UNIVERSITÉ DE BOURGOGNE
Laboratoire LE2I - Ecole doctorale E2S

THÈSE

présentée par **Elie ABI LAHOUD**

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE BOURGOGNE

spécialité : **Informatique**

Composition dynamique de services : application à la conception et au développement de systèmes d'information dans un environnement distribué

Soutenue à Dijon, le 11 février 2010

JURY

Djamal BENSLIMANE	Professeur, Université de Claude Bernard - Lyon 1	Rapporteur
Ernesto DAMIANI	Professeur, Università degli Studi di Milano	Rapporteur
Ahmed LBATH	Professeur, Université de Grenoble - Joseph Fourier	Président
Marinette SAVONNET	Maître de conférences, Université de Bourgogne	Examinateur
Kokou YÉTONGNON	Professeur, Université de Bourgogne	**Directeur**

"Le hasard ne favorise que les esprits bien préparés"

Louis Pasteur (1822-1895)

"Si vous ne pouvez expliquer un concept à un enfant de six ans, c'est que vous ne le comprenez pas complètement"

Albert Einstein (1879-1955)

©Tous droits réservés à l'auteur. Loi sur l'Informatique et les Libertés - 6 Août 2004 - II.2 - Article 9.

"À Pierre, Elham, Georges, Rachelle et Jeannine"

Remerciements

Je tiens à exprimer mes remerciements et toute ma gratitude à Kokou Yétongnon, Professeur à l'Université de Bourgogne, pour avoir accepté de m'encadrer pendant cette thèse ainsi que pour la confiance qu'il m'a accordée. Kokou, je te suis reconnaissant pour le temps et l'effort que tu as consacré à nos discussions.

Je remercie les membres du Jury qui ont accepté d'examiner mon travail : Ernesto Damiani, Professeur à l'Università degli Studi di Milano, de m'avoir accueilli à Crema et d'avoir accepté d'être rapporteur de ma thèse ; Djamal Benslimane, Professeur à l'Université de Claude Bernard - Lyon 1, d'avoir accepté d'être rapporteur de ma thèse ; Ahmed Lbath, Professeur à l'Université de Grenoble - Joseph Fourier, d'avoir accepté de présider le Jury de ma soutenance ; Marinette Savonnet, Maître de Conférences à l'Université de Bourgogne, de son suivi et de ses précieuses remarques.

Je remercie le personnel enseignant et administratif de l'Université de Bourgogne. Merci à Richard Chbeir, Maître de Conférences à l'Université de Bourgogne, qui était à l'origine de mon aventure Dijonnaise et à l'Université Antonine qui m'a accompagné durant mon premier contact avec l'Université de Bourgogne.

Merci à mes collègues, permanents et doctorants, au LE2I - Dijon, plus particulièrement aux membres de l'équipe base de données et à Iwayan, Khalil, Béchara, Joe, Elie, Fekade et Sylvain.

Je tiens à remercier tous les membres de ma famille de leur soutien continu et plus particulièrement mon père, Pierre qui m'a toujours aidé à aller plus loin et qui m'a motivé en étant le modèle à suivre ; ma mère, Elham qui m'a accordé sa confiance, son amour et ses encouragements continus ; mon frère, Georges qui m'a toujours aidé à rebondir et qui m'a inspiré au cours de nos longues discussions ; ma soeur, Rachelle pour son soutien, sa confiance et son sourire et Jeannine mon ange gardien Dijonnais.

Finalement, je remercie tous mes amis et plus particulièrement Mafita, Naty, Baba, Mohanita, Dobrikita et Onomastico.

Résumé

L'orientation service occupe de plus en plus une place importante dans la structuration des systèmes complexes. La conception et le développement d'applications évoluent progressivement d'un modèle traditionnel vers un modèle plus dynamique orienté services où la réutilisation et l'adaptabilité jouent un rôle important. Dans cette thèse, nous proposons une étude portant sur la conception et le développement d'applications par composition de services. Nous décrivons un environnement de partage de services : DyCoSe. Il consiste en un écosystème coopératif où les entreprises membres, organisées en communautés, partagent un consensus global représentant les fonctionnalités métier récurrentes et les propriétés non fonctionnelles communes. La composition d'applications dans DyCoSe repose sur une architecture à trois niveaux combinant à la fois une démarche descendante et une autre ascendante. La démarche descendante permet de décrire l'application à travers une interaction de composants haut niveau et de la raffiner en une ou plusieurs orchestrations de services. La démarche ascendante projette les caractéristiques de l'infrastructure réseau sous-jacente au niveau services. Un processus d'instanciation visant à réaliser une application composite est détaillé. Il formalise le choix des services, selon un ensemble de contraintes données, comme un problème d'optimisation de coûts. Deux solutions au problème d'instanciation sont étudiées. Une solution globale tient compte de l'ensemble des services disponibles dans l'écosystème et une solution locale favorise les services de certaines communautés. Un algorithme génétique est décrit comme implémentation de l'instanciation globale. Une simulation stochastique de l'environnement DyCoSe est proposée. Elle permet d'étudier les possibilités d'instanciation d'une application donnée dans un environnement où la disponibilité des services n'est pas garantie. Elle permet d'étudier aussi, le taux de réussite de l'exécution d'une instance d'une application donnée.

Mots clés : Orientation service - Composition de services - Propriétés non fonctionnelles des services - Optimisation de processus métier - Instanciation - Applications composites - Pair-à-pair.

Abstract

The service oriented computing paradigm plays an increasingly important role in structuring complex systems. Application design and development approaches are witnessing a shift from traditional models towards a more dynamic service oriented model promoting reuse and adaptability. In this thesis, we study an approach for application design and development based on services' composition. We propose DyCoSe, an environment for sharing services. It consists of an enterprise ecosystem wherein members, organized in communities, share a global agreement describing traditional business functionalities and common non-functional properties. DyCoSe relies on a three level architecture for service based application composition combining both, a top down and a bottom up composition approach. The top down part describes an application using high-level components and refines it to an orchestration of services. The bottom up part projects network characteristics to the services' level. An instantiation process aiming at realizing an application is described. It formalizes services' selection as a cost optimization problem considering a set of user given constraints. Two solutions of the instantiation problem are proposed. A global solution considers all the available services in an ecosystem. A local solution gives priority to services of selected communities. A genetic algorithm implements the global instantiation. A stochastic simulation of DyCoSe is proposed. It allows studying the instantiation success rate and the application execution success rate both in a dynamic environment wherein services availability is not guaranteed.

Keywords : Service oriented computing - Service composition - Service non-functional properties - Business process optimization - Instantiation - Service-based applications - Composite applications - Peer-to-peer.

Table des matières

1 Introduction **25**

1.1 Orientation service 26

1.2 Interopération et SOA 28

1.3 Objectifs et contributions 29

1.4 Organisation de la thèse 30

2 État de l'Art **33**

2.1 Description de services 34

2.1.1 Description syntaxique 34

2.1.2 Ajout de la sémantique 37

2.1.2.1 Annotations sémantiques 38

2.1.2.2 Ontologies de services 41

2.1.3 Description comportementale 44

2.2 Découverte de services 47

2.2.1 Localisation de services 48

2.2.1.1 Approches centralisées 48

2.2.1.2 Approches distribuées 49

2.2.2 Niveau d'automatisation 51

2.2.2.1 Approches manuelles et semi-automatiques 52

2.2.2.2 Approches automatisées 52

2.2.3 Matching basé sur le calcul de similarité 54

2.2.3.1 Similarité syntaxique 54

2.2.3.2 Similarité sémantique 56

2.2.4 Autres approches de matching 58
2.3 Composition de services 60
2.3.1 Description de la composition 61
2.3.2 Aspect transactionnel 65
2.3.3 Notion de contexte 66
2.3.4 Composition automatique 67
2.3.4.1 Approches sémantiques 68
2.3.4.2 Automates et réseaux petri 69
2.3.4.3 Techniques de planification 70
2.4 Discussion 71

3 Applications composites : l'environnement DyCoSe 73
3.1 Contexte et Objectifs 74
3.1.1 Modèle du service 75
3.1.2 Pair-à-Pair 77
3.1.3 Objectifs de la thèse 78
3.1.4 Contributions de la thèse 81
3.2 Écosystème de DyCoSe 82
3.2.1 Définition 82
3.2.2 Description 83
3.2.3 Modélisation 86
3.2.3.1 Consensus global 87
3.2.3.2 Consensus local 88
3.2.3.3 Services abstraits 88
3.2.3.4 Services membres 90
3.2.4 Réalisation 91
3.2.4.1 Propriétés de l'écosytème 91
3.2.4.2 Ressources de l'écosytème 92
3.2.4.3 Base de connaissances de l'écosytème 94
3.2.5 Exemple d'un écosystème 94
3.3 Composition dans DyCoSe 97
3.3.1 Niveau fonctionnel 98
3.3.1.1 Couche domaine 98

3.3.1.2 Couche acteurs 100
3.3.1.3 Couche métiers 103
3.3.2 Niveau services 105
3.3.2.1 Couche generic business process 106
3.3.2.2 Couche instances 109
3.3.3 Niveau infrastructure 111
3.3.3.1 Couche réseau virtuel 111
3.3.3.2 Couche réseau physique 119
3.4 Discussion 119

4 Applications composites : déploiement par instanciation 121
4.1 Définitions et notations 122
4.1.1 Graphe d'un GBP 122
4.1.2 Chemin d'exécution d'un GBP 124
4.1.3 Propriétés non fonctionnelles d'un service 127
4.1.3.1 Définition 128
4.1.3.2 Travaux connexes 129
4.1.4 Propriétés non fonctionnelles d'un GBP 130
4.1.4.1 Fonctions d'agrégation 130
4.1.4.2 Contraintes sur un GBP 131
4.2 Problème d'instanciation de GBP 132
4.2.1 Définition d'une instance 132
4.2.2 Définition du processus d'instanciation d'un GBP 133
4.2.3 Exemple d'instanciation 134
4.2.4 Énoncé du problème d'instanciation d'un GBP 137
4.2.5 Instance d'un chemin d'exécution 138
4.2.6 Coût d'une instance d'un chemin d'exécution 139
4.2.7 Choix de l'instance de départ d'un chemin d'exécution 141
4.2.8 Reconstruction d'une instance d'un GBP 143
4.3 Solutions au problème d'instanciation 144
4.3.1 Solution globale 144
4.3.1.1 Exemples de solutions globales 147
4.3.1.2 Protocole et algorithmes d'exécution 148

4.3.2 Solution locale 150
4.3.2.1 Exemple d'une solution locale 152
4.3.2.2 Protocole et algorithmes d'exécution 154
4.4 Discussion 156

5 Implémentation et simulation : le prototype de DyCoSe 159
5.1 Simulations et PeerSim 160
5.1.1 Aperçu sur les types de simulations 160
5.1.2 Librairie de simulation PeerSim 161
5.2 Description du prototype de DyCoSe 163
5.2.1 Composants du niveau infrastructure 165
5.2.1.1 Générateur du réseau physique 165
5.2.1.2 Générateur de réseaux virtuels 166
5.2.1.3 Générateur de pannes 167
5.2.1.4 Moteur de simulation 168
5.2.1.5 Module de communication 170
5.2.2 Composants du niveau services 172
5.2.2.1 Module d'instanciation 172
5.2.2.2 Module d'exécution 172
5.2.2.3 Générateur de GBP 172
5.2.3 Base de connaissances 173
5.3 Analyse de l'instanciation initiale d'une application composite 173
5.3.1 Aperçu sur les algorithmes génétiques 174
5.3.2 Description de l'algorithme génétique implémenté 175
5.3.3 Expérimentation 178
5.3.3.1 Evaluation de l'instanciation globale 178
5.3.3.2 Réussite de l'instanciation d'une application composite 183
5.3.3.3 Impact des communautés sur l'instanciation locale . 185
5.4 Analyse de l'exécution d'une application composite 190
5.4.1 Exécution d'une instance 190
5.4.2 Réussite de l'exécution d'une application composite 192

6 Conclusion **195**
6.1 Discussion 196
6.2 Perspectives futures 199

A Exemple d'interfaces WSDL **201**
A.1 Interface d'un service d'information météo 201
A.2 Interface d'un service de sécrétariat médical 202

B Exemple d'une description comportementale d'un service **207**

C Exemple d'ontologie non fonctionnelle **209**

D Exemple d'une interface d'un service abstrait **211**

E Extraits du prototype **215**
E.1 Extrait du code d'un pair 215
E.2 Extrait du code du générateur de réseau physique 216
E.3 Extrait du code du générateur de réseaux virtuels 218
E.4 Fichier sortie 219

Liste des tableaux

3.1 La table d'état global sur chaque *super-pair* dans le réseau virtuel basé sur la distance physique séparant les nœuds 117

3.2 Exemple des tables d'état local sur les super-pairs (réseau virtuel basé sur la distance) 118

4.1 Table rappelant les principales notations définies 133

4.2 Exemple de services, de liens et des instances possibles 136

4.3 Table résumant les notations supplémentaires proposées 141

4.4 Exemple d'instances et de coûts associés 147

4.5 Exemple de propriétés non fonctionnelles du réseau de la figure 4.11 154

Table des figures

1.1 Modèle d'interaction SOA 29

2.1 Éléments de description d'une interface WSDL d'un service 35
2.2 Illustration des éléments WSDL <*types*> et <*message*> 36
2.3 Illustration des éléments WSDL <*portType*>, <*binding*> et <*service*> 38
2.4 Interface WSDL-S 39
2.5 Éléments d'une ontologie OWL-S 42
2.6 Éléments d'une ontologie WSMO (source [36]) 44
2.7 Résumé comparatif des approches académiques de description sémantique 45
2.8 Exemple d'une interface WSCI d'un service de secrétariat médical . 46
2.9 Composition de services : Orchestration et Chorégraphie 61
2.10 Éléments de description BPEL 2.0 62
2.11 Eléments de description WS-CDL 64
2.12 Description, découverte et composition de services web 72

3.1 Illustration d'un service 76
3.2 Services et applications composites 79
3.3 Deux configurations d'une application composite 80
3.4 Description d'un Écosystème d'Entreprises 85
3.5 Modélisation d'un écosystème d'entreprises 87
3.6 Illustration d'un service abstrait et de ses implémentations 91
3.7 Illustration d'un exemple d'ontologie 95

3.8 Illustration d'un service abstrait 96
3.9 Architecture de composition 98
3.10 Niveau Fonctionnel 99
3.11 Représentation graphique des élements d'un diagrame de processus BPD 100
3.12 Fonctionnalités haut-niveau d'une application d'achat en ligne 101
3.13 Différents acteurs d'une application d'achat en ligne 102
3.14 Variante des différents acteurs d'une application d'achat en ligne . . 103
3.15 Métiers sollicités dans une application d'achat en ligne du point de vue du vendeur 104
3.16 Variante des métiers sollicités dans une application d'achat en ligne du point de vue de l'opérateur 105
3.17 Niveau Services 106
3.18 Exemple d'un Generic Business Process avec regroupement des activités 107
3.19 Exemple d'un Generic Business Process (avec annotations) 109
3.20 Exemple d'une interface simplifiée d'un GBP en BPEL 110
3.21 Exemple d'une interface simplifiée d'une instance d'un GBP en BPEL 111
3.22 Niveau Infrastructure 112
3.23 Exemple d'organisation du réseau virtuel 113
3.24 Exemple d'un réseau virtuel basé sur la distance physique séparant les nœuds 114
3.25 Exemple d'un réseau virtuel basé sur les fonctionnalités fournies par les nœuds 115

4.1 Déduction du graphe d'un GBP décrivant une application 123
4.2 Différentes combinaisons des passerelles (contrôles de flux) 124
4.3 Exemple d'un GBP et des chemins d'exécution associés 125
4.4 Attributs d'un GBP consititué d'un chemin d'exécution 126
4.5 Déduction des probabilités associées aux chemins d'exécution 128
4.6 Exemple de contraintes sur un GBP 131
4.7 Exemple d'un GBP attribué et d'une instance 135
4.8 Représentation de l'espace des instances des services abstraits d'un GBP 138
4.9 Exemple de représentation des variables décisionnelles 142

4.10 Illustration d'un exemple de séquences identifiées pour le calcul d'une solution locale 152
4.11 Exemple d'un réseau virtuel basé sur la distance physique séparant les nœuds 153

5.1 Architecture de composition implémentée par le prototype DyCoSe . 162
5.2 Composants du prototype de DyCoSe 163
5.3 Interface d'entrée des paramètres (alternative au fichier texte de la figure 5.9) 164
5.4 Extrait du générateur de réseaux physiques 165
5.5 Réseaux générés par simulation (cf. annexe E.4) 167
5.6 Extrait du générateur de réseaux virtuels 168
5.7 Communication par files d'attentes 169
5.8 Exécution des composants et phases d'une simulation 170
5.9 Fichier de paramètres PeerSim adopté par DyCoSe 171
5.10 Codage d'une instance 175
5.11 Méthode de croisement simple en un point 176
5.12 Comparaison avec la solution exacte : temps de calcul 179
5.13 Comparaison avec la solution exacte et la génération de départ : coût de l'instance retenue 180
5.14 Comparaison entre deux stratégies de mutations 181
5.15 Écart de AlgoG par rapport à la solution exacte 182
5.16 Taux d'échec de l'instanciation en fonction de la probabilité de départ des pairs 184
5.17 Exemples de distributions de poisson 185
5.18 Exemple d'un GBP et de services membres correspondant 187
5.19 Exemple 1 : jeu de données pour une instanciation 188
5.20 Exemple 2 : jeu de données pour une instanciation 189
5.21 Exemples d'exécution d'une instance 191
5.22 Résultats combinés d'instanciation et d'exécution (100 GBP) 193

E.1 Réseaux simples générés par simulation 220

Chapitre 1

Introduction

Sommaire

1.1 Orientation service **26**
1.2 Interopération et SOA **28**
1.3 Objectifs et contributions **29**
1.4 Organisation de la thèse **30**

1.1 Orientation service

L'orientation service occupe de plus en plus une place importante dans les approches de structuration des systèmes complexes. La notion de service n'est pas nouvelle, pourtant il est difficile de lui donner une définition précise car son utilisation trouve des échos dans plusieurs domaines.

Par exemple, dans le domaine de l'électronique domestique ou l'électroménager, un lecteur de disques fournit un service de lecture de disques, une imprimante fournit un service d'impression, etc. Dans le domaine des télécommunications, un opérateur fournit un service connectivité mettant en liaison deux utilisateurs, un service de messagerie SMS, etc. Dans le domaine de l'informatique l'orientation service est omniprésente. Un hébergeur fournit un service de stockage de données, un système d'exploitation fournit plusieurs services d'abstractions du matériel, etc.

Plus particulièrement, dans le domaine du génie logiciel, la notion de service est un paradigme en cours d'évolution. Elle trouve ses racines dans les méthodologies et les technologies existantes. Historiquement, nous avons assisté à un passage du *modèle procédural* de programmation où la structuration se faisait par fonctionnalité, au *modèle objet* où le principe d'encapsulation permet de regrouper dans un objet des données et les traitements associés. Parallèlement, le *modèle des composants* a suivi cette évolution. Un composant permet d'exposer les fonctionnalités d'une application ou d'un système d'information à travers un ensemble de points d'entrée. L'*orientation service* est une suite logique à ces avancements. Elle consiste à présenter un accès à un ensemble de traitements liés à une fonctionnalité quelconque, sans fournir les détails des traitements. Les données ne sont visibles qu'à travers les messages échangés. Sur une autre dimension, le modèle des *objets distribués* permet de partager des objets à travers un réseau. Cependant, la réutilisation de ces objets est freinée par les dépendances entre les objets comme l'héritage par exemple [42]. Le modèle des *composants distribués* propose de regrouper plusieurs objets inter-dépendants dans un composant distribué facilitant ainsi leur réutilisation. Avec l'affirmation du web grâce à des technologies accessibles à tous les acteurs, le modèle des *services web* fait suite à ces méthodologies et ces technologies de partage.

Intuitivement, la notion de service représente une abstraction des fonctionnalités d'une entité qui peut être aussi simple qu'une donnée ou qu'un objet sur le web ou aussi complexe qu'un système d'information adaptable d'une entreprise. Un service possède une description de la fonctionnalité qu'il accomplit en termes d'opérations

et de leurs paramètres d'entrées et de sorties. Il précise un modèle de communication qui régit les interactions avec d'autres services ou clients. Par exemple, un service de conversion monétaire sur le web fournit une opération *convertir* ayant comme paramètres d'entrée le type de monnaie et la valeur à convertir et comme paramètre de sortie la valeur de la monnaie convertie. Le modèle de communication associé décrit les modalités d'échange de messages avec le service de conversion. Il indique que le service reçoit les requêtes de conversion véhiculées par le protocole HTTP selon un format décrit au préalable. Pratiquement, nous distinguons des services accomplissant des fonctionnalités simples comme par exemple un service de conversion monétaire, un service qui valide un numéro international de code postal ou un service qui récupère un cours de bourse d'une action donnée etc. et d'autres services accomplissant des fonctionnalités plus complexes comme par exemple un service de gestion de voyages, un service de streaming vidéo, etc. Nous constatons que les services accomplissant des fonctionnalités complexes font appel à d'autres accomplissant des fonctionnalités simples.

Plusieurs modèles de services sont étudiés dans la littérature, notamment les services web, les services Grid, les services des réseaux domestiques, etc. Le modèle qui a le plus alimenté le débat autour des services est celui des *services web*. Plusieurs auteurs ont proposé des définitions plus ou moins précises des services web [1]. IBM définit un service web comme "une unité logicielle, autonome et modulaire accessible à travers un réseau pour répondre à des besoins définis et éventuellement pour créer de nouveaux composants logiciels". *Austin et al.* définissent un service web comme "une application identifiée par un URI, dont *l'interface* et le *binding* peuvent être décrits et découverts et qui supporte des interactions directes avec d'autres applications à travers des messages XML véhiculés par des protocoles internet" [7]. Booth *et al.* donnent une définition plus technique d'un service comme un "système applicatif désigné pour supporter une interaction machine-à-machine sur un réseau. Il possède une interface décrite avec un format compréhensible par la machine. Les services interagissent avec un service donné selon des modalités décrites dans l'interface de ce dernier et à travers des messages SOAP véhiculés par HTTP" [16]. Nous retenons de ces définitions, à part les technologies adoptées, trois notions liés aux services : (*i*) l'interface d'un service décrivant la fonctionnalité accomplie, (*ii*) le binding détaillant les messages échangés à travers un protocole de communication et (*iii*) la possibilité d'interaction avec d'autres services.

1. Dans la suite de cette thèse nous utilisons les termes *service* et *service web* de manière interchangeable.

1.2 Interopération et SOA

Traditionnellement l'interopération était centrée sur le partage des données provenant de sources différentes. Elle consistait à créer un schéma global à partir des différents schémas décrivant les données des sources partagées. Ce schéma global joue le rôle d'une interface vis-à-vis de laquelle d'une part, les requêtes des utilisateurs seront évaluées, et d'autre part, les données disponibles seront conciliées. "Un utilisateur a l'impression d'interroger un système centralisé et homogène alors que les sources interrogées sont réparties, autonomes et hétérogènes" [57]. Trois approches sont récurrentes dans la littérature : le GAV, le LAV et le GLAV [51, 78]. Elles adoptent des techniques différentes pour décrire les correspondances entre schémas des sources partagées et le schéma global.

L'approche GAV ou Global As View consiste "à définir le schéma global en fonction des schémas des sources de données partagées" [57]. En d'autres termes le schéma global est défini comme une vue sur les différents schémas sources. La sélection d'une nouvelle source de données ou l'élimination d'une autre entraînent une modification du schéma global. L'approche GAV implique un couplage fort entre les sources de données. Inversement, l'approche LAV ou Local As View consiste à stocker dans les sources distantes les correspondances avec le schéma global sans modifier ce dernier. Chaque source définit sa vue sur le schéma global. De cette manière, les changements dans une source distante n'affectent pas le schéma global ni les autres sources. L'approche LAV implique un couplage faible entre les sources de données. L'approche GLAV ou Global and Local As View combine les deux approches précédemment décrites. Elle emprunte les techniques de l'approche LAV mais utilise les correspondances décrites pour mettre à jour le schéma global. Cette combinaison permet à l'approche GLAV de gagner en pouvoir d'expression du schéma global et de maintenir la flexibilité grâce aux correspondances locales.

Récemment avec l'affirmation de l'orientation service, l'interopération est centrée sur le partage des traitements proposés par les services. La visibilité des données se résume aux paramètres d'entrée - sortie. Le modèle orienté service repose sur un modèle d'interopération SOA. La "Service Oriented Architecture" (SOA) [25, 101], illustrée par la figure 1.1, décrit "l'organisation et les modalités d'utilisation de fonctionnalités distribuées pouvant être sous le contrôle de différents participants" [25, 101]. Elle implique trois types d'acteurs, à savoir, les fournisseurs de services, les registres ou annuaires de services, les consommateurs ou utilisateurs de services. Les fournisseurs décrivent leurs services, les rendent publics ou autorisent un accès

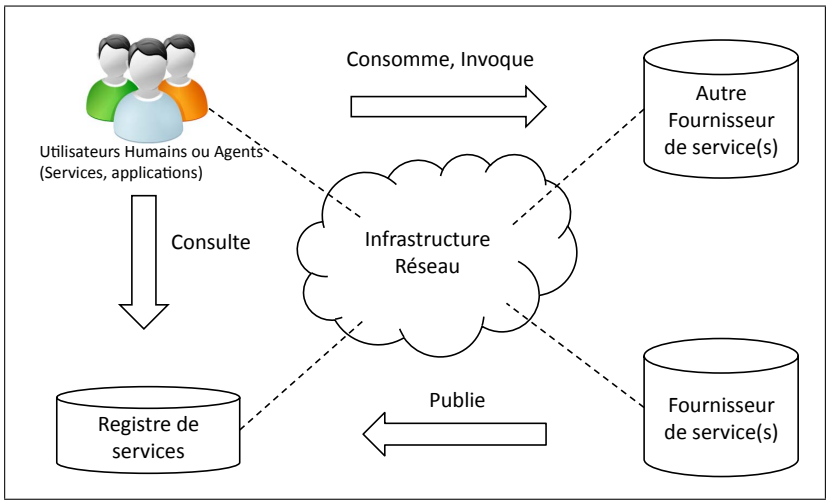


Figure 1.1: Modèle d'interaction SOA

contrôlé et les publient dans un ou plusieurs annuaires. Les annuaires, accessibles à tous les utilisateurs, accueillent les descriptions de plusieurs services. Les utilisateurs découvrent les services répondant à leurs besoins en interrogeant les annuaires. Une fois la correspondance entre un ou plusieurs services et le besoin de l'utilisateur établie, l'utilisateur invoque les services en question à travers les interfaces publiées par leurs fournisseurs respectifs. Le modèle d'interopération SOA met en avant le faible couplage comme caractéristique de l'interaction entre fournisseurs de services. Papazoglou et Georgakopoulos proposent une représentation plus récente de SOA désignée par Extended SOA [111]. ESOA complète SOA pour prendre en compte divers aspects de la composition de services tels que la coordination, la surveillance de l'exécution, la sécurité, etc.

1.3 Objectifs et contributions

L'objectif principal de cette thèse est l'étude de la mise en place des applications composées à base de services dans un environnement distribué. En un premier temps, nous cherchons à définir la structure d'un environnement de partage propice à la création d'applications par composition de services. Ensuite, nous souhaitons définir une démarche visant à décrire une application par une composition de services existants de sorte à favoriser la réutilisation des services partagés. La composition résultante doit tenir compte des propriétés de l'environnement de partage,

notamment du dynamisme qui se manifeste par le départ plus moins contrôlé des fournisseurs de services. Nous cherchons aussi à considérer des critères permettant à un utilisateur d'exprimer ses préférences sur le choix des combinaisons de services. Finalement, nous souhaitons étudier la possibilité de réalisation d'une application donnée face aux changements de l'environnement de partage. Les objectifs de cette thèse sont détaillés davantage en section 3.1.3.

Les contributions de cette thèse se résument comme suit : d'abord, nous avons décrit la structure de l'environnement DyCoSe pour le partage des services. DyCoSe repose sur un écosystème d'entreprises organisées pour contribuer à la composition d'applications. Ensuite, nous avons présenté une architecture à trois niveaux permettant de raffiner la description d'une application composite en vue de l'exprimer en fonction des services disponibles. Puis, nous avons étudié une méthode de déploiement d'applications composites désignée par instanciation. Elle est associée à une approche de composition qui considère les propriétés non fonctionnelles des services. Nous avons formalisé le choix des services en le présentant comme un problème d'optimisation de coûts, auquel nous avons suggéré deux types de solutions. Finalement, nous avons proposé une implémentation du processus d'instanciation et un prototype réalisant la structure de DyCoSe. Il permet de simuler l'instanciation et l'exécution d'une application composite. Les contributions sont détaillées en section 3.1.4.

1.4 Organisation de la thèse

Le reste de ce document est organisé comme suit : le chapitre 2 décrit un état de l'art sur les services web. Il propose une classification en trois parties de la recherche sur les services web et des technologies résultantes. La première partie couvre les modèles de description des services. La deuxième partie présente les approches de découverte de services. La troisième partie décrit les modèles et les techniques récurrentes de composition de services. Le chapitre 3 présente la contribution de ce travail de recherche dans le domaine des applications composites. D'abord, il situe la composition d'applications à base de services vis-à-vis de la recherche sur les services web et la recherche sur les environnements distribués, plus particulièrement le pair-à-pair. Il détaille les objectifs et les contributions de cette thèse. Ensuite, il développe l'environnement de partage de services DyCoSe. Finalement, il décrit une architecture multi-niveaux et une démarche de composition de services associée.

Le chapitre 4 décrit le déploiement d'une application composite par instanciation. D'abord, il propose un ensemble de définitions et une notation formelle représentant une application composite. Ensuite, il présente le processus d'instanciation comme un problème d'optimisation de coûts. Finalement, il décrit deux types de solutions à ce problème. Le chapitre 5 couvre la description d'un prototype permettant de simuler l'instanciation d'un GBP. Il propose une analyse du processus d'instanciation et de l'exécution d'une application composite. Le chapitre 6 conclue ce document par une discussion des travaux. Il souligne les perspectives futures.

Chapitre 2

État de l'Art

Sommaire

2.1 Description de services 34

2.1.1 Description syntaxique 34

2.1.2 Ajout de la sémantique 37

2.1.3 Description comportementale 44

2.2 Découverte de services 47

2.2.1 Localisation de services 48

2.2.2 Niveau d'automatisation 51

2.2.3 Matching basé sur le calcul de similarité 54

2.2.4 Autres approches de matching 58

2.3 Composition de services 60

2.3.1 Description de la composition 61

2.3.2 Aspect transactionnel 65

2.3.3 Notion de contexte 66

2.3.4 Composition automatique 67

2.4 Discussion 71

La recherche dans le domaine du modèle orienté service a accordé une attention significative à trois aspects principaux que nous présentons dans ce chapitre. D'abord dans la section 2.1, nous étudions les modèles de description des services. Nous mettons un accent particulier sur les constructions syntaxiques de représentation des services et leurs extensions sémantiques. Ensuite dans la section 2.2, nous étudions la découverte des services permettant de localiser les services et de les comparer. Nous décrivons en particulier les structures de publication de services et les architectures de découvertes et leurs techniques de correspondances associées. Finalement dans la section 2.3, nous étudions la composition de services. Nous décrivons la coordination entre deux ou plusieurs services en mettant l'accent sur les modèles de description de l'interaction entre services dans la cadre de processus complexes et sur les modèles de composition automatique.

2.1 Description de services

La description d'un service consiste à définir une interface exposant les opérations accomplies par le service et à lier chaque opération à sa réalisation. Dans cette section nous présentons les modèles de description des services. Nous détaillons d'abord le modèle de description syntaxique WSDL qui est devenu le standard incontournable pour les services web. Nous présentons ensuite les diverses approches sémantiques visant à préciser la description d'un service en mettant l'accent sur les approches d'annotation sémantique et sur les ontologies de services. Nous exposons ensuite les modèles de description comportementale d'un service. Finalement nous résumons d'autres approches ne s'inscrivant pas dans la classification proposée.

2.1.1 Description syntaxique

Le *Web Service Description Language* (WSDL) [29] est devenu le standard actuel pour la définition des services. Par le biais d'un document XML, il décrit un service à travers une interface présentant un ensemble d'opérations et leurs paramètres d'entrée et de sortie respectifs. Nous considérons par exemple un service d'information météo qui donne la température dans une ville à une date précise. L'interface WSDL de ce service, fournie en annexe A.1, est un document XML décrivant une opération qui prend en entrée une ville et une date et qui fournit en sortie une température. L'interface WSDL décrit la fonctionnalité accomplie par le service (*ce que le service fait*) mais il ne décrit pas les modalités d'accomplissement de

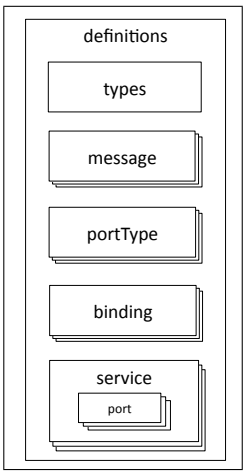
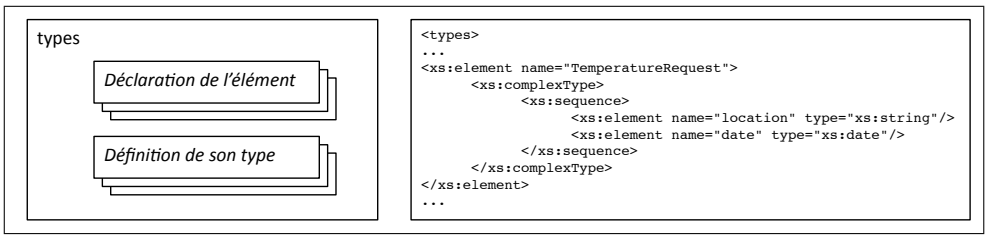


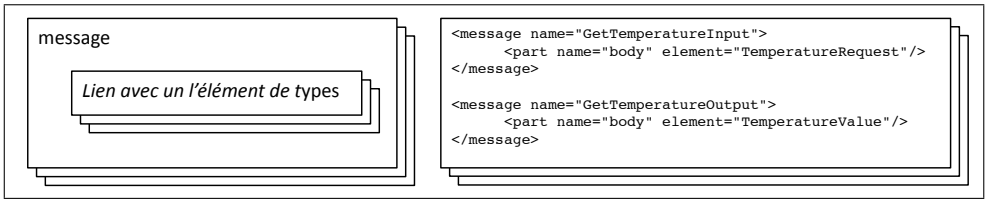
Figure 2.1: Éléments de description d'une interface WSDL d'un service

cette fonctionnalité (*comment le service le fait*). Un document WSDL est un fichier XML constitué de cinq éléments principaux : <*types*>, <*message*>, <*portType*>, <*binding*> et <*service*>. La figure 2.1 illustre la structure d'une interface décrite en WSDL avec les éléments suivants :

- <*types*>, illustré par la figure 2.2a, est un conteneur définissant les données figurant dans les messages échangés par le service. Il définit à travers un schéma XML les types de ces données. Les données peuvent être simples comme par exemple une température qui prend une valeur décimale ou complexes comme par exemple l'entrée *TemperatureRequest* d'un service d'information météo. Cette entrée est un donnée de type complexe qui est constituée d'un élément *location* et d'un élément *date* tel illustré par la figure 2.2a.
- <*message*>, illustré par la figure 2.2b, définit les messages échangés par le service. Il est composé d'un ensemble de <*part*>. Un <*part*> est associé à une donnée décrite dans <*types*>. Par exemple, le message *GetTemperatureInput* de la figure 2.2b est composé d'un élément *part* relié à *TemperatureRequest* défini dans <*types*>.
- <*portType*>, illustré par la figure 2.3a, désigne un ensemble d'*operations*. Une <*operation*> est une description abstraite d'une action accomplie par le service. Elle contient un sous-élément <*input*> et un sous-élément <*output*>. Ils décrivent les paramètres d'entrée et de sortie de l'opération. Un <*input*> ou un



(a) Illustration de l'élément WSDL <*types*>



(b) Illustration de l'élément WSDL <*message*>

Figure 2.2: Illustration des éléments WSDL <*types*> et <*message*>

<*output*> possède un attribut *message* qui référence un élément <*message*>. Par exemple dans la figure 2.3a, le *TemperaturePortType* décrit l'opération *GetTemperature* à travers le message d'entrée *GetTemperatureInput* et le message de sortie *GetTemperatureOutput*.

– <*binding*>, illustré par la figure 2.3b, reprend les opérations de l'élément <*portType*> et leurs associe un protocole de transfert de message. Concrètement, le <*binding*> décrit comment les messages seront échangés. Il spécifie un seul protocole d'échange mais il ne précise pas les adresses entre lesquelles les messages seront échangés. La spécification WSDL décrit le <*binding*> avec les méthodes GET et POST du protocole HTTP et présente un protocole d'échange plus élaboré pour HTPP, le protocole SOAP [18]. Par exemple la figure 2.3b illustre le <*binding*> de *TemperaturePortType* du service d'information météo avec le protocole SOAP. L'élément <*binding*> possède deux attributs. L'attribut *name* définit le nom du binding. Il est égal à *TemperatureSoapBinding* dans cet exemple. L'attribut *type* indique le <*portType*> pour qui le <*binding*> a lieu. Il est égal à *tns :TemperaturePortType* dans cet exemple. L'élément *soap :binding* possède aussi deux attributs. L'attribut *style* qui peut être égal à *rpc* ou *document*. Il précise si l'échange a lieu à travers un appel à une procédure distante ou à travers un envoi - réception d'un document. Dans cet exemple un *document* est utilisé. L'attri-

but *transport* définit le protocole réseau qui va véhiculer les messages SOAP. Dans cet exemple HTTP est utilisé.

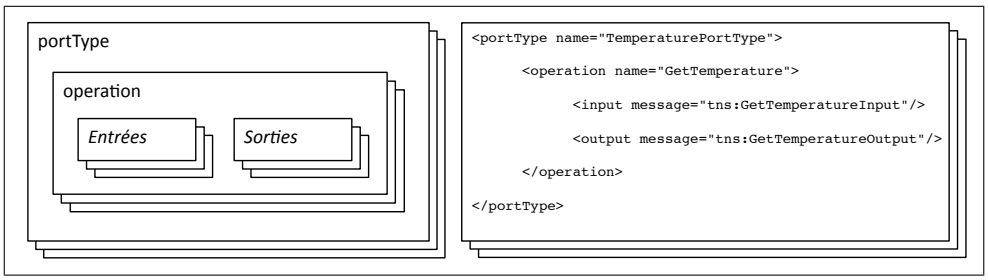
– <*service*>, illustré par la figure 2.3c, précise l'adresse ou les adresses auxquelles se trouve le service. Un <*service*> est un ensemble de <*port*>. Un <*port*> spécifie une adresse pour un <*binding*> donné. Par exemple, la figure 2.3c définit le service *TemperatureService* à travers un port, *TemperaturePort*, associé au binding *tns :TemperatureSoapBinding*. Elle précise l'adresse de *TemperaturePort* qui a comme valeur "http ://example.com/temperature".

En résumé, les <*portType*> décrivent les opérations du service en terme d'entrées et de sorties. Les <*binding*> décrivent concrètement les messages échangés par les opérations à travers un protocole choisi. Le <*service*> associe à chaque <*binding*> une adresse à laquelle se trouve l'implémentation.

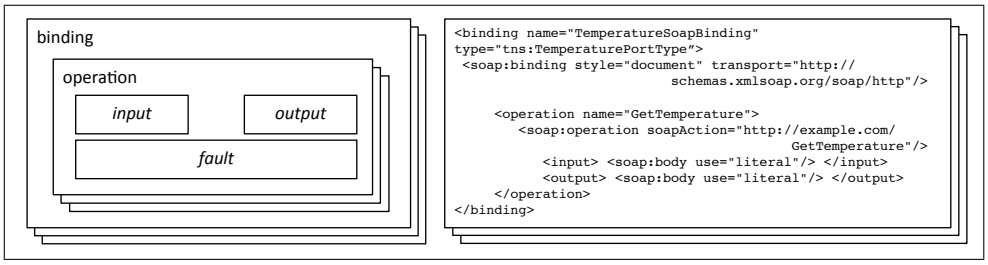
Chronologiquement IBM, Microsoft et Ariba ont développé WSDL 1.0 en 2000. WSDL 1.1 est publié en mars 2001 comme une note W3C [30]. WSDL 1.2 est une nouvelle version simplifiée, mais n'a pas été à ce jour validée par W3C. En juin 2007, le WSDL 2.0 est publié comme une recommandation W3C. WSDL 2.0 n'est autre que la spécification WSDL 1.2 dont le nom a changé vu les différences majeures qui le démarque de WSDL 1.1. Nous retenons les changements suivants : (*i*) <*portType*> est devenu <*interface*>, (*ii*) <*port*> est renommé en <*endpoints*>, (*iii*) <*message*> n'est plus utilisé, un <*xs :element*> suffit pour définir les données échangées.

2.1.2 Ajout de la sémantique

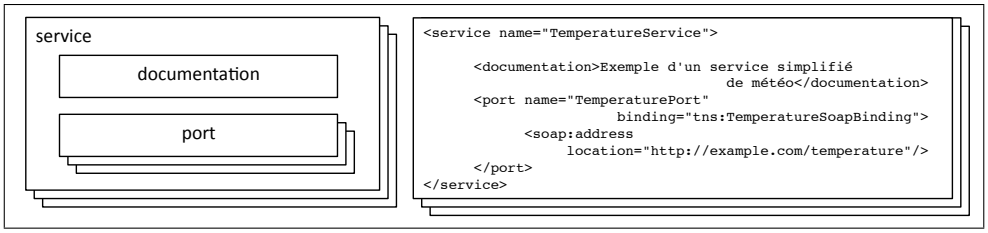
Le standard WSDL présente des lacunes de précision dans la description d'un service. Ces lacunes sont liées au niveau d'expressivité faible de la description syntaxique car elle est limitée à l'énumération des opérations et à la description des types des paramètres d'entrée et de sortie associés. Elle ne caractérise pas la sémantique de la fonctionnalité accomplie par le service. Pour pallier le manque de sémantique de WSDL, plusieurs approches proposent de rajouter une couche au dessus de WSDL complétant la description syntaxique par des précisions sémantiques. Par exemple le service web d'*Amazon.com*, décrit en WSDL, a pour objectif la vente de livre. Il permet à un utilisateur de parcourir les catalogues d'*Amazon*, pour rechercher des livres et les acheter. Cependant, un utilisateur pourra s'en servir pour parcourir les catalogues en guise de recherche d'information, sans acheter. Syntaxiquement, le terme *rechercher* est différent du terme *acheter*, cependant sémantiquement ces



(a) Illustration de l'élément WSDL <*portType*>



(b) Illustration de l'élément WSDL <*binding*>



(c) Illustration de l'élément WSDL <*service*>

Figure 2.3: Illustration des éléments WSDL <*portType*>, <*binding*> et <*service*>

deux concepts sont liés, le premier étant un pré-requis du second [86]. Une recherche par mots clés d'un service permettant de *rechercher un livre* ne retournera pas les services décrits comme permettant d'*acheter un livre*, par suite l'utilisateur ne sera pas informé de tous les services potentiellement pertinents. Une subtilité de ce genre sera contournée si le service fourni une description sémantique plus élaborée qu'un fichier WSDL.

2.1.2.1 Annotations sémantiques

L'annotation sémantique consiste à enrichir et à compléter la description d'un service. Elle établit des correspondances entre des éléments de la description et des concepts d'un ensemble d'ontologies de référence. Une ontologie de référence permet

```
<wssem:category name="payment" taxonomyURI="http://myTaxonomy.edu" taxonomyCode="1414" />

<operation name = "getpaymentOrder" wssem:modelReference="myOntology#RequestPaymentOrder">

      <input message = "tns:paymentOrderRequest"/>
      <output message = "tns:paymentOrderDispatch"/>

      <wssem:precondition name="ExistingPurchase" wssem:modelReference="myOntology#PurchaseExists"/>
      <wssem:effect name="paymentOrderDispatchEffect" wssem:modelReference="myOntology#wait4payment"/>

</operation>
```

Figure 2.4: Interface WSDL-S

de représenter un domaine par des structures interprétables par une machine. Trois modèles principaux suivent l'approche d'annotation sémantique, à savoir WSDL-S, SAWSDL et METEOR-S. Les deux premiers modèles permettent d'annoter manuellement une description WSDL avec des éléments faisant référence à des ontologies tandis que le dernier permet de générer les annotations de l'interface d'un service à partir des annotations du code source de son implémentation.

WSDL-S C'est une spécification commune à IBM et au laboratoire LSDIS. Elle a été soumise au W3C en 2005 [4]. Son objectif principal est de fournir un processus d'annotation sémantique compatible avec les technologies existantes. À cette fin, Akkiraju *et al.* proposent le *WSDL-S meta-model* [93]. Concrètement, ce méta-modèle étend le WSDL en rajoutant trois éléments majeurs <*category*>, <*precondition*>, <*effect*> et deux attributs *modelReference* et *schemaMapping*. Les éléments introduits permettent de rajouter des informations qui n'étaient pas prises en compte dans WSDL comme les pré-conditions et les effets d'une opération. Tandis que les attributs permettent de référencer des concepts dans des ontologies de référence. Nous illustrons dans la suite les annotations WSDL-S à travers l'exemple de l'opération *getPaymentOrder* présentée dans la figure 2.4. Elle représente une demande de facture adressée à un service de facturation. Elle prend en entrée le message *paymentOrderRequest* qui contient la demande de facture et retourne en sortie le message *paymentOrderDispatch* contenant la facture. Les extensions WSDL-S indiquée en gras sur la figure 2.4 sont les suivantes :

– L'élément <*category*> est un sous-élément de <*portType*>. Il précise la catégorie d'un service lors de la publication dans un annuaire ou registre. Par exemple, pour l'opération *getPaymentOrder* de la figure 2.4, <*category*> établit la relation avec *payment* (code 1414) de la taxonomie disponible à

l'adresse "http ://myTaxonomy.edu".

- L'élément <*precondition*>, sous-élément de <*operation*>, indique les préconditions à vérifier pour que l'opération s'exécute comme prévu. Par exemple, pour l'opération *getPaymentOrder* de la figure 2.4, <*precondition*> indique que pour qu'un ordre de paiement puisse être sollicité, il faut que l'achat associé existe.
- L'élément <*effect*>, sous-élément de <*operation*>, indique les effets de l'exécution de l'opération. Par exemple, pour l'opération *getPaymentOrder* de la figure 2.4, <*effect*> indique que l'effet de l'envoi d'un ordre de paiement est l'attente d'un paiement référencée par *wait4payment*.
- L'attribut *modelReference* peut être ajouté à un <*xs :element*> dans une grammaire XML et aux éléments <*operation*>, <*precondition*> et <*effect*>. Il indique une association avec un élément correspondant dans une ontologie donnée. Par exemple, pour l'opération *getPaymentOrder* de la figure 2.4, l'attribut *modelReference* établit le lien avec le concept *RequestPaymentOrder* de l'ontologie *myOntology*.
- L'attribut *schemaMapping* peut être ajouté à un <*xs :element*> dans une grammaire XML. Il permet de décrire les correspondances entre la grammaire annotée et les ontologies de référence.

SAWSDL Le *Semantic Annotations for Web Services Description Language and XML Schema*, recommandé par W3C en avril 2007, est une des approches les plus récentes initiée par la communauté des services web sémantiques [45]. Il présente un mécanisme permettant d'annoter sémantiquement les services décrits avec WSDL et leurs schémas XML associés. Les auteurs affirment que SAWSDL ne spécifie pas un langage pour représenter les modèles sémantiques, mais plutôt il "fournit un mécanisme à travers lequel des concepts appartenant à des modèles sémantiques existants peuvent être référés à partir d'un document WSDL 2.0". SAWSDL propose des extensions à WSDL 2.0 similaires à celles proposées par WSDL-S. Sa particularité réside dans l'annotation supplémentaire des schémas XML. Les principales extensions permettant d'annoter un document WSDL 2.0 sont les attributs suivants : *modelReference*, *liftingSchemaMapping* et *loweringSchemaMapping*. L'attribut *modelReference* permet d'annoter tous les éléments WSDL 2.0. En particulier, il figure comme attribut de <*interface*>, <*operation*> et <*fault*>. Il pointe vers le concept

équivalent en rajoutant son adresse. Nous reprenons par exemple, l'opération *getPaymentOrder* qui prend en entrée un message de demande de facture *paymentOrderRequest* et retourne en sortie un message *paymentOrderDispatch* contenant la facture. Pour associer cette opération avec le concept *RequestPaymentOrder* de l'ontologie *myOntology*, il suffit d'ajouter à l'élément définissant l'opération l'attribut suivant *sawsdl :modelReference="myOntology#RequestPaymentOrder"*. Les attributs *liftingSchemaMapping* et *loweringSchemaMapping* permettent d'associer à un *schema type* ou à un *element* un concept dans une ontologie de référence adoptée. Des prototypes d'environnement de travail implémentant SAWSDL, pour décrire et manipuler les services, émergent [71, 87].

METEOR-S Le projet METEOR-S [5] initié par le laboratoire LSDIS vise l'intégration des technologies de services avec celles du web sémantique. Il ne fournit pas un nouveau modèle de représentation des services mais propose de générer des annotations sémantiques pour les fichiers de type WSDL. METEOR-S est basé sur le projet METEOR qui signifie Managing End To End OpeRations. Ce dernier traite de la gestion des workflows de processus à grande échelle dans des environnements hétérogènes. L'idée principale de METEOR-S est de générer une annotation sémantique de l'interface d'un service à travers des références à une ontologie ajoutées manuellement dans le code source de l'implémentation du service [5, 113, 117, 135]. Les auteurs présentent un prototype permettant de générer des interfaces de services en (*i*) WSDL 1.1 où les liens avec les concepts d'une ontologie de référence sont décrits avec les éléments XML d'extensibilité autorisés par la spécification du langage ou en (*ii*) WSDL-S à travers les attributs *modelReference*, *schemaMapping* et l'élément *<category>*.

2.1.2.2 Ontologies de services

Une ontologie de services saisit les différents aspects liés à la description des services et leur utilisation à travers un ensemble de concepts, de propriétés et de relations entre eux. Deux modèles d'ontologies de services sont décrits ci-après : OWL-S et WSMO.

OWL-S : Ontology Web Language for Services OWL-S désigné par DAML-S dans les versions antérieures [86], est une ontologie de description des services web, dont les objectifs sont de résoudre les ambiguïtés et de rendre la description

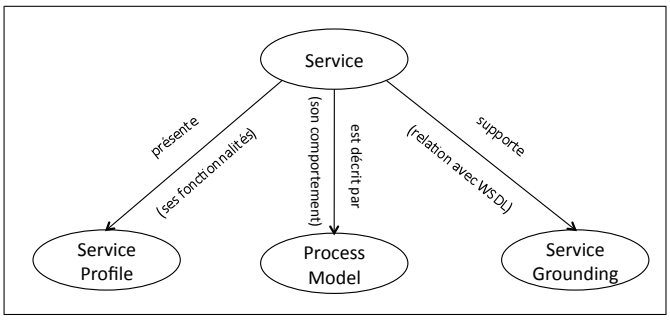


Figure 2.5: Éléments d'une ontologie OWL-S

d'un service compréhensible par une machine. Ankolekar *et al.* [128] présentent une ontologie pour les services web dans le but d'automatiser la découverte, l'invocation, la composition et la surveillance de l'exécution des services. Les auteurs reprennent la notion de classes d'OWL [1] et proposent l'ontologie OWL-S. La figure 2.5 décrit le premier niveau de l'ontologie OWL-S qui présente une structure tripartite. Elle est composée d'un *service profile*, d'un *service grouding* et d'un *process model*. Ces trois concepts permettent de décrire la fonctionnalité accomplie par le service, de relier cette description avec la description WSDL du service et de décrire le comportement d'un service lors une interaction avec d'autres services.

"Le *service profile* fournit une description haut niveau d'un service et de son fournisseur" [72, 128, 138]. Le *service profile* contient trois types d'information : (*i*) une description du service et de son fournisseur, (*ii*) le comportement fonctionnel du service et (*iii*) un ensemble d'attributs comme la catégorie du service. Une liste de ces attributs est disponible dans [84, 128]. Les auteurs soulignent qu'un utilisateur peut interagir avec un service d'une façon non prévue par les concepteurs du service, comme par exemple un service de vente de livre qui peut servir pour récupérer des informations concernant le livre sans l'acheter. Par conséquent en utilisant un OWL-S profile les concepteurs peuvent décider de publier seulement la fonctionnalité *achat de livre* pour réduire la charge ou publier en plus la fonctionnalité *rechercher un livre* pour augmenter le nombre d'utilisateurs potentiels. Dans [35, 84, 108] les auteurs étendent la classe *service profile* en ajoutant la possibilité d'exprimer les préconditions et les effets de chaque opération.

Le *process model* définit plusieurs types de processus décrivant l'ordonnancement des opérations et le comportement d'un service dans une interaction. La description

1. précédemment DAML+OIL [9]

comportementale d'un service est abordée en 2.1.3 et la composition de plusieurs services est décrite en 2.3. Nous retenons qu'un service selon OWL-S est modélisé comme une entité désignée par processus atomique. Ce dernier est un sous-concept de *process model*. Le processus atomique décrit un service en terme d'entrées, de sorties, de pré-conditions et d'effets.

Le *service grounding* décrit les modalités de communication avec un service en précisant le protocole, le format des messages, la sérialisation et l'adressage. Il est le résultat d'une correspondance entre le processus OWL-S décrivant le service et la description WSDL originale. Finalement, nous soulignons qu'OWL-S était soumis au W3C en Novembre 2004 comme une proposition s'intitulant "a semantic markup for web services". Les détails sont fournis dans [84, 85]. Jusqu'au jour de rédaction de cette thèse, la proposition n'est pas encore validée comme recommandation W3C.

WSMO : Web Service Modeling Ontology L'ontologie WSMO est présentée par Roman *et al.* dans [123] . Initiée par l'ESSI WSMO workgroup [153], elle est basée sur le Web Service Modeling Framework WSMF proposé par Fensel et Bussler [46]. WSMO partage avec OWL-S l'objectif d'automatiser les tâches liées aux services. La séparation entre la description des services web sémantiques et les technologies exécutables est un aspect essentiel de WSMO. En effet WSMO fournit un modèle de description d'ontologies indépendant du langage utilisé pour les décrire. Le diagramme de classes UML, illustré par la figure 2.6, est extrait de [36]. Il représente les éléments principaux d'une ontologie WSMO et leurs interactions. Le concept central est le *WSMO element* qui se spécialise en une *ontology* ou un *webService* ou un *goal* ou un *mediator*. Dans la suite nous présentons les éléments *ontology* et *webService* permettant de décrire les services. Les éléments *goal* et *mediator* traitent de la découverte. Ils sont présentés en section 2.2.3.2.

Un élément *ontology* désigne une ontologie de référence importée par un élément de l'ontologie WSMO afin de décrire ses propriétés. Un élément *webService* est "une unité de calcul capable de répondre à une requête utilisateur" [123]. Les *webService* WSMO sont définis d'une manière uniforme comprenant les éléments suivants : la fonctionnalité du service, les interfaces de description, les propriétés non fonctionnelles, les ontologies importées et les médiateurs utilisés. La fonctionnalité du service, désignée par *capability*, est décrite en terme d'opérations associées à des pré-conditions, hypothèses (*assumptions*), post-conditions et effets. Les pré-conditions et les post-conditions décrivent les exigences sur les données avant l'exécution du

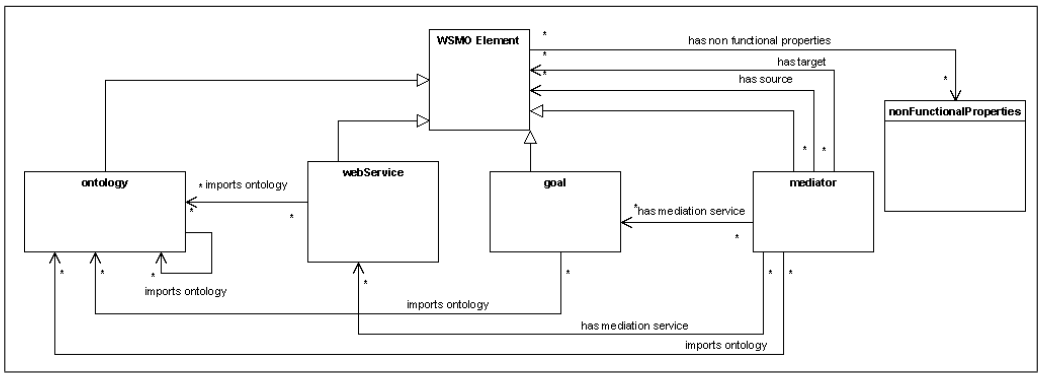


Figure 2.6: Éléments d'une ontologie WSMO (source [36])

service et les changements qu'elles subissent après exécution ; les hypothèses et effets décrivent les exigences et les changements relatifs aux services en interaction avec le service décrit. Les interfaces de description décrivent le comportement du service, en d'autres termes l'ordonnancement des opérations. Les propriétés non fonctionnelles d'un webService décrivent les attributs qui ne se rapportent pas directement aux données traitées, comme par exemple la durée d'exécution d'une opération. Les ontologies importées sont utilisées pour référencer les éléments de la description du service. Les médiateurs sont utilisés si deux webServices en interaction importent deux ontologies différentes.

Nous proposons dans la figure 2.7 un résumé et une comparaison la richesse sémantique des trois approches académiques principales de description sémantique des services, à savoir les ontologies de services OWL-S et WSMO et l'approche d'annotation METEROR-S et son environnement de travail.

2.1.3 Description comportementale

La description comportementale consiste à décrire l'ordre d'invocation des opérations d'un service. Un service peut avoir plusieurs descriptions comportementales tel que chaque description est une vue sur le comportement possible du service dans une interaction donnée. La littérature étudie plusieurs approches de description comportementale.

WSCI Le Web Service Choreography Interface [146] décrit, par le biais d'un document XML, l'ordre des échanges de messages d'un service à travers un ensemble

	(Absent - Mauvais - Bon)	OWL-S	WSMO	METEOR-S
Définition - Description	**Intégration avec WSDL**	**Bon**, à travers le « service grounding »	**Mauvais**, pas encore complètement défini	**Bon**, car l'objectif principal est d'annoter un fichier WSDL
	L'ontologie principale illustre les aspects essentiels des services	**Bon**, définit une ontologie avec quatre concepts de haut niveau (service, service profile, service model et service grounding)	**Bon**, définit un méta modèle basé sur une ontologie avec quatre composants (ontologies, web services, goals et mediators)	**Absent**, l'objectif est d'annoter le fichier WSDL vis à vis des ontologies fournies
	Le modèle supporte les ontologies externes	**Absent**, tous les éléments nécessaires à la description d'un service sont fournis dans l'ontologie OWL-S. Aucun mécanisme d'extension n'est fourni	**Bon**, à tous les niveaux des « imported ontologies » peuvent être référencées et les médiateurs sont utilisés pour résoudre les conflits	**Bon**, l'annotation est faite vis à vis des ontologies fournies (externes)
	Le modèle permet de définir des propriétés non fonctionnelles	**Mauvais**, certains travaux complémentaires étudient les possibilités d'intégration des propriétés non fonctionnelles dans le « service profile »	**Bon**, chaque composant de WSMO a des propriétés non fonctionnelles.	**Mauvais**, les propriétés non fonctionnelles seront annotées sémantiquement si elles sont fournies dans le fichier WSDL de départ
	Le modèle supporte la médiation	**Absent**, owl-s ne supporte pas directement la médiation. Les médiateurs ne sont pas décrits par l'ontologie, ils font partie de l'infrastructure sous-jacente	**Bon**, quatre types de médiateurs sont décrits par WSMO	**Absent**, le Framework de METEOR-S consiste à annoter les services, la médiation n'est pas abordée
	Le modèle garantit une indépendance entre les modules de l'ontologie principale	**Mauvais**, l'ontologie est basée sur la « service class » de laquelle sont dérivés « service, service profile, service model et service grounding » mais ils ne peuvent pas être développés indépendamment	**Bon**, un des objectifs de WSMO est la modularité qui permet de définir les ressources séparément et indépendamment des interactions possibles	**Absent**, aucune ontologie principale n'est définie
	Le modèle décrit l'interaction avec d'autres services	**Bon**, le « service process model » propose des processus composites abstraits mais aussi exécutables	**Bon**, l'interface d'un « WSMO web service » peut être représentée par une chorégraphie et son interaction avec d'autres services par une orchestration	**Bon**, le « METEOR-S Web Service Composition Framework (MWSCF) » est basé sur le « METEOR-S Composer » pour créer des processus
	Expressivité du langage sémantique et puissance de raisonnement logique	**Bon**, OWL et SWRL (pour la logique de type Horn)	**Bon**, WSML supporte la logique descriptive et la F-Logic à travers cinq variantes : WSML-Core, WSML-DL, WSML-Flight, WSML-Rule et WSML-Full	**Mauvais**, supporte seulement OWL
Implémentation	**Prototypes et outils disponibles**	Une liste d'outils est disponible à l'adresse suivante : http://www.daml.org/services/owl-s/tools.html	Le « WSMO studio » est disponible à l'adresse suivante : http://www.wsmo.org/wsmo_tools.html	Le projet Meteor-S est le premier à proposer un plug-in complet pour Eclipse, cependant il est limité à l'annotation seulement
	Contribution à la découverte et l'indexation de services	Le « OWL-S service profile » a ouvert la voie à la nouvelle tendance de décrire sémantiquement les services Web et par suite à automatiser la découverte	WSMO différencie un service accompli d'un service web. La description des « capabilities » d'un service a marqué une nouvelle étape dans la description sémantique des services Web	Une découverte semi automatique est possible après annotation des services

Figure 2.7: Résumé comparatif des approches académiques de description sémantique

de séquences d'opérations. WSCI permet de définir plusieurs comportements d'un même service selon le contexte d'invocation. WSCI définit des processus complexes à travers les messages échangés par un ensemble de services. Cet aspect qui relève de la composition de services est développé en section 2.3.1. Afin d'illustrer la description comportementale d'un service en WSCI, nous prenons l'exemple de l'interface

```
<correlation name = "appointmentCorrelation"
          property = "tns:appointmentID”>
  </correlation>

  <interface name = "MedicalSecretary">
    <process name = "assignAppointment” instantiation = "message">
     <sequence>
       <action name = "ReceiveAppointmentRequest” role = "tns:MedicalSecretary"
                                        operation = "MedicalSecretarytoPatient/getAppointment">
       </action>
      <action name = "ReceiveDateConfirmation" role = "tns:MedicalSecretary”
                                        operation = "MedicalSecretarytoPatient/confirmAppointment”>
         <correlate correlation="tns:appointmentCorrelation"/>
       </action>
  </interface>
```

Figure 2.8: Exemple d'une interface WSCI d'un service de secrétariat médical

WSDL d'un service représentant l'activité d'un secrétariat médical, décrit en annexe A.2. Ce service présente trois opérations : *getAppointment*, *confirmAppointment* et *getPrescription*. La figure 2.8 illustre une interface WSCI associée. Elle précise l'ordre d'invocation des opérations *getAppointment* et *confirmAppointment* dans le cadre d'un échange de messages entre un patient et le service de secrétariat médical visant à un fixer un rendez-vous. En effet, l'interface *MedicalSecretary* définit un processus *assignAppointment* constitué d'une séquence de deux actions : *ReceiveAppointmentRequest* qui est liée à l'opération *ReceiveAppointmentRequest* de l'interface WSDL et *ReceiveDateConfirmation* qui est liée à l'opération *confirmAppointment*. L'action *ReceiveDateConfirmation* définit une corrélation à travers l'élément *appointmentCorrelation*. Cette corrélation sert à établir le lien avec l'opération qui a précédé à travers la propriété *appointmentID*.

WSCL Le Web Services Conversation Language [147] décrit les conversations auxquelles un service peut participer. Une conversation est un échange de messages entre services. WSCL décrit les documents XML échangés et les séquences d'opérations impliquées d'une façon similaire à un diagramme d'état, sans décrire les détails au niveau du protocole d'échange. La spécification WSCL propose un ensemble d'éléments XML et leurs attributs associés pouvant être utilisés en conjonction avec WSDL. De nos jours des langages plus élaborés sont proposés. Ils reposent sur des modèles formels comme l'algèbre de processus, les Petri Nets, etc. Nous les décrivons en section 2.3.

OWL process model L'ontologie OWL-S définit le concept processus simple comme sous-concept de *process model*. Les processus simples sont utilisés pour

décrire l'interaction avec un service à un niveau conceptuel, sans obligation d'avoir un lien avec un document WSDL (*grounding*). Par conséquent ces processus ne sont pas invocables. Un processus simple peut être utilisé pour fournir une vue sur un processus atomique, comme une chorégraphie détaillant les possibilités d'interaction avec le service décrit par le processus atomique. Dans ce cas nous disons que le processus simple est réalisé par le processus atomique associé [84, 128].

WSMO webService Interface L'ontologie WSMO associe à chaque élément *webService* des interfaces pour représenter son comportement. Les interfaces WSMO d'un *webService* sont représentées par une chorégraphie ou par une orchestration. La chorégraphie selon WSMO représente les messages échangés entre un service et son utilisateur pour accomplir la fonctionnalité du service. WSMO différencie la chorégraphie d'exécution et la méta chorégraphie : la première définit un protocole d'interaction et la deuxième définit un protocole de négociation et de suivi d'exécution. Scicluna *et al.* proposent pour WSMO un modèle de représentation de l'échange basé sur les machines abstraites d'états (ASM) [124]. L'orchestration décrit la façon selon laquelle un service interagit avec d'autres. Cet aspect qui relève de la composition de services est décrit en section 2.3.

Nous soulignons que certaines approches de découverte et de composition automatique de services proposent de modéliser le comportement d'un service par un automate comme par exemple, Bordeaux *et al.* qui suggèrent de représenter le comportement du service par un système de transition d'états, où le service est décrit par un ensemble d'états reliés par des actions [17]. Nous présentons ces modèles en section 2.3.4.2.

2.2 Découverte de services

La *découverte de services* est abordée dans un grand nombre de projets et travaux de recherche. Elle est définie par Keller *et al.* comme la "localisation automatique des services répondant à une requête utilisateur" [69]. Booth *et al.*, décrivent le processus de découverte comme étant "la localisation d'une description compréhensible par la machine d'un service éventuellement inconnu au préalable et correspondant à certains critères fonctionnels" [16]. Toma *et al.* définissent la découverte comme "le processus qui prend en entrée une requête utilisateur et retourne une liste de ressources ou services pouvant combler éventuellement le besoin décrit" [140] . Trois

aspects majeurs de la découverte ressortent de ces définitions. Premièrement, l'aspect localisation des services qui consiste à trouver l'adresse à laquelle est fournie la description d'un service. Deuxièmement, l'aspect traitement des données qui indique le degré d'automatisation de la découverte. Troisièmement, l'aspect mise en correspondance ou "matching" qui compare la requête utilisateur aux services répertoriés.

Dans cette section nous présentons d'abord les différentes approches de localisation de services basées sur des modèles centralisés ou des modèles distribués. Ensuite nous classifions les approches de découverte selon le degré d'automatisation du traitement des données. Puis nous étudions la correspondance ou "matching" basé sur le calcul de similarité. Finalement, nous concluons la section en présentant d'autres approches de découverte.

2.2.1 Localisation de services

La découverte de services consiste, en premier, à localiser les fournisseurs proposant des services répondant à une requête utilisateur. Les approches de localisation de services récurrentes dans la littérature sont classées en deux catégories, à savoir les approches centralisées et les approches distribuées.

2.2.1.1 Approches centralisées

Le modèle du UDDI, Universal Description Discovery and Integration [10] s'est imposé comme le modèle de référence des approches centralisées de publication et découverte de services. En Juillet 2004, UDDI sous ses deux versions UDDI 2.04 [10] et UDDI 3.0 [12], est devenu un standard OASIS [2]. UDDI définit un modèle de représentation des données et des méta-données nécessaires à la découverte des services. Un annuaire UDDI (*UDDI registry*) est un ensemble de catalogues fournissant des mécanismes de classification et de gestion des services pour faciliter leur découverte et leur invocation. Un UDDI peut appartenir à un domaine public comme internet ou tout autre réseau accessible à un nombre non limité d'utilisateurs, comme il peut appartenir à un domaine restreint comme l'intranet d'une entreprise ou d'un groupe d'entreprises. Les structures de données d'un UDDI sont les suivantes :

- <*businessEntity*> contient les informations décrivant une organisation fournissant des services.

2. Organization for the Advancement of Structured Information Standards [47]

- <*businessService*> décrit une collection de services fournis par une organisation. Un <*businessService*> est contenu dans une <*businessEntity*>.
- <*bindingTemplate*> définit les informations techniques nécessaires pour invoquer un service. Un <*bindingTemplate*> est contenu dans un <*businessService*>.
- <*tModel*> définit un concept représentant un type de services, un protocole utilisé par les services ou une catégorie de services. Un élément <*tModel*> est "réutilisable" dans la mesure où il peut être référencé par plusieurs <*bindingTemplate*>.
- <*publisherAssertion*> définit un lien entre la <*businessEntity*> qui la contient et une autre <*businessEntity*>. Quand deux éléments <*businessEntity*> référencent la même <*publisherAssertion*>, on parle de relation *relationship* entre ces <*businessEntity*>.

Dans [136], les auteurs étendent le modèle UDDI pour prendre en compte des descriptions OWL-S. L'objectif est de permettre le stockage des fonctionnalités des services désignées par *capabilities* afin de rendre la découverte plus précise. Pour les auteurs argumentent, bien que UDDI permet une grande variété de recherches par catégories (par nom, localisation, métier ou autres), UDDI est limité à la recherche par mots clés, ne permettant aucun raisonnement ni inférence par rapport à des taxonomies sur la possibilité de correspondance entre une description et une requête utilisateur. Ils présentent *OWL-S Matchmaker*, un composant qui assure le raisonnement logique nécessaire pour inférer la correspondance entre le *profil* d'un service publié par UDDI et le *profil* d'un besoin décrivant la requête utilisateur. Ils décrivent une implémentation basée sur jUDDI [3] et le raisonneur Racer [4]. Quelques résultats préliminaires sont proposés. Cette approche bien que riche sémantiquement ne traite pas l'aspect automatisation de la découverte et repose sur un UDDI centralisé.

2.2.1.2 Approches distribuées

Les premières approches distribuées de découverte de services consistaient à mettre en place une fédération d'annuaires UDDI [126]. Dans [126], les auteurs proposent de fournir un service qui agit comme une couche d'abstraction reliant plusieurs instances UDDI à accès public. Chaque instance de UDDI est responsable

3. Implémentation en java du modèle UDDI, `http://ws.apache.org/juddi/`
4. `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`

des données concernant les services auxquels elle fait référence. Des procédures de réplication de données sont définies pour garantir la cohérence des données entre les nœuds. Des problèmes d'invalidité de liens pointant vers les services ont été constatés, dus notamment à la mise en cache statique.

Verma *et al.* dans [143] présentent une infrastructure pair-à-pair extensible, d'annuaires pour la publication de services sémantiques et leur découverte intitulée METEOR-S Web Service Discovery Infrastructure (MWSDI). Cette infrastructure n'est pas limitée au modèle UDDI. Elle permet la mise en relation de plusieurs modèles d'annuaires propriétaires. L'infrastructure présente quatre couches :

- La couche "données" contient un ensemble d'annuaires UDDI pointant vers les services.
- La couche "communication" est le réseau pair-à-pair sous-jacent abritant les annuaires, les services et les clients. Les nœuds sont classés en quatre catégories : *Operator Peer*, *Gateway Peer*, *Auxiliary Peer* et *Client Peer*.
- La couche "services d'opérateurs" contient l'ensemble des services fournis par les *Operator Peers*. Si ces derniers implémentent un annuaire différent de UDDI, ils doivent en fournir les méta-données et les APIs permettant de les interroger.
- La couche de "spécifications sémantiques" est en relation avec les trois précédentes. Elle contient un ensemble d'ontologies de domaine et les correspondances entre elles. Les éléments de la couche de spécifications sémantiques servent à annoter les annuaires et les services pour automatiser l'interaction et rendre la découverte plus précise.

Cette infrastructure a été implémentée dans un prototype, Speed-r, présenté dans [133].

Dans [134], les auteurs se basent sur le modèle MWSDI, présenté dans [143] et décrit ci-dessus, pour mettre en place une fédération d'annuaires publics et privés à la fois. Ils partent du principe qu'une entreprise peut être amenée à partager son infrastructure d'annuaires avec ses partenaires pour mettre en évidence et partager ses services. Ils présentent l'ontologie XTRO (Extended Registries Ontology) regroupant des ontologies d'annuaires et des fédérations d'annuaires et de domaines, permettant ainsi de décrire les relations entre ces différents éléments. La particularité de ce travail réside dans le fait qu'il tient compte à la fois de l'aspect sémantique

du processus de publication - découverte et de la possibilité d'interaction entre les annuaires publics et privés.

La spécification UDDI 3.0.2, dernière version standardisée en Février 2005 [11], reprend le principe d'une fédération d'annuaires UDDI. Elle décrit un annuaire UDDI comme un ensemble de nœuds *UDDI nodes*. Les nœuds d'un annuaire collaborent pour gérer un ensemble de structures de données UDDI. Un nœud fait partie d'un seul annuaire. Chaque nœud possède une copie répliquée du schéma global de la structure de données de l'annuaire pour permettre une meilleure gestion des requêtes qui lui sont adressées.

Paolucci et al. présentent dans [109] une approche distribuée pour la découverte de services. Les services sont supposés être décrits par des *service profiles* de l'ontologie OWL-S. Le protocole pair-à-pair adopté pour la recherche d'information est le Gnutella. En combinant ces deux modèles, OWL-S et Gnutella, les auteurs estiment pouvoir allier la précision de description de l'approche OWL-S et l'efficacité de la technique de recherche et de localisation Gnutella. Les détails de l'implémentation ne sont pas discutés.

Dans [130], Schmidt et Parashar présentent un système d'indexation et de stockage pair-à-pair. L'approche d'indexation est basée sur la fonction de correspondance *Hilbert Space-Filling Curve*. Après avoir partitionné l'espace de données, ils utilisent le protocole CHORD pour répartir les données dans un réseau virtuel visant à faciliter la recherche. Les données sont naturellement les descriptions des services fournis sur le réseau. Après mise en place et évaluation du système, les auteurs affirment que leur modèle présente des capacités de recherche et de localisation en temps réel et qu'il garantit que "tous les éléments correspondant à un besoin seront trouvés s'ils sont fournis sur le réseau" et cela en un nombre de messages et un coût limités. Nous résumons la technique de mise en correspondance proposée en section 2.2.3.1.

2.2.2 Niveau d'automatisation

Selon le niveau de richesse sémantique de la description des services, le traitement des données lors de la découverte peut nécessiter une intervention humaine. Dans la suite nous résumons, en premier, les approches manuelles et semi-automatiques de traitement de données et ensuite nous décrivons les approches automatisées exigeant des descriptions plus riches sémantiquement.

2.2.2.1 Approches manuelles et semi-automatiques

La découverte basée sur la recherche de descriptions dans un annuaire UDDI repose sur une comparaison entre mots clés. C'est une approche manuelle dans la mesure où l'utilisateur devra contacter le service de l'annuaire et parcourir les résultats renvoyés afin de sélectionner le service le plus pertinent. La recherche par mots clés nécessite l'intervention de l'utilisateur pour vérifier la pertinence des résultats (faux synonymes, etc.).

L'approche de découverte proposée dans le cadre de METEOR-S, présente un degré d'automatisation plus avancé qu'une simple recherche dans un annuaire. En effet, l'annotation sémantique des fichiers de description des services permet une découverte semi-automatique et plus précise. L'utilisateur exprime sa requête en fonction des concepts et des instances de l'ontologie de référence utilisée pour annoter les descriptions des services. Le système pré-sélectionne les services susceptibles de satisfaire la requête de l'utilisateur. Ce dernier intervient pour choisir le service qui lui semble le plus pertinent parmi ceux pré-sélectionnés.

Dans une approche plus récente, Sabou et Pan [129] étudient les lacunes des systèmes de découverte de services proposés dans la littérature. Ils présentent les annuaires disponibles en ligne. Ils concluent que la recherche de services dans ces annuaires n'est pas suffisamment précise car elle consiste à faire correspondre les mots clés d'une requête avec une classification des services publiés. Ils ajoutent qu'il est difficile de maintenir à jour les classifications des services car ils évoluent rapidement. Ils proposent alors les bases d'une approche semi-automatique pour la découverte de services qui repose sur des techniques empruntées au web sémantique. Elle consiste, d'abord, à adapter les techniques d'apprentissage à base d'ontologies pour permettre une mise à jour automatique des schémas de classification des services. Des ontologies seront générées à partir de ces schémas. Un utilisateur "expert" peut intervenir à ce niveau pour vérifier la cohérence des ontologies générées. Elle consiste, ensuite, à réutiliser les techniques de matching entre ontologies pour comparer ces schémas et les intégrer en méta-schémas. Elle utilise, finalement, des méthodes de visualisation pour représenter les services par rapport aux méta-schémas générés.

2.2.2.2 Approches automatisées

Benatallah *et al.* présentent une approche pour automatiser la découverte des services [13]. Ils proposent de formaliser la découverte comme un problème de ré-

écriture de concepts en logique descriptive. Ils décrivent un algorithme pour découvir la meilleure couverture sémantique d'une requête par les concepts d'une ontologie donnée. Ils démontrent que le problème de découverte de la meilleure couverture sémantique présente des similitudes avec le problème de calcul, à coût minimal, des transversales minimes d'un hypergraphe attribué (*computing the minimal transversals with minimum cost of a weighted hypergraph*). Les résultats expérimentaux sont présentés dans [20]. Les auteurs affirment que l'approche décrite n'est pas encore adaptée pour traiter un grand nombre d'ontologies hétérogènes, néanmoins les premières expériences présentées sont prometteuses.

Mendell et McIlraith dans [83] présentent une approche proposant d'automatiser la découverte de services. Elle réutilise les technologies service existantes et les combine avec celle du web sémantique. Les auteurs proposent d'étendre BPEL [100]. Dans le document BPEL [5], une description OWL-S de type *service profile* remplace la référence statique aux services impliqués. Ils mettent en place SDS, un service de découverte sémantique décrit par un *service profile* OWL-S. SDS utilise le raisonneur JTP (Java Theorem Prover) pour génèrer des requêtes DQL [6]. Au cours de l'interprétation du document BPEL, le service SDS intervient et envoie des requêtes DQL générées à partir des *service profile* du process BPEL vers un annuaire contenant des services sémantiquement décrits en OWL-S. De cette façon chaque service sera découvert juste avant son invocation durant l'exécution du processus composite.

Keller *et al.* étudient un modèle de localisation automatique de services [68]. L'étude suppose que les services sont décrits via le modèle WSMO. La technique de matching utilisée est choisie en fonction de la richesse de la description des services fournis. Les auteurs adoptent la démarche de découverte de WSMO basée sur le matching entre un objectif *goal* et le raffinement de la requête de l'utilisateur en *intention*. Nous développons le matching sous WSMO dans la section suivante. Le modèle proposé traite à la fois l'aspect statique et l'aspect dynamique de la découverte. L'aspect statique consiste à trouver les correspondances entre les descriptions des services et une requête utilisateur. L'aspect dynamique raffine le choix des services pré-sélectionnés durant la phase statique en interrogeant le fournisseur pour vérifier certaines conditions d'utilisation telles que les pré-conditions, les effets et la disponibilité, etc. L'aspect architectural n'est pas abordé par cette étude, les auteurs ne discutent pas la structure des annuaires publiant les fonctionnalités des

5. Langage de description d'une composition de service sous forme d'une orchestration, décrit en section 2.3.1.

6. DAML Query Language, `http://www.daml.org/dql/`

services.

2.2.3 Matching basé sur le calcul de similarité

Le matching basé sur la similarité consiste à établir des correspondances entre deux ou plusieurs services en calculant des valeurs de similarité indiquant le degré de rapprochement entre leurs descriptions respectives. Le calcul de similarité peut être basé sur des données syntaxiques ou sémantiques plus expressives. Dans la suite nous résumons les approches décrites dans la littérature.

2.2.3.1 Similarité syntaxique

Dans [43], Ernst *et al.* étudient les problèmes de substituabilité et de composabilité des services. En supposant que les informations concernant des exécutions passées des services sont disponibles, ils proposent un algorithme en deux phases. À l'issue de la première phase, l'algorithme décide si les deux services sont composables ou non. Sinon, il les marque en tant qu'éventuellement substituables. Pendant la première phase l'algorithme teste la correspondance entre les valeurs des paramètres entrée d'un service et les valeurs des paramètres de sortie de l'autre. Ces valeurs sont extraites d'un journal d'exécutions passées. Si la correspondance est positive, les deux services comparés sont marqués comme composables. Pendant la deuxième phase si les services sont considérés comme équivalents, les opérations similaires sont marquées comme interchangeables après découverte de leurs paramètres d'entrées - sorties et génération des éventuelles correspondances. L'algorithme teste les entrées entre elles et les sorties entre elles pour confirmer ou infirmer la substituabilité. Les résultats présentés sont prometteurs. Les auteurs insistent sur la précision de leur algorithme : "il ne trouve pas de faux positifs". Ils prévoient de pousser les tests en utilisant des services concrets avec des valeurs pour les entrées et les sorties provenant des cas d'utilisation réels. De plus, ils envisagent d'adapter une technique d'apprentissage machine pour améliorer la correspondance entre les paramètres et détecter la substituabilité non symétrique entre deux services. La substituabilité non symétrique entre un service A et un service B implique que A peut remplacer B mais pas inversement.

Dans [40], Dong *et al.* présentent Woogle un moteur de recherche de services basé sur la similarité syntaxique. Woogle permet à l'utilisateur de rechercher des services similaires à un service donné, des services ayant les mêmes entrées qu'un

service donné (respectivement sorties) et des services composables avec un service donné. L'algorithme présenté étudie la similarité entre les descriptions textuelles des opérations des services et entre les identifiants des paramètres d'entrée - sortie. Il est basé sur une technique de *clustering* qui, en fonction de leurs entrées et sorties, relie les services à des concepts. Pour décider sur la substituabilité ou la composabilité, le système compare les concepts rattachés aux services. L'algorithme de *clustering* est basé sur l'heuristique suivante : "*les paramètres ont tendance à indiquer le même concept s'ils apparaissent souvent ensemble*". Des règles d'association sont définies entre les identifiants des paramètres. La *règle d'association* entre deux identifiants t_1 et t_2 est désignée par $t_1 \rightarrow t_2\,(s, c)$ où le support s est la probabilité que le paramètre t_1 apparaît dans une entrée une ou sortie et la confidence c est la probabilité que le paramètre t_2 apparaît dans une entrée ou une sortie sachant qu'elle contient t_1. Ces règles ne sont pas symétriques : $t_1 \rightarrow t_2$ et $t_2 \rightarrow t_1$ peuvent avoir des valeurs de supports et de confidences différentes. Les auteurs utilisent, en plus, des techniques de fusion (*merging*) et de scission (*splitting*) pour raffiner le clustering et garantir une meilleure cohésion. Finalement, une autre heuristique est utilisée pour éliminer les bruits. Un terme rattaché à un concept est considéré comme bruit si "*dans la moitié de ses occurrences il n'est pas associé à d'autres termes du même concept*". Les expériences conduites ont permis aux auteurs de retenir deux conclusions : (*i*) une condition de cohésion trop rigide empêche les grands clusters étroitement associés de fusionner et (*ii*) une fusion prématurée peut être inappropriée car elle empêchera une éventuelle fusion ultérieure. Woogle, le moteur de recherche présenté supporte en plus de la recherche par mots clés deux types de recherche plus avancée basés sur l'algorithme de clustering : le *template search* et le *composition search*. Le premier récupère les paramètres de l'utilisateur et au lieu de lancer la recherche par mots clés, il les considère comme une opération d'un service et applique l'algorithme de clustering pour la recherche d'un équivalent (un substitut). Le second type de recherche retourne l'éventuelle composition de services qui répond au besoin de l'utilisateur si ce besoin n'est pas satisfait par un des services existants.

Tel que décrit précédemment en section 2.2.1.2, Schmidt et Parashar présentent une technique d'indexation basée sur la fonction de correspondance *Hilbert Space-Filling Curve* [130]. La courbe de Hilbert est une courbe continue de forme irrégulière remplissant un plan. Elle partage le plan en plusieurs zones appelées *clusters*. Elle peut être utilisée pour créer une correspondance entre un espace de dimension d et un autre de dimension 1. Une des particularités de cette courbe est sa propriété de préserver la localité qui se traduit par le fait suivant : deux points proches dans

l'espace de dimension 1 (la courbe) sont les correspondances de deux points proches dans l'espace de dimension d. Schmidt et Parashar adoptent cette technique et la combine avec la technique de répartition des données CHORD pour créer les clusters de description des services. En résumé, la description de chaque service est associée à des mots clés. Les correspondances entre l'espace multi-dimensionnel des mots clés et un espace de dimension 1 sont créées avec la technique de *Hilbert Space-Filling Curve*. Les descriptions des services sont réparties sur les pairs du réseau virtuel selon le protocole CHORD à travers leurs index générés dans l'espace de dimension 1.

2.2.3.2 Similarité sémantique

Les approches de calcul de similarité sémantique reposent sur des modèles de description sémantique. Par la suite nous résumons les deux approches majeures, basées respectivement sur le modèle OWL-S et sur le modèle WSMO.

Approches basées sur OWL-S Dans [108], Paolucci *et al.* présentent une approche de matching sémantique entre des services définis avec OWL-S. Ils décrivent un algorithme qui assure la correspondance entre les profiles des services décrits par des *service profile* et le besoin de l'utilisateur défini aussi par une structure *service profile*. Une fonction de classement permet de différencier les profils grâce à plusieurs degrés de correspondance. L'algorithme consiste d'abord à comparer les paramètres de sortie des profils publiés avec ceux du profil requis ensuite il compare les paramètres d'entrées des profils publiés avec ceux du profil requis. Le service publié est considéré équivalent à la description recherchée si ses paramètres de sortie et ses paramètres d'entrée coïncident respectivement avec ceux du profil recherché. Les auteurs dénotent trois niveaux de *match*. Le premier est le match *exact*. Il a lieu si les paramètres de sortie du profil recherché sont équivalents ou constituent une sous-classe des paramètres de sortie du profil publié. Le second niveau de match est le *plugIn*. Il a lieu si les paramètres de sortie du profil recherché sont englobés (*subsumed*) par les paramètres de sortie du profil publié. Le troisième degré de match est le *subsumes*. Il a lieu quand les paramètres de sortie du profil requis englobent (*subsume*) les paramètres de sortie du profil publié. Dans ce cas le service publié répond partiellement au besoin recherché. Finalement les auteurs notent que le processus de matching décrit n'est pas symétrique. En effet, $Match(A, B) <> Match(B, A)$ parce que l'ordre dans lequel les paramètres d'entrée et de sortie sont comparés

est important. Le calcul de matching est basé sur la distance minimale entre les concepts dans l'ontologie. Le tri final des résultats est basé sur les valeurs du matching des entrées. Dans la même étude [108], Paolucci *et al.* proposent une approche pour appliquer leur modèle au *UDDI tmodels*. Ils comparent leur approche avec des techniques basées sur des agents utilisant le *case based reasoning*, affirmant que ces dernières extraient les informations de correspondance que le système a déjà "appris" tandis que leur approche "découvre et déduit" le match.

Benatallah *et al.* dans [13, 14] étendent le modèle de Paolucci *et al.* pour aller au delà de la simple notion de *subsumption*. Ils présentent une approche de ré-écriture de concepts basée sur la notion de meilleure couverture et sur l'opération de différence entre concepts. Étant donné un profil de service recherché désigné par *requête* Q et une *ontologie* T, l'objectif est de découvrir la meilleure combinaison de services satisfaisant au maximum les sorties de Q. Ils désignent ce processus par "la meilleure couverture de Q en utilisant T".

Approche basée sur WSMO Dans [69], Keller *et al.* présentent deux méthodes pour la découverte de services basées sur le matching sémantique entre les descriptions WSMO des services. La différence majeure entre les méthodes réside dans la richesse sémantique des descriptions des services et des objectifs (*goals*). La première méthode est basée sur des descriptions simplifiées tandis que la deuxième traite des descriptions complexes, plus complètes et permettant de représenter de façon plus réaliste les aspects métier. Dans WSMO, la requête d'un utilisateur recherchant un service est décrite par la structure *goal*. Un médiateur *WG Mediator*, intervient durant le processus de découverte pour pré-lier les services web aux objectifs *goals*. La première méthode de matching basée sur WSMO utilise une approche de modélisation basée sur les ensembles *set-based modeling* pour formaliser la requête de l'utilisateur et la description du service comparé en termes de sorties et effets. Les auteurs désignent par R_G l'ensemble des éléments significatifs à prendre en compte dans la description de la requête utilisateur et par R_W l'ensemble des éléments significatifs à prendre en compte dans la description du service. Ils définissent quatre niveaux de match. (i) Un *exact-match* a lieu quand R_G est équivalent à R_W. (ii) Un *subsumption-match* a lieu quand R_G est un sous-ensemble de R_W. (iii) Un *plugin-match* a lieu quand R_W est un sous-ensemble de R_G, c'est-à-dire quand le service répond partiellement au besoin de l'utilisateur. Finalement (iv) un *intersection-match* a lieu quand la description du service et celle du besoin ont au moins un élément en commun. Cette méthode présente des aspects similaires à celle proposée

par Paolucci *et al.* [108]. La deuxième méthode est une application de la première sur des services et des besoins dont la description est sémantiquement plus riche. Les auteurs redéfinissent les quatre niveaux de match et étendent la définition formelle des critères de correspondance en se basant sur la logique de transaction (*transaction logic*). Afin de valider ces deux méthodes, le groupe de travail WSMO en propose une réalisation en logique descriptive non restreinte à un langage spécifique. Des exemples en SHIQ et SHIQ(D) sont fournis et une proposition de réalisation en WSML-Full (langage développé par le groupe de travail de WSMO) [37] est discutée.

Keller *et al.* proposent également dans [69], la notion d'intentions comme un raffinement applicable aux deux méthodes décrites ci-dessus. Ils définissent les ensembles I_G et I_W telles que $I_G \subseteq R_G$ et $I_W \subseteq R_W$. En effet, I_G désigne l'intention réelle de l'utilisateur. Ce dernier peut être réellement intéressé par une partie de sa requête. I_W représente l'intention du fournisseur. Ce dernier peut fournir réellement une partie des éléments mentionnés dans la description de son service.

2.2.4 Autres approches de matching

D'autres approches de matching sont étudiées dans littérature. Certains proposent de classer les services dans des communautés selon leur adéquation fonctionnelle, d'autres modélisent le comportement d'un service par des systèmes de transition d'états et les comparent.

Dans [139] Taher *et al.* proposent le concept de communautés de services. Ils définissent la communauté comme un groupe de services adressant la même fonctionnalité. La communauté facilite le regroupement des services en fournissant une interface unique pour les descriptions. Elle sert de point d'entrée à tous les services de la communauté. Selon les auteurs, trois éléments définissent une communauté : (*i*) une interface commune représentée par un service abstrait, (*ii*) un ensemble de services concrets réalisant la fonctionnalité décrite par le service abstrait, (*iii*) un ensemble de correspondances *mappings* entre les opérations des services concrets et celles du service abstrait. Pour implémenter cette approche, les auteurs définissent un annuaire de services abstraits sous un format UDDI. Cet annuaire associe à chaque service abstrait (AWS) une catégorie et un ensemble de services concrets. Ils définissent le OSC Open Service Connectivity driver, composant de gestion des interactions entre les clients et une communauté. Il offre une liste de commandes désignées par les méthodes suivantes : *Find()*, *GetDetail()*, *GetListOfCWS()* et *Bind()*. Elles permettent l'interaction entre le client et les communautés. Par exemple le client ou

l'application cliente utilise la méthode *Find(F1)* pour rechercher une fonctionnalité *F1*. *Find(F1)* retourne le service abstrait *AWS* correspondant à la fonctionnalité *F1*. *GetListOfCWS(AWS)* retourne la liste des services concrets rattaché à AWS. *GetDetail(CWS)* permet de récupérer les informations nécessaires au *binding*[7] du service concret *CWS*. Les auteurs ne développent pas l'implémentation des fonctions décrites. La mise en place des communautés est brièvement discutée. Les auteurs indiquent que le fournisseur d'un service concret établit des règles permettant d'identifier les correspondances entre son service et un service abstrait.

Nous soulignons que Zeng *et al.* adoptent le même principe de communautés de services dans une étude de composition de services [158]. Ils ne précisent pas les mécanismes de correspondance ou de calcul de similarité.

Dans [17], Bordeaux *et al.* suggèrent de représenter le comportement du service par un système de transition d'états. Ils modélisent le service par un ensemble d'états reliés par des actions. Les auteurs précisent que ce modèle est déterministe dans la mesure où "une action qui a lieu à un état donné conduit à un état determiné". En supposant que les interactions entre les services sont synchrones, les auteurs proposent trois définitions de compatibilité et deux définitions de substituabilité. La première définition de compatibilité affirme que deux services sont compatibles s'ils ont deux comportements opposés. En d'autres termes si la sortie de l'un est égale à l'entrée de l'autre : $emissions(s1) = receptions(s2) \wedge emissions(s2) = receptions(s1)$. La deuxième définition généralise la première en remplaçant la relation d'égalité par l'opérateur ensembliste inclusion : $emissions(s1) \subseteq receptions(s2) \wedge emissions(s2) \subseteq receptions(s1)$. La troisième définition de la compatibilité considère deux services comme compatibles si "l'état initial n'est pas une impasse (*deadlock*) et s'il existe au moins une exécution conduisant à une paire d'états finals". La première définition de substituabilité est liée au contexte de composition. Elle considère qu'un service *A'* est un substitut potentiel à un service *A* participant dans une composition avec un service *B*, si *A'* est compatible avec *B*. La deuxième définition de substituabilité est plus générique. Elle considère qu'un service *A'* est équivalent à un service *A*, si *A'* est compatible avec tous les services avec lesquels *A* est compatible. Les deux premières notions de compatibilité sont applicables aux deux définitions de substituabilité, tandis que la troisième notion de compatibilité, qui nécessite la connaissance d'un partenaire d'interaction, est applicable seulement à la première définition de substituabilité liée

7. au sens WSDL du terme, cf. 2.1.

au contexte.

2.3 Composition de services

Martin *et al.* définissent la composition comme "le processus de sélection, de combinaison et d'exécution de services en vue d'accomplir un objectif donné" [86]. La composition de services désigne une interaction entre deux ou plusieurs services en vue d'accomplir des objectifs déterminés. Un exemple type est la composition séquentielle entre deux services $S_1(i_1, o_1)$ où i_1 et o_1 désignent respectivement l'entrée et la sortie de S_1 et $S_2(i_2, o_2)$ où i_2 et o_2 désignent respectivement l'entrée et la sortie de S_2. Elle consiste à invoquer S_1 puis S_2, tel que S_2 prend comme entrée i_2 la sortie o_1 de S_1. S_1 et S_2 doivent coordonner leurs activités pour que S_2 reçoive le message contenant o_1.

- Du point de vue de S_1, il doit s'exécuter, envoyer le résultat de son exécution, et attendre un accusé de réception. Du point de vue de S_2, il doit être en attente d'un message en entrée. Un fois le message reçu il doit envoyer un accusé de réception et s'exécuter avant de renvoyer sa sortie o_2 si nécessaire. Ces deux points de vue représentent des descriptions comportementales des services S_1 et S_2 tel décrit en section 2.1.3.
- Du point de vue de l'acteur qui exécute la composition, une séquence de deux étapes a lieu. La première étape correspond à l'invocation de S_1. Ce dernier reçoit une entrée i_1 et s'exécute. Une fois son exécution terminée, la deuxième étape commence par l'exécution de S_2. L'exécution de S_2 se termine par une sortie o_2. Cette vision est communément désignée dans la littérature par *orchestration* de services. L'orchestration illustrée par la figure 2.9, décrit l'ordre dans lequel les opérations ont lieu dans une composition de services. Papazoglou & Dubray définissent l'orchestration comme un processus multi-étapes impliquant un certain nombre de *transactions* [112].
- Du point de vue d'un acteur externe, S_1 reçoit un message, il s'exécute et envoie un message contenant o_1 à S_2. Ce dernier envoie un message à un S_1 pour accuser la réception de o_1 et s'exécute à son tour. Cette vision est communément désigné dans la littérature par *chorégraphie* de services. La chorégraphie illustrée par la figure 2.9, exprime une vue d'ensemble des services interagissant dans le cadre d'une composition de services. Selon Peltz,

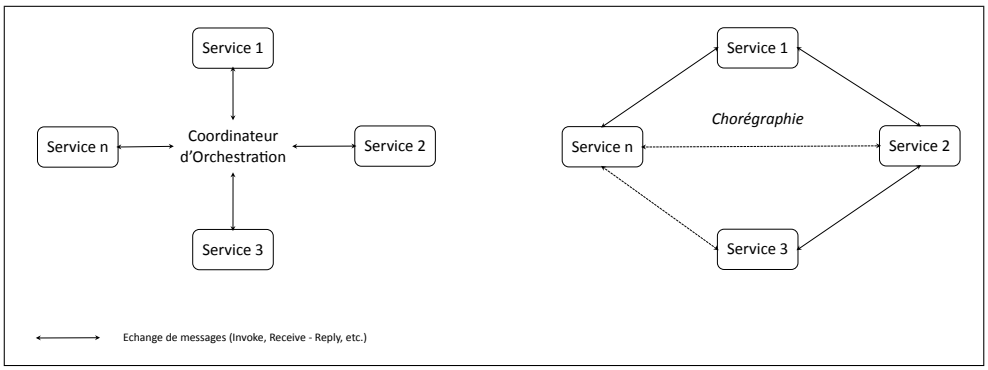


Figure 2.9: Composition de services : Orchestration et Chorégraphie

la chorégraphie illustre les différents échanges de messages entre les participants [114].

La composition requiert la description et l'organisation de l'interaction entre les services. Elle nécessite la gestion de plusieurs aspects comme les échanges de données entre les services, les pannes ou erreurs éventuelles, le contexte d'interaction, le degré d'automatisation des tâches, etc. Dans la littérature, une variété de spécifications, de langages et d'approches formelles ont étudié la composition. Dans la suite nous présentons les deux approches principales de description de la composition en section 2.3.1. Nous soulignons l'aspect transactionnel d'une composition de service en section 2.3.2. Nous mettons l'accent sur le rôle de la description du contexte d'interaction en section 2.3.3. Finalement nous classifions un ensemble d'approches de composition automatique.

2.3.1 Description de la composition

La littérature traitant de la composition des services comporte une multitude d'approches visant à décrire l'interaction entre les services. Dans cette section nous exposons les deux approches principales de description de la composition.

Le **Business Process Execution Language**, désigné par l'acronyme BPEL [60, 100], repose sur le Web Services Flow Language (WSFL) de IBM et le Web Services for Business Process Design (XLANG) de Microsoft. WS-BPEL 2.0 est devenu un standard OASIS en 2004. C'est un langage permettant de décrire une composition de services par un processus métier d'une façon similaire à un *workflow*. BPEL fournit des blocs structurels pour définir le type d'interaction entre deux ou plusieurs

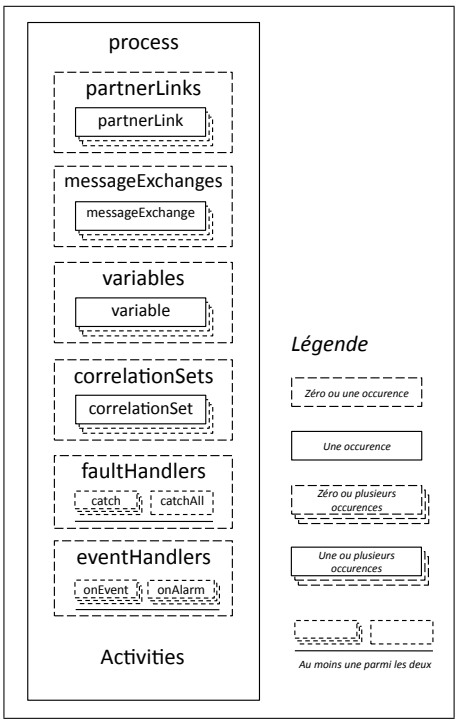


Figure 2.10: Éléments de description BPEL 2.0

composants services. Il décrit le flux d'information et l'ordre d'invocation des composants services. Il permet de définir une composition de services par un *processus abstrait* ou par un *processus exécutable*. Le premier décrit un processus métier de haut niveau. Il illustre les échanges de messages entre les différents composants services, sans obligation de révéler le comportement interne. Il constitue une description publique de la composition [100]. Tandis que le second illustre l'ordre d'exécution des activités, les partenaires invoqués, les messages échangés et les mécanismes de gestion des erreurs éventuelles.

Concrètement, un document BPEL propose un ensemble d'éléments XML représentant les activités impliquées et les séquences de contrôles entre elles. Dans la suite nous décrivons ces différents éléments qui sont illustrés par la figure 2.10. Notre objectif n'est pas de résumer la spécification BPEL mais d'illustrer la façon dont BPEL représente une composition de services. L'élément <*partnerLinks*> contient un ou plusieurs sous-éléments <*partnerLink*>. Un <*partnerLink*> permet d'établir un lien avec un service. Parmi ses attributs, *myRole* précise le rôle du processus BPEL et *partnerRole* indique le rôle du service ou du processus partenaire. Les

attributs *myRole* et *partnerRole* prennent comme valeur un <*partnerLinkType*>. L'élément <*partnerLinkType*> caractérise la conversation (l'échange) entre deux services. Il contient des éléments <*role*>. Un élément <*role*> décrit le rôle d'un service en précisant le <*portType*> qui recevra les messages lors de cette conversation. L'élément <*variables*> permet de définir les données traitées lors de l'exécution du processus. Un processus peut être exécuté plusieurs fois. Chaque exécution génère une instance de ce processus. Afin d'acheminer les données échangées à l'instance correspondante, un élément <*correlationSets*> est défini. L'élément <*correlationSets*> définit, à son tour, un ensemble de corrélations. Une corrélation <*correlationSet*> attribue des identifiants aux messages comprenant les données échangées. L'élément <*faultHandlers*> contient un ensemble de gestion d'exceptions de type <*catch*>. L'élément <*eventHandlers*> modélise les événements susceptible d'avoir lieu au cours de l'exécution et leurs associe des traitements.

Les activités d'un processus sont décrites par un ensemble de structures définissant les patterns d'interaction. Les principales structures représentées par des éléments dans le fichier XML d'un document BPEL, sont les suivantes : <*sequence*> qui contient une séquence d'activités, <*flow*> qui contient des activités en parallèle, <*switch*> qui modélise un branchement décisionnel, <*if*> qui modélise un autre type de branchement décisionnel, <*while*> qui modélise une boucle conditionnelle, etc. Les primitives de base associées aux activités modélisent les différents types d'opérations pouvant être accomplies par les services. Elles sont les suivantes : <*invoke*> qui représente un appel à une opération d'un service donc l'envoi d'un message, <*receive*> qui représente la réception d'un message, <*reply*> qui représente l'envoi d'un message suite à une sollicitaion, <*wait*> qui représente une attente, <*assign*> qui attribue une valeur à une variable, <*throw*> qui renvoie à un élément <*catch*> de <*faultHandlers*> etc. L'élément <*compensate*> représente une activité particulière qui doit être référencée dans un élément appartenant à <*faultHandlers*>. Elle consiste à invoquer des opérations de compensation comme par exemple l'annulation d'un ordre d'achat. L'élément <*scope*> délimite un ensemble d'activités. Il permet par exemple d'associer ces activités à un événement. Généralement un processus BPEL commence par une activité <*receive*> et se termine par une activité <*send*>. Un document BPEL est extensible car il est décrit en XML, d'où la possibilité d'étendre les structures utilisées et de définir les types de données requis par des XML schémas. La spécification BPEL ne précise aucun langage pour l'illustration graphique des descriptions. Cependant la majorité des outils de description BPEL adoptent la notation graphique BPMN (Business Process Modeling Notation) que

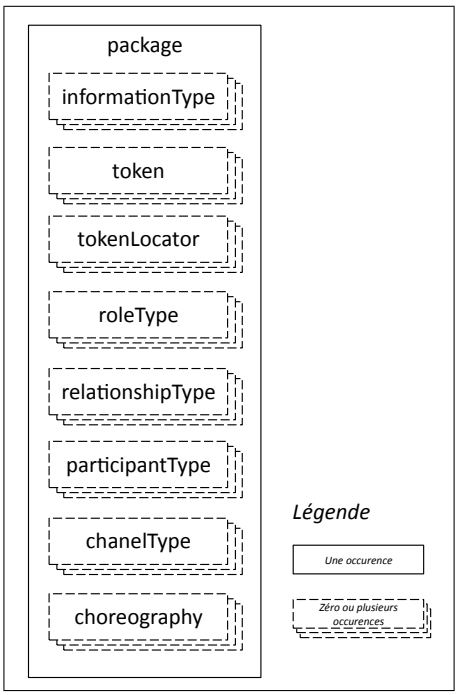


Figure 2.11: Eléments de description WS-CDL

nous décrivons en section 3.3.1.

Le **Web Services Choreography Description Language**, désigné par l'acronyme WS-CDL [145] est un langage qui permet de décrire une vision globale des collaborations entre les services. Kavantzas *et al.* insistent sur le fait que WS-CDL n'est pas un langage de description de processus exécutables, ni un langage d'implémentation [145]. Il décrit les séquences ordonnées de messages impliquant plusieurs entités visant à accomplir un objectif commun. WS-CDL reprend et développe la spécification Web Service Choreography Interface WSCI [146]. WS-CDL permet (*i*) de désigner les variables et les types de données échangées, (*ii*) de décrire les activités impliquées et (*iii*) de décrire les structures illustrant les interactions entre les activités. WS-CDL consiste à définir un fichier XML décrivant une chorégraphie. Les éléments d'un document WS-CDL sont illustrés en figure 2.11. Nous détaillons l'élément <*choreography*> décrivant des règles générales d'échange de messages. Il permet de définir trois aspects d'une chorégraphie : (*i*) *Choreography Life-line* décrit une collaboration et indique son état d'avancement, (*ii*) *Choreography Exception Blocks* exprime les actions à entreprendre en cas d'erreur, (*iii*) *Choreography Finalizer Blocks* indique les méthodes permettant de valider les résultats d'une

chorégraphie ou de les annuler. En résumé WS-CDL permet de décrire les règles selon lesquelles une collaboration doit avoir lieu. Il fournit une structuration globale de l'interaction en fonction de laquelle chaque participant décrit son processus métier et par suite ses services.

2.3.2 Aspect transactionnel

Une transaction "classique" est une unité composée d'un ensemble de tâches. Une transaction réussit si toutes les tâches sont accomplies complètement et correctement. Le cas échéant, la transaction échoue et les tâches partielles qui ont été accomplies sont annulées. Dans les bases de données, les transactions ont des propriétés dites ACID : (*i*) l'atomicité indique que toutes les activités d'une transaction doivent être exécutées correctement sinon toute la transaction est annulée. (*ii*) La cohérence indique qu'une transaction doit préserver la cohérence des données vis-à-vis du schéma global. (*iii*) L'isolation indique qu'au cours de l'exécution d'une transaction, aucune autre transaction ne peut être exécutée. (*iv*) La durabilité indique qu'une fois la transaction accomplie les résultats de la transaction sont permanents.

Il est évident qu'une composition de service ne peut pas être considérée comme une transaction "classique" car elles ne partagent pas les mêmes propriétés. Cependant, une composition de services présente des aspects transactionnels. En effet, plusieurs études et spécifications ont adressé cet aspect. Notre objectif n'est pas de recenser les spécifications existantes mais de souligner l'aspect transactionnel d'une composition de services.

La spécification WS-Transaction [21, 49, 50], est proposée par IBM. Elle définit deux types de composition : *atomic transaction* et *business activity*. Une composition est de type *atomic transaction* quand elle décrit une "courte" unité de travail dans un domaine de confiance comme le réseau interne d'une entreprise. Une *atomic transaction* présente des propriétés similaires aux transactions des bases de données. Une composition est de type *business activity* quand elle décrit un long processus d'échange d'information entre plusieurs domaines de confiance. Elle nécessite des mécanismes de gestion des erreurs qui sont plus complexes en comparaison avec ceux d'une transaction de type *atomic transaction*.

WS-TXM est une spécification similaire proposée par Bunting *et al.* dans le cadre du Web Service Composite Application Framework [19].

La notion de transaction est étudiée davantage dans [161]. Les auteurs discutent

les inconvénients des protocoles transactionnels existants basés sur des mécanismes de compensation. Ils proposent un protocole de gestion des transactions entre services basé sur le principe de réservation des ressources. Ils redéfinissent les propriétés ACID de façon à convenir aux propriétés transactionnelles des services. Le protocole de réservation proposé gère la transaction en deux phases : une première phase durant laquelle les ressources nécessaires sont réservées et une deuxième phase qui confirme ou annule les réservations.

2.3.3 Notion de contexte

Dans [104], Ouksel définit le contexte d'une interaction, au cours de laquelle un utilisateur veut satisfaire un ou plusieurs objectifs, comme un ensemble de données classées en deux groupes. Le premier groupe de données est construit en se basant sur des agréments communs (*mutual beliefs*). Ces derniers sont des informations et des propriétés relatives au domaine d'interaction et aux membres participants. Le deuxième groupe de données est constitué des correspondances entre les schémas fournis par les membres et des correspondances établies en vue de répondre à la requête exprimant l'objectif de l'utilisateur.

Plusieurs approches ont abordé le contexte d'une composition de services. Cabrera *et al.* décrivent la spécification WS-Coordination dans [22]. Elle définit un modèle dans lequel deux ou plusieurs services coordonnent leurs activités en échangeant des messages de type $<CoordinationContext>$.

La spécification WS-Context est présentée par OASIS dans [102]. WS-Context fournit les définitions et les mécanismes de structuration pour organiser et partager un contexte d'interaction lors de l'exécution de deux ou plusieurs services. Elle consiste à regrouper un ensemble de services dans une activité *activity*. Un contexte est dès lors attaché à cette activité. WS-Context définit un service web de type *Context Service* pour gérer les activités. WS-Coordination et WS-Context proposent d'intégrer la notion de contexte au processus composite à un bas niveau, à savoir, au niveau de l'échange de messages et des protocoles associés, ce qui réduit le contexte à un ensemble de variables d'exécution.

Le contexte d'interaction est étudié par Maamar *et al.* dans [81,97]. Ils présentent une architecture de composition à quatre niveaux : le niveau des composants, le niveau composite, le niveau sémantique et le niveau des ressources. La notion de contexte est introduite verticalement en relation avec les quatre niveaux. À chaque

niveau, le contexte se spécifie en sous-contexte. Au niveau *composant*, le contexte contient les informations concernant les services participant à la composition. Au niveau *composite*, le contexte permet de suivre l'évolution de l'exécution de la composition. Au niveau *sémantique*, le contexte contient les informations nécessaires à la médiation entre les entrées et les sorties des différents services. Et finalement au niveau *ressource*, le contexte supervise l'exécution du service sur cette ressource. Le contexte joue le rôle de méta-données apportant des précisions supplémentaires jusqu'alors absentes dans certaines descriptions des compositions. Ces méta-données permettent la résolution des conflits par médiation entre contextes au moment de l'exécution.

Sheng *et al.* étudient le contexte dans une approche plus récente [132]. Les auteurs proposent d'exprimer le contexte à travers des structures génériques *generic policies*. Une structure de type *policy* est décrite indépendamment de la description du service. Cela favorise un découplage maximal entre les définitions des services et celles des contextes. Le modèle présenté permet de définir plusieurs types de *policies* haut niveau et de les réutiliser, évitant ainsi d'intégrer les contraintes directement au processus décrivant la composition. Ces structures de type *policies* permettent de définir : (*i*) un contexte d'exécution qui est exprimé à travers des contraintes sur des propriétés d'exécution telles que le temps de réponse par exemple, (*ii*) un contexte de données qui est exprimé à travers des contraintes sur les entrées et les sorties et (*iii*) un contexte d'exceptions qui est exprimé à travers un ensemble de règles de type événement-action-condition. Dans une composition, une *policy* est attachée à chaque service composant. Elle permet au service d'adapter son comportement selon les règles spécifiées par les contraintes qu'elle décrit.

2.3.4 Composition automatique

La composition automatique permet un développement plus rapide des applications à base de services. Elle consiste à préciser la requête d'un utilisateur sous forme d'objectifs à satisfaire. Un moteur de composition "intelligent" choisit la combinaison de services répondant à l'objectif décrit. Il génère la composition de service adéquate de manière transparente à l'utilisateur. Ce principe a interpellé plusieurs communautés de recherche travaillant dans le domaine de l'Intelligence Artificielle. Dans la suite nous classifions les approches de composition automatique en trois catégories : les approches sémantiques, les approches basées sur les automates et les approches basées sur les techniques de planification.

2.3.4.1 Approches sémantiques

L'ontologie OWL-S différencie trois types de processus. (*i*) Les processus atomiques, *atomic processes*, qui n'ont pas de sous-processus. Un processus atomique est "une description d'un service qui attend un message en entrée et retourne un autre message" [84]. Il est décrit en terme d'entrées, sorties, pré-conditions et effets. Du point de vue de l'utilisateur, un processus atomique est une "boîte noire" accomplissant une fonctionnalité. Le comportement interne d'un processus atomique n'est pas visible à l'utilisateur. Un processus atomique modélisant une fonctionnalité F doit fournir un *grounding* avec une description WSDL réalisant la fonctionnalité F. (*ii*) Les processus composites, *composite processes*, sont constitués d'un ensemble de processus atomiques et de contrôles de séquence. Un processus composite maintient un état. Chaque message reçu d'un client constitue une itération d'une phase dans ce processus [84]. (*iii*) Les processus simples, *simple processes*, ne sont pas exécutables. Tel décrit en section 2.1.3, un processus simple est une vue sur un processus atomique décrivant les possibilités d'interaction avec lui. Dans ce cas, nous disons que le processus simple est realisé par le processus atomique associé [84]. Un processus simple peut fournir une représentation moins complexe d'un cas d'utilisation d'un processus composite. Nous disons que le processus simple *étend* le processus composite en question. Le raisonnement sur une composition OWL-S est assuré par le raisonneur associé au langage OWL choisi (OWL-Lite, OWL-DL ou OWL-Full).

Dans [128], les auteurs expliquent que le niveau d'expressivité de OWL n'est pas suffisant pour raisonner correctement sur les processus. En effet, si un processus OWL-S est décrit en OWL, sa description peut avoir plusieurs interprétations parmi lesquelles il y aura des interprétations non conformes aux intentions de départ. Ankolekar *et al.* [6] présentent un modèle formel pour représenter les services décrits selon OWL-S. Ils proposent le noyau d'un langage fonctionnel pour décrire formellement les services. Ils adoptent la sémantique opérationnelle d'Erlang [59] et les *Concurrent Haskell Programs* [115] pour décrire la sémantique d'exécution de la composition étudiée.

Certains environnements de composition adoptent l'approche descriptive de WSMO pour étendre leur modèle de composition de service. Par exemple, IRS-III *Internet Reasoning Service and Infrastructure for Semantic Web Services*, proposé par Domingue *et al.* [39] décrit les services selon le modèle WSMO afin d'améliorer la découverte et par suite la composition.

2.3.4.2 Automates et réseaux petri

Les automates ou systèmes de transition d'états annotés offrent une sémantique précise et adaptée à la représentation de la composition. Les techniques de raisonnement applicables aux automates permettent de vérifier la structure de la composition et sa cohérence. Un automate est constitué d'un ensemble d'états, d'un ensemble de transitions annotées entre états et d'un ensemble d'actions. Les annotations des transitions contiennent des actions représentant le passage d'un état à un autre.

L'ontologie WSMO associe à chaque élément *webService* des interfaces pour représenter son comportement. L'interface est décrite par une chorégraphie ou par une orchestration. Scicluna *et al.* proposent un modèle de représentation de la chorégraphie et de l'orchestration en WSMO basé sur les machines abstraites d'états (ASM) [124]. Les auteurs ne développent que l'aspect modélisation qui insiste sur la présence d'états et de transitions. Cette étude illustre le rôle des médiateurs WSMO dans une composition.

Dans [48], Foster *et al.* suggèrent de représenter la composition de services par l'implémentation d'une couche *multi-stakeholder distributed system (MSDS)*. MSDS est un environnement distribué où des nœuds définis par des fournisseurs différents interagissent de façon concurrente sans connaissance totale de l'environnement. Les auteurs introduisent la notion de compatibilité entre services à trois niveaux : interface, comportement et paramètres d'entrée et de sortie. Ils présentent une approche de vérification qui consiste à traduire une composition exprimée en BPEL en un modèle d'automates finis et à compiler ce dernier en un système de transition d'états. Ils utilisent un outil de vérification (LTSA pour le model checking) qui permet de s'assurer de la consistance du graphe et d'affirmer qu'il ne contient pas des "impasses" (*deadlock free*).

Bordeaux *et al.* suggèrent de représenter le comportement du service par un système de transition d'états [17]. Cela revient à modéliser le service par un ensemble d'états reliés par des actions. Les auteurs précisent que ce modèle est déterministe dans la mesure où "une action qui a lieu à un état donné conduit à un état determiné". Cependant, tel décrit en section 2.2.4 les auteurs proposent une approche de vérification de la compatibilité entre deux services participant à une composition. Ils n'adressent pas les modalités de composition.

Un réseau Petri est un ensemble de *places* et de *transitions* reliées par des *arcs*. L'aspect dynamique du réseau est décrit par un ensemble de règles. Chaque règle

définit (*i*) les conditions à valider pour qu'une transaction ait lieu et (*ii*) les effets de cette transaction. Les réseaux Petri (Petri Nets) fournissent une représentation précise et sémantiquement riche de l'ordre d'exécution des activités. Ils permettent un raisonnement plus efficace sur la composition. Pratiquement, les réseaux Petri sont utilisés pour modéliser des systèmes concurrentiels.

Narayanan *et al.* [98] proposent une traduction du *process model* de OWL-S en un réseau Petri afin de raisonner sur la vérification automatique de la composition. Ils proposent un outil qui génère automatiquement un réseau Petri pour une composition exprimée par un *process model* de OWL-S. Cet outil vérifie les propriétés de la composition générée. Il juge la consistance du graphe modélisant le flux d'interaction. Il détermine, par exemple, si tous les états (nœuds) peuvent être atteints, si les contraintes décrites sont respectées, si des impasses existent, etc.

Ouyang *et al.* proposent une traduction des contrôles de flux BPEL en réseaux Petri annotés [105]. Cette traduction permet de raisonner automatiquement sur la structure du graphe d'interaction et sur ses propriétés. L'étude de Ouyang *et al.*, menée en 2005, a révélé certaines ambiguïtés dans la description des contrôles de flux de la version de BPEL contemporaine. Une autre étude, menée par Yi et Kochut, décrit la traduction d'une composition BPEL en réseaux Petri [156]. Les auteurs proposent un outil qui permet de décrire des processus BPEL et de les traduire automatiquement en réseaux Petri afin de vérifier leurs propriétés.

2.3.4.3 Techniques de planification

Dans [90], McIlraith *et al.* proposent une extension de Golog. Ce dernier est un langage de programmation logique basé sur la *situation calculus*. La technique présentée repose sur un ensemble de procédures génériques de haut niveau et sur la personnalisation des contraintes associées. Les procédures génériques représentent la requête utilisateur. Les contraintes à respecter sont écrites avec un langage de premier ordre. Chaque service est modélisé comme une action. Deux types d'actions sont possibles : des actions primitives et des actions complexes telles que ces dernières sont des compositions d'actions individuelles. Les auteurs proposent une approche qui permet à l'utilisateur de traduire manuellement les descriptions OWL-S des services impliqués en Prolog. Après cette traduction, le système utilise des règles d'inférences logique pour automatiser le choix d'un service pour chaque action en respectant les contraintes associées.

Dans [92] Medjahed présente une approche de *rule-based planning* pour générer des services composites à partir des déclarations de haut niveau. Il décrit une approche de composition en quatre phases. Il présente des règles syntaxiques et des règles sémantiques. Ces règles décrivent les attributs nécessaires à la composition des services impliqués.

Shankar *et al.* présentent SWORD, un outil de composition basé sur le *rule-based planning* [116]. SWORD raisonne sur des services décrits selon le modèle Entité - Relation. Il associe aux services des règles logiques de type *Horn*. La création d'un service web composite est ramenée à la spécification d'un état de départ et d'un état d'arrivée. Les états intermédiaires seront générés par le moteur de planification à partir des descriptions des services disponibles.

Le *Planning Domain Definition Language (PDDL)* est un langage de planification largement adopté par les communautés de l'Intelligence Artificielle. Dans [89], McDermott décrit les similarités entre l'approche de description et de composition OWL-S et le *Planning Domain Definition Language (PDDL)*. Il présente une approche pour traduire les descriptions OWL-S en PDDL afin de modéliser la composition comme un problème de satisfaction d'un "planning". Il propose la notion de *value of an action* qui modélise l'exécution d'un service. Elle exprime des changements qui peuvent avoir lieu dans l'environnement.

2.4 Discussion

Dans ce chapitre nous avons abordé la description, la découverte et la composition de services. D'abord, nous avons étudié les différents modèles de description des services. Après la présentation du WSDL, modèle omniprésent de description syntaxique, nous avons souligné la nécessité de compléter la description syntaxique par des précisions sémantiques. À cette fin nous avons revu les approches d'annotation sémantique des descriptions des services et les approches proposant des ontologies de services. Finalement, nous avons présenté certaines approches permettant de décrire le comportement d'un service en précisant l'ordre d'invocation de ses opérations.

Ensuite, nous avons étudié la découverte des services sous trois aspects à savoir, la localisation des services, le traitement des données et le matching entre la requête de l'utilisateur et la description d'un service. Nous avons différencié des approches de découverte centralisée et d'autres distribuée, des approches proposant une découverte manuelle et d'autres proposant une découverte automatique, des

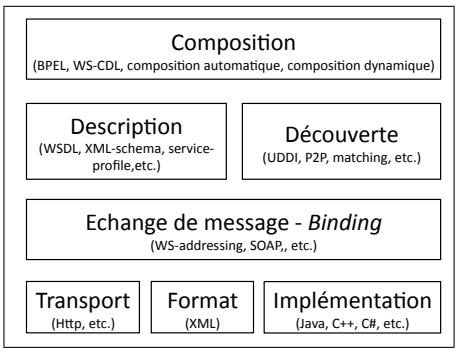


Figure 2.12: Description, découverte et composition de services web

approches basées sur le calcul d'une similarité syntaxique et d'autres proposant le calcul d'une similarité sémantique.

Finalement, nous avons étudié la composition des services. Après avoir présenté BPEL et WS-CDL, les deux approches principales de description d'une composition, nous avons souligné l'aspect transactionnel d'une composition de service et le rôle de la description du contexte d'interaction dans une composition de service. Nous avons différencié ensuite, la composition automatique en trois catégories, à savoir les approches sémantiques, les approches basées sur les automates et les approches basées sur les techniques de planification.

La figure 2.12 représente les services web et certaines technologies associées. Elle permet de situer la composition de services vis-à-vis de la description de services, la découverte de services et les technologies sous-jacentes.

Nous constatons que les approches de description et de découverte présentées ont atteint un certain niveau de complétude vis à vis de la description de l'aspect fonctionnel d'une part et de la comparaison entre deux descriptions fonctionnelles d'autre part. Cependant, nous remarquons que la prise en compte des propriétés non fonctionnelles des services reste limitée. Nous constatons aussi que les approches de composition présentées, bien qu'elles permettent de décrire le schéma d'interaction voire de le générer automatiquement dans certains cas, ne réussissent pas à tenir compte de l'aspect dynamique de l'environnement des services. L'aspect dynamique se manifeste par des changements liés aux services et à leurs fournisseurs.

Chapitre 3

Applications composites : l'environnement DyCoSe

Sommaire

3.1 Contexte et Objectifs 74

3.1.1 Modèle du service 75

3.1.2 Pair-à-Pair 77

3.1.3 Objectifs de la thèse 78

3.1.4 Contributions de la thèse 81

3.2 Écosystème de DyCoSe 82

3.2.1 Définition 82

3.2.2 Description 83

3.2.3 Modélisation 86

3.2.4 Réalisation 91

3.2.5 Exemple d'un écosystème 94

3.3 Composition dans DyCoSe 97

3.3.1 Niveau fonctionnel 98

3.3.2 Niveau services 105

3.3.3 Niveau infrastructure 111

3.4 Discussion 119

Dans ce chapitre nous décrivons une approche de conception et de développement d'applications ou de systèmes d'information [1] à base de services. La première section présente le contexte et les objectifs de la thèse. La deuxième section décrit un environnement de partage de services. La troisième section présente une architecture multi-niveaux de composition et la démarche de composition associée.

3.1 Contexte et Objectifs

La conception et le développement d'applications évoluent progressivement d'un modèle traditionnel vers un modèle orienté services. Traditionnellement, la création d'applications reposait sur un paradigme en trois étapes, notamment, la conception, l'implémentation et le déploiement. La conception consiste à modéliser le fonctionnement de l'application et à effectuer les choix technologiques. L'implémentation consiste à coder les différentes parties de l'application et à les intégrer si nécessaire. Le déploiement consiste à installer l'application et à la configurer sur la plateforme prévue. Des outils CASE [144] accompagnaient le créateur à partir de la phase de conception jusqu'à la phase de déploiement, en passant par le développement ou codage.

Récemment nous assistons à une évolution de cette vision statique de l'implémentation et du déploiement vers une vision plus dynamique orientée services où la réutilisation et l'adaptabilité jouent un rôle important. Avec l'étendue des environnements distribués, protégés tels les réseaux intra-entreprise ou publics tel Internet, la création d'applications par composition de services existants propose une alternative avantageuse par rapport aux méthodes traditionnelles de création d'applications. Après la conception qui identifie les fonctionnalités de l'application, l'implémentation revient à découvrir les services correspondant aux fonctionnalités identifiées et à les composer. Le déploiement est réduit aux séquences d'invocation des services selon l'ordre décrit par la composition. Cette nouvelle approche met l'accent sur la réutilisation des services existants selon le principe d'externalisation "outsourcing". Elle permet une adaptabilité qui se manifeste par la possibilité de remplacer un service par un autre. Nous désignons dans la suite de la thèse les applications composées à base de services par *applications composites*.

Le développement d'applications composites se produit dans un cadre d'entre-

1. Dans la suite de thèse nous utilisons les termes *application* et *système d'information* de façon interchangeable.

prises indépendantes dans leurs cœurs de métier respectifs mais qui coopèrent en partageant des services. Cette nouvelle tendance de création d'applications dynamiques et adaptables s'appuie sur deux domaines de recherche : l'orientation service et les architectures distribuées, plus particulièrement le pair-à-pair.

3.1.1 Modèle du service

Comme défini dans le chapitre 1, un service est une unité logicielle, indépendante, accomplissant une fonctionnalité donnée. Un service illustré par la figure 3.1, est décrit comme une entité composée d'une partie logique et d'une autre physique tel que :

- La partie logique est une *interface* décrivant un service à travers un ensemble d'opérations en corrélation avec une fonctionnalité métier. Chaque opération possède :
  - Des *propriétés fonctionnelles* qui caractérisent les opérations décrites dans l'interface et leurs paramètres. Chaque opération accepte en entrée un ensemble de paramètres et retourne en sortie après exécution un ensemble de paramètres. Les propriétés fonctionnelles décrivent l'activité accomplie par une opération et les différents types de données de ses paramètres d'entrée et de sortie.
  - Des *propriétés non fonctionnelles* qui sont associées aux opérations pour caractériser les attributs ne se rapportant pas directement aux données traitées. Elles décrivent la configuration du service comme par exemple le niveau de sécurité, le cryptage des données, etc. Elles décrivent aussi la performance de chaque opération en terme de durée d'exécution ou de coût d'exécution.
- La partie physique ou *implémentation* d'un service consiste en : (*i*) un fichier semi-structuré contenant la description de l'interface du service et (*ii*) un ensemble de réalisations des différentes opérations décrites dans l'interface. Ces réalisations dépendent des technologies adoptées par le fournisseur du service.

La recherche dans le domaine des services a abordé les modèles de description des services, les architectures et techniques de découverte des services et les modèles de composition de services. La description d'un service consiste à créer une interface représentant les opérations accomplies par le service. Nous avons revu la

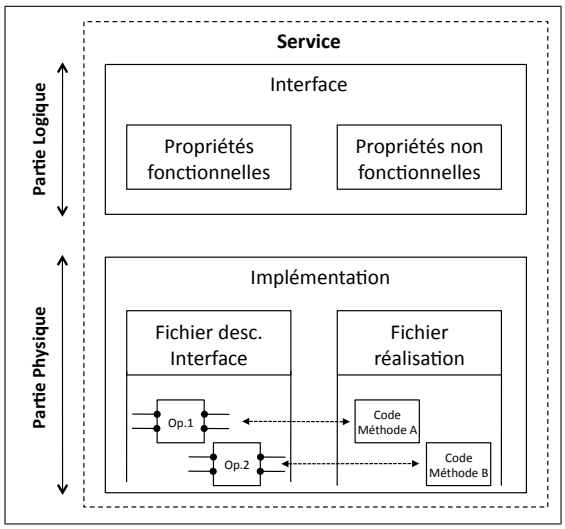


Figure 3.1: Illustration d'un service

littérature sur la description des services en section 2.1. Plusieurs approches visent à compléter le modèle de description WSDL largement adopté en rajoutant une couche sémantique. La découverte de services permet de comparer les descriptions des services entre elles ou avec une requête utilisateur. Nous avons revu les modèles de découverte de services en section 2.2. Plusieurs techniques permettent d'inférer une correspondance fonctionnelle entre deux interfaces de services ou entre l'interface d'un service et une requête utilisateur. Cependant, un manque de précision dans la description d'un service peut compromettre le résultat du processus de mise en correspondance. Par ailleurs, avec l'ajout de la sémantique des conflits émergent. Ils sont dus aux différentes représentations adoptées pour des données identiques et aux différentes interprétations d'une donnée. La composition de services repose sur la description du schéma d'interaction entre les services découverts. Nous avons revu les approches de composition de services en section 2.3. Certaines approches permettent de décrire manuellement le schéma de composition à travers des structures regroupant les services et des contrôles de flux entre elles. D'autres approches permettent de générer automatiquement le schéma de composition à travers des techniques empruntées au web sémantique, aux automates ou à l'intelligence artificielle. Cependant, ces approches considèrent que les services interagissant sont connus a priori au moment de la description du schéma de composition. Nous pouvons imaginer des scénarios où l'utilisateur préfère repousser le choix des services jusqu'à l'exécution.

3.1.2 Pair-à-Pair

Une architecture Pair-à-Pair est un environnement distribué composé d'éléments distincts désignés par pairs. Les pairs présentent des capacités plus ou moins similaires et partagent des ressources de même nature. Ils interagissent afin de consommer des ressources et en fournir d'autres. La littérature a largement adressé divers aspects des architectures pair-à-pair. La recherche de ressources est étudiée dans [62, 121, 127]. Des systèmes analysant la motivation et la récompense sont proposés dans [54, 99] et d'autres approfondissent la notion de confiance dans [54, 149]. La préservation de l'anonymat est abordée dans [154]. Inspiré de cette littérature, des applications ou systèmes applicatifs basés sur des architectures pair-à-pair sont proposés. Une application de gestion de stockage d'information dans un environnement à niveau de confiance faible est présentée dans [1]. Des applications de type composition de streaming sont étudiées dans [27, 58, 73, 141]. Les architectures Pair-à-Pair présentent deux caractéristiques majeures, à savoir, l'indépendance entre les pairs et la flexibilité de réorganisation des pairs.

D'abord l'indépendance se traduit à deux niveaux. Premièrement l'indépendance vis-à-vis de la description des ressources partagées. En effet, les pairs ne sont pas tenus de décrire leurs ressources par rapport à un schéma global. Deuxièmement l'indépendance vis-à-vis de la durée de participation. En effet, un pair peut participer en mettant en commun des ressources, comme il peut se retirer à n'importe quel moment.

Ensuite la flexibilité se traduit par une réorganisation des pairs membres en un réseau virtuel. Deux approches de réorganisation des architectures pair-à-pair sont récurrentes dans la littérature. La première résulte en une architecture structurée tandis que la deuxième résulte en une architecture non structurée. Dans les architectures structurées, les pairs sont organisés en une topologie bien définie et les ressources sont distribuées sur les pairs de façon précise respectant une stratégie d'indexation donnée. Une grande majorité des architectures structurées utilisent une technique basée sur les tables de hachage dynamiques (DHT) [62, 121]. En effet, le DHT contrôle la distribution des ressources sur les pairs pour garantir la localisation des ressources recherchées en un minimum de sauts. Dans les architectures non structurées, les pairs sont organisés selon des graphes arbitraires sans aucun contrôle sur leurs contenus. Chaque pair est responsable de la gestion du contenu et de l'accès aux ressources qu'il propose. Les architectures pair-à-pair non structurées sont classées en trois catégories : les architectures centralisées, les architectures décentralisées et

les architectures hybrides. Les architectures centralisées reposent sur un annuaire central pour stocker l'information globale concernant tous les membres de l'environnement. Les architectures décentralisées ne fournissent pas d'informations globales mais plutôt un ensemble d'informations locales gérées par chaque pair. Les architectures hybrides combinent les caractéristiques des architectures centralisées et des architectures décentralisées offrant un ensemble d'informations globales et locales. Elles reposent sur le principe de super-pair, décrit dans [155].

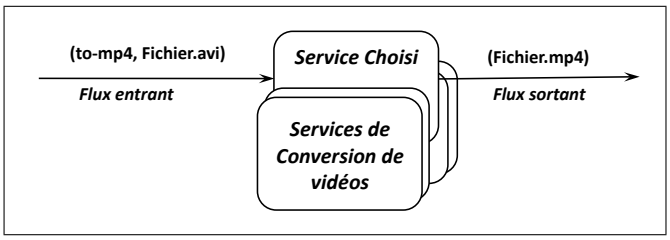
Dans la pratique, les pairs entrent en relation avec d'autres pairs partageant un intérêt commun. Des groupes d'intérêt émergent. Ils sont désignés dans la littérature par communautés de pairs. Khambatti *et al.* définissent une communauté de pairs comme un ensemble de membres actifs, impliqués dans un processus de partage de ressources et dans une série de communications liés à un intérêt commun [70]. Dans les architectures hybrides, la tendance est de regrouper les pairs en communautés [54, 70, 107, 118, 122, 149, 159, 160, 162]. Chaque communauté est un ensemble de pairs partageant des propriétés communes et gérés par un super-pair. Nous distinguons des communautés de pairs coopératives où les pairs proposent des ressources différentes et collaborent pour accomplir un objectif commun et d'autres communautés compétitives où les pairs proposent plusieurs versions de la même ressource et sont en compétition pour mettre en avant leurs versions respectives.

3.1.3 Objectifs de la thèse

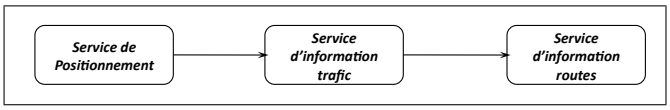
Dans cette thèse l'accent est mis sur la composition dynamique de services. Nous ne sommes pas intéressés par l'inférence des correspondances fonctionnelles entre les descriptions des services. Cependant, ayant l'équivalence fonctionnelle d'un ensemble de services, nous proposons (*i*) d'étudier la sélection d'une combinaison de services respectant des contraintes associées à une composition donnée et (*ii*) d'analyser les possibilités de réalisation de cette combinaison dans un environnement dynamique. Avant de souligner les objectifs principaux de cette thèse, nous proposons d'illustrer certains aspects des applications composites par les exemples suivants :

Exemple 1 ***Échange avec un service*** :

Un utilisateur a besoin de convertir une vidéo en format "avi" au format "mp4". Il fait appel à un service de conversion vidéos. D'abord il recherche les services publiant la fonctionnalité "conversion de vidéos" dans la description de leurs opérations. Ensuite il vérifie la conformité des entrés - sorties du service



(a) Exemple d'un échange avec un service de conversion vidéo



(b) Exemple d'une application composite simple

Figure 3.2: Services et applications composites

choisi avec son besoin. En effet, le service retenu doit accepter en entrée un fichier en format "avi" et retourner en sortie un fichier en format "mp4". Tous les services de conversion de vidéos disponibles ne proposent pas forcément la possibilité de convertir le format "avi" en format "mp4". La figure 3.2a illustre l'échange de données avec un service de conversion vidéo.

Exemple 2 ***Composition pour répondre à un besoin*** :

Une application d'assistance de voyageurs à travers leurs mobiles est composée à partir des services disponibles sur un ensemble de réseaux d'opérateurs. Un voyageur en voiture sur une autoroute part d'une villeX pour aller à une villeY. Il s'aperçoit que la circulation est bloquée et décide de poursuivre sur un trajet alternatif pour arriver à sa destination. Il décrit à son application le besoin suivant : "changer d'itinéraire et arriver à la même destination". Aucun service disponible sur le réseau ne répond directement à son besoin. Cependant une composition de services existants peut satisfaire sa requête. La figure 3.2b illustre une composition séquentielle d'un service de positionnement suivi d'un service d'information de trafic, suivi à son tour d'un service d'information sur les autoroutes et les routes. Le service de positionnement envoie au service d'information de trafic les coordonnées de l'utilisateur. Le service de trafic situe l'utilisateur par rapport au bouchon. Il communique ces informations à un service d'information sur les routes qui détermine la sortie la plus proche et l'itinéraire à suivre pour atteindre la destination désirée.

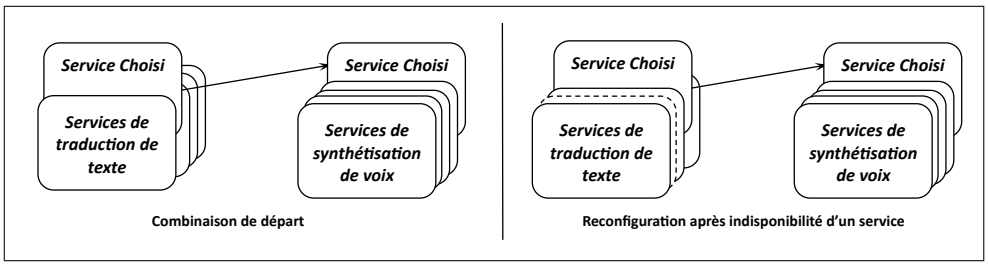


Figure 3.3: Deux configurations d'une application composite

Exemple 3 ***Sélection d'une combinaison convenable et adaptabilité*** :

Un utilisateur veut traduire un texte de l'Allemand au Français. Il veut que son PDA lise le texte traduit. Son budget pour l'opération est fixé à une somme donnée. L'application multimédia répondant à ce besoin consiste en une composition d'un service de traduction qui coordonne avec un service de synthétisation de voix. Les coûts d'invocation des deux services ne doivent pas dépasser le budget fixé. Si le service de traduction n'est plus disponible il doit être remplacé par un autre équivalent en tenant compte du budget fixé tout en conservant les caractéristiques de l'application comme par exemple la durée de traduction et de synthétisation ou d'autres mesures de performance. La figure 3.3 illustre deux configurations de cettre composition.

Les objectifs de cette thèse sont classés en deux groupes :

1. Les objectifs liés à l'environnement de partage des services :
   - Proposer les bases d'un environnement d'interopération délimité pour définir l'étendue logique de la recherche. En effet dans les environnements complètement ouverts comme le web, l'accessibilité inconditionnelle devient un inconvénient et prête à des confusions inutiles. Afin d'éviter ce genre de problèmes il est possible de définir l'étendue logique de l'environnement d'interaction en indiquant explicitement les domaines d'intérêt. Par ailleurs, il est possible d'identifier un ensemble de propriétés significatives aux participants voire même de reconnaître les fonctionnalités métier traditionnellement récurrentes.
   - Fournir les "règles" de participation pour organiser les fournisseurs de services. En effet, l'organisation des fournisseurs doit permettre d'augmenter l'efficacité des applications composites sans pour autant ajouter des "règles"

très restrictives contraignant l'interaction. Pour cela, il est possible d'organiser les fournisseurs en groupes partageant des propriétés communes.

2. Les objectifs liés à la composition des services :
   - Proposer une démarche de composition d'applications suffisamment flexible pour tenir compte à la fois de (*i*) l'évolution des besoins de l'utilisateur et des (*ii*) changements dans l'environnement d'exécution. En effet, une description "statique" de la composition précise les services impliqués au moment de la description. Des changements affectant la disponibilité des services peuvent avoir lieu entre le moment où la composition est décrite et son exécution. Il est possible de décrire la composition indépendamment des services composants et de choisir ces derniers au moment de l'exécution.
   - Tenir compte des propriétés non fonctionnelles des services. En effet, les paramètres non fonctionnels permettent de différencier deux services fonctionnellement identiques. Il est possible, au niveau de l'application composite, de choisir une composition de services qui satisfait des contraintes décrites sur des paramètres non fonctionnels.
   - Etudier les possibilités de réalisation d'une application composite dans un environnement dynamique où la disponibilité des services n'est pas garantie. Si l'application est réalisable, examiner l'effet des changements de l'environnement sur son exécution.

3.1.4 Contributions de la thèse

Dans cette thèse nous proposons l'environnement DyCoSe pour la conception et le développement d'applications composites. DyCoSe s'appuie sur des concepts étudiés dans la littérature mais il propose de les réorganiser de façon à étudier l'aspect dynamique des applications composites. Dans le cadre de DyCoSe :

- Nous avons décrit un écosystème d'entreprises, où les membres partagent leurs services selon un consensus global. Dans cet écosystème, les fonctionnalités métiers traditionnelles sont reconnues et certaines propriétés significatives aux membres sont identifiées. Les entreprises membres sont organisées en communautés selon un ensemble de consensus locaux reposant sur des propriétés partagées.

- Nous avons proposé une architecture de composition à trois niveaux. Elle permet de décrire une application composite à travers des fonctionnalités haut niveau et de les raffiner pour traduire l'application en une orchestration d'activités métiers représentées par des services abstraits. Un processus associé, désigné par instanciation, permet de substituer pendant l'exécution chaque service abstrait par une implémentation équivalente proposée par une entreprise membre de l'écosystème.
- Nous avons décrit l'instanciation comme un problème d'optimisation de coûts, les coûts étant exprimés sur des propriétés non fonctionnelles partagées par les fournisseurs. Nous avons proposé deux algorithmes pour le déploiement d'une application composite. Un premier qui nécessite une connaissance de l'état global du réseau des fournisseurs et un deuxième qui considère que l'utilisateur veut favoriser un certain nombre de fournisseurs "locaux".
- Nous avons proposé une réalisation du déploiement d'une application composite par un algorithme génétique. Nous avons implémenté un prototype simulant l'environnement DyCoSe. Cette implémentation permet d'étudier les possibilités de réalisation d'une application composite en tenant compte des paramètres de l'environnement d'exécution. Nous avons examiné en particulier, la probabilité de réussite du déploiement d'une application et la probabilité de réussite de son exécution.

3.2 Écosystème de DyCoSe

Dans cette section nous étudions un modèle d'écosystème d'entreprises. Après la définition des écosystèmes nous décrivons l'organisation de l'écosystème proposé. Nous proposons ensuite une modélisation des différents éléments constituant l'écosystème, suivie d'une proposition de réalisation de ces différents éléments. Finalement nous concluons la section en détaillant un exemple simplifié d'un écosystème d'entreprises.

3.2.1 Définition

Les écosystèmes d'entreprises émergent comme un nouveau concept de description des environnements d'interopération entre plusieurs réseaux d'entreprises

[8, 31, 76]. Un écosystème comprend un ensemble d'organisations autonomes et leurs ressources respectives. Ces organisations partagent un ensemble de consensus définissant les domaines concernés, les règles associées et décrivant les relations entre les différents membres de l'écosystème. Dans [76], Lau définit un écosystème d'entreprises comme un environnement propice à la création d'une entreprise virtuelle. Cette dernière est une entité logique dont l'existence débute suite à la mise en place d'une orchestration de plusieurs composants services accomplissant une nouvelle fonctionnalité métier non proposée jusqu'alors dans l'écosystème. Les composants services mis en jeu sont fournis par une ou plusieurs entités physiquement séparées d'où la désignation d'entreprise virtuelle attribuée à l'entité logique qui les rassemblent en vue d'une coopération précise. L'écosystème décrit les règles d'interaction entre ces composants services et leurs fournisseurs. Selon [8], un écosystème impose les règles conformément auxquelles les services sont publiés, partagés et réutilisés dans des compositions favorisant une externalisation fonctionnelle (*outsourcing*). L'écosystème fournit des moteurs de paiement gérant les procédures de mise en commun d'un savoir-faire. Un tel écosystème joue le rôle d'une place de marché pour les services (*marketplace*). En résumé, un écosystème fournit une formalisation des concepts et des propriétés communes et un cadre pour partager un savoir-faire et des services afin de favoriser l'interaction inter-membres. Cette interaction est caractérisée par un faible couplage entre les services mis en jeu.

Essentiellement, nous définissons dans cette thèse un écosystème d'entreprises, par opposition à un environnement ouvert, comme un environnement d'interaction contrôlée, délimitant la coopération [2] à un ensemble d'acteurs respectant des règles métiers communes. Il forme un environnement propice à la composition d'applications à base de services. En effet, il fournit un environnement où l'aspect sémantique est défini, les propriétés métiers sont décrites et dans lequel l'interaction entre membres est basée mais non restreinte à un consensus global partagé. Cela favorise l'identification des fonctionnalités métiers requises (par l'application), les relations entre elles et les propriétés qu'elles partagent.

3.2.2 Description

Dans la suite nous décrivons un écosystème et les éléments qui le constituent. Lau décrit dans [76] un modèle d'écosystèmes d'entreprises composé de ressources, d'une infrastructure et d'opérations. (*i*) Les ressources sont des "actifs tangibles

2. Dans la suite nous utilisons de façon interchangeable les termes coopération et interopération.

et intangibles" accessibles et gérables par une entreprise virtuelle. On dénombre trois types de ressources dans un écosystème : les applications, les participants et l'ensemble de connaissances. Les applications représentent des tâches automatisées, tandis que les participants s'engagent à réaliser les tâches non adaptées à la machine. Les applications et les participants constituent les actifs tangibles de l'écosystème. L'ensemble de connaissances, un actif intangible, est formé d'un historique des données récupérées à partir d'interactions précédentes et des analyses effectuées sur ces données. Cet ensemble de connaissances intervient dans les décisions futures. Un répertoire de ressources permet de recenser les descriptions des interfaces des ressources disponibles. Un contrat entre une entreprise virtuelle et un ensemble de ressources définit un guide d'interaction à suivre et un consensus d'interaction à respecter. Ce consensus quantifie des propriétés telles que le niveau de sécurité, le coût d'exploitation et le temps de réponse. (*ii*) L'infrastructure est composée de la structure organisationnelle, la structure de gestion et la structure de communication. La première définit les relations entre les différents membres. La deuxième définit les règles métier de l'entreprise virtuelle. La troisième décrit la plateforme de communication entre les différents membres. (*iii*) Les opérations peuvent être explicites ou implicites. Une opération est qualifiée d'explicite si elle représente un processus métier décrivant précisément la séquence des activités à accomplir en vue de réaliser l'entreprise virtuelle. Tandis qu'une opération est qualifiée d'implicite si elle représente un objectif atteint par une interaction *ad-hoc* (open-ended).

L'écosystème d'entreprises proposé dans le cadre de cette thèse emprunte les concepts de ressources, de participants et d'infrastructure du modèle étudié dans [76] mais propose de les organiser différement. En effet, un écosystème d'entreprises tel que nous le définissons est constitué d'*entreprises membres* jouant le rôle de *participants*. Ces dernières ont un double objectif résumé par la proposition de nouvelles *ressources* et par la consommation des services existants. En d'autres termes les entreprises membres proposent un savoir-faire aux participants sous forme d'activités et collaborent en consommant d'autres activités disponibles offrant des fonctionnalités complémentaires utiles. Ces ressources sont classées en deux catégories : (*i*) des ressources représentant des activités accomplies automatiquement par la machine et (*ii*) d'autres représentant des activités accomplies manuellement et nécessitant une intervention humaine. Dans la suite nous désignerons uniformément les ressources par *services*, la différenciation entre une opération automatisée ou non sera faite au stade de l'implémentation. Nous développons cette idée dans la section 3.2.4.2. Les entreprises membres présentent leurs services conformément à des interfaces com-

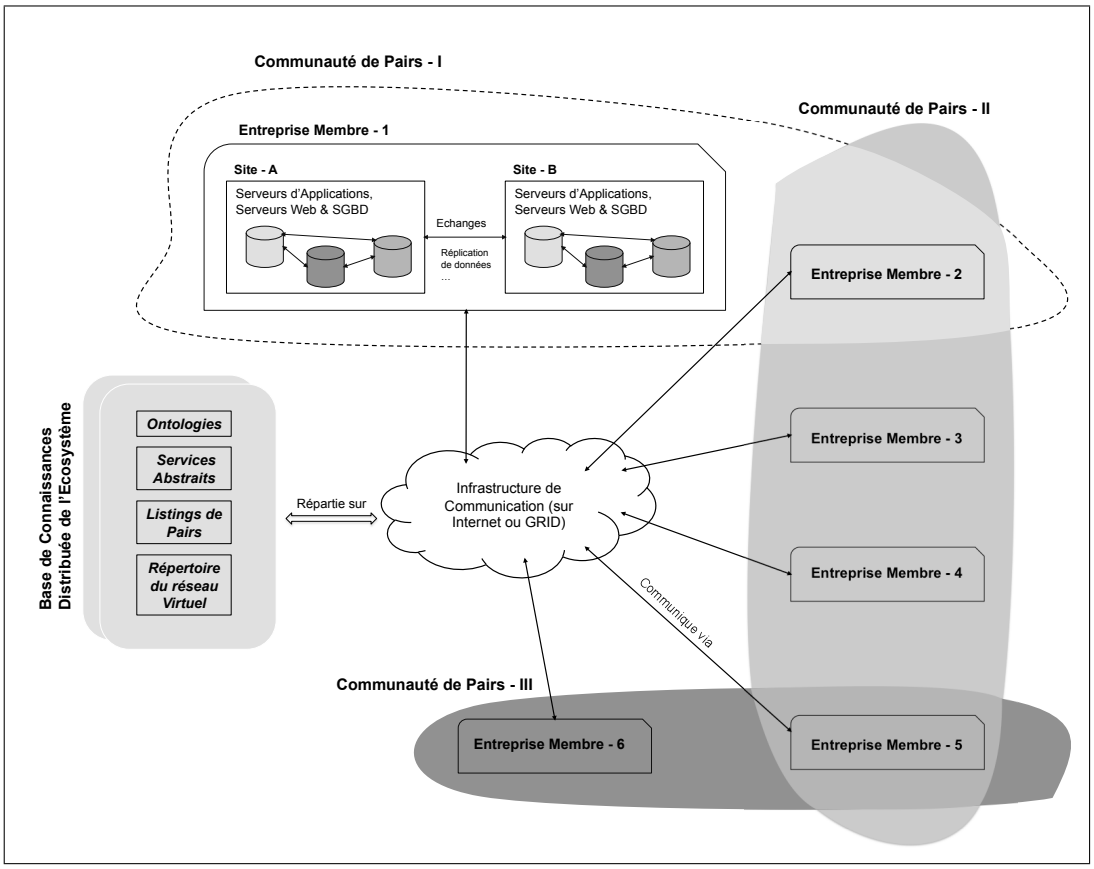


Figure 3.4: Description d'un Écosystème d'Entreprises

munes proposées dans l'écosystème. En d'autres termes, chaque ressource partagée dans le cadre de l'écosystème, qu'elle soit une activité automatisée ou non, est associée à une interface parmi les interfaces communes de description. Dans la suite nous désignons ces interfaces de ressources par *services abstraits*. Les entreprises membres sont reliées par une *infrastructure de communication*. Cette infrastructure consiste en un réseau pair-à-pair sur internet ou un réseau GRID. Les entreprises membres sont regroupées en *communautés*. Une communauté est construite selon une ou plusieurs propriétés qui peuvent être liées aux entreprises ou aux services fournis. Une entreprise fait partie d'une ou de plusieurs communautés selon les propriétés qu'elle remplit ou qu'elle décide de respecter. L'écosystème repose sur une *base de connaissances distribuée* contenant les différents éléments nécessaires à l'interaction des membres. Les détails concernant la structure et le contenu de cette base de connaissances sont discutés dans la section 3.2.4.3.

La figure 3.4 illustre un exemple d'écosystème d'entreprises. Elle détaille pour une entreprise membre, ses différents sites et sa structure de serveurs de bases de données, de serveurs web et de serveurs d'applications abritant son offre de services. Cette figure représente la base de connaissances de l'écosystème. Une entreprise membre utilise un ou plusieurs éléments de la base de connaissances afin de présenter et de décrire son savoir-faire, mais aussi comme guide pour interagir avec les autres membres. La figure 3.4 illustre aussi un exemple de trois communautés de pairs formées par les entreprises membres et l'infrastructure de communication les reliant.

3.2.3 Modélisation

Dans le cadre d'un écosystème, une entreprise membre est modélisée par un ou plusieurs *pairs* selon les points d'entrée qu'elle propose (URI, URL publique), tel que chaque point d'entrée constitue une entité séparée désignée par *pair*. Un ensemble de propriétés décrit le *consensus global* sur lequel repose l'écosystème. Ces propriétés sont respectées par l'ensemble des pairs membres [3]. L'écosystème définit et propose un ensemble de *services abstraits* basés sur le consensus global. Ces derniers (les services abstraits) modélisent les connaissances du domaine et les exigences métiers à travers des interfaces de programmation. Un pair peut fournir un ou plusieurs *services* [4] implémentant des interfaces de services abstraits, comme il peut en consommer (invoquer) un ou plusieurs. Un pair accède à un service membre à travers une communauté. Les communautés participent à organiser la collaboration entre les pairs fournisseurs de services et à l'orienter vers le métier correspondant. Les propriétés selon lesquelles une communauté est populée constitutent le *consensus local*. En résumé, un pair respecte à la fois le consensus global et un ou plusieurs consensus locaux. En effet, il s'engage à respecter les spécifications métiers et garantit le respect d'un nombre de contraintes sur les services qu'il fournit comme implémentation des services abstraits. La figure 3.5 illustre un diagramme de classe simplifié (UML), décrivant un écosystème d'entreprises où les pairs membres sont organisés en communautés. Chaque classe modélise un élément, une entité ou un concept de l'écosystème. Elle illustre les différentes relations entre les acteurs principaux de l'écosystème. Les différents éléments membres sont répertoriés dans la *base de connaissances distribuée* de l'écosystème décrite en section 3.2.4.3. Dans la suite nous définissons les différents éléments membres et l'ensemble de règles associées.

3. Dans la suite nous utilisons les termes *entreprise membre* et *pair membre* de façon interchangeable.

4. Désigné dans la suite par *service*, *service membre* ou *service fourni* de façon interchangeable.

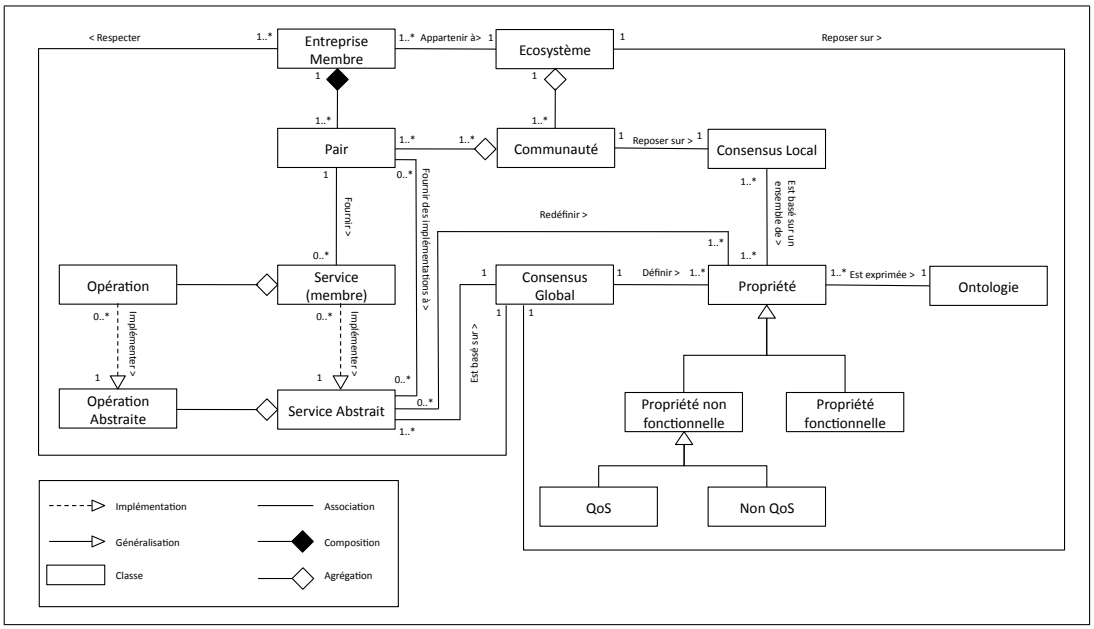


Figure 3.5: Modélisation d'un écosystème d'entreprises

3.2.3.1 Consensus global

Étant donné un écosystème E, le consensus global sur lequel repose E est un ensemble de propriétés significatives partagées par tous les membres de l'écosystème. Ces propriétés appartiennent à deux catégories :

- Les propriétés fonctionnelles représentent les fonctionnalités métiers communes par un ensemble de concepts. Elles sont décrites par rapport à une ontologie de référence adoptée par les membres de l'écosystème. Elles permettent de définir des éléments tels que les catégories et les paramètres d'entrée et de sorties des fonctionnalités proposées.

- Les propriétés non fonctionnelles représentent les caractéristiques qui ne se rapportent pas directement aux fonctionnalités métiers fournies par les membres de l'écosystème. Ces caractéristiques sont modélisées par un ensemble de "mesures" significatives pour les membres de l'écosystème. Elles sont décrites par une ontologie de référence.

Nous désignons par *Glob* l'ensemble des propriétés définies par le consensus global de l'écosystème. Nous proposons une notation formelle pour les propriétés non fonctionnelles qui sera utilisée dans la suite pour l'étude des coûts. L'ensemble des propriétés

non fonctionnelles, $GlobA \subset Glob$, est défini comme suit $GlobA = \{p^p\}_{p=1..q}$ où q est le nombre de propriétés non fonctionnelles significatives dans l'écosystème, telles que le *temps de réponse*, *le coût d'exploitation*, *etc.* qui sont des propriétés largement utilisées.

3.2.3.2 Consensus local

Le consensus local d'une communauté de pairs PC est un ensemble de propriétés partagées par les membres de la communauté. Il est noté par $Loc(PC) = \{p^r\}_{r=1..s}$ tel que $Loc(PC) \subset Glob$. Ce consensus est exprimé en termes de propriétés des services ou tout autre propriété significative pour les membres de l'écosystème étudié. Par exemple la fonctionnalité fournie, la distance physique, le niveau de confiance, etc.

Nous désignons par $P_{N_i} = \{p^r_{N_i}\}_{r=1..s',i\in\mathbb{N}}$ l'ensemble regroupant les propriétés d'un pair N_i et celles de ses services. Les pairs appartenant à la communauté PC_{Loc} doivent respecter le consensus local $Loc(PC)$. Cela se traduit par l'équation suivante :

$$\forall p^r_{N_i} \in Loc(PC),\ \forall N_i \in PC_{Loc} \Leftrightarrow p^r_{N_i} \in P_{N_i} \text{ où } r = 1..s,\ i \in \mathbb{N} \tag{3.1}$$

Par exemple, une communauté PC_{Loc} basée sur le consensus local Loc : *égalité-des-niveaux-de-confiance* englobera des pairs ayant la même valeur pour la propriété *niveau-de-confiance*.

Nous notons par $S(N_i)$ l'ensemble des services fournis par le pair N_i et par $S(PC)$ les services disponibles dans la communauté PC. $S(PC)$ est alors décrit comme l'union des ensembles de services fournis par des pairs respectant Loc(PC).

$$S(PC) = \bigcup_{i\in\mathbb{N}} S(N_i) \text{ où } N_i \text{ vérifie (3.1).} \tag{3.2}$$

3.2.3.3 Services abstraits

L'écosystème définit des services abstraits afin de disséminer des fonctionnalités et des connaissances métiers. Un service abstrait tel décrit par la figure 3.6, est une interface contenant un ensemble d'opérations abstraites nécessaires à l'accomplissement (exécution) d'une fonctionnalité associée. Le service abstrait ne fournit pas d'implémentation concrète permettant de réaliser les opérations en question. Chaque

service abstrait propose une définition de ses opérations, notamment à travers un ensemble de paramètres d'entrée - sortie et un ensemble de restrictions associées. En effet, il est possible de restreindre les valeurs des paramètres d'entrée et de sortie de chacune des opérations décrites. Cela permet de réduire le champ d'application de l'opération et par suite de spécialiser le service abstrait. Par exemple un service abstrait de conversion monétaire peut limiter sa sortie à une monnaie donnée. Un service abstrait de transformation audio - texte peut, par exemple, imposer les restrictions suivantes : (i) le fichier audio en entrée est de type "wave (.wav)" et (ii) le fichier texte en sortie est encodé en "ISO latin1" ou "MacOS Roman". Les services fournis dans l'écosystème comme implémentations de cette interface n'accepteront en entrée que des fichiers de type ".wav" et fourniront la transcription textuelle selon un des deux encodages acceptés. L'ensemble des descriptions des paramètres d'entrée et de sortie et les restrictions associées forment les propriétés fonctionnelles du service abstrait.

Les propriétés non fonctionnelles d'un service abstrait sont exprimées en fonction des propriétés non fonctionnelles définies par le consensus global. Elles peuvent exiger simplement l'évaluation d'une propriété p^p donnée, comme elles peuvent définir en plus une contrainte de réalisation sur p^p. Par exemple, pour la propriété *temps de réponse* définie par le consensus global, un service abstrait peut simplement définir la propriété rattachée *temps d'exécution*. De cette façon toute implémentation de ce service abstrait doit évaluer son *temps d'exécution*. Il peut en plus associer une contrainte de réalisation à la propriété *temps d'exécution* comme par exemple *temps d'exécution* $<$ 1 seconde. De cette façon toute implémentation de ce service abstrait doit évaluer son *temps d'exécution* et garantir qu'il sera inférieur à une seconde.

Un service abstrait est désigné par A_i. Nous dénotons par A l'ensemble des services abstraits de l'écosystème étudié. $|A| = n$, n étant le nombre total de ces services. Pour chaque service abstrait A_i nous définissons un ensemble de propriétés non fonctionnelles P_i tel que $P_i = \{p_i^{m_i}\}_{m_i=1..q_i,\ i\in[1;n]}$ où $q_i \leq q$ et $\exists\ f :\ p_i^{m_i} \rightarrow p^p$. Nous soulignons qu'un service abstrait donné ne doit pas forcément redéfinir toutes les propriétés non fonctionnelles de l'écosystème. En effet, certaines propriétés ne sont pas significatives par rapport à l'activité modélisée.

Les services abstraits d'un écosystème E doivent être conformes au consensus global de E. Cela ce traduit pour chaque service abstrait par (i) l'adoption de l'ontologie de référence pour définir ses propriétés fonctionnelles et par (ii) la description

de ses propriétés non fonctionnelles à l'aide des propriétés décrites par $GlobA$:

$$\forall p_i^{m_i} \in P_i \ \exists \ p^p \in GlobA \text{ tel que } f(p_i^{m_i}) = p^p \ . \tag{3.3}$$

$$\text{où } i \in [1;n], m_i \in [1;q_i], p \in [1;q]$$

3.2.3.4 Services membres

Les services sont définis par les pairs membres afin de présenter leur offre d'activités métiers dans le cadre de l'écosystème. Les services permettent à un pair de décrire son savoir-faire et de le partager avec les autres membres. Chaque service constitue une implémentation d'un service abstrait correspondant. Un service $S_{i\,k_i}$ implémentant un service abstrait A_i doit redéfinir l'opération abstraite de A_i ou les opérations abstraites si le cas se présente et doit respecter la totalité des contraintes associées. La figure 3.6 reprend le concept de service décrit par la figure 3.1 et situe les services membres vis-à-vis des services abstraits. Deux services $S_{i\,k}$ et $S_{i\,k'}$ bien qu'ayant des interfaces fonctionnellement similaires peuvent différer par leurs propriétés non fonctionnelles.

Les propriétés non fonctionnelles d'un service $S_{i\,k_i}$ sont déduites par évaluation de l'ensemble des propriétés non fonctionnelles du service abstrait correspondant A_i. Par exemple la propriété *temps d'exécution* de A_i est redéfinie et évaluée par $S_{i\,k_i}$. Nous notons par $P_{i\,k_i}$ l'ensemble des propriétés non fonctionnelles du service $S_{i\,k_i}$, $P_{i\,k_i} = \{p_{i\,k_i}^{m_i}\}_{i\in[1..n],\ m_i=1..q_i,\ k\in\mathbb{N}}$.

Dans le cadre des partenariats entre les membres d'un écosystème, les services fournis comme implémentations des services abstraits respectent une relation d'implémentation notée par $IMPD(service,\ serviceAbstrait)$. Cette relation définit l'association entre un service membre et le service abstrait qu'il implémente et se traduit par (i) la redéfinition des opérations abstraites de A_i en respectant les propriétés fonctionnelles associées et par (ii) le respect de l'équation suivante :

$$\forall S_{i\,k_i}, \forall A_i \in A,\ i \in [1;n],\ k_i \in [1;n_i]$$

$$IMPD(S_{i\,k_i}, A_i) \implies \forall p_i^{m_i} \in P_i,\ \exists\ p_{i\,k_i}^{m_i} \in P_{i\,k_i} \text{ où } m_i \in [1;q_i] \tag{3.4}$$

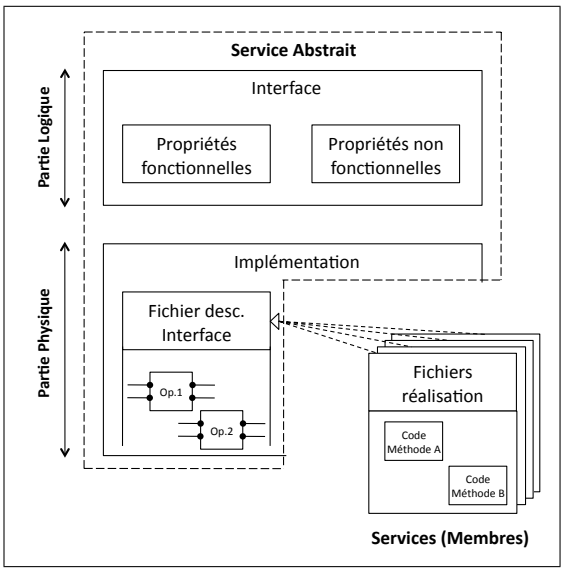


Figure 3.6: Illustration d'un service abstrait et de ses implémentations

3.2.4 Réalisation

Dans cette partie nous décrivons les modalités de réalisation de l'écosystème et de ses divers éléments.

3.2.4.1 Propriétés de l'écosytème

Les propriétés de l'écosystème sont exprimées à travers deux types d'ontologies de référence partagées par les membres de l'écosystème. Uschold et Grüninger [142] définissent une ontologie comme l'ensemble de connaissances partagées d'un domaine donné. Selon [56], une ontologie est une spécification explicite d'une conceptualisation. Nous retenons qu'une ontologie est une modélisation formelle et structurée. Elle fournit une représentation hiérarchique des différents éléments symbolisant les constituants d'un domaine donné et les relations entre ces éléments. La description d'une ontologie est accessible à travers le web et consitue un modèle informatif compréhensible autant par l'Homme que par la machine. D'où la possibilité d'automatiser le traitement d'une ontologie. Les principaux éléments d'une onotlogie décrite en logique descriptive sont : (*i*) les concepts, (*ii*) les relations et les (*iii*) instances. Un concept représente un objet, un terme ou une notion comme par exemple un véhicule, une personne, une facture etc. Une relation ou rôle est une association

binaire entre deux concepts. Elle est exprimée par un prédicat logique. Parmi les relations usuelles nous retenons les relations suivantes : *isA*, *partOf*, *hasPart*, *closeTo*, *over*, *under*, *contain*, *connectedTo*. Une instance ou extension de concept est une valeur attribuée à un concept donné. Par exemple, *Pierre* est une instance du concept *Personne*. Les ontologies servent à annoter les propriétés de l'écosystème et celles des services abstraits. Parmi les ontologies de référence indispensables à la description de l'écosystème et de ses propriétés nous retenons deux types majeurs :

- L'ontologie des propriétés fonctionnelles. Elle contient un ensemble de concepts et de relations associées modélisant les fonctionnalités requises et couvertes par les services abstraits. Cette même ontologie ou une ontologie rattachée décrits les concepts et les relations associées modélisant les ensembles d'entrées-sorties des services abstraits décrits dans l'écosystème.
- L'ontologie des propriétés non fonctionnelles significatives aux membres de l'écosystème. Des éléments de cette ontologie sont associés aux propriétés non fonctionnelles des services abstraits.

3.2.4.2 Ressources de l'écosytème

Deux aspects concernant les ressources sont à étudier. D'abord les services abstraits uniformisant les fonctionnalités métiers et ensuite les services membres (ou services fournis) concrétisant le savoir-faire des entreprises membres.

Services Abstraits Précédement, dans la section 3.2.3.3, nous avons défini un service abstrait comme une interface qui se rapporte à une fonctionnalité donnée et possédant un ensemble de propriétés rattachées. Nous proposons de rajouter des annotations sémantiques aux divers éléments de l'interface d'un service abstrait afin d'augmenter la précision de la description. En effet, pour une activité fonctionnelle donnée, automatisée ou non, l'interface du service abstrait est un ensemble de données structurées ou semi-structurées contenant des informations décrivant les opérations nécessaires pour accomplir cette activité. Chaque opération possède un ensemble de paramètres en entrée et fournit un ensemble de paramètres en sortie. Afin de décrire sémantiquement ces paramètres, nous rattachons à chacun une annotation sémantique vis-à-vis d'une ontologie de référence décrite dans l'écosystème. Les restrictions et les contraintes de réalisation sont exprimées sur les concepts des ontologies de référence. Particulièrement, les restrictions peuvent être

exprimées lors de la définition des paramètres d'entrée - sortie à travers l'élément XML $< xs : restriction >$.

Les services abstraits sont définis par les membres de l'écosystème après la mise place du consensus global (section 3.2.3.1). L'uniformisation des interfaces de description des fonctionnalités fournies dans l'écosystème permet de faciliter la découverte du point de vue de l'utilisateur. En effet, une partie du processus de découverte est accomplie par le fournisseur du service dans la mesure où il implémente directement ou indirectement l'interface d'une description communément adoptée par les membres de l'écosystème. L'*implémentation directe* consiste à fournir un service ayant une interface identique à un service abstrait proposé dans l'écosystème ; tandis que l'*implémentation indirecte* consiste à fournir les correspondances entre l'interface du service proposé et un service abstrait équivalent.

Services Membres Concrètement, comme évoqué dans la section 3.2.2, les services membres sont classés en deux catégories : des services encapsulant une activité automatisée et d'autres encapsulant une activité nécessitant une intervention humaine.

Les éléments de la première catégorie sont réalisés par des services web ayant une interface WSDL. Si une description WSDL [5] contient plusieurs opérations correspondant à des activités fonctionnelles différentes, chaque ensemble d'opérations rattachés à une fonctionnalité donnée sera modélisé par un service abstrait dans l'écosystème de DyCoSe. Par exemple les services $S_{1\,3}$, $S_{2\,1}$ et $S_{3\,5}$, constituant des implémentations respectives des services abstraits A_1, A_2 et A_3, peuvent être décrits dans un même fichier WSDL. Les éléments de la deuxième catégorie sont réalisés également par des services web (WSDL) proposant des opérations où les entrées et les sorties sont définies mais le traitement sera accompli par une personne et non pas par la machine. La spécification WS-HumanTask [44], aborde la définition des activités "humaines" (*service-enabled human tasks*), non accomplies par la machine et leurs associe des propriétés et un ensemble d'opérations. WS-HumanTask propose un protocole de coordination afin de gérer la durée de vie des activités humaines de façon uniforme garantissant l'interopérabilité entre elles et avec d'autres services web.

5. Détaillé en section 2.1.1

3.2.4.3 Base de connaissances de l'écosytème

La base de connaissances de l'écosystème, comme décrite auparavant, contient les différents éléments réalisant l'écosystème. Elle regroupe l'ensemble des fichiers décrivant les ontologies nécessaires à la définition du consensus global et des services abstraits. L'ensemble des propriétés définissant le consenus global est recensé dans un fichier semi-structuré. Ce fichier contient aussi la liste des services abstraits disponibles ainsi que les adresses (URL/URI) respectives de leurs interfaces associées. La base de connaissances pourrait comprendre, si les membres de l'écosystème le souhaitent, un historique des applications créées et exécutées, contenu dans un fichier semi-structuré pointant vers le journal d'exécution de chaque application. La base de connaissances de l'écosystème contient aussi un fichier listant les entreprises membres à travers leurs pairs. En effet, comme discuté dans la section 3.2.3, une entreprise est modélisée par un ou plusieurs pairs, tel que chaque pair est un point d'entrée accessible aux membres de l'écosystème (adresse URL/URI d'une interface).

Concrètement, la base de connaissances de l'écosystème est répartie sur le réseau de membres. Les détails de cette répartition sont discutés dans la section 3.3.3.1, après la description de la réorganisation du réseau de membres en un réseau virtuel hiérarchique.

3.2.5 Exemple d'un écosystème

Nous prenons l'exemple d'un écosystème où les fournisseurs sont connectés par un réseau d'opérateurs mobiles. Les entreprises membres coopèrent afin de fournir des éléments fonctionnels pour construire des applications mobiles. Parmi les acteurs possibles nous retenons :

- les opérateurs qui assurent et gèrent (le service de) la connectivité. Ils peuvent fournir des services d'authentification, de paiement ou de recherche, etc. ;
- les fournisseurs de services de recherches (hôtels, restaurants, taxis etc.) ;
- les fournisseurs de services d'utilitaires (station de services, gares, garages, parkings, etc.) ;
- les fournisseurs de services d'assistance (traduction, conversion monétaire, positionnement, etc.) ;
- les fournisseurs de services multimédia (vidéo, réseaux sociaux, etc.) ;

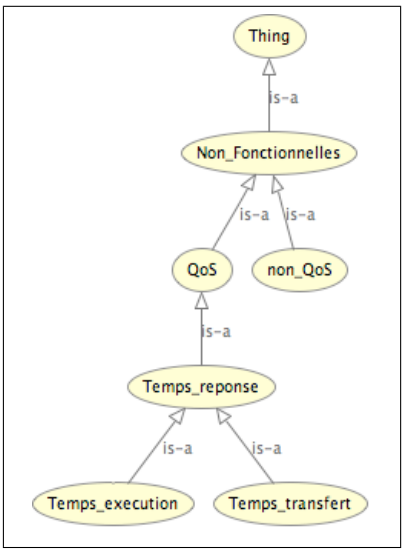


Figure 3.7: Illustration d'un exemple d'ontologie

– les fournisseurs de services logistique (dépannage, livraison, etc.).

Les membres de cet écosystème notamment les fournisseurs de services et les clients mobiles, sont organisés en communautés. Par exemple, les communautés peuvent être formées par opérateur, c'est-à-dire chaque communauté contient des membres connectés par le même opérateur. Nous pouvons imaginer un nombre considérable d'applications composées à base des services fournis dans cet écosystème. À titre d'exemple, l'utilisateur en déplacement sur une route peut invoquer à travers son opérateur un ensemble de services afin de déterminer la station de service la plus proche de sa position actuelle et éventuellement le parking le mieux situé vis-à-vis de sa destination d'arrivée. Un autre utilisateur peut rechercher un itinéraire quelconque et demander au système de lui lire les indications, comme il peut procéder à des achats de dernière minute à travers le réseau de son opérateur et programmer leurs livraisons.

Exemple de propriétés d'un écosystème À titre illustratif, pour exprimer un exemple des ontologies de l'écosystème étudié nous avons choisi le langage OWL [9]. Les principaux éléments d'une ontologie OWL sont : (*i*) les classes et sous-classes, (*ii*) les individus et les (*iii*) propriétés. Une classe représente un concept. Une sous-classe représente une spécification du concept associé. Les individus constitutent les instances des classes et des sous-classes. Les propriétés sont des relations binaires

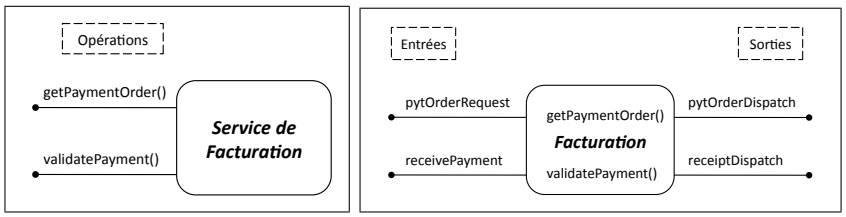


Figure 3.8: Illustration d'un service abstrait

de deux types : (*i*) les propriétés "types de données" qui sont des relations entre les instances des classes et les types de données d'un schéma xml, et (*ii*) les propriétés "objets" qui sont des relations entre instances de deux classes différentes.

Afin de modéliser les aspects fonctionnels de la facturation par exemple, nous adoptons une ontologie du domaine financier proposée dans [80]. Elle nous servira comme référence pour annoter les paramètres des opérations des services abstraits de facturation par exemple. Plusieurs classifications (taxonomies, ontologies) des propriétés non fonctionnelles sont disponibles, notamment les standards ISO 8402 [61] et ITU E.800 [63]. En annexe C, nous présentons un extrait d'une ontologie simplifée, écrite en OWL et qui définit quelques propriétés non fonctionnelles de l'écosystème. La figure 3.7, générée sous OWL protégé [6], propose une réprésentation graphique équivalente au code présenté.

Exemple de services abstraits Nous fournissons en annexe D, un exemple de description d'un service abstrait en WSDL 1.0. Des annotations SAWSDL sont rajoutées afin d'apporter des précisions sémantiques. Le code présenté décrit l'interface du service abstrait de facturation, en particulier les opérations *getpaymementOrder* (ligne 77) et *validatePayment* (ligne 82) du point de vue de l'acteur recevant un paiement. La première opération reçoit en entrée une demande de devis et retourne un devis, tandis que la deuxième opération reçoit en entrée une certificat de paiement et renvoie un reçu. Les annotations SAWSDL de l'opération *validatePayment* (ligne 82) vis-à-vis de l'ontologie présentée en [80] sont les suivantes :

- les lignes 84,86 indiquent que le message en entrée *receivePayment* correspond au concept *InternalTransfer* de l'ontologie [80] ;
- les lignes 87,89 indiquent que le message en sortie *receiptDispatch* correspond au concept *Receipt* de l'ontologie [80].

6. `http://protege.stanford.edu/overview/protege-owl.html`

La figure 3.8 propose deux illustrations d'un service abstrait de facturation. La première décrit le service à travers ses opérations tandis que la deuxième détaille les paramètres d'entrée et de sortie de chaque opération.

3.3 Composition dans DyCoSe

Nous proposons une architecture de composition à trois niveaux, à savoir, le *niveau fonctionnel*, le *niveau services* et le *niveau infrastructure*. Cette architecture est illustrée par la figure 3.9. Chaque niveau est composé de plusieurs couches. D'abord le niveau fonctionnel supporte un raffinement des fonctionnalités de l'application. Une première couche décrit l'application et son domaine. Une deuxième couche constitue un raffinement par acteurs du modèle applicatif décrit et une troisième couche caractérise ce dernier à travers un raffinement par métiers. Ensuite le niveau services constitue la réalisation de l'application raffinée à travers les services disponibles. Une première couche contient la représentation de l'application raffinée à travers un diagramme de services abstraits du point de vue du métier choisi. Une deuxième couche englobe les réalisations possibles du diagramme de services abstraits en utilisant les services fournis dans l'écosystème. Finalement le niveau infrastructure concrétise les échanges de messages entre les membres de l'écosystème. Il comporte deux couches. Une première qui constitue la réorganisation des membres en un réseau virtuel de communautés structurant les échanges de messages. Une deuxième qui n'est autre que le réseau physique sous-jacent évaluant les propriétés des fournisseurs et celles des liens les reliant.

L'architecture proposée permet une approche de composition d'applications à base de services qui combine à la fois une démarche descendante et une autre ascendante tel décrit par la figure 3.9. La démarche descendante, du niveau fonctionnel jusqu'au niveau services, permet de modéliser l'application, de gérer le raffinement fonctionnel et de propager les contraintes de réalisation et les préférences de l'utilisateur en vue de choisir une composition de services répondant aux contraintes de l'utilisateur. La partie ascendante de la démarche projette les caractéristiques de l'infrastructure réseau sous-jacente au niveau services, permettant d'évaluer les propriétés et les contraintes étudiées.

Dans la suite nous développons chacun des niveaux en décrivant les couches qui le constitue. Nous illustrons la démarche de composition d'applications associée à l'architecture proposée à travers des exemples basés sur des applications de l'écosystème

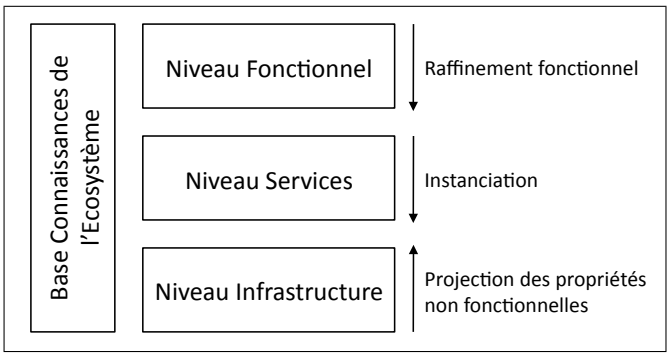


Figure 3.9: Architecture de composition

décrit en section 3.2.5.

3.3.1 Niveau fonctionnel

Ce niveau consiste d'abord à décrire les fonctionnalités accomplies par l'application à travers un "workflow" d'interaction entre composants de haut niveau, ensuite à identifier les différents acteurs impliqués et finalement discerner les métiers associés dans le but de modéliser l'application requise par une ou plusieurs orchestrations de services abstraits disponibles dans l'écosystème. À cette fin, après chaque étape de raffinement une ou plusieurs correspondances entre une activité fonctionnelle et une *opération* d'un service abstrait sont identifées. L'activité ou les activités en question ne seront plus raffinées dans les étapes suivantes et seront illustrées dans les diagrammes des" workflows" par la représentation des services abstraits équivalents (ou des opérations abstraites équivalentes). Un service abstrait (ou une opération abstraite) est représenté graphiquement par un rectangle aux bords arrondis contenant un cercle noirci dans le coin supérieur droit, comme illustré dans la figure 3.11. Dans la suite nous décrivons chacune des trois étapes représentées comme couches composant le niveau fonctionnel dans la figure 3.10.

3.3.1.1 Couche domaine

Au niveau de cette couche, les fonctionnalités accomplies par l'application sont décrites à travers un workflow de composants haut niveau. Nous adoptons le Business Process Modeling Notation (BPMN [55, 151]) comme modèle de description de cette interaction. La notation BPMN définit un diagramme de processus métier (Bu-

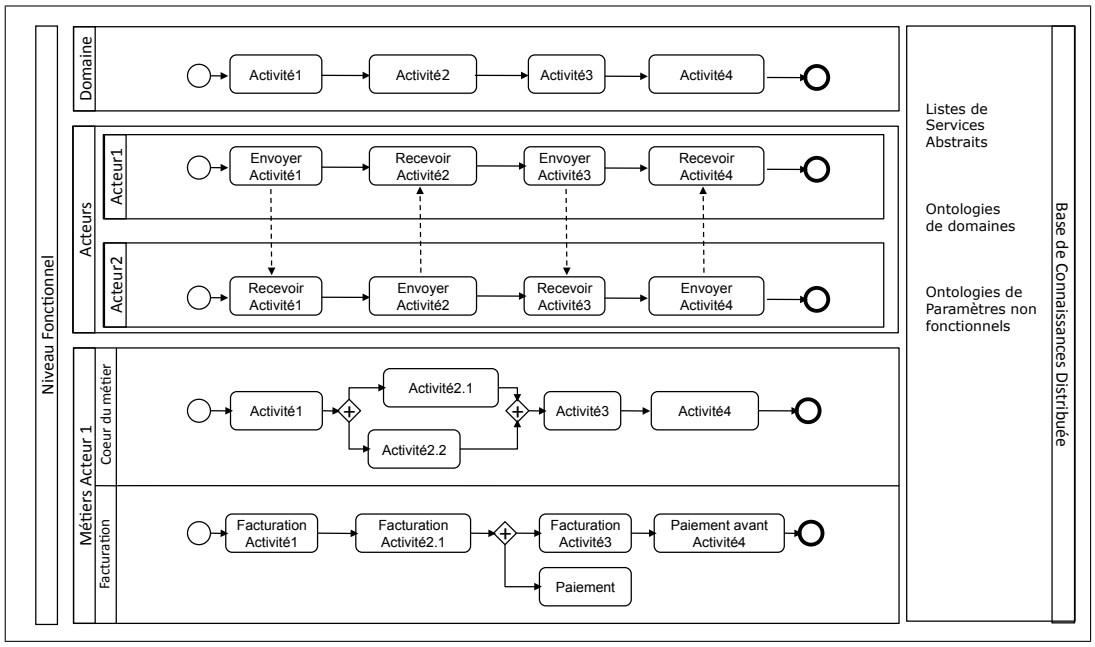


Figure 3.10: Niveau Fonctionnel

siness Process Diagram BPD) qui n'est autre qu'un ensemble d'éléments graphiques constitué d'activités et de contrôles de flux définissant l'interaction entre les activités. Au niveau de cette couche, les représentations graphiques des notations BPMN suivantes suffisent pour décrire la logique applicative. Ces éléments graphiques sont illustrés dans la figure 3.11. De la catégorie *objets de flux*, les représentations *activité* (activity), *sous-processus* (sub-process) et *passerelle* (gateway) sont retenues. Une *activité* est représentée par un rectangle aux bords arrondis. Elle désigne une fonctionnalité applicative ou activité [7]. Un *sous-processus* est représenté par un rectangle aux bords arrondis contenant le symbole *plus* centré en bas du rectangle. Il désigne une fonctionnalité applicative composite. Il permet de mieux illustrer les niveaux de granularité dans le diagramme d'un processus. Une *passerelle* est représentée par un losange. Elle est utilisée pour désigner la convergence ou la divergence d'un ensemble de *flux séquentiels*, notamment pour designer un branchement décisionnel, un branchement en parallèle ou une jointure entre deux ou plusieurs chemins. Un losange contenant le symbole *plus* désigne un branchement ou une jointure parallèle, tandis qu'un losange vide indique un branchement ou une jointure décisionnelle exclusive.

7. Dans la suite nous utilisons de façon interchangeable les termes *fonctionnalité*, *fonctionnalité applicative* et *activité*.

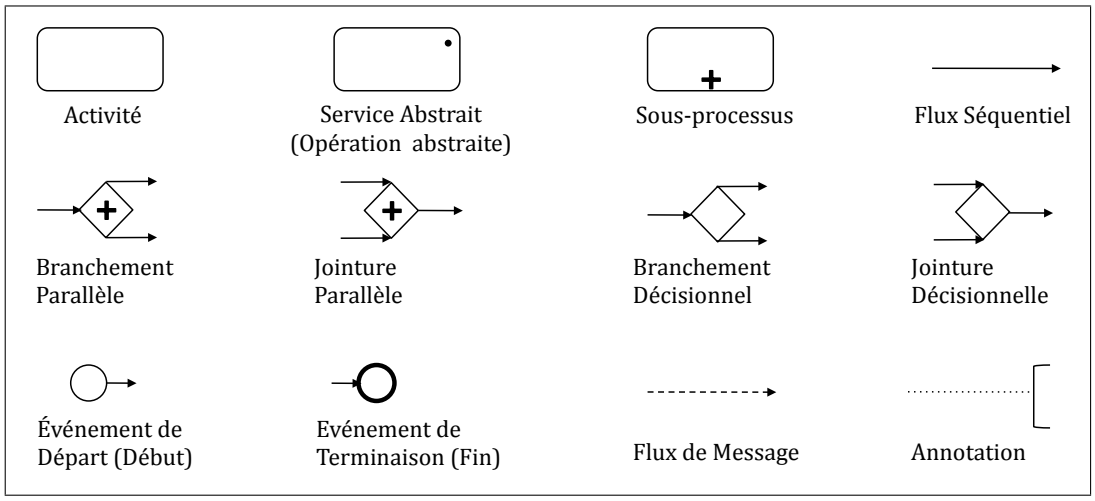


Figure 3.11: Représentation graphique des élements d'un diagrame de processus BPD

De la catégorie *objets de connexion* nous retenons le *flux séquentiel*. Il est représenté par une flèche. Il désigne l'ordonnancement des activités. Finalement de la catégorie *événements* deux représentations sont indispensables, un cercle à périmètre fin indiquant le début d'un processus d'interaction et un cercle à périmètre gras indiquant une fin (terminaison) d'un processus d'interaction. Des exemples d'illustration sont donnés dans les figures 3.12, 3.13, 3.14, 3.15, 3.16.

Nous reprenons l'exemple de l'écosystème décrit en section 3.2.5. Supposons que l'utilisateur connecté à travers son PDA veut faire un achat et programmer sa livraison à distance. La modélisation haut niveau de l'application est illustrée par la figure 3.12. L'application est représentée par un diagramme BPD impliquant une séquence d'interactions entre un service de *sélection de produit*, un service de *paiement* et un service de *livraison*.

3.3.1.2 Couche acteurs

Au niveau de la couche acteurs, un premier raffinement est appliqué au diagramme BPD de la couche domaine. Il consiste d'abord à identifier les différents acteurs impliqués mettant en évidence leurs rôles respectifs. Il consiste ensuite à détailler les fonctionnalités pour chaque acteur. Le choix du niveau de détail dépend du besoin de l'utilisateur (le concepteur). Un acteur est défini comme une entité "physiquement" séparée interagissant avec d'autres entités à travers une (ou plu-

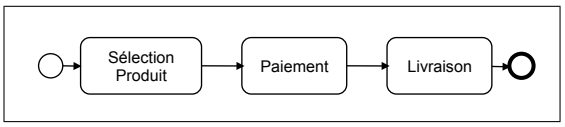


Figure 3.12: Fonctionnalités haut-niveau d'une application d'achat en ligne

sieurs) application(s) commune(s). Par ce fait, à chaque application nous associons un groupe d'acteurs, par exemple le groupe <acheteur - vendeur - livreur - centre de paiement> ou le groupe <patient - médecin - pharmacien - sécurité sociale> ou le groupe <État - Citoyen - Administration> etc. Le but du raffinement par acteurs est d'identifier à travers l'illustration des échanges de messages les rôles des différents acteurs afin de sélectionner la perspective à partir de laquelle l'application sera raffinée davantage et implémentée par la suite.

La figure 3.13 illustre un exemple du premier niveau de raffinement de l'application décrite par la figure 3.12. En effet, elle détaille les activités à accomplir lors d'une exécution de l'application d'achat en ligne mettant en évidence les acteurs suivants : l'acheteur ou l'*utilisateur*, le *vendeur* et le *livreur*. En plus du raffinement fonctionnel, la figure 3.13 illustre l'échange de messages entre ces différents acteurs. Cette description des différents acteurs permet de mettre en avant différents points de vue de la même application. De plus, elle souligne la nature distribuée de l'application illustrée par les échanges de messages décrits.

L'application décrite par la figure 3.13 illustre le rôle des acteurs impliqués en détaillant les activités accomplies par chacun. Tout d'abord l'interaction commence entre l'acheteur et le vendeur. En effet, l'acheteur envoie un message au vendeur demandant un catalogue. Cet dernier renvoie les informations nécessaires dans un message correspondant. L'acheteur, après sélection du (des) produit(s) désiré(s), envoie un ordre d'achat au vendeur qui répond par un ordre de paiement. Après réception du paiement, le vendeur entre en contact avec un livreur lui envoyant un message contenant la demande de livraison. Le livreur confirme au vendeur la livraison et entame l'envoi. Après réception du message de confirmation de la livraison, le rôle du vendeur dans cette interaction prend fin. L'exécution du processus métier de l'application d'achat s'achève une fois que l'acheteur reçoit ses produits.

La figure 3.13 met en évidence, à titre illustratif, six exemples de correspondances entre des activités métiers et des opérations abstraites proposées par les services abstraits de l'écosystème, à savoir, *Recevoir Demande Catalogue*, *Envoyer*

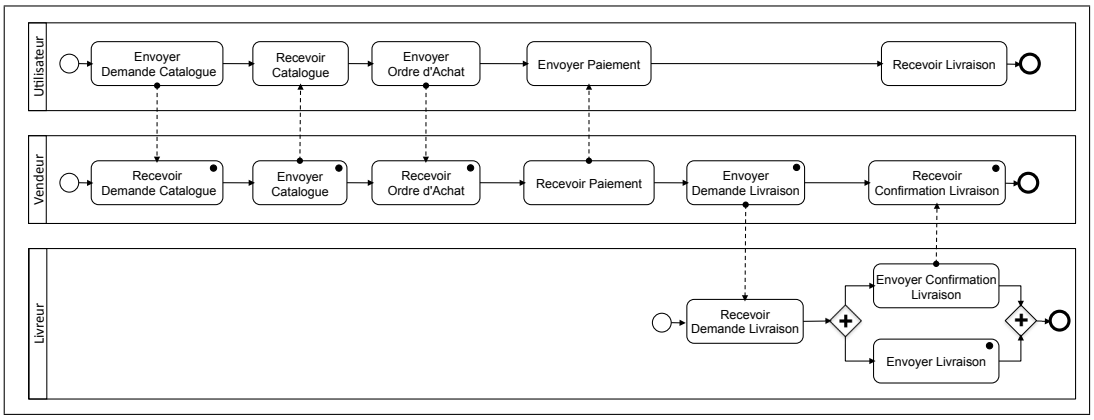


Figure 3.13: Différents acteurs d'une application d'achat en ligne

Catalogue, *Recevoir Ordre d'Achat*, *Envoyer Demande Livraison*, *Recevoir Confirmation Livraison*, *Envoyer Livraison*. L'identification des activité abstraites implique qu'elles ne seront plus raffinées dans les étapes suivantes. Nous soulignons l'activité *Envoyer Livraison* qui correspond à l'acteur fournisseur et qui constitue un exemple d'une opération abstraite pouvant avoir des implémentations concrètes où les activités représentent des tâches manuelles nécessitant une intervention humaine comme discuté dans les sections 3.2.2 et 3.2.4.2.

Le raffinement par acteurs fournit un modèle de description applicative illustrant une vision globale de l'interaction entre deux ou plusieurs acteurs (participants). Cette vision est décrite à travers les séquences d'échanges de données. Cela rappelle le modèle de chorégraphie entre services web *WS-CDL* [145]. En effet, la chorégraphie de services est définie par Peltz comme une description collaborative où chaque participant présente le rôle qu'il joue dans l'interaction [114]. Elle met en évidence les séquences de messages impliquant plusieurs entités afin d'illustrer le comportement de plusieurs systèmes interagissant. Elle donne une vue du comportement externe visible. Par analogie, la couche acteurs fournit un raffinement du modèle d'interaction à travers une chorégraphie entre les différents acteurs.

D'autre éléments de représentation BPMN sont utiles à ce niveau. L'élément graphique *pool* (*bassin*) de la catégorie *swimlane* de BPMN est utilisé pour séparer les vues des différents acteurs. De la catégorie objets de connexion nous retenons le *flux de messages*. Il est représenté par une flèche en pointillés. Il désigne un échange de messages entre deux entités séparées du processus notamment deux acteurs. Contrairement au *flux séquentiel*, le *flux de messages* peut traverser les bordures d'un *pool*.

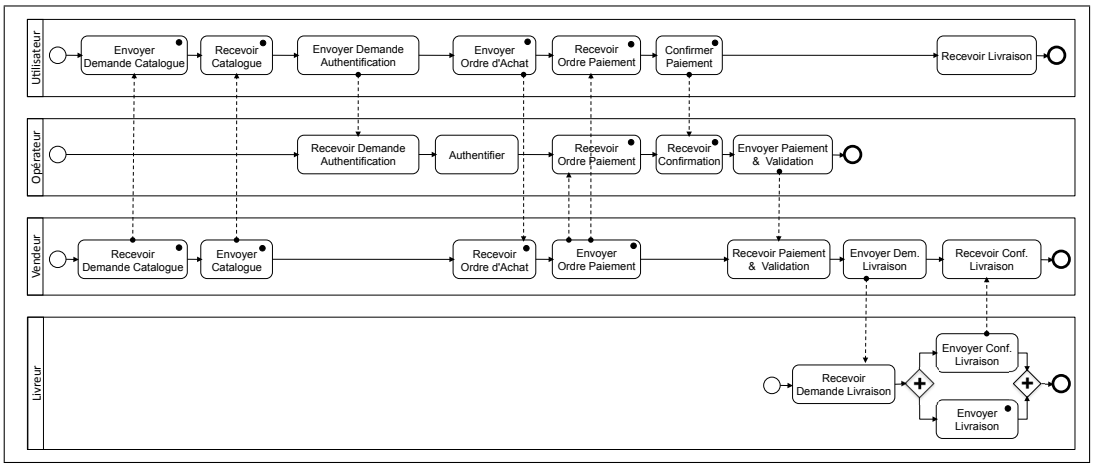


Figure 3.14: Variante des différents acteurs d'une application d'achat en ligne

La figure 3.14 illustre une variante plus détaillée du raffinement proposé dans la figure 3.13. En effet la figure 3.13 décrit trois acteurs tandis que la figure 3.14 introduit un acteur supplémentaire qui intervient dans l'interaction : l'opérateur de téléphonie mobile à travers lequel l'acheteur est connecté via son PDA. L'introduction de ce nouvel acteur permet une vue (visibilité) plus large du domaine et par suite une application englobant de nouvelles fonctionnalités. Effectivement, la figure 3.14 décrit l'échange de messages avec l'opérateur mettant en évidence le rôle qu'il peut jouer dans l'authentification de l'utilisateur et l'éventuel paiement. Nous pouvons imaginer qu'un fournisseur de services de vente passent un contrat avec un ou plusieurs opérateurs afin de publier ses services sur leurs réseaux. L'acheteur pourra alors effectuer le paiement par l'intermédiaire de son opérateur.

3.3.1.3 Couche métiers

À ce niveau, partant de la modélisation du point de vue de chaque acteur du niveau supérieur, un raffinement par métier est conduit pour un acteur sélectionné. L'utilisateur (concepteur) choisit le point de vue de l'acteur qui lui convient afin de développer le processus métier de sa perspective. Généralement, l'utilisateur assure le rôle d'un des acteurs de l'application étudiée. Par exemple dans le cadre d'une application de gestion de voyages, si l'utilisateur de cette démarche est le client qui cherche à composer son itinéraire, ce dernier décrira le processus métier et les échanges de messages correspondant à son point de vue. Cependant si l'utilisateur

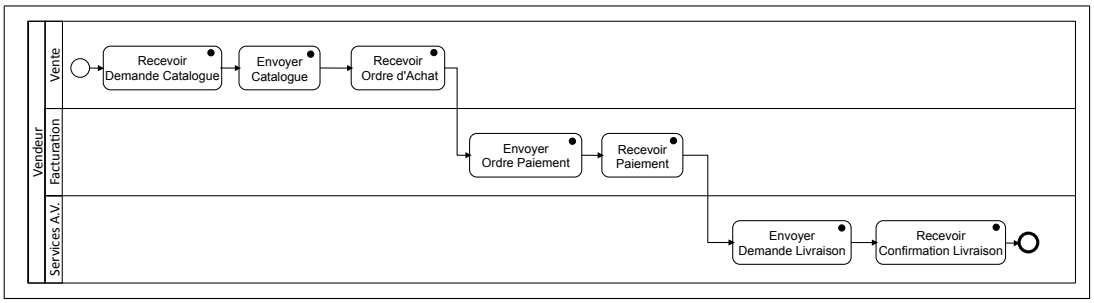


Figure 3.15: Métiers sollicités dans une application d'achat en ligne du point de vue du vendeur

de cette démarche est une compagnie aérienne, le processus métier décrit de sa perspective sera, bien évidemment, structurellement différent de celui décrit par le client mais aussi fonctionnellement distinct. En effet, le processus métier décrivant l'interaction de la perspective du client cherchant à composer son itinéraire offre une vision plus détaillée du domaine en comparaison le processus métier décrit du point de vue d'une compagnie aérienne.

La figure 3.15 illustre l'application étudiée du point de vue du vendeur. Les entités acheteur et livreur n'apparaissent pas dans le BPD à ce niveau. Le BPD de la figure 3.15 met en évidence trois métiers liés au vendeur, en l'occurrence le service de vente qui traite la gestion des ventes et qui constitue le cœur du métier, la facturation qui gère l'aspect financier de l'interaction et les services après-vente qui gèrent ou sous-traitent, entre autres, la livraison. Les activités décrites dans la figure 3.13 sont reprises dans le diagramme de la figure 3.15. Une première différenciation (raffinement) les classe par métier correspondant. Par exemple les activités reliées au paiement telles que *Envoyer Ordre Paiement* et *Recevoir Paiement* sont regroupées dans le *couloir* (*lane*) réservé au métier de facturation. Éventuellement plusieurs étapes de raffinement fonctionnel peuvent suivre selon le niveau de granularité choisi par le concepteur.

Afin de représenter les BPD illustrant le raffinement de la couche métiers, un élément supplémentaire de la notation BPMN est adopté. La représentation graphique de l'élément *lane* (*couloir*) de la catégorie *swimlane* permet de séparer entre les différents métiers dans le *pool* (*bassin*) de l'acteur choisi. En effet comme l'illustre les figures 3.15 et 3.16 les *lanes* séparent les groupes d'activités par métiers. Le *flux séquentiel* peut traverser les *couloirs* dans un même *bassin* d'un acteur donné.

Le raffinement par métiers fournit un modèle de description applicative

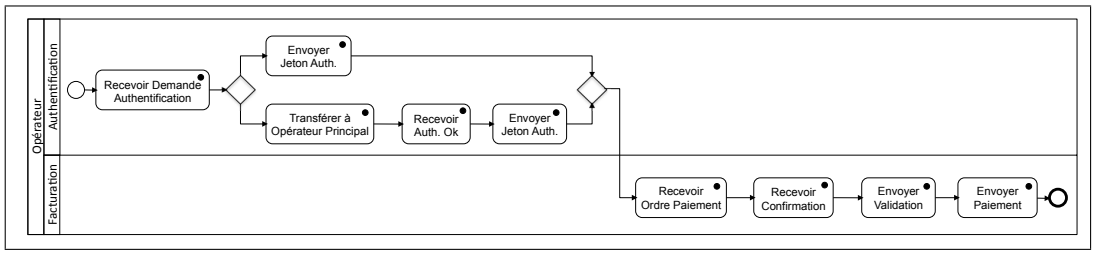


Figure 3.16: Variante des métiers sollicités dans une application d'achat en ligne du point de vue de l'opérateur

catégorisant les activités par groupes fonctionnels et illustrant leur ordre d'exécution. Cela rappelle le modèle d'orchestration entre services web : *web services orchestration*. En effet, l'orchestration de services fournit un modèle de description de l'interaction entre deux ou plusieurs services web. Le modèle de l'orchestration illustre la logique métier à travers l'ordre d'exécution des interactions prévues. Dans une orchestration de services "le processus d'interaction est toujours contrôlé de la perspective d'un participant" [114]. Par analogie, la couche métiers fournit un raffinement du BPD à travers une orchestration entre les activités des différents métiers impliqués de la perspective de l'acteur choisi.

La figure 3.16 illustre un raffinement par métiers du BPD de la figure 3.14. Ce raffinement est réalisé du point de vue de l'opérateur. Deux métiers se rapportant à l'opérateur sont identifiés dans ce BPD. Le premier métier est l'authentification. Après réception de la demande d'authentification l'opérateur a deux choix modélisés par l'emploi d'une passerelle décisionnelle dans le BPD. Si l'utilisateur qui demande l'authentification appartient à son réseau, il sera directement authentifié. S'il appartient au réseau d'un autre opérateur, il sera authentifié une fois que l'opérateur en question confirme son identité. Le deuxième métier est la facturation. Après réception d'un ordre de paiement, l'opérateur procède aux opérations financières permettant à l'utilisateur de son réseau d'effectuer un paiement à travers ses services.

3.3.2 Niveau services

Ce niveau consiste à réaliser l'application à travers les services fournis par les membres de l'écosystème. Le niveau services est composé de deux couches, la couche Generic Business Process et la couche instances. La première décrit l'application à

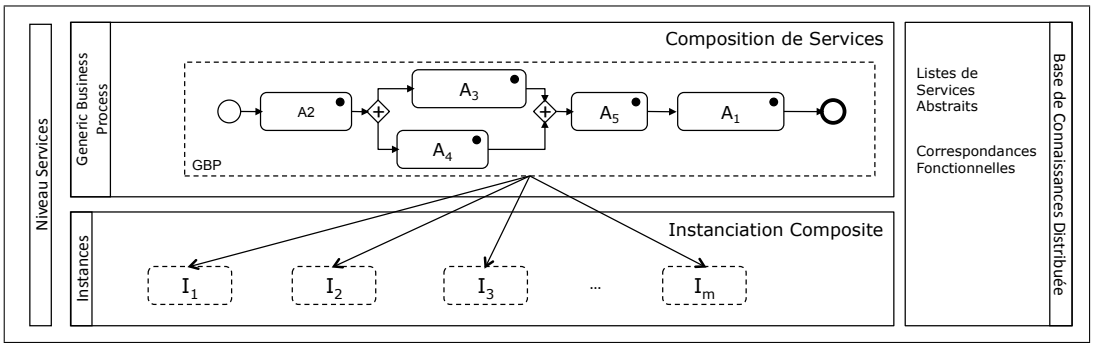


Figure 3.17: Niveau Services

travers une interaction entre services abstraits, tandis que la deuxième détaille les réalisations possibles de l'application à travers des diagrammes de services membres fournis dans l'écosystème. Dans la suite nous décrivons chacune des couches composant le niveau service dans la figure 3.17.

3.3.2.1 Couche generic business process

À ce niveau, une application composite est représentée par un *Generic Business Process*. Le GBP est la version de l'application exprimée à travers une interaction entre services abstraits. Un GBP est défini par deux éléments : (*i*) une représentation graphique et (*ii*) une interface (de programmation). La représentation graphique reprend le résultat du raffinement fonctionnel qui a permis d'identifier les services abstraits impliqués, tandis que l'interface est la traduction de la représentation graphique en un fichier décrivant l'orchestration de services nécessaire à la réalisation de l'application.

La représentation graphique d'un GBP est un BPD attribué où les activités sont des services abstraits (ou opérations abstraites) et les flux séquentiels expriment les dépendances entre services (ou opérations). Généralement le GBP est le dernier BPD résultant du raffinement fonctionnel où toutes les activités ont été raffinées en services abstraits (ou opérations abstraites). Les diagrammes des figures 3.15 et 3.16 illustrent deux exemples de GBP. Quand deux opérations abstraites d'un même service abstrait se suivent séquentiellement dans la représentation graphique d'un GBP, il est possible de les représenter par le même élément graphique contenant l'identifiant du service abstrait en question. Par exemple la figure 3.18 présente un diagramme d'un GBP simplifiant celui illustré par la figure 3.15. En effet les activités

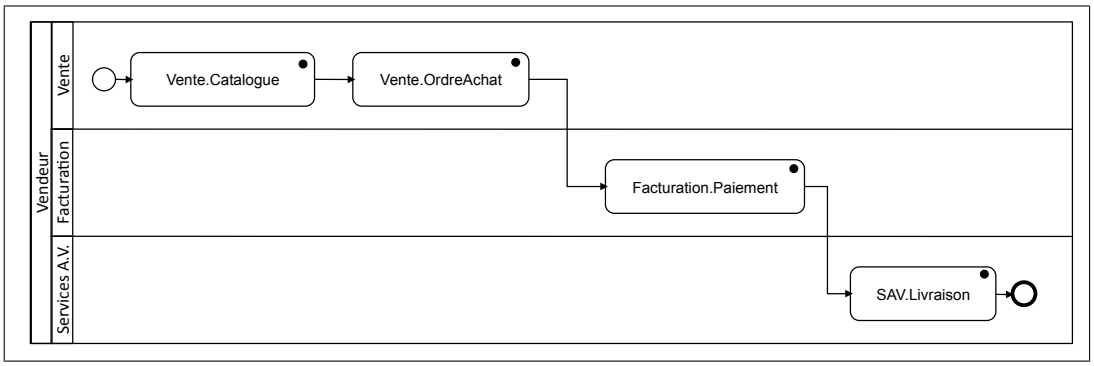


Figure 3.18: Exemple d'un Generic Business Process avec regroupement des activités

abstraites *Recevoir Demande Catalogue* et *Envoyer Catalogue* sont regrouppées dans *Vente.Catalogue*, le service abstrait de gestion de catalogues de vente. De même, les activités abstraites *Envoyer Ordre Paiement* et *Recevoir Paiement* sont regroupées dans *Facturation.Paiement*, service abstrait représentant l'activité de facturation et les activités abstraites *Envoyer Demande Livraison* et *Recevoir Confirmation Livraison* sont regroupées dans *SAV.Livraison*, service abstrait représentant l'activité de livraison

Des attributs peuvent être associés aux services abstraits, aux sous-processus et aux passerelles, et aux flux séquentiels d'un GBP. Ces attributs sont rajoutés au GBP à travers l'artefact *annotation*, tel proposé dans [82]. Une annotation est un crochet alignant une représentation textuelle d'information pertinentes et reliée à l'élément concerné par une association (ligne pointillée, comme illustré dans la figure 3.11). Les attributs d'un GBP sont classés en deux catégories : attributs locaux et attributs globaux. La première catégorie représente des attributs sur un service abstrait ou sur un lien. La deuxième catégorie représente des attributs sur une propriété partagée par tous les éléments comme par exemple le prix désignant le prix d'invocation pour un service et le prix d'exploitation pour un lien. Parmi les attributs d'un lien figure la probabilité que ce lien soit emprunté. En effet, dans le cas d'un branchement décisionnel une probabilité est associée à chacun des branchements possibles. Par défaut la probabilité d'un lien est égale à un, cependant suite au raffinement fonctionnel en services abstraits les probabilités des branchements décisionnels sont définies. Deux stratégies sont envisageables [23, 24]. Une première stratégie consiste à fixer les probabilités lors de la conception (raffinement fonctionnel). Ces probabilités restent les mêmes pour un GBP donné et sont par suite

qualifiées comme probabilités statiques. Une deuxième stratégie consiste à donner des valeurs de départ aux probabilités lors de la conception et les ajuster après chaque exécution de l'application résultante. Une technique de re-calcul des probabilités basée sur un journal d'exécutions est proposée dans [23]. Ces probabilités sont qualifiées d'évolutives. D'autres types d'attributs sont possibles, comme par exemple les propriétés non fonctionnelles.

Grâce aux attributs d'un GBP, l'utilisateur peut exprimer des contraintes d'exécution sur l'application. Toujours dans l'écosystème décrit en section 3.2.5, nous reprenons l'exemple de l'application mobile d'assistance de voyageurs décrite en section 3.1.3. Étant sur la route, le voyageur a besoin de refaire le plein de carburant dans les 100 km qui suivent. Il décide aussi de faire une pause dans l'hôtel le plus proche juste après le plein. Le plein étant prioritaire, le voyageur a besoin de trouver un hôtel à proximité de la station de carburants. Le gestionnaire d'itinéraires génère alors une composition séquentielle entre un service de positionnement et un service de recherche d'utilitaires (station de carburants, garages, etc.) suivi d'un service de recherche d'hôtels. Cette composition est illustrée par le GBP de la figure 3.19. La figure représente le BPD d'une composition séquentielle entre trois services abstraits, *Positionnement*, *Recherche d'utilitaires* et *Recherche d'hôtels*, au niveau domaine. La contrainte garantissant le bon fonctionnement de l'application résultante impose une vitesse minimale de connexion à la liaison (*Positionnement*, *Recherche d'utilitaires*) dans la mesure où le temps maximal de l'échange de données sur celle-ci, doit durer moins que le temps restant avant la panne d'essence duquel on soustrait le temps d'exécution du service implémentant le service abstraits *Recherche d'utilitaires*. En effet il faut bien trouver une station avant d'avoir une panne d'essence. L'annotation de la figure 3.19 illustre cette contrainte décrite comme suit *self.TempsTransfert* $<$ *Utilisateur.GetTempsEssenceRestant - RechercheUtilitaires.TempsExecution*. L'utilisateur peut aussi fixer un budget global pour son application. Effectivement, il peut exiger que la somme des coûts d'invocation des services réalisant son application ne dépasse un montant donné.

L'interface d'un GBP est un fichier "BPEL executable process" décrivant l'orchestration de services abstraits impliqués. Le langage BPEL et les structures qu'il propose sont décrits dans la section 2.3.1. Nous motivons ce choix de BPEL par deux arguments. D'abord le langage BPEL, standardisé par OASIS, est accessible à tous les acteurs car il repose sur XML. Ensuite, une notation BPMN est traduisible en

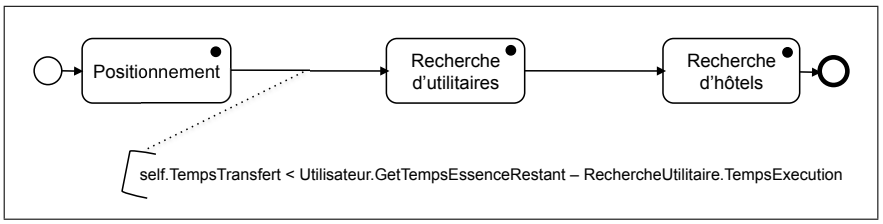


Figure 3.19: Exemple d'un Generic Business Process (avec annotations)

BPEL comme discuté dans [106, 150], de plus des outils de traduction automatique sont disponibles. La figure 3.20 décrit un exemple d'interface de GBP en BPEL et correspondant à la représentation graphique du GBP de la figure 3.18.

3.3.2.2 Couche instances

À ce niveau, les instances des services abstraits et celles des GBP sont modelisées. L'instance d'un service abstrait est un service membre implémentant l'interface en question tout en respectant les contraintes de réalisation associées. Pour chaque service abstrait $A_i \in A$, il existe une ensemble de services membres I_{A_i} implémentant son interface.

$$I_{A_i} = \{S_{i\,k_i}\}_{k_i=1\ldots n_i} \tag{3.5}$$

où $S_{i\,k_i}$ désigne un service implémentant l'interface de A_i, n_i indique le nombre de services membres implémentant A_i.

Nous rappelons qu'un service S_{ik} définit comme implémentation de A_i respecte la relation d'implémentation notée par $IMPD(S_{i\,k}, A_i)$ et décrite en section 3.2.3.4. La relation d'implémentation garantit que chaque service membre $S_{i\,k_i}$ redéfinit tous les éléments de l'interface du service abstrait correspondant A_i, en particulier les propriétés non fonctionnelles P_i. Par conséquence, chaque service $S_{i\,k_i}$ possède un ensemble de propriétés non fonctionnelles $P_{i\,k} = \left\{p^1_{i\,k_i}, p^2_{i\,k_i}, \ldots, p^p_{i\,k_i}, \ldots, p^q_{i\,k_i}\right\}$ où $p^p_{i\,k_i}$ constitue une évaluation de la propriété p^p_i pour le service $S_{i\,k_i}$. À titre d'exemple $p^2_{3\,1}$: $TempsExecution = 1.8$ *secondes* indique la valeur de la deuxième propriété, le *temps d'exécution*, de la première instance du service abstrait A_3.

Ayant les services implémentant les services abstraits d'un GBP donné, un ensemble d'instances possibles est généré. Soit Ω l'ensemble des instances possibles d'un GBP G donné. Ω contient l'ensemble des instances valides de G désigné par I. $\Omega = I \cup \overline{I}$. Le processus d'instanciation déterminant les éléments appartenant à

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3
4  <process name="GBP"
5    ...
6
7    <receive name= RecevoirDemandeCatalogue>
8    <sequence>
9        <invoke name="Vente-Catalogue"/>
10       <invoke name="Vente-OrdreAchat"/>
11       <invoke name="Facturation-Paiement"/>
12       <invoke name="SAV-Livraison"/>
13   </sequence>
14
15   ...
16
17 </process>
```

Figure 3.20: Exemple d'une interface simplifiée d'un GBP en BPEL

l'ensemble I est détaillé dans le Chapitre 4.

Une instance est définie par deux éléments dérivés du GBP correspondant, notamment (i) une représentation graphique et (ii) une interface de programmation. La representation graphique reprend celle du GBP en remplaçant les activités abstraites et les services abstraits par leur implémentations respectives. De même, l'interface de programmation reprend celle du GBP correspondant en remplaçant les activités abstraites et les services abstraits par leur implémentations respectives dans le BPEL exécutable. Pratiquement cela se traduit par l'évaluation des *partners links* respectifs. La figure 3.21 décrit une instance du GBP illustré par les figures 3.20 et 3.18. Elle contient les définitions des *partners links* faisant les liens avec les services fournis. Chaque *partner link* est référencé par une ou plusieurs activités, désignant ainsi le service qui réalisera l'activité en question. Ainsi dans la figure 3.21 le partner link *maFacturation* est référencé par l'opération *Facturation-Paiement* et désigne l'opération équivalente implémentée par le service `http://www.myServices/maFacturation.wsdl`.

À ce niveau les éléments appartenant à Ω sont définis en utilisant les services membres correspondant aux services abstraits du GBP. Cependant, les instances valides ne peuvent pas être determinées parce que l'évaluation des contraintes n'est pas possible à ce stade. En effet la détermination des éléments de I nécessite des données relatives aux pairs et au réseau provenant du niveau inférieur : le niveau infrastructure.

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="GBP"
...
<partnerLinks>
   <partnerLink name="maFacturation"
           xmlns:tns="http://www.myServices/maFacturation.wsdl"/>
   ...
</partnerLinks>

...
 <sequence>
      <invoke name="Facturation-Paiement" partnerLink="maFacturation"/>
   ...
   </sequence>

</process>
```

Figure 3.21: Exemple d'une interface simplifiée d'une instance d'un GBP en BPEL

3.3.3 Niveau infrastructure

Ce niveau représente l'architecture sous-jacente reliant les membres de l'écosystème d'entreprises. Il permet de représenter les caractéristiques du réseau afin d'évaluer certaines propriétés qui en dépendent. Ce niveau, illustré par la figure 3.22, est composé de deux couches, la première étant la couche réseau virtuel et la deuxième la couche réseau. Dans la suite nous détaillons chacune des couches.

3.3.3.1 Couche réseau virtuel

Cette couche constitue la réorganisation des pairs du réseau sous-jacent en un réseau virtuel basé sur les communautés. Nous adoptons une architecture pair-à-pair non structurée, hybride. Elle est hiérarchisée en deux niveaux et elle comprend des communautés de pairs tel que chaque communauté est gérée par un super-pair.

La figure 3.23 illustre un exemple d'architecture d'un réseau virtuel de communautés, adapté à la composition d'applications dans un environnement orienté services. Il consite en un ensemble de pairs dont l'objectif principal est de fournir des implémentations concrètes des services abstraits disponibles en vue de collaborer avec les autres pairs. Les pairs sont organisés en communautés gérées par des super-pairs, qui à leur tour sont organisés en une topologie de communication. Par exemple, la *communauté de pairs - 1* est gérée par le super-pair $SN1$ et contient

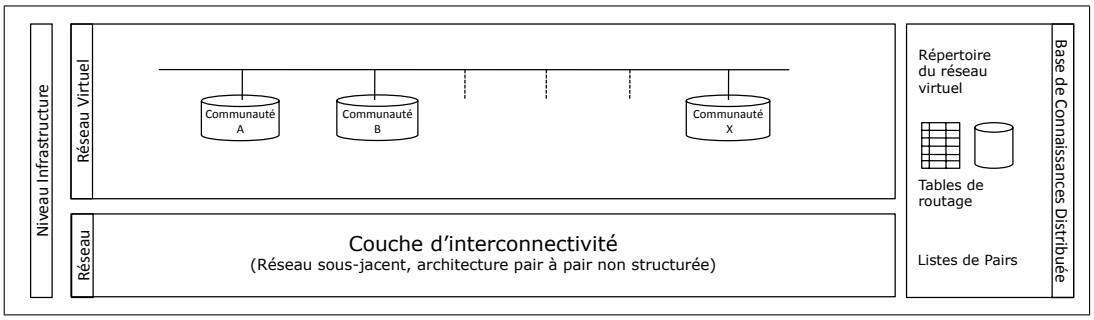


Figure 3.22: Niveau Infrastructure

quatre pairs $N_1, \ldots, N_4$. Nous notons qu'un pair s'il fournit des services peut en fournir un ou plusieurs ; d'autre part un service abstrait peut être implementé par plusieurs pairs.

Dans la suite nous détaillons l'organisation du réseau virtuel proposé et plus particulièrement les caractéristiques des communautés et le rôle de super-pair. Ensuite nous décrivons le protocole de gestion des événements liés au départ et à l'arrivée des pairs.

Communautés de pairs Dans un écosystème donné, une communauté de pairs est un ensemble de pairs partageant une ou plusieurs caractéristiques communes et gérés par le même super-pair. Formellement une communauté de pairs est designée par

$$PC_{Loc} = \Big(SN_k,\ \{N_r\}_{r=1..n_k}\Big)$$

où SN_k est le super-pair gérant un ensemble de n_k pairs respectant le consensus local $Loc(PC)$.

Le consensus local d'une communauté de pairs est basé sur (i) des caractéristiques du réseau ou (ii) sur des propriétes liées aux services comme décrit précédemment dans la section 3.2.3. En effet, selon [107] deux stratégies sont récurrentes dans littérature pour regrouper les pairs en communautés. La première stratégie considère des propriétés rattachées au réseau, tandis que la deuxième stratégie prend en compte les intérêts des pairs, or dans l'écosystème étudié les pairs sont intéressés par le partage et par la consommation des services. Dans la suite nous détaillons ces deux approches de construction de communautés.

(i) La construction des communautés basées sur les caractéristiques du réseau

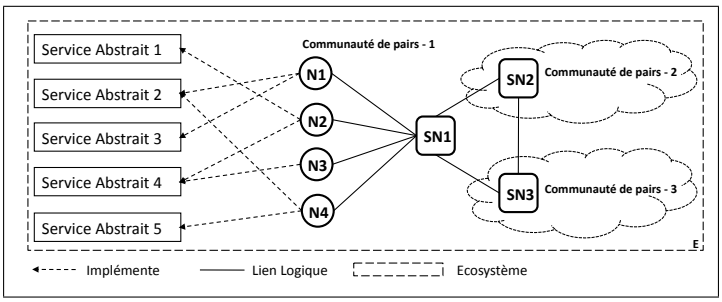


Figure 3.23: Exemple d'organisation du réseau virtuel

génère un réseau virtuel organisé de telle sorte que les pairs d'une même communaué peuvent fournir des services différents. Dans un tel scénario, il est très probable que les pairs d'une même communauté coopèrent en vue de combler un besoin utilisateur. Effectivement, quand un pair est choisi, la probabilité de choisir un pair de la même communauté augmente, dans la mesure où il est plus judicieux de rechercher des services parmi ses voisins proches (dans la même communauté) avant de s'adresser à des voisins plus éloignés (appartenant à d'autres communautés).

Plusieurs approches ont proposé de regrouper les pairs selon des caractéristiques du réseau. Ratnasamy *et al.* présentent dans [122], une démarche qui permet de regrouper les pairs dans des groupes désignés par "bins" de telle sorte que les pairs appartenant au même "bin" sont relativement proches en terme de *temps de latence*. De la même façon, dans [162], Zheng *et al.* utilisent une approche de regroupement de pairs basée sur le délai de communication. Zhang *et al.* proposent dans [160] un système de construction de réseaux virtuels exploitant la notion de "voisinage" dans le réseau physique sous-jacent. Chaque pair de la couche virtuelle exécute le protocole de communication proposé afin de communiquer avec les autres pairs. Chacun des pairs contient une liste de voisins avec lesquels il peut communiquer. Deux pairs sont dit voisins s'ils sont reliés au niveau de la couche virtuelle. Le protocole MetaStream adapté aux applications de streaming et étudié dans [159], permet une découverte de pairs basée sur la distance séparant deux pairs fournisseurs. À cette fin les auteurs dans [159] proposent une technique de réorganisation des pairs en une hiérarchie de groupes selon des critères relatifs à la topologie du réseau.

(*ii*) Les propriétés relatives aux services sont classées en deux catégories : *propriétés fonctionnelles* et *propriétés non fonctionnelles* [77, 123]. Les propriétés fonctionnelles décrivent la fonctionnalité accomplie par le service et représentent la liste des paramètres d'entrées et de sorties et les conditions associées, si nécessaire.

Construire des communautés basées sur des propriétés fonctionnelles reliées aux services permet d'obtenir des communautés au sein desquelles les pairs sont en compétition pour fournir des services ayant des attributs non fonctionnels différents, bien que couvrant la même fonctionnalité. De cette façon les pairs entreront en compétition pour être sélectionné par l'utilisateur. Le choix de ce dernier est basé sur des critères relatifs aux propriétés non fonctionnelles.

Les propriétés non fonctionnelles son classées en deux catégories : propriétés quantitatives et propriétés qualitatives [23]. Les propriétés quantitatives sont majoritairement en rapport avec la qualité de service (QoS) et par suite ils sont en relation étroite avec les propriétés du réseau, comme par exemple le temps de réponse. Les autres critères non fonctionnels ne représentant pas des propriétés de qualité de service, comme par exemple le niveau de sécurité *security-level* et la confiance *trust* peuvent aussi constituer des propriétés sur lesquelles les communautés sont construites [54, 149]. Une étude exhaustive de ces critères ou une classification n'est pas visée dans le cadre de cette thèse, dans nos exemples nous illustrons souvent *le temps de réponse* et *le prix d'exécution*. Nous résumons les modèles de description des critères non fonctionnels rattachés aux services dans la section 4.1.3.2.

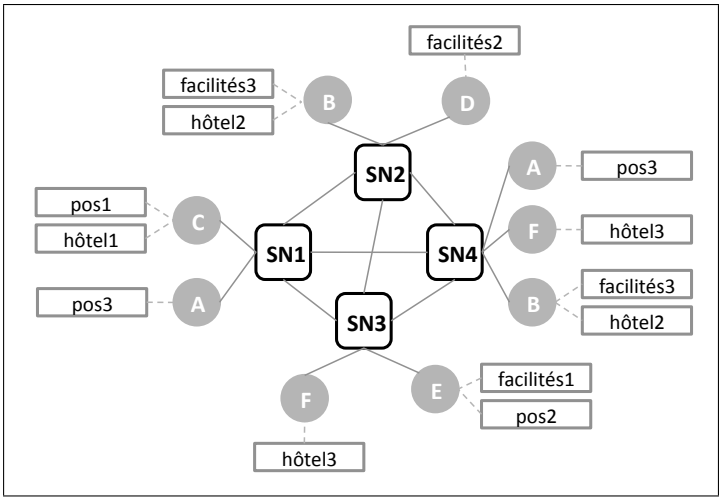


Figure 3.24: Exemple d'un réseau virtuel basé sur la distance physique séparant les nœuds

Afin d'illustrer les communautés dans le réseau virtuel nous reprenons l'exemple de l'application mobile de gestion d'itinéraires. Le GBP de cette application est décrit dans la figure 3.19. Pour la suite de l'exemple nous supposons que les services pos1, pos2 et pos3 implémentent le service abstrait

Positionnement, les services facilités1, facilités2 et facilités3 implémentent le service abstrait *Recherche d'utilitaires* et les services hôtel1, hôtel2 et hôtel3 implémentent le service abstrait *Recherche d'hôtels*. La figure 3.24 illustre un réseau virtuel basé sur la distance physique entre les nœuds. Il est composé de quatre super-pairs et leurs communautés correspondantes. Chaque super-pair gère une communauté de pairs séparés au plus par deux nœuds dans le réseau sous-jacent. Un pair peut appartenir à plusieurs communautés, par exemple, le pair B est membre de la communauté gérée par SN2 et celle gérée par SN4.

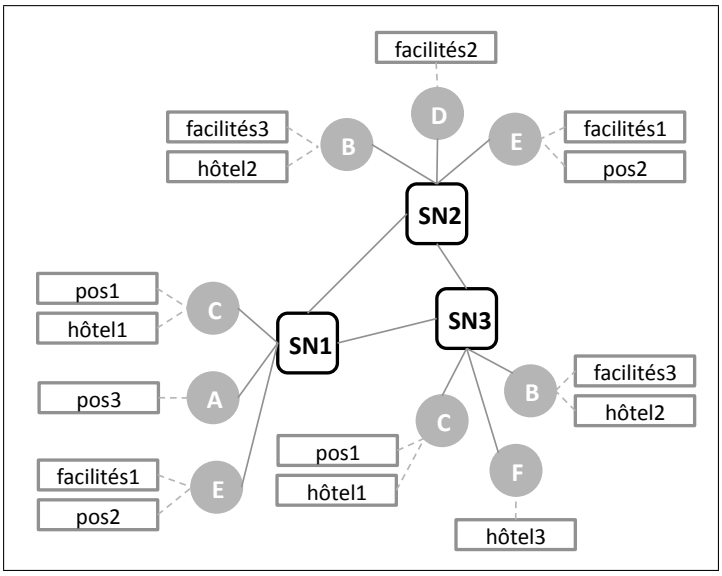


Figure 3.25: Exemple d'un réseau virtuel basé sur les fonctionnalités fournies par les nœuds

Toujours dans le cadre de l'exemple de l'application mobile de gestion d'itinéraires, la figure 3.25 illustre une configuration du réseau virtuel où les communautées sont basées sur les fonctionnalités fournies. Chaque communauté de pairs implémente un service abstrait dans la mesure où les pairs membres fournissent des implémentations du même service abstrait, des implémentations qui diffèrent par leurs propriétés non fonctionnelles. Trois communautés de pairs sont construites pour representer les implémentations des trois services abstraits utilisés dans l'exemple de la figure 3.19. Par exemple le super-pair SN1 gère la communauté du service abstrait *Positionnement* ; les pairs membres A, C et E fournissent des implémentations à *Positionnement*. Nous soulignons qu'un pair membre peut fournir des implémentations de plusieurs services abstraits différents, il appartiendra alors à plusieurs communautés. Par exemple le pair C fournit des implémentations à

Positionnement et à *Recherche d'hôtels*, il appartient alors à la communauté gérée par SN1 et celle gérée par SN3. Formellement les communautés gérées par SN1 et SN3 sont décrites respectivement comme suit :

$$PC_{(Positionnement)} = (\text{SN1},\ \{\text{A},\ \text{C},\ \text{E}\})$$

et

$$PC_{(Recherche\ d'hôtels)} = (\text{SN3},\ \{\text{B},\ \text{C},\ \text{F}\})$$

Super-pairs Un super-pair est un nœud dans un réseau pair-à-pair qui joue à la fois le rôle de serveur pour un ensemble de nœuds clients et le rôle d'un égal dans le réseau de super-pairs [155]. Les supers-pairs sont sélectionnés selon leurs capacités de calcul. Un raffinement de la sélection peut avoir lieu selon le niveau de confiance attribué à un super-pair. Plusieurs stratégies envisageables pour le choix de super-pairs sont décrites dans [155]. Dans le cadre de l'écosystème coopératif étudié dans cette thèse, nous adoptons le modèle de super-pairs dédiés et disponibles en ligne. Cependant nous admettons que les super-pairs peuvent fournir des services comme implémentation des services abstraits. Du point de vue implémentation, un super-pair est un serveur d'application d'une entreprise membre. Les liens entre super-pairs sont choisis comme le chemin le plus court du réseau physique sous-jacent. Chaque lien entre deux super-pairs est un bus de communication bidirectionnelle. Chaque super-pair possède une copie de la base de connaissances de l'écosystème décrite en section 3.2.4.3. Une copie qu'il entretient et transmet à ses pairs clients après chaque mise à jour, de telle sorte que chaque pair possède à tout instant une copie à jour de la base de connaissances.

Les super-pairs maintiennent et gèrent le répertoire distribué du réseau virtuel. Le répertoire distribué est une structure de données relative à l'organisation du réseau virtuel contenant deux tables : la *table d'état local* et la *table d'état global*. La table d'état local contient (i) l'ensemble des pairs clients du super-pair courant ; (ii) un état $St\,(N_r)$ pour chaque pair. $St\,(N_r)$= ON si le pair N_r est en ligne. $St\,(N_r)$ = OFF si le pair N_r est hors ligne ; (iii) pour chaque pair N_r, une liste de services fournis S_{ik} et leurs services abstraits correspondant A_i.

La table d'état global représente pour chaque super-pair SN_k appartenant au réseau virtuel, l'ensemble des services abstraits $\{A_i\}_{i=1..n_k}$ implémentés dans sa communauté.

La table d'état global décrite par la table 3.1 est basée sur l'exemple du réseau

Super-pair	Service Abstrait Implémenté
SN1	{*Positionnement*, *Recherche d'hôtels*}
SN2	{*Recherche d'utilitaires*, *Recherche d'hôtels*}
SN3	{*Positionnement*, *Recherche d'utilitaires*, *Recherche d'hôtels*}
SN4	{*Positionnement*, *Recherche d'utilitaires*, *Recherche d'hôtels*}

Table 3.1: La table d'état global sur chaque *super-pair* dans le réseau virtuel basé sur la distance physique séparant les nœuds

virtuel de la figure 3.24. Cette table est sujette à des mise à jours régulières. Par exemple, suite au départ du pair A, la communauté gérée par SN4 ne fournit plus d'implémentation pour le service abstrait *Positionnement*, alors *Positionnement* sera éliminé de la liste des services abstraits implémentés par la communauté de SN4. Cependant, aucune modification est faite sur la liste des services abstraits implémentés par la communauté de SN1 car une implémentation du service abstrait *Positionnement* est fournie par le pair C.

La table 3.2a illustre la table d'état local sur le super-pair SN2 et la table 3.2b résume l'état local de la communauté de SN4

Gestion des événements Deux événements majeurs sont considérés, d'abord l'arrivée d'un pair ensuite le départ d'un pair.

Quand un pair rejoint le réseau, trois acteurs ou groupes d'acteurs sont impliqués. Premièrement le pair lui même, (*i*) lance un processus de sondage (*probing*) afin de découvrir le super-pair le plus proche en terme de distance physique, ensuite (*ii*) il interroge le super-pair choisi demandant une copie de la base de connaissances de l'écosystème et une copie du répertoire du réseau virtuel. Ayant ces informations, (*iii*) le pair, s'il est intéressé par la fourniture de services, décide des services abstraits à implémenter ou crée les correspondances entre ses services existants et les services abstraits adéquats. De cette façon le pair respecte indirectement le consensus global de l'écosystème à travers les services qu'il fournit. Finalement, (*iv*) le pair envoie des demandes d'adhésions aux super-pairs des communautés qu'il a l'intention de rejoindre. Il est évident que le respect du consensus local de la communauté sollicitée est un pré-requis d'adhésion. Cela est traduit par l'équation 3.1. Deuxièmement, chacun des supers-pairs concernés (*i*) reçoit la demande d'adhésion du pair, (*ii*) met à jour sa table d'information globale et sa table d'information locale, si et seulement si le pair respecte les conditions d'adhésion (à la fois le consensus global et le consensus local) et (*iii*) envoie aux autres super-pairs des notifications de présence

Pair		(Service, *Service Abstrait*)
P_{ID}	$St\,(P_i)$	
B	ON	{(facilités3, *Recherche d'utilitaires*), (hôtel2, *Recherche d'hôtels*)}
D	ON	{(facilités2, *Recherche d'utilitaires*)}

(a) Table d'état local sur SN2

Pair		(Service, *Service Abstrait*)
P_{ID}	$St\,(P_i)$	
B	ON	{(facilités3, *Recherche d'utilitaires*), (hôtel2, *Recherche d'hôtels*)}
F	ON	{(hôtel3, *Recherche d'hôtels*)}
A	ON	{(pos3, *Positionnement*)}

(b) Table d'état local sur SN4

Table 3.2: Exemple des tables d'état local sur les super-pairs (réseau virtuel basé sur la distance)

d'un nouveau pair et de mise à jour des copies du répértoire. Troisièmement, chacun des "autres" super-pairs (i) reçoit la notification de mise à jour et (ii) met à jour ses informations soit en interrogeant le super-pair qui a envoyé la demande, soit en interrogeant le nouveau pair.

Quand un pair se déconnecte deux stratégies sont envisageables. La première stratégie considère le modèle de "départ propre" partant du principe que l'intérêt des pairs membres de l'écosystème est la coopération en vue d'améliorer le réseau et sa valeur ajoutée : les applications. Dans ce cas, trois acteurs ou groupes d'acteurs sont impliqués. Premièrement le pair lui même envoie des messages de déconnection à ses super-pairs. Deuxièmement, chacun des super-pairs gérant les communautés du pair partant (i) marque le pair comme hors ligne. Si le pair en question est le dernier dans la communauté à fournir une implémentation à un service abstrait donné, (ii) le service abstrait correspondant est supprimé de la table d'information globale. Finalement, (iii) le super-pair envoie des demandes de notifications aux autres super-pairs du réseau virtuel contenant la nouvelle version de la table d'information globale. Troisièmement, les autres supers-pairs (i) reçoivent la mise à jour et (ii) procèdent à la copie. La deuxième stratégie considère que le départ d'un pair est dû à une panne. Dans ce cas, deux acteurs ou groupes d'acteurs sont impliqués. Premièrement, le super-pair en charge du pair en panne et qui a détecté son départ, accomplit les mêmes étapes que la stratégie précédente. Deuxièmement, les autres super-pairs, (i)

reçoivent la mise à jour et (*ii*) procèdent à la copie.

3.3.3.2 Couche réseau physique

Ce niveau permet de saisir les caracteristiques du réseau sous-jacent. À ce stade, les propriétés des services membres et celles des liens entre les pairs fournisseurs d'une instance peuvent être évaluées. Les propriétés des pairs sont projetées sur les graphes d'instances qui correspondent. Les éléments de l'ensemble des instances valides I sont alors déterminés.

3.4 Discussion

L'écosystème d'entreprises décrit dans le cadre de DyCoSe propose, d'abord, une solution aux conflits sémantiques entre membres participants. En effet, il fournit une ontologie afin d'unifier la terminologie utilisée par les membres, le but étant de partager les concepts communs modélisant les métiers des différents membres. Il fournit aussi une ontologie de paramètres non fonctionnels qui permet de déclarer explicitement les propriétés non fonctionnelles significatives aux membres. L'écosystème propose aussi, une solution aux défis de la découverte de services. Premièrement, le contexte de l'écosystème réduit l'étendue de la découverte en la limitant aux domaines des membres participants. Deuxièmement, l'écosystème propose des interfaces de description communes décrivant les fonctionnalités métiers usuelles. Quand un fournisseur de services adhère à l'écosystème, il a le choix entre implémenter directement une ou plusieurs de ces interfaces ou aligner les descriptions des services qu'il propose avec celles des interfaces correspondantes. Le processus de découverte de services est ainsi simplifié pour le consommateur de services.

L'architecture de composition proposée s'accorde avec une approche de composition en deux étapes. En effet, la première étape consiste à définir l'application en terme d'interactions entre composants de haut niveau ; tandis que la deuxième étape consiste à réaliser l'application à travers une ou plusieurs orchestrations de services disponibles implémentant les composants de haut niveau. Cette dernière étape permet de tenir compte des changements de l'environnement au cours de l'exécution. Dans cette perspective, le choix des services est décalé jusqu'à l'exécution, permettant ainsi un processus de sélection dynamique basée sur un calcul d'optimisation de coûts. Cette architecture supporte une approche de composition d'applications à

base de services qui combine à la fois une démarche descendante et une autre ascendante. La démarche descendante du niveau *fonctionnel* jusqu'au niveau *services*, en passant par les couches (sous-niveaux) intermédiaires, permet de modéliser l'application, de gérer l'aspect fonctionnel et de propager les contraintes de réalisation et les préférences de l'utilisateur en vue de choisir une composition de départ. La partie ascendante de la démarche projette les caractéristiques du réseau sous-jacent au niveau *services*, plus particulièrement à l'instance de processus, permettant d'évaluer les propriétés et les contraintes étudiées.

Chapitre 4

Applications composites : déploiement par instanciation

Sommaire

4.1 Définitions et notations 122

4.1.1 Graphe d'un GBP 122

4.1.2 Chemin d'exécution d'un GBP 124

4.1.3 Propriétés non fonctionnelles d'un service 127

4.1.4 Propriétés non fonctionnelles d'un GBP 130

4.2 Problème d'instanciation de GBP 132

4.2.1 Définition d'une instance 132

4.2.2 Définition du processus d'instanciation d'un GBP 133

4.2.3 Exemple d'instanciation 134

4.2.4 Énoncé du problème d'instanciation d'un GBP 137

4.2.5 Instance d'un chemin d'exécution 138

4.2.6 Coût d'une instance d'un chemin d'exécution 139

4.2.7 Choix de l'instance de départ d'un chemin d'exécution . . . 141

4.2.8 Reconstruction d'une instance d'un GBP 143

4.3 Solutions au problème d'instanciation 144

4.3.1 Solution globale 144

4.3.2 Solution locale 150

4.4 Discussion 156

Dans ce chapitre nous étudions le processus d'instanciation d'une application composite. Il consiste à sélectionner un ensemble de services membres répondant aux préférences de l'utilisateur pour réaliser une application exprimée en terme de services abstraits. Nous proposons tout d'abord un ensemble de définitions et une notation formelle pour décrire l'instance d'un GBP. Ensuite, nous formulons le processus d'instanciation comme un problème décisionnel d'optimisation de coûts. Puis nous décrivons deux types de solutions au problème d'instanciation : une solution globale et une solution locale.

Dans le cadre de l'écosystème de DyCoSe, présenté dans le chapitre précédent, nous avons décrit une application composite par un GBP qui est constitué d'une représentation graphique et d'une interface de programmation. La représentation graphique d'un GBP est un diagramme BPMN annoté. Il contient un ensemble de services abstraits, des passerelles représentant les contrôles de flux et des liens entre eux. Un extrait des éléments graphiques d'un diagramme BPMN est présenté dans la figure 3.11. Les annotations associent des attributs aux nœuds et aux liens du GBP, caractérisant ainsi l'application. L'interface de programmation d'un GBP est un fichier BPEL traduisant la représentation graphique. Il propose une description plus complète de la logique applicative du diagramme BPMN par un fichier semi-structuré. Il contient des liens vers les descriptions des services abstraits composant le GBP et les listes d'attributs associés au services abstraits et aux liens du diagramme BPMN. La figure 4.1 illustre les différentes étapes pour déduire un graphe attribué contenant les informations pertinentes indispensables au processus d'instanciation.

4.1 Définitions et notations

Dans cette section nous donnons un ensemble de définitions relatives au GBP et aux propriétés non fonctionnelles des services. Nous introduisons une notation formelle basée sur la théorie des ensembles et sur les graphes.

4.1.1 Graphe d'un GBP

Dans la suite de la thèse, un graphe G est décrit par le triplet : $G = (V,\ E,\ \psi())$ tel que : (1) $V = \{v_1, v_2, \ldots, v_n\}$ où n est le nombre de nœuds de G, désigne l'ensemble des nœuds de G. (2) $E \subset V \times V$ désigne l'ensemble des liens entre les nœuds de G. $E = \{e_1, e_2, \ldots, e_l\}$ où l désigne le nombre de liens dans G. (3) $\psi() : E \rightarrow V \times V$ est une fonction d'incidence tel que si $\psi(e_j) = (v_i, v_{i'})$ alors e_j lie v_i à $v_{i'}$.

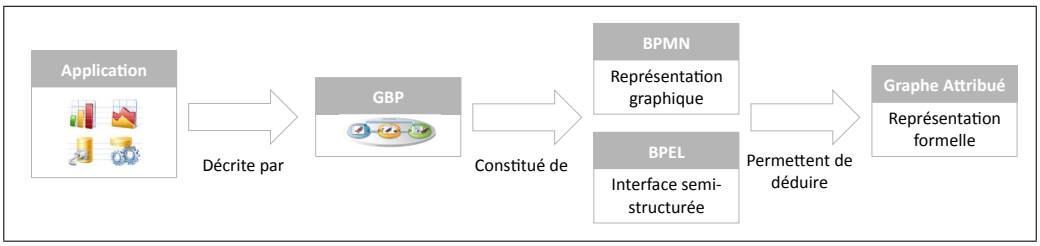


Figure 4.1: Déduction du graphe d'un GBP décrivant une application

Définition 4.1. ***GBP***
Un GBP est formellement représenté par un graphe attribué

$$GBP = \left(\widetilde{V},\ \widetilde{E},\ \widetilde{\psi()},\ \widetilde{\omega()},\ \widetilde{\epsilon()}\right) \tag{4.1}$$

tel que :

- *L'ensemble des nœuds $\widetilde{V}$ du GBP contient les nœuds utilisés dans le GBP. $\widetilde{V} = \widetilde{V1} \cup \widetilde{V2}$ où $\widetilde{V1}$ désigne l'ensemble des services abstraits du GBP et $\widetilde{V2}$ désigne l'ensemble des passerelles exprimant les contrôles de flux du GBP.*
- *L'ensemble des liens $\widetilde{E} \subset \widetilde{V} \times \widetilde{V}$ contient les liens entre les nœuds du GBP.*
- *La fonction d'incidence $\widetilde{\psi()}$ associe à chaque lien $\widetilde{e_j}$ de $\widetilde{E}$ un couple de nœuds $(v_i, v_{i'}) \in \widetilde{V} \times \widetilde{V}$. $\widetilde{\psi(\widetilde{e_j})} = (v_i, v_{i'}) \Leftrightarrow v_i$ et $v_{i'}$ sont les extrémités de $\widetilde{e_j}$.*
- *La fonction d'attributs $\widetilde{\omega()} : \widetilde{V} \rightarrow GlobA$ associe un ensemble de propriétés non fonctionnelles, définies par le consensus global de l'écosystème GlobA, à chaque nœud du GBP. Si $\widetilde{v_i} \in \widetilde{V2}$, alors $\widetilde{\omega(\widetilde{v_i})} = \emptyset$.*
- *La fonction d'attributs $\widetilde{\epsilon()} : \widetilde{V} \rightarrow GlobA$ associe aux liens $\widetilde{e_j}$ du GBP un ensemble de propriétés non fonctionnelles définies par le consensus global de l'écosystème GlobA.*

La figure 4.2 décrit les différentes combinaisons de passerelles usuelles d'un GBP. Un branchement parallèle suivi d'une jointure parallèle désignent une exécution concurrente des chemins intermédiaires et indiquent que finalement tous les chemins sont attendus. Un branchement parallèle suivi d'une jointure décisionnelle indiquent que tous les chemins intermédiaires sont exécutés en même temps et que finalement un chemin sera choisi. Un branchement décisionnel suivi d'une jointure décisionnelle indiquent qu'un chemin intermédiaire sera exécuté. La spécification de BPMN admet

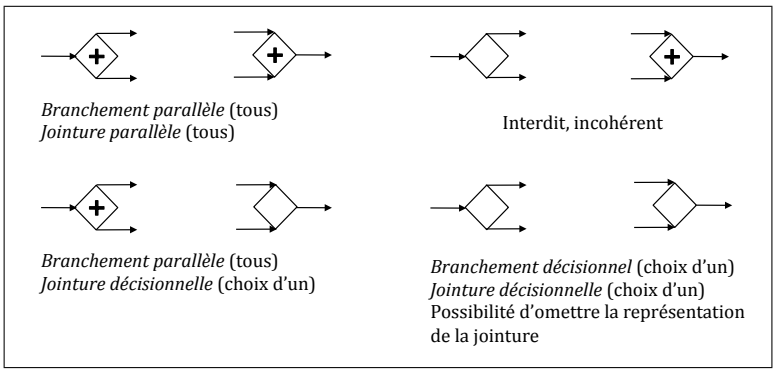


Figure 4.2: Différentes combinaisons des passerelles (contrôles de flux)

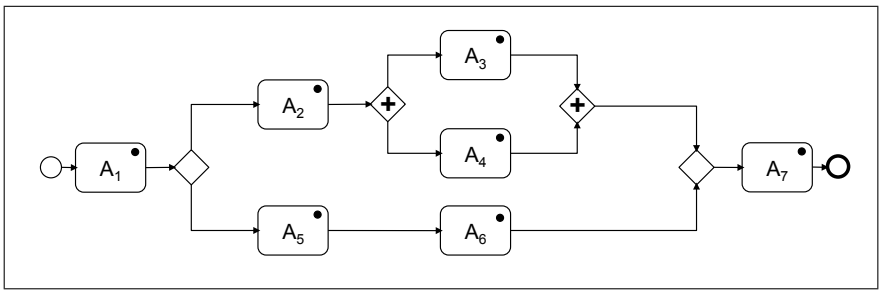
que la simplification d'une jointure décisionnelle suivant un branchement décisionnel n'entraîne pas de perte sémantique dans un diagramme (cf. pages 115-116 dans [55]). La combinaison entre un branchement décisionnel et une jointure parallèle n'est pas possible.

4.1.2 Chemin d'exécution d'un GBP

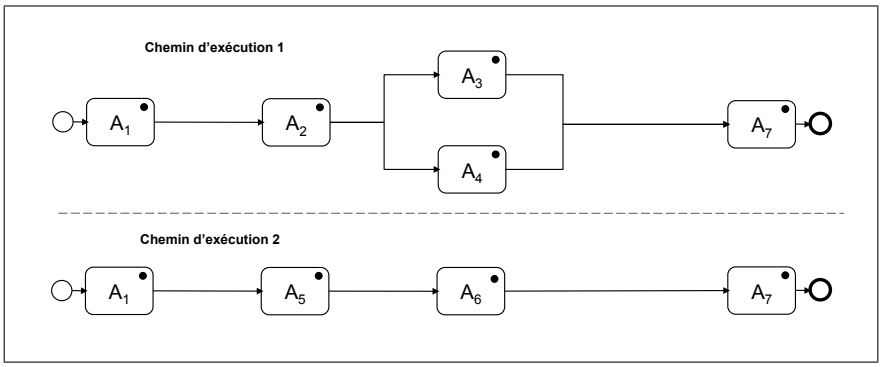
Un GBP contenant des nœuds de type branchement décisionnel contient plusieurs flux d'exécution possibles. Un flux d'exécution désigne un chemin d'exécution entre le nœud de départ et le nœud de terminaison. Chaque chemin d'exécution modélise une réalisation possible de l'application. Nous définissons un chemin d'exécution d'un GBP en adaptant une définition proposée par Zeng *et al.* pour un chemin d'exécution dans un diagramme d'états [158].

Définition 4.2. ***Chemin d'exécution d'un GBP.***
Un chemin d'exécution est une séquence de services abstraits représentée par le n-uplet $(A_1, A_2, \ldots, A_n)$ *où* A_1 *est le premier service,* A_n *est le dernier service et* A_i *où* $1 < i < n$ *vérifie les règles suivantes :*

- A_i *est un successeur direct d'un des services* $(A_1, \ldots, A_{i-1})$.
- A_i *n'est pas un successeur direct d'un des services* $(A_{i+1}, \ldots, A_n)$.
- *Un chemin d'exécution peut contenir des passerelles parallèles.*
- $\forall$ A_i *appartenant à un chemin d'exécution 1 et* $\forall$ A_j *appartenant à un chemin d'exécution 2, il n'existe pas un lien* (A_i, A_j) *entre* A_i *et* A_j. *Nous en déduisons*



(a) Illustration d'un GBP



(b) Illustration de deux chemins d'exécution

Figure 4.3: Exemple d'un GBP et des chemins d'exécution associés

que deux chemins d'exécution sont indépendants.

La figure 4.3b illustre deux chemins d'exécution associés au GBP de la figure 4.3a. En effet, après simplification du branchement décisionnel et de la jointure décisionnelle, deux chemins possibles lient le nœud de départ au nœud d'arrivée.

Nous proposons de simplifier la représentation graphique d'un chemin d'exécution en éliminant les passerelles de type branchement parallèle et jointure parallèle. En effet, dans un chemin d'exécution donné, il n'est pas nécessaire de préciser si la passerelle est décisionnelle ou parallèle. Par exemple, dans la figure 4.3b, les flux séquentiels sortant du service abstrait A_2 seront interprétés comme des branchements en parallèle et les flux séquentiels arrivant au service abstrait A_7 seront interprétés comme une jointure parallèle.

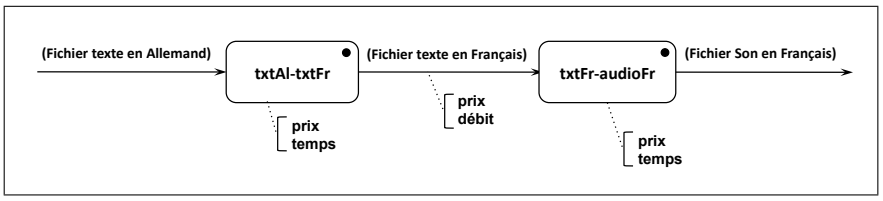


Figure 4.4: Attributs d'un GBP consititué d'un chemin d'exécution

Définition 4.3. ***Graphe d'un chemin d'exécution.***
En se basant sur la définition 4.1 d'un GBP, un chemin d'exécution g est formellement représenté par le graphe attribué

$$g = \left(\overline{V},\ \overline{E},\ \overline{\psi()},\ \overline{\omega()},\ \overline{\epsilon()}\right) \tag{4.2}$$

où :

- *L'ensemble des nœuds $\overline{V} \subset \widetilde{V1}$ contient les services abstraits utilisés dans g. $\overline{V} = \{A_1, A_2, \ldots A_n\}$ où n est le nombre de services abstraits de g.*
- *L'ensemble des liens $\overline{E} \subset \overline{V} \times \overline{V}$ contient les liens entre les services abstraits de g. $\overline{E} = \{\overline{e_1}, \overline{e_2}, \ldots, \overline{e_l}\}$ où l désigne le nombre de liens dans le chemin d'exécution g.*
- *La fonction d'incidence $\overline{\psi()}$ associe à chaque lien $\overline{e_j}$ de $\overline{E}$ un couple $(A_i, A_{i'})$. Si $\overline{e_j}$ est un lien et A_i et $A_{i'}$ sont des nœuds tel que $\overline{\psi(\overline{e_j})} = (A_i, A_{i'})$ alors $\overline{e_j}$ lie A_i à $A_{i'}$. Fonctionnellement cette liaison implique que la sortie du service abstrait A_i est utilisée en entrée du service abstrait $A_{i'}$.*
- *La fonction d'attributs $\overline{\omega()} : \overline{V} \rightarrow GlobA$ associe un ensemble de propriétés non fonctionnelles, parmi celles définies par le consensus global de l'écosystème GlobA, à chaque service abstrait de g. $\overline{\omega(A_i)} = P(A_i) = P_i$ tel que $P_i = \{p_i^p\}_{p=1..q_i}$, $P_i \subset GlobA$.*
- *La fonction d'attributs $\overline{\epsilon()} : \overline{E} \rightarrow GlobA$ associe un ensemble de propriétés non fonctionnelles, parmi celles définies par le consensus global de l'écosystème GlobA, à chaque lien entre les services abstraits de g.*

Considérons par exemple le GBP de la figure 4.4. Il est constitué d'un chemin d'exécution représenté formellement par :(i) l'ensemble de services abstraits $\overline{V} = \{A_1, A_2\}$ tel que A_1 et A_2 modélisent respectivement les fonctionnalités txtAl-txtFr

et txtFr-audioFr. (*ii*) L'ensemble de liens $\overline{E} = \{\overline{e_1}\}$ modélise le lien entre les deux nœuds du chemin d'exécution. (*iii*) La fonction d'incidence $\overline{\psi()}$ tel que $\overline{\psi(\overline{e_1})} = (\text{txtAl-txtFr}, \text{txtFr-audioFr})$, indique que la sortie de txtAl-txtFr est liée à l'entrée de txtFr-audioFr. Ce lien est désigné par $\overline{e_1}$. (*iv*) La fonction d'attributs $\overline{\omega()}$ telle que $\overline{\omega(\text{txtAl-txtFr})} = \overline{\omega(\text{txtFr-audioFr})} = \{p^1, p^2\}$ où p^1 = prix et p^2 = temps, associe aux services abstraits les propriétés non fonctionnelles prix et temps. (*v*) La fonction d'attributs $\overline{\epsilon()}$ telle que $\overline{\epsilon(\overline{e_1})} = \{p^1, p^3\}$ où p^1 = prix et p^3 = débit, associe au lien $\overline{e_1}$ les propriétés non fonctionnelles prix et débit qui correspondent aux propriétés du transfert de données entre txtAl-txtFr et txtFr-audioFr.

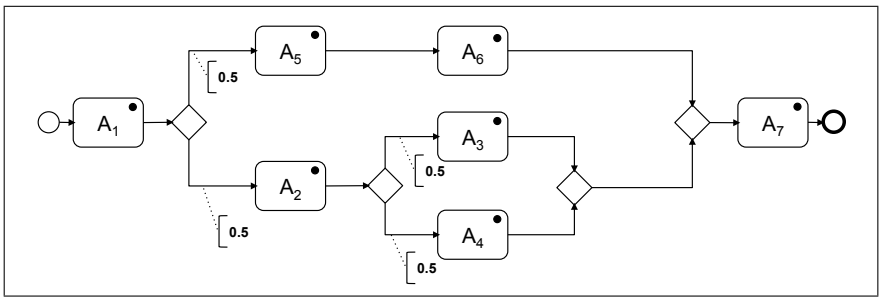
Chaque chemin d'exécution, g, a une probabilité d'être suivi $p(g)$. Cette probabilité est calculée en combinant les probabilités des liens de type branchement décisionnel et/ou jointure décisionnelle qui ont donné naissance au chemin d'exécution. En effet, tel expliqué en section 3.3.2.1, chaque lien représenté par un flux séquentiel suivant un branchement décisionnel ou précédant une jointure décisionnelle possède parmi ses attributs, la probabilité qu'il soit emprunté. Nous rappelons que ces probabilités sont spécifiées par l'utilisateur lors du raffinement du GBP. La mise à jour de ces probabilités n'est pas abordée dans le cadre de DyCoSe. La figure 4.5 illustre un GBP annoté, montrant les probabilités associées aux passerelles décisionnelles.

Définition 4.4. ***GBP en fonction des chemins d'exécution.***
Étant donné les chemins d'exécution $g_i\,\{i = 1, 2, \ldots s\}$ *d'un GBP* G *et* $p(g_i)\,\{i = 1, 2, \ldots s\}$ *les probabilités associées aux chemins d'exécution* g_i*, le GBP* G *est représenté par :*

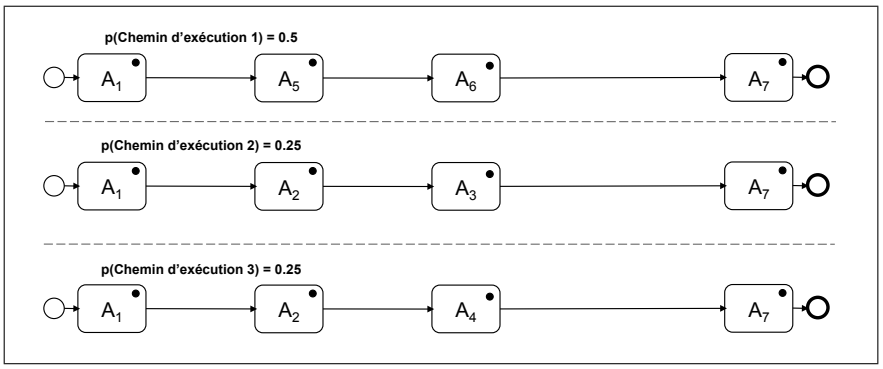
$$G = \bigcup_{i=1}^{s} (g_i, p(g_i)) \quad \textit{tel que} \sum_{i=1}^{s} p(g_i) = 1 \tag{4.3}$$

4.1.3 Propriétés non fonctionnelles d'un service

La réalisation d'une application composite nécessite une différenciation entre plusieurs services fonctionnellement équivalents. Les critères de différenciation sont souvent basés sur les propriétés non fonctionnelles des services. Dans la suite nous donnons un ensemble de définitions des propriétés non fonctionnelles des services.



(a) Probabilités comme attributs d'un GBP



(b) Probabilités associées aux chemins d'exécution

Figure 4.5: Déduction des probabilités associées aux chemins d'exécution

4.1.3.1 Définition

Les propriétés non fonctionnelles des services sont un ensemble paramètres reflétant l'environnement d'exécution ou des caractéristiques opérationnelles des services. Nous distinguons deux catégories de propriétés non fonctionnelles. La première catégorie regroupe les paramètres de Qualité de Service (QoS) qui peuvent être en étroite relation avec l'infrastructure abritant le service en question comme par exemple le serveur d'application, l'architecture réseau, etc. [1]. Ces propriétés sont souvent quantitatives. La deuxième catégorie de propriétés non fonctionnelles regroupe celles qui ne relèvent pas de l'aspect infrastructure, comme par exemple la capacité de gestion des pannes, des transactions, etc. Ces propriétés sont généralement qualitatives.

1. Dans la littérature, QoS et paramètres non fonctionnels sont utilisés parfois sans distinction, ne tenant pas compte de la subtilité évoquée.

Définition 4.5. ***Propriétés non fonctionnelles.***
Un service abstrait A_i possède un ensemble de propriétés non fonctionnelles désigné par $P_i = \{p_i^p\}_{p=1..q_i}$ où q_i désigne le nombre de propriétés non fonctionnelles significatives au service, tel que $P_i \subset GlobA$ où $GlobA$ désigne l'ensemble des propriétés non fonctionnelles définies dans l'écosystème. Chaque service implémentant A_i fournit des valeurs aux propriétés non fonctionnelles P_i de A_i.

4.1.3.2 Travaux connexes

La littérature a adressé les propriétés non fonctionnelles des services à plusieurs niveaux. Certains modèles proposent de définir et de décrire un ensemble de propriétés relatives à l'utilisation des services. D'autres proposent des modèles et des langages de description sémantique de QoS.

Lee *et al.* [77] fournissent une liste de paramètres QoS significatifs pour les services web : *performance, reliability, scalability, capacity, robustness, exception handling, accuracy, integrity, accessibility, availability, interoperability, security.* Les auteurs proposent une définition pour chaque paramètre sans préciser une méthode d'évaluation ou de calcul. O'Sullivan *et al.* présentent dans [103] un ensemble de paramètres non fonctionnels qualifiant les services. Maximilien et Singh présentent dans [88] un modèle pour la réputation de services. La réputation du service est composée de deux types d'attributs : des attributs génériques et d'autres spécifiques au domaine. Dans [119,120], Ran propose d'étendre le modèle de découverte de services basé sur UDDI avec un certificateur de QoS (QoS certifier). Il présente de nouvelles structures de données UDDI pour exprimer les valeurs des paramètres de QoS. Son modèle présente des similitudes avec celui proposé par Lee *et al.* dans [77], ainsi ces modèles sont limités à des descriptions syntaxiques des paramètres non fonctionnels proposés par le fournisseur du service.

Dans [110], Papaioannou *et al.* décrivent un langage d'ontologies de QoS pour les services. Ils étudient le processus de développement d'une ontologie de QoS. Cependant, les techniques adaptées aux matching entre concepts de QoS ne sont pas discutées. Zhou *et al.* décrivent dans [163] une ontologie de QoS spécifique à un domaine pour les services décrits par rapport à l'ontologie de services OWL-S. Dans [148], Wang *et al.* présentent un modèle plus élaboré de propriétés non fonctionnelles des services basé sur l'ontologie de services WSMO. La notion de tendance (inspirée de la logique floue) est introduite notamment en pondérant les propriétés non fonctionnelles. Kritikos & Plexousakis dans [74] proposent OWL-Q,

une ontologie pour décrire les paramètres de QoS des services. Ils définissent la découverte d'un service tenant compte de ses propriétés non fonctionnelles comme un CSP non linéraire (Constraint Solving Problem). Le but est d'inférer l'équivalence de deux paramètres QoS à partir de leurs définitions.

4.1.4 Propriétés non fonctionnelles d'un GBP

Une propriété non fonctionnelle $p^p \in GlobA$ est considérée comme une propriété d'un GBP G si elle associée à au moins un service abstrait ou un lien entre services abstraits de G. Dans la suite de cette étude, nous considérons que les propriétés non fonctionnelles sont quantitatives. Ce choix se justifie par l'existence de techniques de quantification de variables qualitatives [28,125]. Par exemple une propriété non fonctionnelle se rapportant à la sécurité peut être quantifiée en une propriété équivalente indiquant le niveau de sécurité sur une échelle de un à cinq.

4.1.4.1 Fonctions d'agrégation

Detynieki définit une fonction d'agrégation comme une fonction combinant les éléments d'un n-uplet appartenant à un même ensemble de définition en un seul élément appartenant à cet ensemble de définition [38]. Dans le cadre de DyCoSe, nous définissons pour chaque propriété non fonctionnelle p^p décrite dans le consensus global, une fonction d'agrégation $\sigma^p()$ qui permet de combiner les valeur d'un ensemble de propriétés $p^p_{i\,k_i}$ se rapportant à un ensemble de services $S_{i\,k_i}$ et leurs liens.

Prenons l'exemple de la propriété non fonctionnelle *prix*. Le prix d'invocation d'un ensemble de services est la somme des prix d'invocation de chacun des services. La fonction d'agrégation $\sigma^{prix}()$ n'est autre que la fonction somme (Σ). Prenons comme autre exemple, la propriété non fonctionnelle *temps*. La fonction d'agrégation des temps d'exécution d'un ensemble de services est la fonction somme, s'ils sont liés séquentiellement (en série). Cependant, si certains services sont liés parallèlement à d'autres, la fonction d'agrégation est la somme sur le chemin critique. Dans le cas de branchements en parallèle, le chemin critique est la séquence de services ayant le temps d'exécution le plus long. Un service appartenant au chemin critique ne peut pas être invoqué avant que l'exécution de ses prédécesseurs soit correctement complétée. D'autres fonctions d'agrégations peuvent être utilisées selon la propriété concernée, comme par exemple la moyenne arithmétique, la moyenne pondérée, etc.

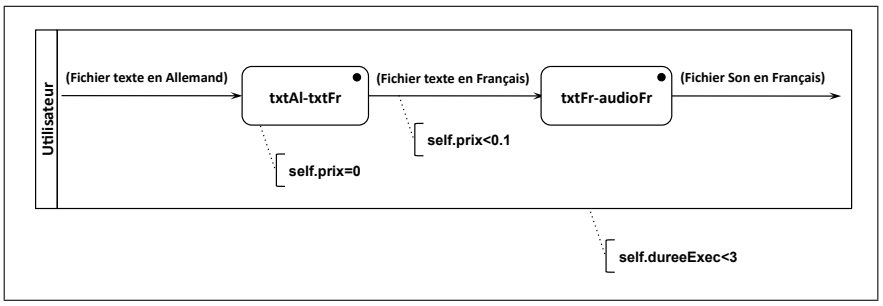


Figure 4.6: Exemple de contraintes sur un GBP

4.1.4.2 Contraintes sur un GBP

L'utilisateur exprime ses préférences par rapport à une application composite à travers un ensemble de contraintes sur le GBP de l'application. Deux types de contraintes sont possibles : des contraintes locales et d'autres globales. Les contraintes locales sont relatives à un service abstrait ou à un lien entre services abstraits. Une contrainte se rapportant à un service abstrait appartenant au GBP, indique les exigences (préférences) de l'utilisateur sur le service réalisant ce service abstrait. Une contrainte se rapportant à un lien entre services abstraits indique les exigences de l'utilisateur sur la réalisation du lien entre deux services abstraits. Les contraintes globales se rapportent à l'ensemble des services abstraits du GBP. Chaque contrainte globale est exprimée sur une propriété non fonctionnelle partagée par les services du GBP. Une contrainte sur une propriété donnée doit être vérifiée par l'ensemble des services du GBP redéfinissant cette propriété dans leur interface respective. Par exemple, la contrainte *coût d'exécution total inférieur à un euro*, implique que la somme des coûts d'exécution de tous les services participants doit être inférieure à un euro.

Nous reprenons l'application décrite en 3.1.3 pour illustrer un exemple de contraintes sur un GBP. L'application consiste en une composition d'un service de traduction de texte de l'Allemand au Français, suivi d'un service de synthétisation de voix à partir d'un texte en Français. La figure 4.6 illustre le GBP annoté par des contraintes. Une première contrainte indique que le prix d'invocation de l'opération de traduction de texte doit être gratuit. Une deuxième contrainte indique que le prix de transfert de données entre l'opération de traduction et l'opération de synthétisation doit être inférieur à dix centimes d'euros. Ces deux contraintes sont locales car elles sont relatives à un service et un lien. Une troisième contrainte indique

que la durée d'exécution totale de l'application ne doit pas dépasser trois minutes.

4.2 Problème d'instanciation de GBP

Tout d'abord, avant de définir le problème d'instanciation de GBP, nous définissons une instance d'un chemin d'exécution puis nous illustrons divers aspects du processus d'instanciation à travers un exemple simplifié. Ensuite, nous décrivons l'instanciation d'un GBP comme un problème d'optimisation de coûts.

4.2.1 Définition d'une instance

Pour un GBP G contenant un ensemble de services abstraits $\{A_i\}_{i=1..n}$, une instance $i_t(G)$ est un ensemble de services membres $\{S_{i\,k_i}\}_{i=1..n,\ k_i\in[1;n_i]}$ tel que chaque service $S_{i\,k_i}$ est une implémentation d'un service abstrait A_i. Un service $S_{i\,k_i}$ appartenant à une instance $i_t(G)$ vérifie un ensemble de règles relatives à la fois à ses propriétés fonctionnelles et à ses propriétés non fonctionnelles. Il doit :

1. Implémenter une fonctionnalité décrite par un service abstrait de G.
2. Respecter la conformité des paramètres d'entrée et de sortie. En effet, les paramètres d'entrée de $S_{i\,k_i}$ doivent être compatibles avec les paramètres de sortie de $S_{i-1\,k_{i-1}}$. Par exemple pour un paramètre simple désignant un salaire, la conformité de type implique que le salaire en entrée de $S_{i\,k_i}$ doit avoir le même type que le salaire en sortie de $S_{i-1\,k_{i-1}}$ ou sinon un type plus générique. Si le salaire en sortie de $S_{i-1\,k_{i-1}}$ est de type décimal, l'entrée de $S_{i\,k_i}$ doit être de type décimal. Elle pourra éventuellement être de type réel.
3. Redéfinir et évaluer les propriétés non fonctionnelles décrites par le service abstrait. Par exemple un service $S_{i\,k_i}$ implémentant un service abstrait A_i doit redéfinir et évaluer l'ensemble des propriétés non fonctionnelles de A_i. Dans l'écosystème de DyCoSe cette règle est vérifiée de facto. En effet, la relation d'implémentation [2] $IMPD(S_{i\,k_i}, A_i)$ exige la redéfinition et l'évaluation des différentes propriétés non fonctionnelles significatives au service abstrait implémenté.
4. Respecter les contraintes locales associées aux services abstraits. Un service $S_{i\,k_i}$ appartenant à une instance $i_t(G)$ doit respecter les contraintes locales

2. La relation d'implémentation est décrite en section 3.2.3.4

décrites par l'utilisateur de l'application composite. Il doit respecter aussi les contraintes sur les liens avec les services $S_{i-1\,k_{i-1}}$ et $S_{i+1\,k_{i+1}}$.

Une instance $i_t(G)$ doit vérifier les contraintes globales décrites par l'utilisateur sur une ou plusieurs propriétés non fonctionnelles partagées par l'ensemble des services $\{S_{i\,k_i}\}$ appartenant à $i_t(G)$.

Notation	**Elément associé**
A_i	Service abstrait i
$S_{i\,k_i}$	Service implémentant A_i
p^p	Propriété non fonctionnelle définie dans l'écosystème
$GlobA$	Ensemble de propriétés non fonctionnelles définies dans l'écosystème
p_i^p	Propriété non fonctionnelle redéfinie par A_i
P_i	Ensemble de propriétés non fonctionnelles redéfinies par A_i
$p_{i\,k_i}^p$	Propriété non fonctionnelle évaluée par $S_{i\,k_i}$
$P_{i\,k_i}$	Ensemble de propriétés non fonctionnelles évaluée par $S_{i\,k_i}$
$\sigma^p()$	Fonction d'agrégation applicable à un ensemble de $p_{i\,k_i}^p$
G	Un GBP
$\widetilde{V}$	Ensemble de services abstraits utilisés dans G
$\widetilde{E}$	Ensemble de liens entre services abstraits de F
$\widetilde{\psi()}$	Désigne A_i et $A_{i'}$ comme extrémités de $\widetilde{e_j} \in \widetilde{E}$
$\widetilde{\omega()}$	Associe des propriétés non fonctionnelles à $A_i \in \widetilde{V}$
$\widetilde{\epsilon()}$	Associe des propriétés non fonctionnelles à $\widetilde{e_j} \in \widetilde{E}$
g	Chemin d'exécution d'un GBP
$\overline{V}$	Ensemble de services abstraits utilisés dans g
$\overline{E}$	Ensemble de liens entre services abstraits de g
$\overline{\psi()}$	Désigne A_i et $A_{i'}$ comme extrémités de $\overline{e_j} \in \overline{E}$
$\overline{\omega()}$	Associe des propriétés non fonctionnelles à $A_i \in \overline{V}$
$\overline{\epsilon()}$	Associe des propriétés non fonctionnelles à $\overline{e_j} \in \overline{E}$
$p(g)$	Probabilité qu'un chemin d'exécution g soit emprunté
$\Omega(g)$	Ensemble des instances possibles de g
$I(g)$	Ensemble des instances valides de g

Table 4.1: Table rappelant les principales notations définies

4.2.2 Définition du processus d'instanciation d'un GBP

Le processus d'instanciation consiste à réaliser l'application décrite par un GBP à travers les services fournis par les membres de l'écosystème. Fonctionnellement, vis à vis de l'architecture à trois niveaux illustrée par la figure 3.9, le processus d'ins-

tanciation est un processus descendant qui a lieu au niveau services. Il traduit le passage de la couche GBP à la couche instance tenant compte des valeurs projetées du niveau infrastructure. Techniquement, tel décrit en 3.3.2.2, le processus d'instanciation d'un GBP consiste à remplacer des services abstraits dans l'interface BPEL par leurs implémentations respectives.

Pratiquement, le processus d'instanciation [3] d'un GBP correspond à décomposer le GBP en un ensemble de chemins d'exécution et à réaliser pour chaque chemin d'exécution les étapes suivantes :

1. Evaluer Ω, l'ensemble des instances possibles du chemin d'exécution g étudié. Cela revient à trouver l'ensemble des services membres fonctionnellement équivalents aux services abstraits de g et vérifiant la conformité des entrées - sorties. L'entier m désigne le nombre d'instances possibles,

$$|\Omega| = m = \prod_{i=1..n} n_i \tag{4.4}$$

   où n_i désigne le nombre de services membres implémentant le service abstrait A_i.
2. Evaluer $I = \{i_1, i_2, \ldots, i_t, \ldots, i_v\}$, l'ensemble des instances valides du chemin d'exécution étudié. Cela revient à retenir les combinaisons de services membres répondant aux préférences de l'utilisateur exprimées comme contraintes sur les propriétés non fonctionnelles des services. L'entier v désigne le nombre d'instances vérifiant ces contraintes.
3. Choisir i_d, l'instance valide retenue. i_d définit l'orchestration des services membres qui seront invoqués afin de réaliser l'application.

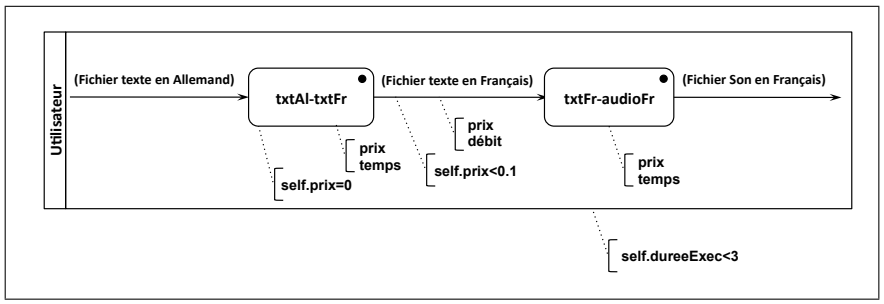
Le déploiement d'une application composite représentée par un GBP G, revient à décomposer

La table 4.1 rappelle les principales notations définies dans les parties précédentes et réutilisées par la suite. Elle associe à chaque notation un élément parmi ceux définis dans cette étude.

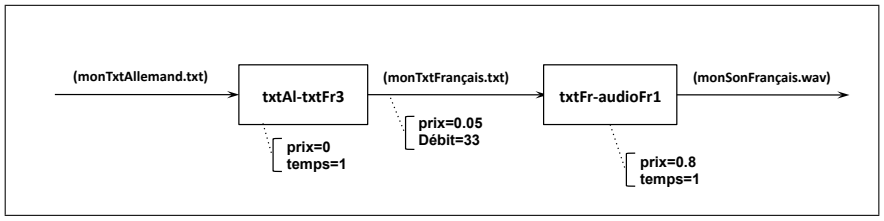
4.2.3 Exemple d'instanciation

La figure 4.7a illustre un GBP attribué composé d'un chemin d'exécution. Les attributs expriment les propriétés non fonctionnelles des deux services abstraits et du

3. Dans la suite nous utilisons uniformément et de manière interchangeable les termes instanciation et processus d'instanciation.



(a) Un GBP ayant un chemin d'exécution



(b) Une instance possible

Figure 4.7: Exemple d'un GBP attribué et d'une instance

lien. Les propriétés prix et temps sont associées aux services abstraits. Les propriétés prix et débit sont associées au lien. Les attributs expriment aussi trois contraintes, dont deux locales une globale, identiques à celles décrites par la figure 4.6. La table 4.2a décrit un ensemble de services réalisant les services abstraits du GBP de la figure 4.7a. Chaque service évalue les propriétés non fonctionnelles associées au service abstrait correspondant. Dans cet exemple, tous les services évaluent le prix d'invocation et le temps d'exécution. La table 4.2b décrit les liens entre les services. Chaque lien évalue les propriétés non fonctionnelles associées au lien correspondant dans le GBP. Dans cet exemple, le lien décrit par le GBP évalue le prix de communication et le débit. Nous supposons que les services txtAl-txtFr3 et txtFr-audioFr1 sont abrités par le même serveur d'application. D'où l'explication des valeurs avantageuses pour le prix de communication et le débit.

Une instance possible de ce GBP sera composée de deux services implémentant respectivement la fonctionnalité traduction et conversion texte - audio. L'implémentation du service abstrait de traduction doit être compatible avec celle du service abstrait de conversion texte - audio. Par exemple, si s(txtAl-txtFr1) désigne les paramètres de sortie du service txtAl-txtFr1 et e(txtFr-audioFr1) désigne les paramètres d'entrée du service txtFr-audioFr1, pour que l'instance <txtAl-

Service	**Prix** euro	**Temps** s
txtAl-txtFr1	0.1	0.8
txtAl-txtFr2	0	1.1
txtAl-txtFr3	0	1
txtFr-audioFr1	0.5	2
txtFr-audioFr2	0.8	1

(a) Services

Lien	**Prix** euro	**Débit** Mo/s
(txtAl-txtFr1, txtFr-audioFr1)	0.05	10
(txtAl-txtFr1, txtFr-audioFr2)	0.03	20
(txtAl-txtFr2, txtFr-audioFr1)	0.15	50
(txtAl-txtFr2, txtFr-audioFr2)	0.09	10
(txtAl-txtFr3, txtFr-audioFr1)	0.05	33
(txtAl-txtFr3, txtFr-audioFr2)	0	∞

(b) Liens

Instance	**Prix total** euro	**Temps lien** s	**Temps total** s
<txtAl-txtFr1, txtFr-audioFr1>	0.65	0.1	2.9
<txtAl-txtFr1, txtFr-audioFr2>	0.93	0.05	1.85
<txtAl-txtFr2, txtFr-audioFr1>	0.65	0.02	3.12
<txtAl-txtFr2, txtFr-audioFr2>	0.89	0.1	2.2
<txtAl-txtFr3, txtFr-audioFr1>	0.55	0.03	3.03
<txtAl-txtFr3, txtFr-audioFr2>	0.8	0	2

(c) Instances possibles et coûts associés

Table 4.2: Exemple de services, de liens et des instances possibles

txtFr1, txtFr-audioFr1> appartienne à l'ensemble des instances possibles il faut que : s(txtAl-txtFr1) $\subseteq$ e(txtFr-audioFr1). Pour qu'une instance possible soit marquée comme valide, l'ensemble des services et des liens la composant doivent vérifier la totalité des contraintes décrites.

La première étape du processus d'instanciation consiste à générer les combinaisons entre services implémentant les fonctionnalités requises et respectant la conformité des paramètres d'entrée et de sortie. Ainsi l'ensemble des instances possibles est engendré. En se basant sur les services de la table 4.2a, et en supposant que tous les services décrits respectent la conformité des paramètres d'entrée et de sortie, l'ensemble des instances possibles Ω du GBP étudié contient les instances suivantes :

- <txtAl-txtFr1, txtFr-audioFr1>
- <txtAl-txtFr1, txtFr-audioFr2>
- <txtAl-txtFr2, txtFr-audioFr1>
- <txtAl-txtFr2, txtFr-audioFr2>
- <txtAl-txtFr3, txtFr-audioFr1>
- <txtAl-txtFr3, txtFr-audioFr2>

La deuxième étape du processus d'instanciation consiste à éliminer les instances ne vérifiant pas les contraintes décrites. D'abord la vérification des contraintes locales permet un premier "filtrage". En effet, toutes les instances avec txtAl-txtFr1 sont

éliminées car l'invocation de txtAl-txtFr1 n'est pas gratuite. Par suite les instances <txtAl-txtFr1, txtFr-audioFr1> et <txtAl-txtFr1, txtFr-audioFr2> ne sont pas valables. De plus, l'instance possible <txtAl-txtFr2, txtFr-audioFr1> est éliminée car le lien (txtAl-txtFr2, txtFr-audioFr1) ne respecte pas la contrainte de prix associée. Ensuite la vérification des contraintes globales permet d'évaluer l'ensemble des instances valides I. En supposant qu'après la traduction du fichier monTxtAllemand.txt, la taille du fichier transmis monTxtFrançais.txt est de 1Mo, la table 4.2c indique pour chaque instance possible les coûts associés en terme de prix et de temps. En effet pour chaque paramètre non fonctionnel défini dans l'instance, un coût est calculé en agrégeant les valeurs de ce paramètre sur l'ensemble des services et des liens de l'instance. L'instance possible <txtAl-txtFr3, txtFr-audioFr1> est éliminée car elle ne respecte pas la contrainte globale sur la durée totale d'exécution. L'ensemble des instance valides est $I = \{$ <txtAl-txtFr2, txtFr-audioFr2>, <txtAl-txtFr3, txtFr-audioFr2> $\}$.

4.2.4 Énoncé du problème d'instanciation d'un GBP

Le problème d'instanciation peut être énoncé comme suit : étant donné un GBP G ayant un ensemble de services abstraits $\{A_i\}_{i=1..n}$ auquel correspond fonctionnellement un ensemble de services membres $\{S_{i\,k_i}\}_{i=1..n,\ k_i=1..n_i}$ où n est le nombre de services abstraits et n_i le nombre de services membres implémentant le service abstrait A_i, quelle est la combinaison de services membres générant la meilleure instance dans un environnement d'exécution dynamique ? La figure 4.8 illustre un exemple d'un GBP simplifié. Elle liste pour chaque service abstrait $A_i,\ i \in [1;n]$, l'ensemble des services membres $S_{i\ k_i},\ k_i \in [1;n_i]$, où n est le nombre de services abstrait dans le GBP et n_i est le nombre de services membres $S_{i\ k_i}$ implémentant le service abstrait A_i. Une instance de ce GBP est définie à travers un ensemble de $S_{i\ k_i}$ tel que $\forall\ i \in [1;n]$ un et un seul $S_{i\ k_i}$ est choisi comme instance de A_i.

Résoudre le problème d'instanciation revient à choisir une instance de départ vérifiant les préférences de l'utilisateur dans un environnement dynamique où certains fournisseurs ne sont pas continuellement présents. Le choix de l'instance de départ parmi les instances possibles est basé sur une différenciation entre propriétés non fonctionnelles, le but étant de satisfaire un ensemble de contraintes exprimant les préférences de l'utilisateur. Nous avons classifié les contraintes en deux catégories : les contraintes locales et les contraintes globales. Nous soulignons que la satisfaction des contraintes locales est un sous-problème de la satisfaction des contraintes glo-

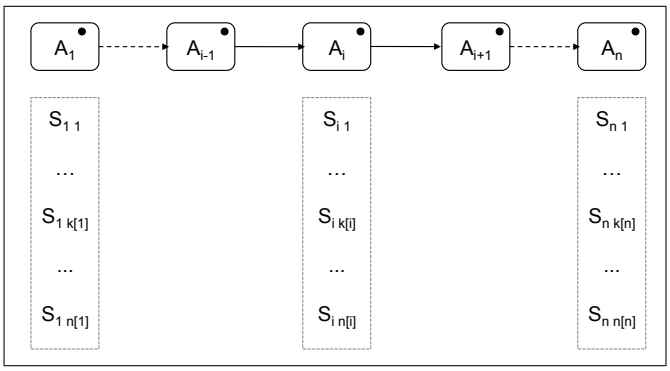


Figure 4.8: Représentation de l'espace des instances des services abstraits d'un GBP

bales. En effet, une contrainte globale porte sur l'ensemble des services et des liens d'un GBP, tandis qu'une contrainte locale porte sur une partie (sous-ensemble) des services et des liens d'un GBP. Les contraintes locales permettent un premier filtrage qui élimine certaines instances possibles, réduisant ainsi l'espace de recherche. Dans la suite nous étudions le problème d'instanciation visant à satisfaire des contraintes globales.

4.2.5 Instance d'un chemin d'exécution

Une instance $i_t(g)$ d'un chemin d'exécution g, est représentée par :

$$i_t(g) = (V,\ E,\ \psi(),\ \omega(),\ \epsilon(),\ C) \tag{4.5}$$

où

- V est l'ensemble des services membres utilisés dans $i_t(g)$. Il contient pour chaque service abstrait $A_i \in \overline{V}$ un service membre $S_{i\,k_i}$ vérifiant la relation $IMPD(S_{i\,k_i}, A_i)$ décrite en 3.2.3.4. $V = \{S_{1\,k_1}, S_{2\,k_2}, \ldots, S_{n\,k_n}\}_{k_i \in [1 \ldots n_i]}$ où n est le nombre de services abstraits dans g et n_i est le nombre de services membres implémentant A_i.
- $E \subset V \times V$ est l'ensemble de liens entre les services de l'instance $i_t(g)$. $E = \{e_1, e_2, \ldots, e_l\}$ où l désigne le nombre de liens dans le chemin d'exécution g. E contient pour chaque lien $\overline{e_j} \in \overline{E}$ un lien e_j entre les services de l'instance $i_t(g)$.
- La fonction d'incidence $\psi() : E \rightarrow V \times V$ associe à chaque lien e_j de E un

couple $(S_{i\,k_i}, S_{i'\,k_{i'}})$. Si e_j est un lien et $S_{i\,k_i}$ et $S_{i'\,k_{i'}}$ sont des nœuds tels que $\psi(e_j) = (S_{i\,k_i}, S_{i'\,k_{i'}})$ alors e_j lie $S_{i\,k_i}$ à $S_{i'\,k_{i'}}$. Fonctionnellement cette liaison implique que la sortie du service membre $S_{i\,k_i}$ est utilisée en entrée du service membre $S_{i'\,k'_i}$.

- La fonction $\omega()$ évalue pour chaque service $S_{i\,k_i} \in V$ l'ensemble de ses propriétés non fonctionnelles $P_{i\,k_i}$ correspondant à l'ensemble des propriétés non fonctionnelles définies pas A_i.
- La fonction $\epsilon()$ évalue pour chaque lien entre services membres e_j correspondant à un lien $\overline{e_j}$ entre services abstraits, l'ensemble des propriétés non fonctionnelles décrit par $\overline{\epsilon(\overline{e_j})}$.
- C désigne le coût de l'instance par rapport aux propriétés non fonctionnelles significatives. Nous développons le coût d'une instance dans la section suivante.

Considérons par exemple l'instance du chemin d'exécution illustrée par la figure 4.7b. La notation formelle de cette instance comprend les éléments suivants : (i) l'ensemble $V = \{S_{1\,3}, S_{2\,1}\}$ où $S_{1\,3}$ = txtAl-txtFr3 et $S_{2\,1}$ = txtFr-audioFr1, indique que les services réalisant les services abstraits A_1 et A_2 sont respectivement $S_{1\,3}$ et $S_{2\,1}$. (ii) L'ensemble $E = \{e_1\}$ tel que $\psi(e_1) = (S_{1\,3}, S_{2\,1})$ = (txtAl-txtFr3, txtFr-audioFr1) indique que la sortie de $S_{1\,3}$ est liée à l'entrée de $S_{2\,1}$. Ce lien est désigné par e_1. (iii) La fonction d'attributs $\omega()$ associe au service $S_{1\,3}$ l'ensemble $P_{1\,3}$ et au service $S_{2\,1}$ l'ensemble $P_{2\,1}$ tel que $P_{1\,3} = \{p^1 = 0, p^2 = 1\}$ et $P_{2\,1} = \{p^1 = 0.8, p^2 = 1\}$. (iv) La fonction d'attributs $\epsilon(e_1) = \{p^1 = 0.05, p^3 = 0.03\}$ évalue les propriétés associées au lien par $\overline{\epsilon(\overline{e_1})}$.

4.2.6 Coût d'une instance d'un chemin d'exécution

Le coût d'une instance $i_t(g)$ est un n-uplet désigné par $C(i_t(g)) = \langle c^1(i_t(g)), c^2(i_t(g)), \ldots, c^n(i_t(g))\rangle$ tel que $c^p(i_t(g))$ désigne le coût de l'instance $i_t(g)$ vis-à-vis de la propriété non fonctionnelle p^p. Il est calculé en appliquant la fonction d'agrégation $\sigma^p()$ à l'ensemble des services et des liens entre services appartenant à l'instance et redéfinissant la propriété p^p. Nous rappelons que la fonction $\omega()$ évalue les propriétés non fonctionnelles d'un service $S_{i\,k_i}$ et que la fonction $\epsilon()$ évalue les propriétés non fonctionnelles d'un lien $e_j = (S_{i\,k_i}, S_{i'\,k_{i'}})$.

Dans la suite de cotre étude, nous illustrons le calcul du coût d'une instance par rapport à deux propriétés non fonctionnelles : le prix et le temps. Nous supposons

que l'utilisateur s'intéresse à minimiser le prix d'exécution tout en gardant la durée d'exécution inférieure à une valeur donnée. Pour une instance donnée $i_t(g)$ nous définissons deux mesures de performances liées au prix désigné par p^1 et deux autres mesures liées au temps désigné par p^2.

- Le prix d'invocation d'un service $S_{i\,k_i} \in V$ est désigné par $p^1_{i\,k_i}$.
- Le prix d'exploitation d'un lien $e_j = (S_{i\,k_i}, S_{i'\,k_{i'}}) \in E$ est désigné par $p^1(e_j)$ ou $p^1((S_{i\,k_i}, S_{i'\,k_{i'}}))$.
- Le temps d'exécution prévu pour un service $S_{i\,k_i} \in V$ est désigné par $p^2_{i\,k_i}$.
- La temps de transfert de données via un lien $e_j = (S_{i\,k_i}, S_{i'\,k_{i'}}) \in E$ est désigné par $p^2(e_j)$ ou $p^2((S_{i\,k_i}, S_{i'\,k_{i'}}))$ tel que

$$p^2((S_{i\,k_i}, S_{i'\,k_{i'}})) = \frac{io((S_{i\,k_i}, S_{i'\,k_{i'}}))}{p^3(S_{i\,k_i}, S_{i'\,k_{i'}})}$$

  où $io((S_{i\,k_i}, S_{i'\,k_{i'}}))$ désigne la taille des données transférées entre $S_{i\,k_i}$ et $S_{i'\,k_{i'}}$ et $p^3(S_{i\,k_i}, S_{i'\,k_{i'}})$ indique le débit de la connexion entre $S_{i\,k_i}$ et $S_{i'\,k_{i'}}$.

En se basant sur les mesures de performances décrites ci-dessus le coût d'une instance $i_t(g)$ est donné par $C(i_t(g)) = \langle c^1(i_t(g)), c^2(i_t(g))\rangle$ tel que :

$$c^1(i_t(g)) = \sigma^1\left(\sigma^1_{i=1..n}(p^1_{i\,k_i}), \sigma^1_{j=1..l}(p^1(e_j)\right) \text{ où } k_i \in \mathbb{N} \tag{4.6}$$

et

$$c^2(i_t(g)) = \sigma^2\left(\sigma^2_{i=1..n}(p^2_{i\,k_i}), \sigma^2_{j=1..l}(p^2(e_j)\right) \text{ où } k_i \in \mathbb{N} \tag{4.7}$$

où n désigne le nombre de services dans le chemin d'exécution g et l désigne le nombre de liens entre les services de g. Sachant que la fonction d'agrégation $\sigma^1()$ désigne la somme, l'équation 4.6 devient :

$$c^1(i_t(g)) = \sum_{i=1}^{n} p^1_{i\,k_i} + \sum_{i,i' \in [1;n]}^{i \neq i'} p^1(S_{i\,k_i}, S_{i'\,k_{i'}}) \text{ où } k_i, k_{i'} \in \mathbb{N} \tag{4.8}$$

La table 4.3 résume les notations supplémentaires proposées. Elle complète la table 4.1 rappelant ainsi les notations adoptées dans l'étude du processus d'instanciation.

Notation	Elément associé
$i_t(g)$	Instance d'un chemin d'exécution g
V	Ensemble de services utilisés dans $i_t(g)$
E	Ensemble de liens entre services de $i_t(g)$
$\psi()$	Désigne $S_{i\,k_i}$ et $S_{i'\,k_{i'}}$ comme extrémités de $e_j \in E$
$\omega()$	Evalue les propriétés non fonctionnelles de $S_{i\,k_i} \in V$
$\epsilon()$	Evalue les propriétés non fonctionnelles de $e_j \in E$
$c^p(i_t(g))$	Coût de l'instance $i_t(g)$ par rapport à p^p
$C(i_t(g))$	n-uplet contenant des $c^p(i_t(g))$

Table 4.3: Table résumant les notations supplémentaires proposées

4.2.7 Choix de l'instance de départ d'un chemin d'exécution

Le choix de l'instance de départ $i_d(g)$ consiste d'abord à déterminer l'instance ayant le coût $c^1(i_d(g))$ minimal et ensuite à vérifier que l'instance ayant le coût minimal en terme de prix possède un temps d'exécution inférieur à la valeur précisée par l'utilisateur.

Déterminer l'instance ayant le coût $c^1(i_d(g))$ minimal est exprimé alors comme un problème d'optimisation de coût utilisant la variable décisionnelle $X_i[k_i]$ et défini comme suit :

$$\min_{\substack{k_1 \ldots k_n \\ k_i \in [1;n_i] \\ i=1..n}} \left(\sum_{i=1}^{n} \left(X_i[k_i] * p^1_{i\,k_i} \right) + \sum_{i=1}^{n} \sum_{i'=1}^{n} \left(X_i[k_i] * X_{i'}[k_{i'}] * T[i][i'] * p^1(S_{i\,k_i}, S_{i'\,k_{i'}}) \right) \right) \tag{4.9}$$

tel que

$$X_i[k_i] = \begin{cases} 1 & \text{si } S_{i\,k_i} \text{ est choisi comme implémentation de } A_i \\ 0 & \text{dans le cas contraire} \end{cases} \tag{4.10}$$

et

$$T[i][i'] = \begin{cases} 1 & \text{si } \exists\ e_j\ /\ \psi(e_j) = (A_i, A_{i'}) \\ 0 & \text{le cas échéant} \end{cases} \tag{4.11}$$

et

$$\sum_{k_i=1}^{n_i} X_i[k_i] = 1 \tag{4.12}$$

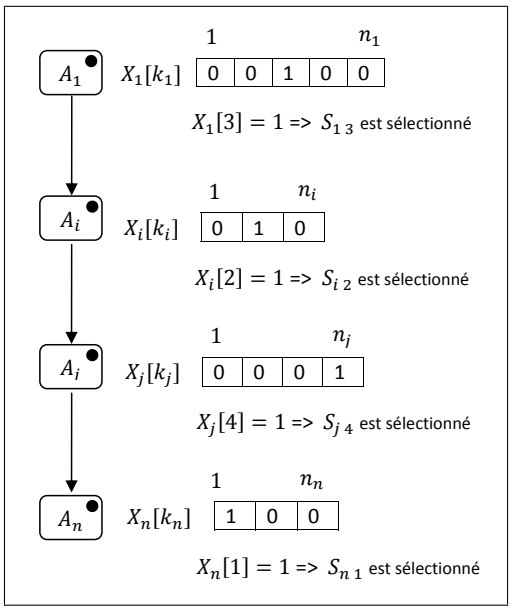


Figure 4.9: Exemple de représentation des variables décisionnelles

et

$$\sum_{i=1}^{n} \sum_{k_i=1}^{n_i} X_i[k_i] = n \tag{4.13}$$

L'équation 4.9 permet de parcourir les instances possibles en calculant leur prix d'invocation respectif et de sélectionner la combinaison optimale. Effectivement, à travers les variations de i nous parcourons l'ensemble des services abstraits A_i appartenant au chemin d'exécution à instancier. À travers les variations de k_i sur chaque A_i, nous parcourons l'ensemble des implémentations possibles $S_{i\,k_i}$ de A_i.

La *variable décisionnelle* $X_i[k_i]$, (4.10), indique si le service membre $S_{i\,k_i}$ est sélectionné comme implémentation pour le service abstrait A_i ou non. Quand la valeur de $X_i[k_i]$ est égale à un, le service $S_{i\,k_i}$ est choisi. $X_i[k_i]$ est égale à zéro si $S_{i\,k_i}$ n'est pas selectionné dans l'instance optimale. Par exemple, $X_2[7] = 1$ indique que le service $S_{2\,7}$, septième implémentation de A_2 sera invoqué lors de l'exécution de l'instance pour accomplir la fonctionnalité décrite par le service abstrait A_2.

Le *paramètre* $T[i][i']$ (4.11), représente la matrice d'adjacence du graphe du chemin d'exécution. La valeur de $T[i][i']$ est égale à un si la sortie de A_i est utilisée par l'entrée de $A_{i'}$. Inversement, la valeur de $T[i][i']$ est égale à zéro si aucun lien existe entre les services A_i et $A_{i'}$.

Deux contraintes sont à respecter sur l'équation 4.9. D'abord, la contrainte sur la variable décisionnelle $X_i[k_i]$ dans le cas où le service A_i est fixé et k_i varie, est exprimée par l'équation 4.12. Elle indique que pour chaque service abstrait A_i une seule implémentation concrète $S_{i\,k_i}$ est choisie. Ce qui est traduit par l'assertion suivante : la somme des $X_i[k_i]$ pour un i donné est égale à un. La figure 4.9, représente les variables décisionnelles associées à chaque service abstrait du graphe d'un chemin d'exécution. Ensuite la contrainte sur la variable décisionnelle $X_i[k_i]$ dans le cas où i et k_i varient simultanément est exprimée dans l'équation 4.13. Elle indique que le nombre total de $X_i[k_i]$, dont la valeur est à un, est égal à n qui désigne nombre total de noeuds dans le graphe du chemin d'exécution. En effet, pour chaque service abstrait A_i dans le graphe du chemin d'exécution, une et une seule implémentation $S_{i\,k_i}$ est choisie.

4.2.8 Reconstruction d'une instance d'un GBP

Nous avons souligné en 4.1.2 qu'un GBP complexe contenant des nœuds de type branchement décisionnel ou jointure décisionnelle est décomposé en un ensemble de chemins d'exécution. De ce fait, la réalisation d'un GBP complexe passe par la réalisation des différents chemins d'exécution constituant ce GBP. Pratiquement, l'instanciation d'un GBP débute par l'identification des différents chemins d'exécution tel décrit en 4.1.2. Chaque chemin est ensuite instancié indépendamment des autres. Une fois les instances de départ respectives sont sélectionnées, elles sont combinées pour déterminer une instance de départ du GBP. Nous adaptons la technique d'agrégation des chemins d'exécution proposée par Zeng *et al.* dans [158]. En effet, après avoir sélectionné les instances des différents chemins d'exécution l'agrégation des instances retenues est accomplie selon les règles suivantes :

- Pour chaque service abstrait A_i appartenant à un seul chemin d'exécution, le service $S_{i\,k_i}$ choisi comme implémentation de A_i dans l'instance de ce chemin est retenu comme implémentation de A_i dans l'instance du GBP.
- Si un service abstrait A_i appartient à deux chemins d'exécution différents, le service $S_{i\,k_i}$ qui sera retenu comme implémentation de A_i dans l'instance du GBP est celui qui appartient à l'instance du chemin d'exécution ayant la probabilité la plus élevée.

4.3 Solutions au problème d'instanciation

Dans cette section nous étudions l'instanciation d'un chemin d'exécution g. Nous avons décrit en section 4.2.8, la déduction de l'instance d'un GBP ayant plusieurs chemins d'exécution à partir des instances des chemins d'exécution possibles. Pour résoudre le problème d'instanciation de g, il ne suffit pas de retenir les services fonctionnellement équivalents aux services abstraits de g, il faut aussi évaluer un ensemble de contraintes sur des propriétés non fonctionnelles des services membres.

Nous rappelons que dans l'écosystème de DyCoSe, les membres sont organisés par communautés. Un membre cherchant à instancier un GBP peut décider de prendre en compte les services fournis par toutes les communautés ou de favoriser les services de certaines communautés par rapport à d'autres. Deux types de solutions au problème d'instanciation sont envisageables. Une solution globale qui tient compte de l'ensemble des services membres de l'écosystème et une solution locale qui sélectionne en priorité les services d'une ou de plusieurs communautés choisies par l'utilisateur.

4.3.1 Solution globale

Une solution globale repose sur la connaissance de l'ensemble des services membres fournis dans l'écosystème et des valeurs de leurs propriétés non fonctionnelles respectives. La détermination d'une solution globale au problème d'instanciation suit les étapes suivantes :

1. Évaluation de l'ensemble des instances possibles Ω, tel que chaque instance appartenant à Ω est composée de services membres qui (i) sont fonctionnellement équivalents aux services abstraits du GBP à instancier et qui (ii) respectent la conformité des entrées et des sorties.

2. Calcul pour chaque instance possible $i_t(g)$, des différents coûts partiels $c^p(i_t(g))$ par rapport à la propriété non fonctionnelle p^p, comme expliqué dans la section 4.2.6.

3. Détermination du coût total $C(i_t(g))$ de chaque instance possible i_t, en combinant les coûts partiels de chaque instance afin d'évaluer les contraintes globales et de retenir les instances valides.

4. Sélection de l'instance de départ i_d ayant le meilleur coût.

Dans la suite une proposons une technique de *multi-attribute decision making* [157] appliquée au problème de l'instanciation globale. Elle permet de choisir la meilleure instance parmi un ensemble d'instances données. Elle consiste (i) à calculer une matrice de coûts, (ii) à éliminer les instances ne vérifiant pas certaines contraintes, (iii) à normaliser la matrice calculée et (iv) à calculer un score pondéré pour chaque colonne.

Pour simplifier la représentation nous désignons dans la suite $i_t(g)$ par i_t. La section 4.2.6 détaille le calcul du coût d'une instance par rapport à une propriété non fonctionnelle donnée. En faisant ce calcul pour l'ensemble des propriétés non fonctionnelles sur lesquelles l'utilisateur a exprimé ses préférences sous forme de contraintes, le n-tuple $C(i_t)$ est déterminé.

Matrice des coûts partiels Soit q le nombre de propriétés non fonctionnelles sur lesquelles l'utilisateur a exprimé des contraintes. Le coût total de chaque instance est le q-tuple $C(i_t) = \langle c^1(i_t), c^2(i_t), \ldots, c^q(i_t) \rangle$. Nous rappelons que l'ensemble des m instances possibles est calculé selon l'équation 4.4 comme le produit des nombres des services membres implémentant chaque service abstrait du GBP. Nous regroupons les coûts des m instances possibles dans une matrice des coûts décrite par l'équation 4.14, comme suit :

$$MC_{q,m} = \begin{bmatrix} c^1(i_1) & c^1(i_2) & \cdots & c^1(i_m) \\ c^2(i_1) & c^2(i_2) & \cdots & c^2(i_m) \\ \vdots & \vdots & \ddots & \vdots \\ c^q(i_1) & c^q(i_2) & \cdots & c^q(i_m) \end{bmatrix} \tag{4.14}$$

Élimination des instances non valides Ayant la matrice des coûts $MC_{q,m}$, nous pouvons déterminer l'ensemble des instances valides I respectant les contraintes de l'utilisateur. Deux types de contraintes sont possibles : des contraintes du genre le coût minimal (maximal) ou des contraintes du genre coût inférieur (supérieur) à un seuil. Nous éliminons les instances ne respectant pas les contraintes du genre coût inférieur (supérieur) à un seuil. Ce premier filtrage permet de réduire le nombre d'instances à traiter. Nous désignons par $m' \leq m$ le nombres d'instances possibles vérifiant les contraintes globales du genre coût inférieur (supérieur) à un seuil.

Matrice des coûts normalisés (ou scores) Soit CU l'ensemble des contraintes globales cu^p du genre le coût minimal (maximal), $|CU| \leq q$. Pour chaque $cu^p \in CU$, nous proposons de normaliser les valeurs de $c^p(i_t)$ afin de les combiner et de les comparer. Nous proposons une technique de normalisation en trois étapes adaptée de [148] et de [158]. La première étape consiste à identifier le maximum c^p_{max} et le minimum c^p_{min} des coûts partiels des instances par rapport à la propriété non fonctionnelle p^p. La deuxième étape consiste à calculer pour chaque coût partiel $c^p(i_t)$, le coût normalisé ou *score* $\overline{c^p(i_t)}$ selon l'équation suivante :

$$\overline{c^p(i_t)} = \begin{cases} \frac{c^p(i_t)-c^p_{max}}{c^p_{max}-c^p_{min}} & \text{si} \quad c^p_{max} - c^p_{min} \neq 0 \\ 1 & \text{si} \quad c^p_{max} - c^p_{min} = 0 \end{cases} \tag{4.15}$$

La troisième étape consiste à générer une matrice des scores des instances $\overline{MC_{q,m'}}$.

Les valeurs de certaines propriétés non fonctionnelles sont inversement proportionnelles au niveau de qualité indiqué. Ces propriétés non fonctionnelles sont dites négatives. Pour une propriété négative, plus sa valeur est grande, moins son niveau de qualité est élevé. Par exemple le prix indiquant le prix d'invocation d'un service ou le prix d'exploitation d'une connexion. Un autre exemple est le temps indiquant une durée d'exécution ou une durée de transmission de données. Le score des propriétés négatives est donnée par l'équation 4.16.

$$\overline{c^p(i_t)} = \begin{cases} \frac{c^p_{max}-c^p(i_t)}{c^p_{max}-c^p_{min}} & \text{si} \quad c^p_{max} - c^p_{min} \neq 0 \\ 1 & \text{si} \quad c^p_{max} - c^p_{min} = 0 \end{cases} \tag{4.16}$$

Calcule du score des instances Nous supposons que l'utilisateur affecte un coefficient $\mathrm{w}^p \in [0;1]$ indiquant l'importance de la propriété non fonctionnelle p^p, tel que

$$\sum_{p=1}^{|CU|} \mathrm{w}^p = 1$$

À partir de $\overline{MC_{q,m'}}$, nous pouvons calculer pour chaque instance i_t un score global $\overline{C(i_t)}$ en faisant la somme pondérée des scores partiels :

$$\overline{C(i_t)} = \sum_{p=1}^{q} (\mathrm{w}^p \times \overline{c^p(i_t)}) \tag{4.17}$$

Instance i_t	**Prix total** $c^1(i_t)$ euro	**Temps total** $c^2(i_t)$ s
i_1, <txtAl-txtFr1, txtFr-audioFr1>	0.65	2.9
i_2, <txtAl-txtFr1, txtFr-audioFr2>	0.93	1.85
i_3, <txtAl-txtFr2, txtFr-audioFr1>	0.65	3.12
i_4, <txtAl-txtFr2, txtFr-audioFr2>	0.89	2.2
i_5, <txtAl-txtFr3, txtFr-audioFr1>	0.55	3.03
i_6, <txtAl-txtFr3, txtFr-audioFr2>	0.8	2

(a) Instances possibles et coûts associés

i_t	$\overline{c^1(i_t)}$	$\overline{c^2(i_t)}$	$\overline{C(i_t)}$
i_1	0.736	0.173	0.567
i_2	0	1	0.3
i_3	0.736	0	0.515
i_4	0.142	0.724	0.316
i_5	1	0.07	0.721
i_6	0.464	0.881	0.588

(b) Coûts normalisés ou scores pour l'exemple 1

Table 4.4: Exemple d'instances et de coûts associés

La comparaison des scores des différentes instances permet de sélectionner l'instance de départ ayant le score le plus élevé.

4.3.1.1 Exemples de solutions globales

Nous reprenons dans la table 4.4a, l'exemple des instances de la table 4.2c avec leurs coûts partiels suivants : $c^1(i_t)$ indiquant le prix total d'exécution et $c^2(i_t)$ indiquant le temps total d'exécution.

Exemple 1 Nous supposons que l'utilisateur veut *minimiser à la fois le prix et le temps d'exécution* en favorisant le *prix par un coefficient de 0.7*. La solution globale consiste (i) d'abord à calculer les coûts partiels qui sont indiqués dans la table 4.4a et retranscrits dans la matrice suivante :

$$MC_{2,6} = \begin{bmatrix} 0.65 & 2.9 \\ 0.93 & 1.85 \\ 0.65 & 3.12 \\ 0.89 & 2.2 \\ 0.55 & 3.03 \\ 0.8 & 2 \end{bmatrix}$$

(*ii*) ensuite à identifier le minimum et le maximum des coûts partiels des instances par rapport aux deux propriétés non fonctionnelles, prix et temps respectivement. Nous identifions pour le prix : $c^1_{min} = 0.55$ et $c^1_{max} = 0.93$ et pour le temps $c^2_{min} = 1.85$ et $c^2_{max} = 3.12$, (*iii*) puis ayant ces valeurs, les coûts partiels des instances par rapport au prix d'exécution et au temps d'exécution sont normalisées en appliquant la formule de l'équation 4.16. Les résultats exprimant les scores partiels de chque instance sont donnés dans les colonnes 2 et 3 de la table 4.4b et retranscrits dans la matrice normalisée ci-dessous :

$$\overline{MC_{2,6}} = \begin{bmatrix} 0.736 & 0.173 \\ 0 & 1 \\ 0.736 & 0 \\ 0.142 & 0.724 \\ 1 & 0.07 \\ 0.464 & 0.881 \end{bmatrix}$$

(*iv*) Finalement, le score total de chaque instance est calculé selon l'équation 4.17 comme suit : $C(i_t) = 0.7 \times c^1(i_t) + 0.3 \times c^2(i_t)$. La colonne 4 de la table 4.4b donne le score total de chaque instance. L'instance ayant le score le plus élevé est sélectionnée : $i_d = i_5$.

Exemple 2 Toujours dans l'exemple des instances de la table 4.2c, nous supposons que l'utilisateur a *un budget maximal de 0.80 euros* et qu'il veut *minimiser le temps d'exécution de l'application.* La solution globale consiste (*i*) d'abord à calculer les coûts partiels qui sont indiqués dans la table 4.4a ; (*ii*) ensuite à évaluer la contrainte sur le budget. Cette contrainte permet d'éliminer les instances i_2 et i_4. (*iii*) Finalement, il reste à choisir l'instance ayant le temps d'exécution le plus faible. L'instance de départ i_d est alors identifiée $i_d = i_6$. Nous soulignons que la normalisation des valeurs du temps d'exécution de chaque instance n'est pas utile dans ce cas car la décision se fait sur un critère, le temps d'exécution.

4.3.1.2 Protocole et algorithmes d'exécution

Dans l'écosystème de DyCoSe, le membre demandant l'instanciation d'un GBP peut être un super-pair ou un pair du réseau virtuel. Nous avons décrit les super-pairs dans la section 3.3.3.1, comme nœuds dédiés à la gestion du réseau virtuel mais qui participent également à la publication et la consommation des services.

Algorithme 1 : Délégation du processus d'instanciation

```
   Entrée  : Demande d'instanciation d'un GBP G sur un pair N_i
1  Début
2  |  si l'utilisateur veut privilégier la communauté PC_X alors
   |     /* Instanciation locale                                  */
3  |     Envoyer la demande d'instanciation au super-pair SP_X
4  |  sinon
   |     /* Instanciation globale                                 */
5  |     vSP = SPselection(strategyID)
6  |     Envoyer la demande d'instanciation de G au super-pair vSP
7  Fin
```

De part l'organisation du réseau virtuel de DyCoSe, un super-pair a une visibilité plus large sur le réseau qu'un pair de sa communauté. Nous supposons dans la suite que le processus d'instanciation s'exécute sur un super-pair. Si la demande d'instanciation provient d'un pair, un super-pair est choisi pour exécuter l'instanciation selon l'algorithme de délégation, Algorithme 1. En effet, nous sommes dans le cadre d'une solution globale au problème d'instanciation, ce qui sous-entend que le pair en question ne veut pas favoriser une communauté particulière mais veut profiter de l'ensemble de l'offre de services de l'écosystème.

L'algorithme de délégation, Algorithme 1, prend en compte le type de solution envisagée par le pair demandant l'instanciation. Si le pair demandant l'instanciation envisage une solution locale car il veut privilégier la communauté PC_X, l'instanciation est déléguée au super-pair SP_X en charge de la communauté PC_X. Si le pair envisage une solution globale, plusieurs stratégies sont possibles pour sélectionner le super-pair qui prendra en charge l'instanciation. Dans DyCoSe, nous étudions deux stratégies. La première est basée sur l'heuristique qui consiste à choisir le super-pair ayant la charge la plus faible. La deuxième est basée sur l'heuristique qui consiste à choisir aléatoirement un super-pair parmi ceux dont la communauté fournit des implémentations au plus grand nombre de services abstraits du GBP à instancier. L'Algorithme 2 décrit la procédure de sélection d'un super-pair en vue d'une instanciation globale. L'algorithme de délégation s'exécute sur un super-pair choisi aléatoirement parmi les super-pairs du pair demandant l'instanciation. L'algorithme de sélection s'exécute sur le même super-pair.

Le processus d'instanciation globale se déroule selon un protocole présentant les étapes suivantes :

1. Réception d'une demande d'instanciation d'un GBP G.
2. Choix du super-pair en charge de l'instanciation de G.

Algorithme 2 : Sélection du super-pair en charge de l'instanciation

```
   Entrée  : strategyID
1  Début
2      si strategyID == 1 alors
3          Demander à chaque super-pair d'envoyer sa charge
4          Retourner vSP ayant la charge la plus faible
5      sinon si strategyID == 2 alors
6          Demander à chaque super-pair le nombre de services abstraits de G implémentés dans
           sa communauté
7          Retourner vSP dont la communauté fournit des implémentations au plus grand
           nombre de services abstraits de G
8  Fin
```

3. Le super-pair instanciant G consulte sa table d'état global pour déterminer les super-pairs en charge des communautés implémentant les services abstraits de G.
4. Le super-pair instanciant G envoie des messages aux super-pairs concernés, leur demandant des informations sur les propriétés non fonctionnelles de leurs services.
5. Réception des informations demandées.
6. Exécution de l'algorithme d'instanciation globale.

L'algorithme d'instanciation globale présenté dans l'Algorithme 3, correspond à la solution globale décrite précédemment au début de cette section.

4.3.2 Solution locale

Une solution locale repose sur le choix de privilégier une ou plusieurs communautés. Ayant les communautés privilégiées, nous pouvons décomposer le chemin d'exécution à instancier en deux types de séquences de services abstraits : des séquences de services abstraits implémentés localement par les membres des communautés privilégiées et des séquences de services abstraits implémentés "à distance" par les membres des autres communautés. La figure 4.10 illustre un exemple de ces séquences. Un cas particulier peut se présenter si tous les services abstraits du chemin d'exécution à instancier sont implémentés localement, c'est-à-dire dans la ou les communautés privilégiées. Dans ce cas, il est évident que le calcul d'une solution locale est un sous-problème du calcul d'une solution globale. Par suite l'algorithme 3 est utilisé pour calculer la solution.

Algorithme 3 : Instanciation Globale

Entrée : Informations Globales
1 **Début**
2 Générer l'ensemble des instances possibles Ω, $|\Omega| = m$
3 **pour** $t = 1$ *à* m **faire**
4 **pour chaque** p^p *parmi les* q *contraintes de l'utilisateur* **faire**
5 Calculer $c^p(i_t)$
6 Générer la matrice des coûts $MC_{q,m}$
7 **pour chaque** *contrainte de type inférieur (supérieur) à un seuil* **faire**
8 Ecarter les instances ne respectant pas cette contrainte
9 **pour chaque** p^p *adressée par les contraintes* $\in CU$ **faire**
10 **pour** $t = 1$ *à* m' **faire**
11 Calculer le score $\overline{c^p(i_t)}$
12 Générer la matrice des scores $\overline{MC_{q,m'}}$
13 **pour** $t = 1$ *à* m' **faire**
14 Calculer le score final $\overline{C(i_t)}$ selon l'équation 4.17
15 **Fin**
Sortie : L'instance i_d ayant le meilleur score

Nous identifions deux stratégies pour déterminer une solution locale au problème d'instanciation. Une *première stratégie* consiste à rassembler d'abord les informations des services membres correspondants aux services abstraits implémentés à distance sur un super-pair et à effectuer ensuite un calcul centralisé déterminant l'instance de départ. La détermination d'une solution locale au problème d'instanciation selon cette première stratégie suit les étapes suivantes :

1. Identifier les services abstraits implémentés localement.
2. Récupérer les propriétés non fonctionnelles des services implémentant le reste des services abstraits.
3. Générer l'ensemble des instances possibles en se basant sur les propriétés des services membres des communautés privilégiées et celles récupérées à l'étape précédente.
4. Calculer une solution globale déterminant les instances valides parmi celles identifiées à l'étape précédente.

Cette stratégie reprend la technique d'instanciation globale décrite dans la section précédente en privilégiant certains services, réduisant ainsi l'espace de recherche. Nous ne la développons pas plus loin.

Une *deuxième stratégie* consiste à calculer d'abord les instances des séquences implémentées localement et à les compléter ensuite par des services membres

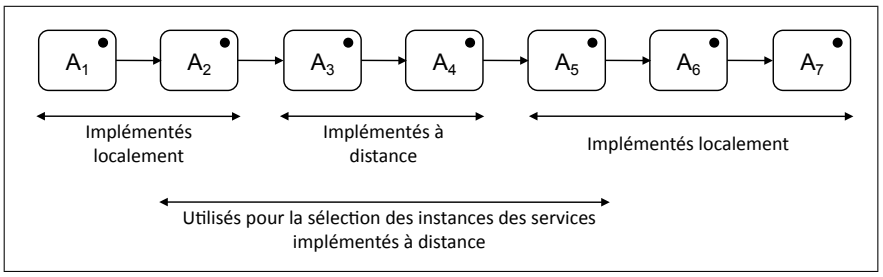


Figure 4.10: Illustration d'un exemple de séquences identifiées pour le calcul d'une solution locale

d'autres communautés en se basant sur une heuristique de distribution du calcul. La détermination d'une solution locale au problème d'instanciation selon cette deuxième stratégie suit les étapes suivantes :

1. Identifier les séquences des services abstraits implémentés localement et sélectionner leurs meilleures implémentations locales.
2. Identifier les communautés susceptibles de compléter l'instance générée localement.
3. Déléguer aux communautés identifiées à l'étape précédente, le calcul complétant l'instance.
4. Choisir une instance parmi celles complétées par délégation.

4.3.2.1 Exemple d'une solution locale

Nous reprenons dans la figure 4.11, l'exemple du réseau virtuel décrit dans le chapitre 3 par la figure 3.24. Le GBP à instancier sur le super-pair SN1 reprend le GBP de la figure 3.19. Il consiste en une composition séquentielle de trois services *Positionnement*, *Recherche d'utilitaires* et *Recherche d'hôtels*. L'utilisateur veut privilégier la communauté de SN1 et minimiser le temps d'exécution pour un budget ne dépassant pas la somme de 1.3 euros. La table 4.5 donne les valeurs du prix d'exécution et celle du temps d'exécution des services et des liens entre eux.

D'après la figure 4.11, les services abstraits implémentés localement dans la communauté de SN1 sont *Positionnement* et *Recherche d'hôtels*. SN1 retient pos1 comme implémentation de *Positionnement*, pos1 ayant le temps d'exécution le plus court. SN1 retient hôtel1 comme implémentation de *Recherche d'hôtels*. Pour sélectionner la meilleure implémentation du service abstrait *Recherche d'utilitaires*

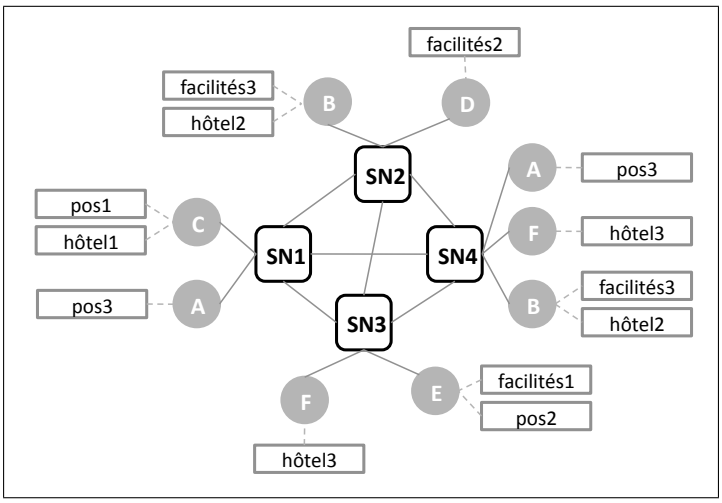


Figure 4.11: Exemple d'un réseau virtuel basé sur la distance physique séparant les nœuds

à distance, SN1 consulte sa copie de la table d'état global et envoie aux super-pairs fournissant une implémentation de *Recherche d'utilitaires* les parties locales de l'instance. Chaque super-pair complète l'instance reçue par une implémentation de *Recherche d'utilitaires* fournie dans sa communauté tel que l'instance résultante possède le temps d'exécution le plus court et respecte le budget indiqué.

Pour des raisons de simplification du calcul nous supposons que la taille des données échangées est égale 1Mo. Le calcul accompli par SN2 est le suivant : le prix d'exécution de l'instance <pos1, facilités2, hôtel1> $= 0.1 + 0.03 + 0.8 + 0.09 + 0.5 = 1.52$ euros. Le temps d'exécution de l'instance <pos1, facilités2, hôtel1> $= 0.8 + 1/20 + 1.3 + 1/20 + 2 = 4.2$ s. Toujours sur SN2, le prix d'exécution de l'instance <pos1, facilités3, hôtel1> $= 0.1 + 0.15 + 0.3 + 0.05 + 0.5 = 1.1$ euros et son temps d'exécution est égale à $0.8 + 1/50 + 1 + 1/10 + 2 = 3.92$ s. Le super-pair SN3 calcule le prix d'exécution de l'instance <pos1, facilités1, hôtel1> $= 0.1 + 0.05 + 0.5 + 0.05 + 0.5 = 1.2$ euros et son temps d'exécution égale à $0.8 + 1/10 + 2 + 1/10 + 2 = 5$ s. Le super-pair SN4 effectue le même calcul que SN2 pour l'instance <pos1, facilités3, hôtel1>.

SN3 et SN4 renvoient à SN1 les instances (<pos1, facilités1, hôtel1>, 1.2 euros, 5 s) et (<pos1, facilités3, hôtel1>, 1.1 euros, 3.92 s) respectivement, tandis que SN2 opère une sélection locale éliminant l'instance ne respectant pas la contrainte du budget (<pos1, facilités2, hôtel1>, 1.52 euros, 4.2 s) et envoie à SN1 l'instance (<pos1, facilités3, hôtel1>, 1.1 euros, 3.92 s).

Service	Prix euro	Temps s
pos1	0.1	0.8
pos2	0.3	1
pos3	0	1.2
facilités1	0.5	2
facilités2	0.8	1.3
facilités3	0.3	1
hôtel1	0.5	2
hôtel2	0	1
hôtel3	0.6	1.5

(a) Services

Lien	Prix euro	Débit Mo/s
(pos1, facilités1)	0.05	10
(pos1, facilités2)	0.03	20
(pos1, facilités3)	0.15	50
(pos2, facilités1)	0	∞
(pos2, facilités2)	0.05	33
(pos2, facilités3)	0.09	10
(pos3, facilités1)	0.05	10
(pos3, facilités2)	0.03	50
(pos3, facilités3)	0.15	20
(facilités1, hôtel1)	0.05	10
(facilités1, hôtel2)	0.2	20
(facilités1, hôtel3)	0.15	30
(facilités2, hôtel1)	0.09	20
(facilités2, hôtel2)	0.05	33
(facilités2, hôtel3)	0	∞
(facilités3, hôtel1)	0.05	10
(facilités3, hôtel2)	0	∞
(facilités3, hôtel3)	0.3	50

(b) Liens

Table 4.5: Exemple de propriétés non fonctionnelles du réseau de la figure 4.11

À la réception, SN1 élimine les instances ne respectant pas la contrainte du budget (aucune instance est à éliminer dans cet exemple) et choisit l'instance ayant le temps d'exécution le plus court parmi les instances restantes. L'instance (<pos1, facilités3, hôtel1>, 1.1 euros, 3.92 s) est retenue comme instance de départ par une solution locale privilégiant la communauté de SN1.

Nous soulignons que dans le réseau de la figure 4.11 le pair B fournissant le service facilités3 appartient à deux communautés, une gérée par SN2 et une autre gérée par SN4. Pour cette raison, SN1 reçoit deux fois la même instance (<pos1, facilités3, hôtel1>, 1.1 euros, 3.92 s).

4.3.2.2 Protocole et algorithmes d'exécution

L'algorithme de délégation, Algorithme 1, permet de transmettre la demande d'instanciation au super-pair choisi par l'utilisateur. L'utilisateur peut décider de privilégier une ou plusieurs communautés. Il peut décider en plus, de sélectionner le super-pair responsable de l'instanciation. Si l'utilisateur choisit plusieurs communautés n'indiquant pas de préférences concernant le super-pair responsable de l'instanciation, ce dernier est sélectionné selon les heuristiques de l'algorithme de

sélection présenté précédemment pour une solution globale dans l'algorithme 2.

Le super-pair responsable de l'instanciation entreprend le processus d'instanciation par la décomposition du chemin d'exécution g en deux types de séquences de services abstraits : des séquences de services abstraits implémentés localement par les membres des communautés privilégiées, désignés pas $sl_i(g)$ et des séquences de services abstraits implémentés "à distance" par les membres des autres communautés, désignés par $sr_i(g)$.

Le processus d'instanciation locale se déroule selon un protocole présentant les étapes suivantes :

1. Un super-pair SP_X reçoit la demande d'instanciation de g avec une liste de r communautés à privilégier $PCP = \{PC_j\}_{j=1..r.}$
2. SP_X consulte sa copie de la table d'état global pour déterminer les ensembles $SL(g) = \{sl_i(g)\}_{i\in\mathbb{N}}$ et $SR(g) = \{sr_i(g)\}_{i\in\mathbb{N}}$.
3. SP_X identifie l'ensemble des r' communautés distantes fournissant des implémentations au premier service abstrait de chaque $sr_i(g) \in SR(g)$. Cet ensemble est désigné par $PCR = \left\{PC_{j'}\right\}_{j'=1..r'}$. Les super-pairs de ces communautés seront sollicités plus tard pour compléter l'instance.
4. SP_X demande à ses pairs concernés les propriétés non fonctionnelles de leurs services.
5. SP_X reçoit les informations demandées et procède au calcul des instances de chaque $sl_i(g)$.
6. SP_X retient une instance pour chaque $sl_i(g)$ et envoie ces instances à chaque $SP_{PC_{r'}}$ tel que $PC_{r'} \in PCR$, en lui demandant de compléter l'instanciation.
7. Chaque $SP_{PC_{r'}}$ tel que $PC_{r'} \in PCR$, reçoit la demande de SP_X et procède à une instanciation locale des séquences $sr_i(g)$ auxquelles sa communauté fournit au moins une implémentation du premier service abstrait. Chaque $SP_{PC_{r'}}$ tient compte des services membres retenus par SP_X encadrant chaque séquence comme indiqué par la figure 4.10.
8. Une fois le calcul des instances distantes $sr_i(g)$ est terminé, SP_X reçoit les instances complétées à distance.
9. SP_X sélectionne l'instance de départ i_d.

L'algorithme 4 décrit les étapes de calcul d'une solution locale. Il consiste à décomposer la séquence de services abstraits à instancier localement en deux ensembles de séquences comme décrit auparavant. L'algorithme propose une procédure

Algorithme 4 : Instanciation Locale

```
   Entrée  : Un chemin d'exécution g ou une séquence de services abstraits à instancier
 1 Début
 2   Décomposer g en deux ensembles de séquences SL(g) et SR(g)
 3   pour chaque sl_i(g) ∈ SL(g) faire
 4     Calculer une instance de sl_i(g)
 5   si |SR(g)| ≠ ∅ alors
 6     pour chaque sr_i(g) ∈ SR(g) faire
         /* appel récursif                                    */
 7       Calculer une instance locale de sr_i(g)
 8   sinon
 9     return SL(g)
10 Fin
```

récursive qui consiste d'abord à calculer les instances des "séquences locales" et ensuite à décomposer chaque "séquence distante" en deux ensembles de séquences et ainsi de suite. Cet algorithme ne détaille pas la procédure de calcul des instances des séquences locales (ligne 4). Nous proposons de calculer le coût réel de chaque instance possible, comme décrit dans l'exemple de la section 4.3.2.1, car dans une instanciation locale l'espace de recherche est réduit par le choix des communautés.

4.4 Discussion

La démarche de composition d'applications étudiée dans DyCoSe permet de repousser de le choix des services composants l'application jusqu'au moment de l'exécution. Elle vérifie un ensemble de contraintes sur les propriétés non fonctionnelles des services. Cette démarche garantit le choix d'une instance de départ respectant ces contraintes dans la limite des services disponibles dans l'écosystème. Nous avons décrit dans ce chapitre, la représentation d'une application composite par un graphe attribué. Ce graphe est déduit du GBP issu du raffinement fonctionnel de l'application composite. En se basant sur la représentation d'une application par un graphe attribué, nous avons modélisé la réalisation d'une application composite, désignée par instanciation, comme un problème d'optimisation de coûts visant à satisfaire un ensemble de contraintes.

Dans DyCoSe, le processus d'instanciation d'une application composite a lieu sur un super-pair. Ce dernier coordonne les échanges de messages permettant de récupérer les informations nécessaires à l'instanciation. De part l'organisation des entreprises membres de l'écosystème de DyCoSe en communautés, deux méthodes

pour déterminer l'instance de départ sont possibles : une méthode d'*instanciation globale* et une méthode d'*instanciation locale*. Nous rappelons que la méthode globale prend en compte tous les services disponibles dans l'écosystème de DyCoSe. Tandis que la méthode locale privilégie les services de certaines communautés choisies par l'utilisateur.

L'instanciation globale décrite en section 4.3.1, est basée sur une vue globale de l'écosystème à un instant donné. Elle consiste à récupérer les informations sur l'ensemble des services correspondant aux services abstraits du GBP en cours d'instanciation. La sélection de l'instance de départ est décrite dans l'algorithme 3. Elle est basée sur un calcul de coût pour chaque instance possible. Une implémentation directe de l'algorithme 3 peut s'avérer coûteuse en ressources et en temps, de part la complexité polynomiale de cet algorithme. En effet la complexité de l'algorithme 3 croît avec le nombre m' des instances possibles respectant les contraintes de type inférieur (supérieur) à un seuil. Quand le nombre de services disponibles fonctionnellement équivalents aux services abstraits du GBP et respectant les conformités d'entrée et de sortie résulte en un nombre m' grand, il est plus judicieux d'utiliser un algorithme d'approximation ou des heuristiques pour calculer une solution globale. Nous proposons en section 5.3.2, un algorithme génétique de la famille des algorithmes évolutionnaires (ou évolutionniste) permettant de sélectionner une instance de départ selon un processus d'instanciation global.

L'instanciation locale décrite en section 4.3.2, est basée sur le choix d'un utilisateur de privilégier une ou plusieurs communautés. Elle peut être envisagée comme une sous partie de l'instanciation globale si les communautés privilégiées proposent des implémentations à tous les services abstraits du GBP en cours d'instanciation. Dans ce cas, l'algorithme 3 est utilisé pour calculer l'instance de départ. Cependant, si les communautés privilégiées ne fournissent pas des implémentations à tous les services abstraits du GBP en cours d'instanciation, l'algorithme 4 est utilisé. Il consiste à décomposer le chemin d'exécution en cours d'instanciation en séquences de services abstraits implémentées localement et d'autres implémentées à distance. Le résultat du calcul des instances des séquences implémentées localement est utilisé pour calculer les instances des séquences implémentées à distance.

Le processus d'instanciation permet de réaliser la phase de déploiement d'une application composite en sélectionnant un ensemble de services composant l'instance de départ. Il est suivi par l'exécution de l'application composite accomplie par l'orchestration des services composant l'instance de départ. Nous étudions dans

le chapitre suivant une implémentation du processus d'instanciation. Nous vérifions sa réalisation dans un environnement dynamique où il est difficile de garantir la disponibilité des pairs fournisseurs.

Chapitre 5

Implémentation et simulation : le prototype de DyCoSe

Sommaire

5.1 Simulations et PeerSim 160

5.1.1 Aperçu sur les types de simulations 160

5.1.2 Librairie de simulation PeerSim 161

5.2 Description du prototype de DyCoSe 163

5.2.1 Composants du niveau infrastructure 165

5.2.2 Composants du niveau services 172

5.2.3 Base de connaissances 173

5.3 Analyse de l'instanciation initiale d'une application composite 173

5.3.1 Aperçu sur les algorithmes génétiques 174

5.3.2 Description de l'algorithme génétique implémenté 175

5.3.3 Expérimentation 178

5.4 Analyse de l'exécution d'une application composite . . . 190

5.4.1 Exécution d'une instance 190

5.4.2 Réussite de l'exécution d'une application composite 192

Dans ce chapitre nous décrivons un prototype implémentant l'environnement DyCoSe pour la conception et le développement d'applications composites. Après un bref aperçu des types de simulation, nous décrivons la librairie de simulation PeerSim sur laquelle repose l'implémentation présentée. Ensuite, nous présentons les différents composants du prototype de DyCoSe. Puis, nous analysons l'instanciation initiale d'une application composite. Finalement, nous étudions le taux de réussite de l'exécution d'une application composite.

5.1 Simulations et PeerSim

Dans cette section nous proposons, en premier lieu, une classification des types de simulations. En deuxième lieu, nous décrivons la libraire de simulation PeerSim.

5.1.1 Aperçu sur les types de simulations

Les simulations permettent d'étudier le comportement d'un système quand la mise en place d'un environnement "réel" d'expérimentation est onéreuse et nécessite un temps considérable. Plusieurs types de simulation sont abordées dans la littérature. Une simulation peut être :

- *Déterministe* ou *stochastique*. Une simulation déterministe est basée sur un ensemble d'algorithmes déterministes, qui produisent le même ensemble de résultats pour le même ensemble d'entrées. Généralement une simulation déterministe est modélisée par une machine abstraite d'état (automates). Une simulation stochastique est basée sur une répétition d'échantillonnages aléatoires. Elle consiste à choisir un domaine de définition pour les entrées, en sélectionner aléatoirement des valeurs, les fournir à un ou plusieurs algorithmes déterministes et agréger les résultats. Les simulations de type Monte Carlo sont des simulations stochastiques.
- *Statique* ou *dynamique*. Dans une simulation statique les relations entre composants sont formalisées par des équations mathématiques. Le déroulement de la simulation se résume par une recherche d'un équilibre donné. Ce type de simulations est généralement utilisé en physique. En revanche dans une simulation dynamique les relations changent au cours de l'exécution et en fonction des entrées.

- *Continue* ou *discrète*. Une simulation continue calcule périodiquement la solution d'une équation différentielle et utilise le résultat pour modifier l'état général de la simulation. Par exemple les simulateurs de vols ou les jeux de construction de cités et d'évolution de civilisations. Cependant une simulation discrète utilise des générateurs de valeurs aléatoires et une horloge pour rythmer les évènements. Elle a toujours une condition marquant la fin ou l'arrêt d'exécution.
- *Locale* ou *distribuée*. Un simulation locale est exécutée sur une machine, tandis qu'une simulation distribuée est exécutée sur plusieurs machines.

5.1.2 Librairie de simulation PeerSim

PeerSim est une librairie de simulation de réseaux pair-à-pair écrite en java sous licence GNU/LGPL pour le projet BISON [1] à l'université de Bologne. Le code source de PeerSim et des documents explicatifs sont disponibles à l'adresse `http://peersim.sourceforge.net/`. L'environnement est codé de façon modulaire augmentant ainsi son extensibilité. PeerSim permet de mettre en place deux types de simulations, à savoir une simulation par événements et une simulation par cycles. Le premier type propose un moteur de simulation qui prend en compte la couche transport d'un réseau. L'échange de messages entre les pairs est simulé en se basant sur les caractéristiques de la couche transport choisie. Le deuxième type propose un moteur de simulation qui ignore la couche transport et par conséquent ne supporte pas la concurrence entre les pairs. Il donne le contrôle séquentiellement à chaque pair du réseau une fois pendant chaque cycle de la simulation. Cette abstraction de la couche transport permet à PeerSim de simuler un réseau contenant jusqu'à un million de pairs. Plusieurs protocoles et extensions ont été développés pour PeerSim, notamment, le protocole de super-pairs SG-1 [95], le protocole OverSat [64,96], le protocole T-MAN [65] et l'extension P2PAM [3] qui permet de simuler la modélisation des architectures pair-à-pair et d'étendre les protocoles de routage de PeerSim pour les rendre génériques.

Les principaux composants d'une simulation écrite avec PeerSim sont :

- Les nœuds ou *pairs* sont des conteneurs de protocoles. Chaque pair fournit les interfaces d'accès à ses protocoles. Il possède un identifiant unique lors d'une exécution la simulation.

1. `http://www.cs.unibo.it/bison/deliv.shtml`

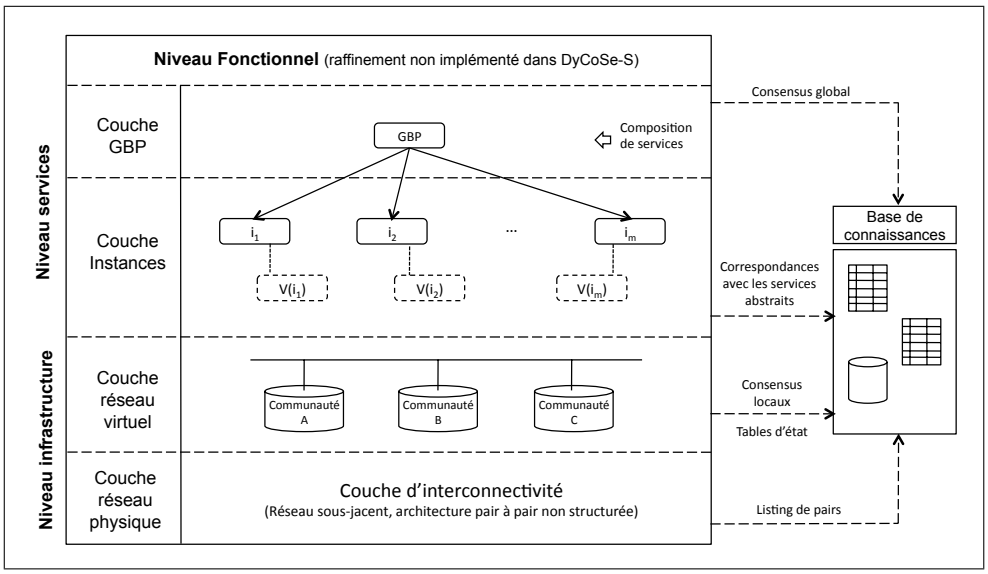


Figure 5.1: Architecture de composition implémentée par le prototype DyCoSe

- Les *protocoles* définissent une ou plusieurs opérations destinées à être exécutées par un pair.
- Les *contrôles* sont des composants programmés pour s'exécuter à des instants précis de la simulation. Un contrôle a une visibilité sur tous les pairs du réseau. Les contrôles servent en général à agir sur l'ensemble des pairs du réseau pour modifier leur comportement et pour récupérer des données relatives à l'état du réseau.
- Les *initiateurs* sont des contrôles spéciaux exécutés au lancement de la simulation. Ils permettent de générer les pairs, leurs propriétés et le réseau les reliant.

Nous avons choisi la librairie PeerSim pour les raisons suivantes : d'abord le passage à l'échelle qui se manifeste par le nombre élevés de pairs supportés par une simulation sous PeerSim (de l'ordre du million [66]). Ensuite la possibilité d'extension qui se manifeste par la personnalisation de l'environnement de simulation en ajoutant de nouveaux composants.

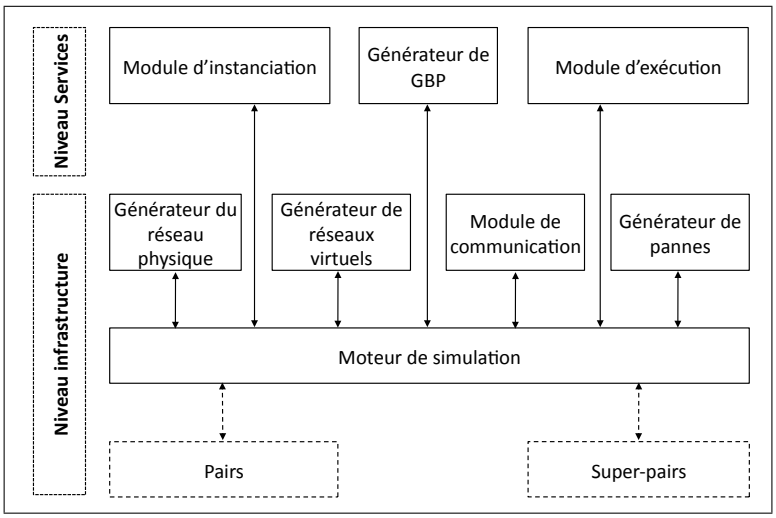


Figure 5.2: Composants du prototype de DyCoSe

5.2 Description du prototype de DyCoSe

Le prototype de DyCoSe implémente l'architecture de composition multi-niveaux décrite dans la section 3.3. La figure 5.1, rappelle cette architecture. Nous proposons à travers ce prototype, une implémentation du niveau infrastructure, du niveau services et de la base de connaissances. Les objectifs du prototype de DyCoSe sont les suivants : (*i*) simuler l'organisation du réseau virtuel décrit dans la section 3.3.3.1, (*ii*) simuler le processus d'instanciantion décrit dans le chapitre 4 et (*iii*) simuler l'exécution d'une instance dans un environnement dynamique.

La figure 5.2, recense les divers composants du prototype de DyCoSe. Elle les classe par niveau selon l'architecture décrite en figure 5.1. L'ensemble de ces composants permet de réaliser le niveau infrastructure et le niveau services de l'architecture de composition. La réalisation de la base de connaissance est décrite en section 5.2.3.

Avant de décrire les composants du prototype, nous proposons d'illustrer l'interaction entre les pairs dans une simulation en décrivant le cycle de vie d'un pair. Pendant la phase d'initialisation :

- Le pair est créé. Il possède un nombre de connexions avec d'autres pairs sélectionnés aléatoirement. Il peut posséder aussi une liste de services fournis.
- Le pair a une probabilité d'être choisi comme super-pair. S'il est choisi comme super-pair il possède une copie de la table d'état global et il génère sa table

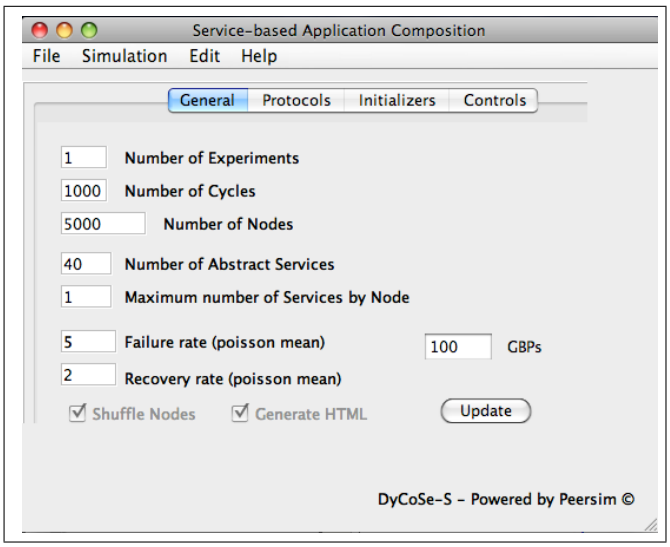


Figure 5.3: Interface d'entrée des paramètres (alternative au fichier texte de la figure 5.9)

d'état local.

- Le pair est placé dans une communauté, s'il n'est pas choisi comme un super-pair.

Pendant la phase d'exécution :

- Le pair change d'etat (en ligne, hors ligne) ou de propriétés (services fournis) en subissant les actions d'un ou de plusieurs composants implémentant l'interface "Control" de PeerSim.
- Le pair, quand il a le contrôle, exécute les composants implémentant l'interface PeerSim "Protocol". Si sa file d'attente contient des services, il les exécute en décrementant d'un cycle le temps d'exécution restant pour chaque service.
- Le pair, s'il est choisi comme super-pair, possède dans sa file d'attente en plus des services, des GBP à instancier et des instances à exécuter. Pour chaque GBP à instancier, il exécute l'algorithme d'instanciation. Pour chaque instance à exécuter, il garde trace de l'avancement de l'exécution de chaque service. Il vérifie l'état du pair exécutant le service courant. Si le pair est hors ligne, il marque l'instance comme en panne.
- Le pair change d'etat (en ligne, hors ligne) ou de propriétés (services fournis) en subissant les actions d'un ou de plusieurs composants implémentant l'interface "Control" de PeerSim.

La figure 5.3 présente une vue de l'interface d'entrée des paramètres du prototype. Certains paramètres de PeerSim apparaissent sur la figure en même temps

```
1
2  public class physicalInit implements Control {
3           ...
4      private void initPeersProperties() {
5          // pour chaque pair
6          // initialiser sa file d'attente
7          // Choisir un nombre de services à implémenter
8          // Choisir les services et initialiser leurs propriétés
9      }
10     private void generateCostMatrix(boolean bi) {
11         ...
12     }
13     public boolean execute() {
14         initPeersProperties();
15         generateCostMatrix(true);
16         return false;
17     }
18 }
```

Figure 5.4: Extrait du générateur de réseaux physiques

que le nombre de services abstraits de l'écosystème et le nombre maximal de services implémentés par un pair. Une deuxième méthode pour préciser les paramètres d'entrée est le fichier de paramètres PeerSim. Nous en présentons un exemple détaillé et commenté dans la figure 5.9.

5.2.1 Composants du niveau infrastructure

Les composants réalisant le niveau infrastructure sont les suivants : le générateur du réseau physique, le générateur de réseaux virtuels, le générateur de pannes, le module de communication et le moteur de simulation. Dans la suite nous décrivons les implémentations de chacun.

5.2.1.1 Générateur du réseau physique

Le générateur du réseau physique est implémenté comme un initiateur PeerSim. La classe GeneralNode modélise les pairs dans PeerSim. Le prototype de DyCoSe propose une spécialisation de cette classe en ajoutant les éléments nécessaires à l'étude des applications composites. Parmi ces éléments figurent les services fournis par un pair. L'implémentation actuelle adopte une abstraction de l'aspect fonctionnel de la couche services pour étudier l'aspect non fonctionnel. Les services membres, tout comme les services abstraits sont modélisés par des objets en mémoire. Ils sont

décrits par des classes possédant comme attributs les propriétés non fonctionnelles définies par le consensus global. Quand un pair est créé, il possède une probabilité de fournir un nombre de services. Dans l'implémentation actuelle, cette probabilité prend une valeur pseudo-aléatoire.

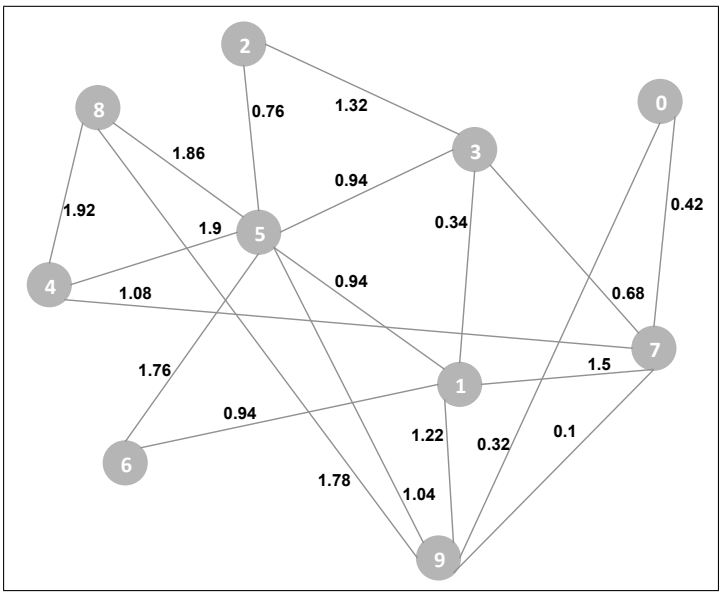
Des composants de type initiateur sont fournis dans PeerSim pour générer les connexions entre les pairs. Nous avons choisi le composant WireKOut qui génère des connexions entre chaque pair et un ensemble de pairs sélectionnés pseudo-aléatoirement. Le prototype de DyCoSe rajoute une couche par dessus WireKOut introduisant la notion de coût de connexion. En effet, nous avons implémenté un composant qui génère, pseudo-aléatoirement, une matrice de coûts des connexions entre les pairs. Ce coût peut représenter, par exemple, le prix d'exploitation du lien entre deux pairs.

La figure 5.5a illustre un extrait d'un réseau physique créé par le composant générateur du réseau physique avec les paramètres suivants : taille du réseau = 10 pairs, initiateur PeerSim choisi : WireKout, en mode bidrirectionnel avec un nombre maximal de deux liens sortant d'un pair [2]. Nous fournissons en Annexe E.1, une partie du code annoté des classes décrivant les pairs créés par ce composant. La classe "physicalInit" décrite dans la figure 5.4 est instanciée après WireKout dans la génération du réseau physique. Elle est responsable de l'initialisation du contenu des pairs notamment les services.

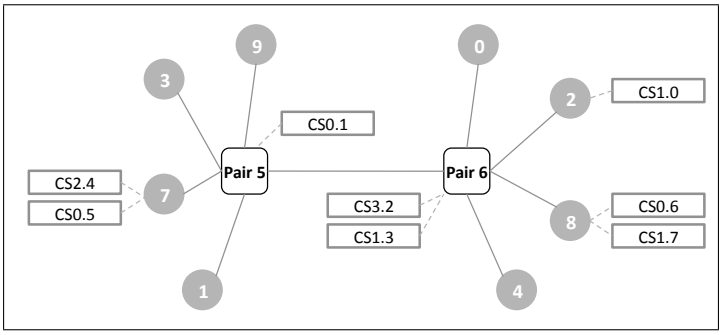
5.2.1.2 Générateur de réseaux virtuels

Le générateur de réseaux virtuels est un composant de type contrôle qui génère un réseau virtuel où les pairs sont distribués uniformément en communautés. Ce composant prend en entrée le réseau physique et un paramètre indiquant le nombre de super-pairs. Il génère en sortie un réseau virtuel comme décrit dans la section 3.3.3.1. Les tables d'état global et d'état local sont implémentées par des structures en mémoire contenant les informations présentées dans la section 3.3.3.1. Un résultat des tables générées est fourni en annexe E.4. Une variante de ce générateur prend en entrée une propriété d'un pair, modélisée comme attribut de l'objet correspondant et regroupe les pairs ayant des valeurs identiques pour cette propriété dans une même communauté. Pour chaque communauté un pair est sélectionné pseudo-aléatoirement comme super-pair. Nous soulignons que durant une simulation donnée, le moteur de

2. Un extrait du fichier sortie décrivant un réseau physique est présenté en annexe E.4.



(a) Réseau physique (par le composant WireKOut)



(b) Réseau virtuel illustrant les communautés et les services fournis

Figure 5.5: Réseaux générés par simulation (cf. annexe E.4)

simulation peut faire appel au générateur de réseaux virtuel à plusieurs reprises, modifiant ainsi la structure des communautés. La figure 5.6 présente un extrait de la classe implémentant le générateur de réseaux virtuel. La figure 5.5b décrit un réseau virtuel généré selon la première stratégie à partir du réseau physique de la figure 5.5a.

5.2.1.3 Générateur de pannes

Le générateur de pannes est un composant de type contrôle. Il s'exécute régulièrement lors d'une simulation pour décider de l'état de présence des pairs.

```
1
2  public class overlayInit implements Control {
3           ...
4     private void createCommunitiesNEW(double vDensite) {
5          // sélectionner les super-pairs
6          // initialiser leurs files d'attente
7                  ...
8              EALparam.superPeersHolder[i].isSuperPeer = true;
9              EALparam.superPeersHolder[i].GBPQueue = new ArrayList();
10             EALparam.superPeersHolder[i].InstancesQueue = new ArrayList();
11         // Remplir sa communauté aléatoirement
12     }
13     public boolean execute() {
14         createCommunitiesNEW(10);
15         return false;
16     }
17 }
```

Figure 5.6: Extrait du générateur de réseaux virtuels

Naturellement, si un pair est sélectionné comme super-pair, il ne sera pas affecté par l'exécution de ce composant. Le générateur de pannes permet de choisir entre deux stratégies de pannes qui sont actuellement implémentées dans le prototype de DyCoSe. La première stratégie modélise une "mort subite" d'un pair. Elle implique une panne définitive du pair. À la création du réseau physique, chaque pair choisit une probabilité de panne dans une distribution donnée. Cette distribution décrit le taux de pannes dans le réseau. La deuxième stratégie modélise une panne temporaire. Le pair partant reviendra après un laps de temps choisi dans une autre distribution au moment de son départ. Cette autre distribution décrit le taux de récupération dans le réseau.

5.2.1.4 Moteur de simulation

Le moteur de simulation implémenté reprend le moteur de simulation par cycles de PeerSim. À chaque cycle tous les pairs ont le contrôle à tour de rôle. Si l'option *shuffle* est activée, l'ordre dans lequel les pairs ont le contrôle est modifié aléatoirement d'un cycle à l'autre. Avec le moteur de simulation par cycles de PeerSim, un pair exécute tous les protocoles (au sens PeerSim du terme) qu'il contient avant de donner la main à un autre. La figure 5.7, décrit le principe de cycles adopté dans l'implémentation de DyCoSe. Un jeton est partagé par les pairs. Quand un pair possède le jeton, il traite les messages de sa file d'attente. Le pair transmet le jeton

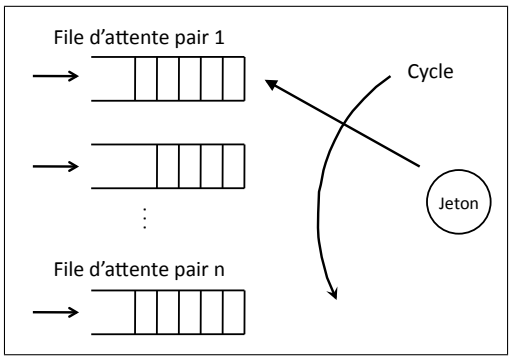


Figure 5.7: Communication par files d'attentes

à un autre pair une fois la durée de possession du jeton est écoulée, une condition quelconque est vérifiée ou quand sa file d'attente est vide. Ce principe nous permet d'éviter, si nécessaire, l'exécution de tous les protocoles contenus dans le pair.

Pour modéliser l'exécution des services nous avons implémenté sur chaque pair fournisseur une deuxième file d'attente contenant les services en cours d'exécution. Elle désignée par file d'exécution. La durée d'exécution des services est exprimée en nombre de cycles. Pour un service dans une file d'exécution, chaque cycle réduit la durée d'exécution restante d'une itération. Ainsi un pair recevant une demande d'invocation d'un service ayant un temps d'exécution égale à 4, mettra 4 cycles à l'exécuter. Si aucune panne n'est simulée, le consommateur de ce service recevra la réponse à son invocation au bout de 4 + C cycles où C désigne le nombre de cycles nécessaires à l'échange de messages entre les pairs.

Une simulation par cycles dans DyCoSe se déroule comme décrit dans la figure 5.8. Le premier cycle est précédé par une phase d'initialisation au cours de laquelle le générateur du réseau physique est exécuté. Ainsi les pairs et leurs services membres sont générés. Il est suivi par l'exécution du générateur de réseaux virtuels. Ainsi les communautés sont construites, les supers-pairs sont sélectionnés et les tables d'état sont calculées. Chaque cycle débute par l'exécution du module de communication suivi du module d'exécution adaptable qui à son tour est suivi du module d'instanciation. Ces modules sont exécutés sur chacun des pairs en ligne. Leur exécution change l'état du pair lui même et des autres pairs avec lesquels il interagit. À la fin de chaque cycle, le générateur de pannes et le générateur de GBP sont exécutés. Ils ont une visibilité sur tous les pairs du réseau. Par conséquent, leur exécution peut changer l'état de pairs qui ne sont pas nécessairement en interaction encore. Dans

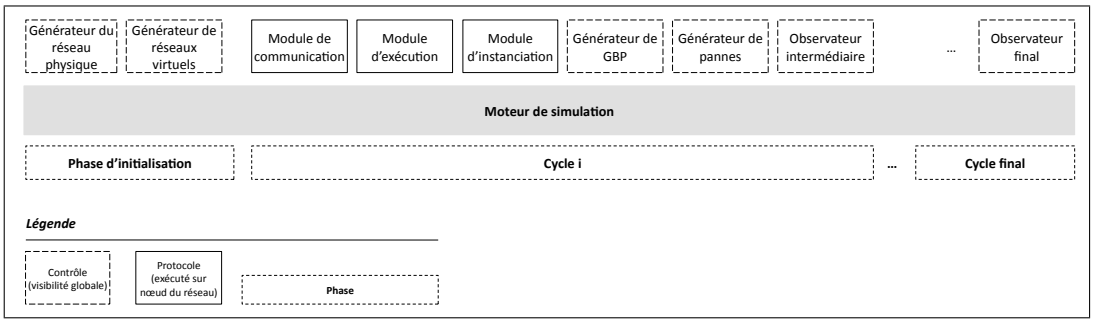


Figure 5.8: Exécution des composants et phases d'une simulation

la figure 5.8, les deux composants observateurs ont pour simple rôle de collecter des données à des fins d'analyse du résultat.

5.2.1.5 Module de communication

Le module de communication est implémenté comme un composant de type protocole. Il s'exécute régulièrement sur chaque pair (y compris les super-pairs). Ce module peut être implémenté de plusieurs façons selon le protocole d'échange de messages adopté et le niveau d'abstraction désiré. Dans PeerSim, la couche transport est simplifiée dans le cadre du moteur de simulation par cycles. Par suite, un pair communique directement avec ses voisins. Dans la version actuelle du prototype et pour des raisons de simplification nous adoptons la version de PeerSim sans modifications. Ainsi quand un pair envoi un message à un autre il le place dans la file d'attente correspondante. Nous constatons que durant un cycle donné, si un pair N_i envoie un message à un autre pair N_j qui est passé avant lui, N_j traitera le message au cours du cycle suivant ce qui semble raisonnable. Cependant, durant un cycle donné, si un pair N_i envoie un message à un autre pair N_j qui aura le contrôle après lui, N_j traitera le message au cours du même cycle. Cela peut s'avérer critique dans certaines études où la durée de l'échange des messages est significative pour l'analyse des résultats. Ce n'est pas le cas dans notre étude car nous sommes intéressé par l'analyse de la complétude de l'instanciation et celle de l'exécution d'une application composite.

Fichier de paramètres

```
1  random.seed 3419085325
2  # Entrée du générateur de valeurs pseudo-aléatoires de PeerSim
3  simulation.cycles 1000 # Nombre de Cycles d'une simulation
4  network.size 1000 # Nombre de pairs dans le réseau
5  network.node DyCoSe.objects.EALNode
6  # Classe modélisant un pair par défaut GeneralNode
7  control.shf Shuffle  # Ordre aléatoire des pairs
8
9  ##### protocoles
10 protocol.lnk IdleProtocol
11 # Protocole PeerSim permettant de stocker les liens entre les pairs
12 protocol.p1 DyCoSe.protocols.modExec
13 # Protocole DyCoSe représentant le module d'exécution adaptable
14 protocol.p1.linkable lnk
15 # Indique que le linkable de p1 est lnk
16 #(c'est-à-dire p1 s'exécute sur le réseau modélisé par lnk)
17 protocol.p2 Dycose.protocols.modInst
18 # Protocole DyCoSe représentant le module d'instanciation
19 protocol.p2.linkable p1 # Indique que le linkable de p2 est p1
20 order.protocol lnk, p1, p2 # Ordre d'exécution des protocoles
21
22 ##### initialiseurs
23 init.rnd WireKOut # Contôle PeerSim de génération de lien
24 init.rnd.protocol lnk # Les liens seront stockés dans lnk
25 init.rnd.k 20 # Nombre de liens sortant d'un pair
26 init.rnd.undir true # Liens bidirectionnels
27
28 init.startHtml startHtmlInit
29 # Contrôle DyCoSe générant l'entête du fichier HTML de sortie
30 init.peers DyCoSe.controls.intializers.physicalInit
31 # Contrôle DyCoSe représentant le générateur réseau physique
32 init.peersOverlay DyCoSe.controls.intializers.overlayInit
33 # Contrôle DyCoSe représentant le générateur réseau virtuel
34 init.communityDirectory communityDirectoryPrint
35 # Contrôle DyCoSe imprimant la table d'état global
36 init.localState localStatePrint
37 # Contrôle DyCoSe imprimant les tables d'état local de chaque SP
38 order.init startHtml rnd peers peersOverlay communityDirectory localState
39 # Ordre d'exécution des initialiseurs
40
41 ##### contrôles
42 control.etat DyCoSe.controls.peerStatusControl # Générateur de pannes
43 control.etat1 DyCoSe.controls.myObserver # Observateur, collecte des données
44 control.GBP DyCoSe.controls.addGBPcontrol # Générateur de GBP
45 control.endHtml endHtmlControl
46 # Contrôle DyCoSe générant la fin du fichier HTML de sortie
47 control.endHtml.FINAL true
48 # Indique que le contrôle d'exéctute suelement à la fin de la simulation
49 order.control  etat GBP etat1 final endHtml # Ordre d'exécution des protocoles
```

Figure 5.9: Fichier de paramètres PeerSim adopté par DyCoSe

5.2.2 Composants du niveau services

Les composants réalisant le niveau service sont les suivants : le module d'instanciation, le module d'exécution et le générateur de GBP. Dans la suite nous les décrivons.

5.2.2.1 Module d'instanciation

Le module d'instanciation est implémenté comme un composant de type protocole. Son rôle est de générer l'instance de départ en exécutant un processus d'instanciation. Il s'exécute sur un super-pair du réseau virtuel. Son exécution réalise le déploiement d'une application composite. La réussite de ce déploiement dépend évidemment de l'algorithme d'instanciation implémenté, mais elle est surtout conditionnée par les services disponibles sur le réseau.

L'objectif du module de d'instanciation est de réaliser les algorithmes d'instanciation globale et d'instantiation locale décrits dans le chapitre 4. Actuellement, un algorithme d'instanciation globale est implémenté. Nous le décrivons et nous en proposons une analyse en section 5.3.

5.2.2.2 Module d'exécution

Le module d'exécution est implémenté comme un composant de type protocole. Il joue un double rôle dans le prototype de DyCoSe. D'abord, il assure le rôle d'un moteur de composition de services. En effet, ce module prend en charge l'orchestration des services composant l'instance de départ. Ensuite, il assure le rôle d'un gestionnaire d'exécution pour une application composite. En effet, ce module prend en charge la surveillance de l'exécution d'une instance donnée. Il s'exécute sur super-pair du réseau virtuel. L'objectif du module d'exécution adaptable est d'abord, de coordonner l'exécution d'une instance sur les pairs fournissant les services correspondant, ensuite de surveiller l'exécution en vérifiant l'état des pairs. Nous étudions l'exécution d'une instance en section 5.4.

5.2.2.3 Générateur de GBP

Le générateur de GBP est implémenté comme un composant de type contrôle. Il s'exécute à plusieurs reprises pendant une simulation injectant ainsi sur les super-

pair un nombre de GBP à instancier. Les paramètres de ce composant sont le nombre de GBP à injecter et la taille d'un GBP exprimée en nombre de services abstraits.

5.2.3 Base de connaissances

La base de connaissances de l'écosystème n'est pas implémentée comme un composant. Elle est une reconstitution logique d'un ensemble d'information provenant des pairs, des services et de certains composants décrits ci-dessus.

Elle est implémentée dans le prototype de DyCoSe comme suit : (*i*) la partie fonctionnelle du consensus global est représentée par un liste de services abstraits disponibles que le pair consulte en phase d'initialisation pour décider des services qu'il veut implémenter. (*ii*) La partie non fonctionnelle du consensus global est représentée par un tableau de propriétés non fonctionnelles. Nous avons adopté pour cette étude deux propriétés, à savoir le temps et le prix. Dans le cas d'un service, le prix désigne le prix d'invocation du service et le temps désigne le temps d'exécution du service. Dans le cas d'un lien entre deux pairs, le prix désigne le prix de connexion entre les pairs et le temps désigne le débit de cette connexion. (*iii*) La correspondance entre un service membre et le service abstrait qu'il implémente est stockée dans le pair. L'implémentation actuelle est réalisée par un pointeur vers l'objet (les objets) représentant le(s) service abstrait implémenté(s). (*iv*) Les consensus locaux définissant les communautés sont pris en charge par le générateur du réseau virtuel.

5.3 Analyse de l'instanciation initiale d'une application composite

Le module d'instanciation proposé actuellement est une implémentation du processus d'instanciation globale décrit dans le chapitre 4. Il repose sur un problème d'optimisation basé sur une fonction coût similaire à celle décrite dans l'équation 4.9. Une variété de techniques permettent de trouver une solution exacte à ce problème d'optimisation. Cependant, le nombre de variables décisionnelles $X_i[k_i]$ augmente exponentiellement avec le nombre n de services abstraits du graphe d'un chemin d'exécution. Il augmente aussi avec le nombre n_i de services membres implémentant le service abstrait A_i du graphe d'un chemin d'exécution. Par conséquent, le calcul d'une solution exacte au problème d'instanciation devient coûteux en terme de temps d'exécution mais aussi en terme de ressources. En effet, avec l'augmentation

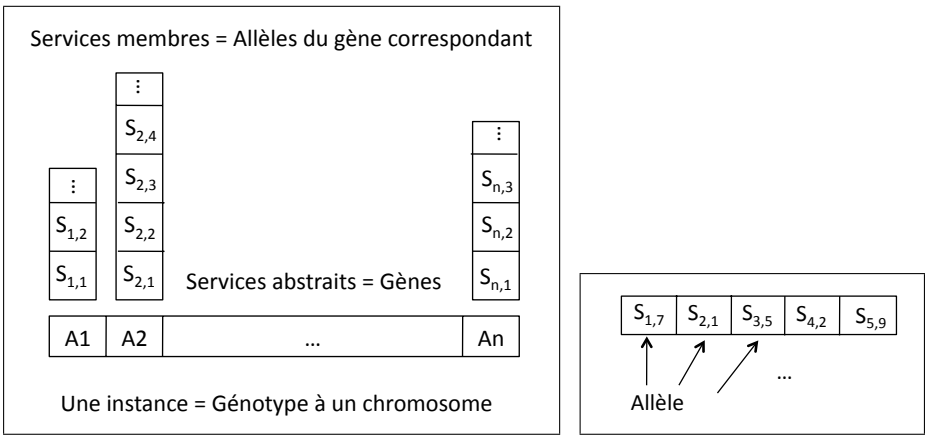
de n et de n_i le nombre d'instance possibles augmentent et le calcul du coût des instances nécessite un temps considérable et un grand espace mémoire. Il est plus adapté alors d'utiliser un algorithme d'approximation de la solution réelle afin de trouver une solution dans un temps acceptable. Plusieurs méthodes d'approximation d'un problème d'optimisation sont étudiées dans la littérature, nous citons la programmation linéaire [67], le "branch and bound" et ses applications en programmation par contraintes [75, 94] et les métaheuristiques comme le tabu search [52, 53] ou les algorithmes génétiques [34, 152].

5.3.1 Aperçu sur les algorithmes génétiques

Les algorithmes génétiques appartiennent la famille des algorithmes évolutionnaires (ou évolutionnistes). Ces derniers sont un ensemble de métaheuristiques servant à résoudre des problèmes d'optimisation et pouvant s'appliquer à différents cas. Quand le calcul d'une solution exacte à un problème d'optimisation donné est difficile à réaliser en un temps raisonnable, un algorithme génétique permet d'obtenir, en un temps "correct", une approximation de cette solution exacte. Les algorithmes génétiques sont basés sur le principe de sélection naturelle de Darwin. Ils appliquent ce principe à une *population* de solutions possibles au problème étudié dans le but d'améliorer les *individus* descendant de cette population.

Un algorithme génétique classique se déroule suit :

- Construire aléatoirement une population de départ et à calculer le score de chaque individu appartenant à cette population grâce à une fonction d'évaluation.
- Sélectionner un nombre donné d'individus. Plusieurs méthodes de sélection sont possibles. La sélection par rang choisit les individus ayant les meilleurs scores calculés par une fonction d'évaluation. La sélection probabiliste calcule pour chaque individu, sa probabilité d'être sélectionné proportionnellement à son adaptation au problème. Cette méthode est désignée par la roue de la fortune. Une variante de cette méthode est la sélection par tournoi qui consiste à choisir des paires d'individus par une sélection probabiliste puis préférer, dans chaque paire, l'individu ayant le meilleur score. La sélection uniforme choisit aléatoirement un individu appartenant à la population étudiée. Ainsi tous les individus ont la même probabilité d'être sélectionné indépendamment de leur



(a) Représentation du génotype d'un individu **(b)** Exemple d'un chromosome

Figure 5.10: Codage d'une instance

score.

- Puis, il consiste à croiser les individus retenus entre eux afin de générer une nouvelle population.
- Opérer des mutations sur certains individus de la nouvelle population. Ces mutations permettent généralement d'éviter les minima locaux.
- Remplacer la population de départ par celle générée récemment et réitère jusqu'à valider une condition donnée ou jusqu'à atteindre le nombre maximal de générations.

Il existe une variété d'algorithmes génétiques qui diffèrent par leur implémentation de la fonction d'évaluation, du processus de sélection, de la méthode de croisement, du phénomène de mutations ainsi que par les paramètres de départ, à savoir, la taille de la population, le nombre d'individus retenus et le nombre maximal de générations. La pertinence du résultat d'un algorithme génétique est étroitement liée à la modélisation du problème et au choix des techniques de sélection, de croisement et de mutations.

5.3.2 Description de l'algorithme génétique implémenté

Nous proposons de modéliser le processus d'instanciation globale par un algorithme génétique comme suit :

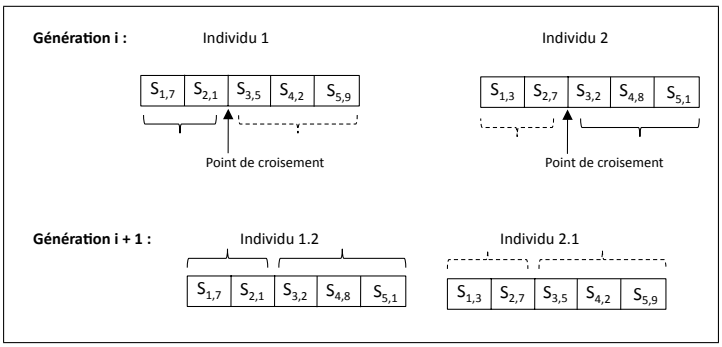


Figure 5.11: Méthode de croisement simple en un point

- Une instance est modélisée par un *individu* dont le *génotype* (ou génome) se résume à un *chromosome*.
- Un chromosome contient n *gènes* , où n désigne le nombre de services abstraits dans le chemin d'exécution en cours d'instanciation.
- L'allèle du gène i d'un chromosome donné représente le service membre S_{ik_i} choisi comme implémentation du service abstrait A_i dans l'instance modélisée par ce gène. La figure 5.10a décrit le chromosome constituant le génotype d'une instance. Elle illustre les services abstraits aux emplacements des gènes. Elle liste pour chaque gène, un exemple d'allèles représentant les services membres correspondant. La figure 5.10b décrit une instance composée des cinq services membres $\{S_{1\,7}, S_{2\,1}, S_{3\,5}, S_{4\,2}, S_{5\,9}\}$ à travers le chromosome correspondant.
- La fonction d'évaluation utilisée est une fonction calculant le coût d'une instance comme décrit dans la section 4.2.6. Nous étudions les contraintes de type trouver le minimum (maximum), les contraintes du type inférieur (supérieur) à un seuil étant plus simples. Dans la suite nous adoptons une fonction d'évaluation similaire à l'équation 4.9.
- Le croisement entre deux individus est une des étapes critiques de l'algorithme. Il consiste à croiser deux chromosomes pères codant les mêmes informations entre eux dans le but générer un chromosome fils contenant une partie des allèles de chaque chromosome père. Nous adoptons la méthode de croisement à un point décrite dans la figure 5.11. Elle consiste à fixer un point de croisement sur chaque chromosome père et à échanger les allèles des gènes de part et d'autre du point de croisement générant ainsi deux chromosomes fils. Dans

Algorithme 5 : Description de l'algorithme génétique implémenté

```
Entrée : taillePop, taillePop2, nbrIndivElite, nbrGen, tauxMut, n, n_i, prix d'exécution des
         services et prix des liens entre pairs
1  Début
     /* Construire aléatoirement une population initiale Pop de taille
        taillePop                                                        */
2    pour i=k à taillePop faire
3      pour i=1 à n faire
4        choisir aléatoirement un service S_{i n_i}
5      Ajouter l'individu généré à la population de départ
6    pour chaque individu dans Pop faire
7      Calculer le coût selon l'équation 4.9
8    Trier la population par ordre croissant de coût
9    répéter
10     Sélectionner nbrIndivElite de la population courante
11     Ajouter nbrIndivElite à la nouvelle génération
12     pour i=j à taillePop2 faire
13       Sélectionner aléatoirement deux individus parmi les élites
14       Effectuer un croisement simple médian entre les individus sélectionnés
15       Injecter des mutations à la descendance selon tauxMut
16       Ajouter l'individu généré à la a nouvelle génération
17     Trier la nouvelle génération par ordre croissant de coût
18     Remplacer Pop par la nouvelle génération
19   jusqu'à nbrGen
20 Fin
```

DyCoSe nous adoptons l'heuristique qui consiste à prendre le premier individu issu du croisement et à éliminer l'autre. Par exemple dans la figure 5.11, notre implémentation maintient l'individu 1.2. Il est cependant probable que l'individu 2.1 réapparaisse dans un croisement ultérieur, si les individus 1 et 2 sont croisés de nouveau dans l'ordre inverse.

- La méthode de sélection implémentée est la sélection par rang. En effet, à l'issue de chaque génération les individus de la nouvelle population sont triés par ordre croissant du coût qu'ils représentent. Un nombre d'individus élites est retenu pour "construire" la nouvelle génération.

- Les mutations permettent d'éviter les minima locaux en choisissant aléatoirement un ou plusieurs gènes et en remplaçant leurs allèles correspondant. Pratiquement, une mutation, si elle a lieu, remplace dans une instance un service membre par un autre choisi aléatoirement parmi les services disponibles.

L'algorithme 5 décrit en pseudo-code l'algorithme génétique implémenté dans le module d'instanciation et basé sur la modélisation décrite ci-dessus.

5.3.3 Expérimentation

Dans cette section nous décrivons les expériences menées pour analyser l'instanciation d'une application composite dans DyCoSe.

5.3.3.1 Evaluation de l'instanciation globale

Pour analyser l'algorithme génétique implémenté dans DyCoSe nous avons développé une extension du module d'instanciation permettant de calculer la solution exacte au problème d'instanciation globale d'un chemin d'exécution contenant jusqu'à six services abstraits. Nous regroupons les paramètres permettant d'analyser l'approximation de l'instanciation globale en trois catégories. La première catégorie correspond aux variables du problème d'optimisation, à savoir, le nombre de services abstraits dans le chemin d'exécution en cours d'instanciation et le nombre de services membres implémentant chaque service abstrait. La deuxième catégorie correspond aux paramètres de l'algorithme génétique, comme par exemple, la taille d'une population, le nombre de générations, le nombre d'individus élites à conserver d'une génération à l'autre et le taux de mutations. La troisième catégorie correspond aux variantes de l'algorithme génétique notamment les stratégies de croisement, de sélection et de mutation. Dans la suite nous décrivons un ensemble d'expériences visant à tester l'algorithme génétique proposé. La configuration matérielle utilisée est la suivante : un processeur Intel Core2 Duo 2.53Ghz, ayant 3Mo cache interne et une mémoire centrale de 4GB de type DDR3. La mémoire attribuée à la machine virtuelle (java max heap size) est de 2Go et le système d'exploitation est le Mac OS X 10.5.8. Pour des raisons de simplification, nous désignons ci-après l'algorithme génétique décrit par AlgoG.

Comparaison de AlgoG avec la solution exacte : temps de calcul La figure 5.12 illustre les résultats de l'expérience visant à comparer le temps de calcul de AlgoG et celui du calcul de la solution exacte. Elle est conduite avec les paramètres suivants :

– Nombre de services abstraits = 6.

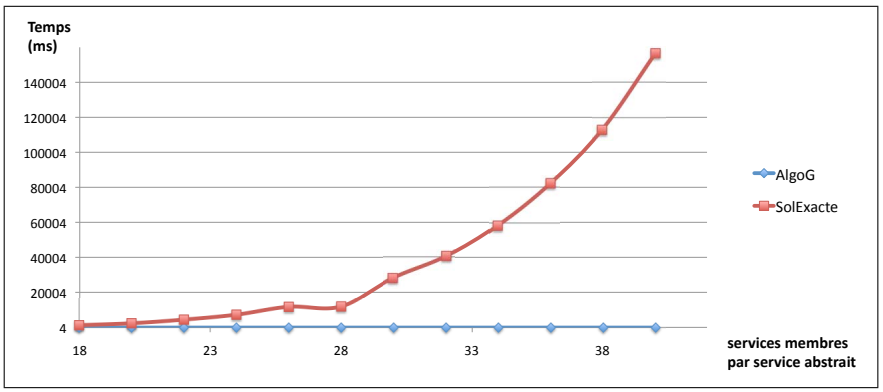


Figure 5.12: Comparaison avec la solution exacte : temps de calcul

- Nombre de services membres implémentant chaque service abstrait varie de 18 à 40 entraînant une variation du nombre d'instances possibles de 34012224 instances jusqu'à 4.10E+09 instances.
- Taille de la population = 30.
- Nombre d'individus élites gardés d'une génération à une autre = 4.

Le temps de calcul de AlgoG est de l'ordre des six millisecondes en moyenne tandis que le temps de calcul de la solution exacte commence à 1.3 secondes pour 18 services membres par service abstrait et croit rapidement jusqu'à 2.33 minutes (140000 millisecondes) pour 40 services membres par service abstrait. Cette expérience permet de vérifier la discussion de début de la section concernant le temps de calcul d'une solution exacte selon l'algorithme 3.

Comparaison de AlgoG avec la solution exacte : coût de l'instance retenue

La figure 5.13 illustre les résultats de l'expérience visant à comparer AlgoG et la solution exacte, la performance étant mesurée en terme de coût de l'instance retenue. L'expérience est conduite avec les paramètres suivants :

- Nombre de services abstraits = 6.
- Taille de la population = 30.
- Nombre d'individus élites gardés d'une génération à une autre = 4.
- Nombre de générations = 100.

Le coût de chaque instance est calculé selon l'équation 4.9 portant sur les propriétés non fonctionnelles suivantes : prix d'exécution d'un service et prix d'exploitation du lien entre les pairs fournissant deux services consécutifs dans l'instance étudiée. Nous avons considéré l'instanciation d'un chemin d'exécution contenant 6 services

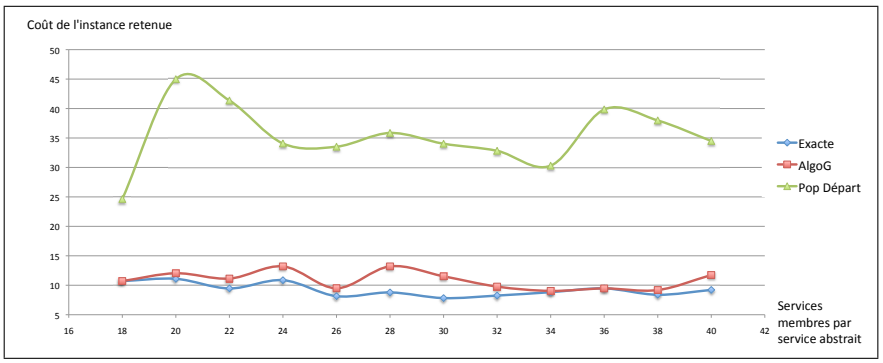


Figure 5.13: Comparaison avec la solution exacte et la génération de départ : coût de l'instance retenue

abstraits et nous avons modifié le nombre de services membres les implémentant à chaque instanciation. Ce nombre, relevé sur l'axe des abscisses, passe de 18 à la première expérience d'instanciation jusqu'à 40 à la dernière. La courbe de la solution exacte donne le coût de la meilleur instance. La courbe AlgoG donne le coût de l'instance retenue par l'algorithme génétique. Nous observons certains cas où AlgoG retrouve la solution exacte. Nous avons ajouter à titre informatif, les coûts de la meilleure instance appartenant à la population de départ pour illustrer l'évolution de l'algorithme génétique au bout de 100 générations seulement.

Comparaison entre deux stratégies de mutations La figure 5.14 illustre les résultats de l'expérience visant à comparer deux variantes de AlgoG. La première variante, mut1 correspond à un croisement avec mutation d'un gène à un taux relativement élevé (76%) mais n'affectant pas les individus élites. La deuxième variante, mut2 correspond à un croisement avec mutation de deux gènes à un taux inférieur à celui de mut1 (10%). Le but est de comparer l'efficacité de calcul des deux variantes en tenant compte d'un faible nombre de générations. Chaque variante est exécutée 50 fois. A chaque exécution mut1 et mut2 reçoivent en entrée la même population de départ. L'expérience est conduite avec les paramètres suivants :

- Nombre de services abstraits = 6.
- Nombre de services membres implémentant chaque service abstrait 100.
- Taille de la population = 30.
- Nombre d'individus élites gardés d'une génération à une autre = 4.
- Nombre de générations = 500.

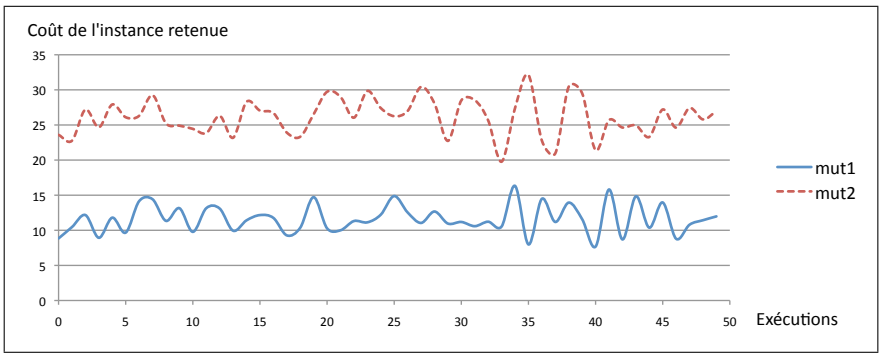


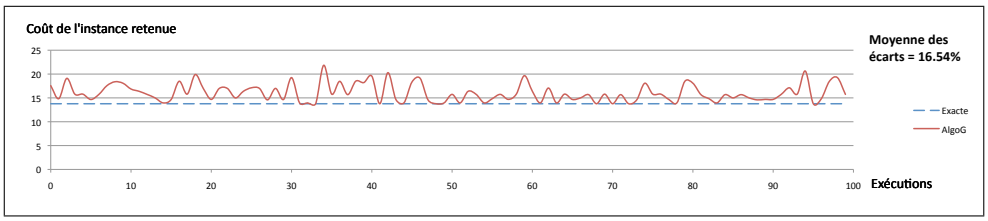
Figure 5.14: Comparaison entre deux stratégies de mutations

Le coût de chaque instance est calculé selon l'équation 4.9 portant sur les propriétés non fonctionnelles suivantes : prix d'exécution d'un service et prix d'exploitation du lien entre les pairs fournissant deux services consécutifs dans l'instance étudiée. La figure 5.14 indique que mut1 surpasse mut2. En effet, à chaque exécution le coût de l'instance retenue par mut1 est inférieur au coût de celle retenue par mut2. Le taux de mutations élevé de mut1 l'empêchera probablement de trouver la solution exacte, cependant il lui permet de tendre plus rapidement que mut2 vers la solution exacte en effectuant le même nombre de générations.

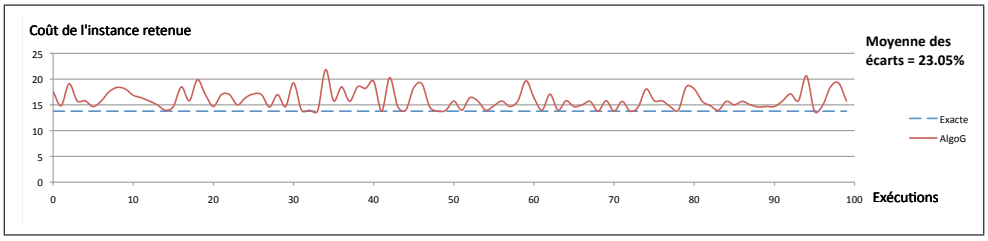
Écart de AlgoG par rapport à la solution exacte La figure 5.15 décrit un ensemble d'expériences visant à comparer AlgoG avec la solution exacte. La mesure de performance est le coût de l'instance retenue. Ces expériences sont conduites avec les paramètres suivants :

- Nombre de services abstraits = 6.
- Taille de la population = 40.
- Nombre d'individus élites gardés d'une génération à une autre = 4.
- Nombre de générations = 400.
- Stratégie de mutation : mutation d'un gène à un taux de 72% épargnant les individus élites.
- Variable : nombre de services membres implémentant un service abstrait.

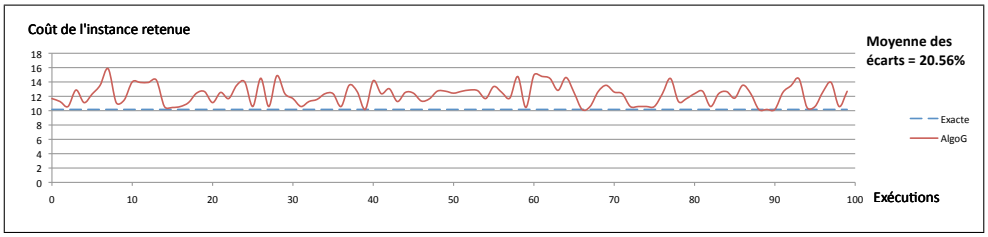
Le coût de chaque instance est calculé selon l'équation 4.9 portant sur les propriétés non fonctionnelles suivantes : prix d'exécution d'un service et prix d'exploitation du lien entre les pairs fournissant deux services consécutifs dans l'instance étudiée. Les trois expériences des figures 5.15a, 5.15b et 5.15c diffèrent par le nombre de services membres implémentant un service abstrait. Nous avons choisit respectivement 20,



(a) 6 services abstraits, 120 services membres, 6.4E7 instances possibles



(b) 6 services abstraits, 150 services membres, 2.44E8 instances possibles



(c) 6 services abstraits, 180 services membres, 7.29E8 instances possibles

Figure 5.15: Écart de AlgoG par rapport à la solution exacte

25 puis 30 services membres par service abstrait. Nous justifions le choix du faible nombre de services membres par la nécessité de calculer la solution exacte afin de la comparer avec AlgoG. Pour chaque configuration nous avons exécuté AlgoG 100 fois. Après chaque exécution de AlgoG, nous avons calculé l'écart entre le coût de l'instance sélectionnée par AlgoG et le coût de l'instance minimale trouvée par la solution exacte comme suit :

$$\text{écart relatif} = \frac{|\text{coût instance retenue} - \text{coût meilleure instance exacte}|}{\text{coût meilleure instance exacte}}$$

Sur chacune des figures 5.15a, 5.15b et 5.15c, nous avons noté la moyenne des écarts calculés. Nous observons que la moyenne de l'écart dans les trois expériences est de l'ordre de 20%. Nous observons aussi un faible nombre de solutions exactes trouvées par AlgoG. En effet, le taux de mutation élevé empêche AlgoG de trouver la solution exacte mais lui permet de tendre rapidement, à 20% près en moyenne, de la solution

exacte.

5.3.3.2 Réussite de l'instanciation d'une application composite

Nous rappelons que l'écosystème de DyCoSe est un environnement dynamique où les pairs fournisseurs de services peuvent partir à tout moment entraînant une modification des services disponibles. Par conséquence, la réussite du processus d'instanciation n'est pas garantie. Nous proposons d'étudier dans la suite, l'impact du taux de départ des pairs fournisseurs sur la réussite de ce processus. La figure 5.16 illustre les résultats des expériences conduites avec les paramètres suivants :

- Nombre de pairs = 5000.
- Nombre de services abstraits dans l'écosystème = 40.
- Nombre maximal de services fournis par un pair = 1.
- Nombre de GBP à instancier = 100.
- Variables : μ désignant le taux de départ des pairs et n désignant le nombre maximal de services abstraits dans un GBP.
- Modèle de panne adopté : la "mort subite", c'est-à-dire si un pair se déconnecte, il ne se reconnecte plus durant la simulation.
- Mesure de performance : nombre de GBP non instanciés.
- Durée de la simulation : 1000 cycles.

Nous avons mené trois séries d'expériences. Pour chaque série, le nombre maximal de services abstraits dans un GBP est fixé à une valeur n mais la moyenne de la distribution de poisson μ varie de 1 à 14. Dans la première série, n est désigné par n-petit. Sa valeur est fixée à 5 services abstraits parmi 40. Par conséquence, les GBP qui seront instanciés auront au maximum 5 services abstraits, modélisant ainsi des compositions à faible nombre de services. Dans la deuxième série, n est désigné par n-moyen. Sa valeur est fixée à 10. Dans la troisième série, n est désigné par n-grand. Sa valeur est fixée à 20.

Dans une série d'expériences, pour chaque valeur de μ, une simulation de 1000 cycles est lancée. Au cours de cette simulation, 100 GBP de taille maximale = n sont générés pseudo-aléatoirement, à raison d'un GBP tous les 6 cycles de sorte à distribuer les demandes d'instanciation uniformément sur la période de simulation. Pour chaque GBP, une demande d'instanciation est placée dans la file d'attente d'un super-pair choisi pseudo-aléatoirement. Quand le moteur de simulation donne le contrôle à un super-pair ayant un GBP à instancier, le super-pair exécute le module d'instanciation décrit ci-dessus. Nous rappelons que le module d'instanciation

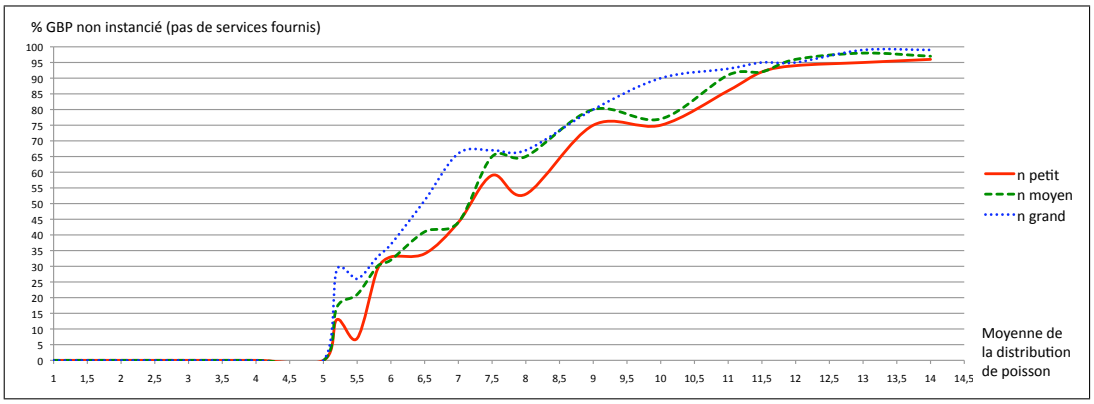


Figure 5.16: Taux d'échec de l'instanciation en fonction de la probabilité de départ des pairs

implémenté dans le prototype est basé sur une instanciation globale. Il propose une approximation de la solution exacte selon l'algorithme 5. La fonction de coût simulée consiste à minimiser le prix total d'exécution selon l'équation 4.9. Le super pair récupère les informations sur tous les services membres disponibles et implémentant les services abstraits du GBP. Il procède à l'instanciation. Si à cet instant, pour un service abstrait du GBP, aucun service membre n'est proposé, le super-pair marque le GBP comme non réalisable et incrémente un compteur partagé. Au bout de 1000 cycles, le moteur de simulation fournit en sortie le nombre de GBP non instancié.

Nous observons sur la figure 5.16, dans les trois séries d'expériences, que pour $\mu < 5$, le taux d'échec de l'instanciation est autour de 1%. Pour μ entre 5 et 12, le taux d'échec croit avec la valeur de μ. Pour $\mu > 12$, le taux d'échec se stabilise dépassant toujours 95%. La valeur de μ, paramètre du générateur de pannes, indique la distribution des probabilités de départ des pairs. Quand un pair est créé, le générateur du réseau physique lui attribut un nombre indiquant le cycle durant lequel il se déconnecte. Ce nombre est extrait d'une distribution de poisson de moyenne μ. Il est calculé en divisant le nombre de cycles par $poisson(\mu)$. La figure 5.17 décrit trois courbes de distribution de poisson. Nous observons que plus la valeur de μ augmente, plus la courbe s'aplatit en s'étalant sur l'axe des abscisses. Nous en déduisons que plus μ augmente, plus les pannes arriverons plutôt dans la simulation, réduisant ainsi le nombre de services disponibles pour les instanciations ultérieures. Nous vérifions sur la figure 5.16, que pour une faible valeur de μ, le taux d'échec est faible.

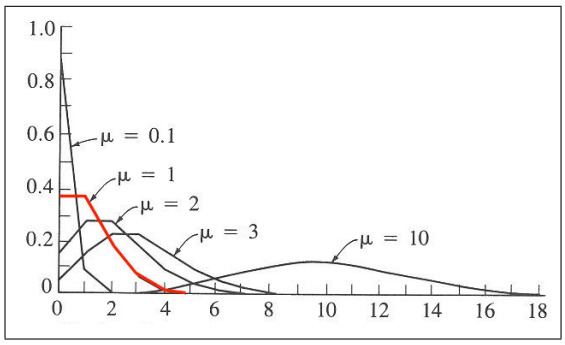


Figure 5.17: Exemples de distributions de poisson

Finalement, la figure 5.16 nous permet de vérifier l'impact du nombre de services abstraits d'un GBP sur le taux de réussite de l'instanciation de ce GBP. En effet, la courbe n-petit est toujours inférieure ou égale aux autres courbes impliquant que le taux d'échec des GBP à faible nombre de services abstraits est inférieur au taux d'échec des GBP contenant plus de services abstraits.

5.3.3.3 Impact des communautés sur l'instanciation locale

Afin d'étudier l'impact de l'organisation des pairs fournisseurs en communautés sur le processus d'instanciation, nous proposons de comparer, dans un écosystème donné, deux instanciations du même GBP en variant l'organisation du réseau virtuel entre les deux instanciations. Nous comparons deux mesures de performance. La première mesure désigne le coût des communications lors du processus d'instanciation. Elle est calculée en additionnant les coûts de communication entre le super-pair en charge de l'instanciation et les pairs sollicités de sa communauté. Mais aussi en ajoutant les coûts de communication entre les super-pairs, auxquels sont additionnés en plus, les coûts des communications entre les super-pairs distants et les pairs qu'ils ont sollicité. La deuxième mesure désigne le prix total d'exécution prévu pour l'instance retenue comme instance de départ. Ce prix total est calculé en additionnant les prix des services composant l'instance auxquels sont ajoutés les prix des transferts de données sur les liens entre pairs fournisseurs. Nous reprenons dans la figure 5.18, une variante du GBP "raffiné" [3] dans la section 3.3.1. Nous supposons que l'utilisateur instanciant ce GBP a deux objectifs : (i) minimiser le prix total

3. Nous avons décrit le raffinement fonctionnel d'une application composite permettant de générer un GBP dans le cadre de DyCoSe, en section 3.3.

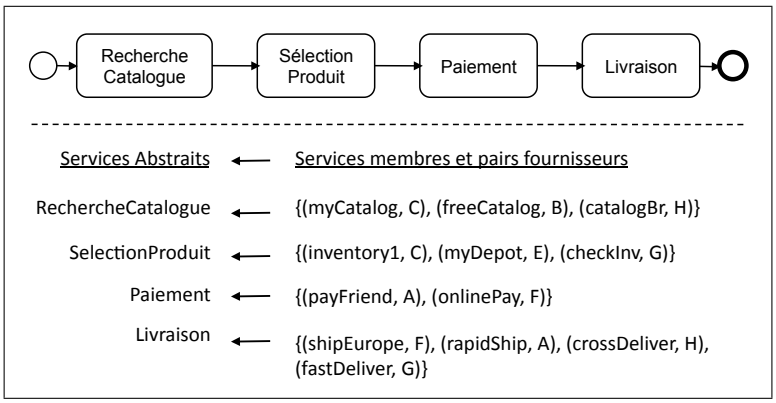
comme décrit dans l'équation 4.9 et (*ii*) privilégier les fournisseurs appartenant à la communauté gérée par SP3.

Le moteur de simulation et le module communications implémentés actuellement dans DyCoSe ignorent la couche transport, par suite ils ne permettent pas d'étudier le coût de communication pendant l'instanciation. Nous proposons dans la suite, deux simulations manuelles de l'instanciation du GBP de la figure 5.18a. La première simulation, décrite dans l'exemple 1, est basée sur le jeu de donnée de la figure 5.19 où le réseau virtuel est construit en se basant sur la distance entre pairs. La deuxième simulation, décrite dans l'exemple 2, est basée sur le jeu de donnée de la figure 5.20 où le réseau virtuel est construit en se basant sur les services abstraits implémentés. Pour des raisons de simplification nous considérons que les prix des liens entre pairs sont constants et identiques. Par suite nous pouvons les simplifier pendant le calcul d'optimisation. Nous considérons aussi que les chemins entre les pairs sont des liens bidirectionnels ayant des coûts constants (indépendant de la quantité de données échangées) et uniformes comme suit : une communication entre deux super-pairs coûte une unité de temps et une communication entre un pair et un super-pair coûte une demie unité de temps. Par exemple, une requête-réponse entre SP3 et SP1 coûte 1 unité tandis qu'une invocation-réponse entre SP3 et H coûte 0.5 unité.

Exemple 1 : Instanciation dans un réseau virtuel basé sur la distance entre pairs

Nous rappelons que l'utilisateur veut privilégier les fournisseurs appartenant à la communauté de SP3, alors il adopte une instanciation locale sur SP3. En se basant sur sa table d'état local illustrée dans la figure 5.19b, le super-pair SP3 détermine les services abstraits implémentés dans sa communauté. Dans cet exemple nous avons respectivement pour *RechercheCatalogue* : (myCatalog, C), (catalogBr, H) ; pour *SelectionProduit* : (inventory1, C), (myDepot, E) et pour *Livraison* : (crossDeliver, H). Les services abstraits *RechercheCatalogue*, *SelectionProduit* et *Livraison* sont "matchés" localement tandis que le service abstrait *Paiement* sera "matché" à distance.

Sur SP3, selon l'algorithme d'instanciation locale (Algorithme 4), le GBP est exprimé comme l'union d'un ensemble de services abstraits implémentés localement et d'un singleton représentant le service abstrait *Paiement* implémenté à distance. Pour calculer l'instance de la séquence implémentée localement, SP3 contacte les pairs appartenant à sa communauté et fournissant des implémentations aux services abstraits de ladite séquence, leur demandant le



(a) Exemple d'un GBP à instancier

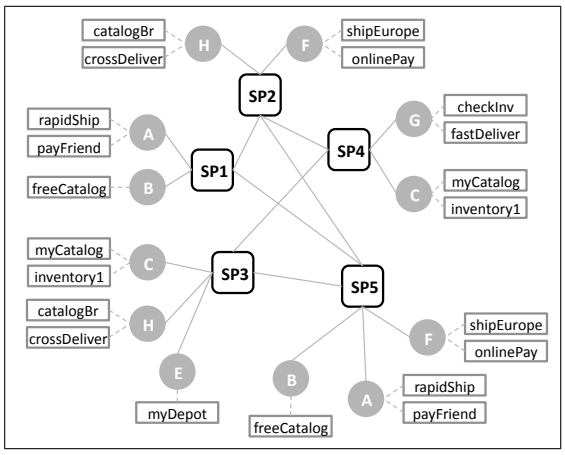
Service	Prix (euro)	Service	Prix (euro)
myCatalog	0.1	payFriend	0.5
freeCatalog	0.0	onlinePay	0.4
catalogBr	0.15	shipEurope	0.9
inventory1	0.2	rapidShip	0.8
myDepot	0.3	crossDeliver	1.2
checkInv	0.15	fastDeliver	1.0

(b) Exemples de services membres

Figure 5.18: Exemple d'un GBP et de services membres correspondant

prix d'exécution de leurs services. En se basant sur les valeurs d'exemple de la figure 5.18b les services suivants sont sélectionnés pour l'instance de départ : {(myCatalog, C), (inventory1, C), (crossDeliver, H)}. Pour calculer la partie de l'instance implémentée à distance, SP3 consulte sa table d'état global, décrite dans la figure 5.19b, pour trouver des supers-pairs implémentant le service abstrait *Paiement*. SP3 envoie des requêtes à SP1, à SP2 et à SP5. SP1 et SP5 répondent avec (payFriend, A, 0.5), tandis que la réponse de SP2 est (onlinePay, F, 0.4). SP3 sélectionne (onlinePay, F) comme implémentation de *Paiement*. Sur SP3, l'instance de départ calculée est i_d = {(myCatalog, C), (inventory1, C), (onlinePay, F), (crossDeliver, H)}. Nous soulignons que dans le cas d'une panne du service myDepot, due au départ du pair E par exemple, SP3 a la possibilité de le remplacer par inventory1.

Dans la suite nous calculons la première mesure de performance qui n'est autre que le coût des communications pendant le processus d'instanciation sur SP3. D'abord le coût des communications intra-communauté est généré par les requêtes-réponses entre SP3 et les pairs C, H et E. SP3 a envoyé une requête



(a) Exemple d'un réseau virtuel basé sur la distance entre pairs

Super-pair	*Services abstraits*
SP1	{*RechercheCatalogue*, *Paiement*, *Livraison*}
SP2	{*RechercheCatalogue*, *Paiement*, *Livraison*}
SP3	{*RechercheCatalogue*, *SelectionProduit*, *Livraison*}
SP4	{*RechercheCatalogue*, *SelectionProduit*, *Livraison*}
SP5	{*RechercheCatalogue*, *Paiement*, *Livraison*}

(a) Table d'état global

Pair		(Service, *Service Abstrait*)
P_{ID}	$St(P_i)$	
C	ON	{(myCatalog, *RechercheCatalogue*), (inventory1, *SelectionProduit*)}
H	ON	{(crossDeliver, *Livraison*), (catalogBr, *RechercheCatalogue*)}
E	ON	{(myDepot, *SelectionProduit*)}

(b) Table d'état local sur SP3

(b) Extraits de la base de connaissance

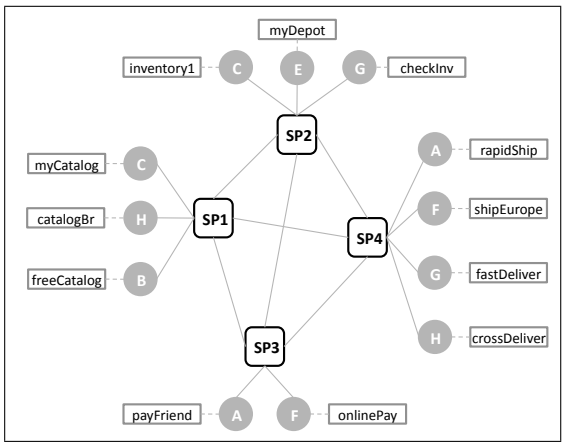
Figure 5.19: Exemple 1 : jeu de données pour une instanciation

à chacun de ces pairs leur demandant le prix d'exécution de leurs services, générant ainsi un coût de communication de 1.5 unités. Ensuite le coût des communications inter-communautés est généré par l'échange de messages entre SP3 et les super-pairs SP1, SP2 et SP5. Chacun des pairs distants SP1, SP2 et SP5 a communiqué avec un pair membre, le coût total de communications inter-communautés est ainsi égale à $3+0.5+0.5+0.5 = 4.5$ unités. Finalement, nous déduisons le coût des communications pendant le processus d'instanciation qui est égale à $1.5 + 4.5 = 6$.

La deuxième mesure de performance reflète le prix de l'instance retenue comme instance de départ. En se basant sur les prix d'exécution des services fournis dans la table 5.18b, le prix d'exécution prévu pour l'instance $i_d = \{$(myCatalog, C), (inventory1, C), (onlinePay, F), (crossDeliver, H)$\}$ est égale à 1.9 euros plus une constante représentant le prix de communications sur les liens entre les services pendant l'exécution de cette instance.

Exemple 2 : Instanciation dans un réseau virtuel basé sur les services abstraits implémentés

Nous reprenons le même GBP à instancier en privilégiant la communauté de SP3, l'utilisateur veut toujours minimiser le temps d'exécution cependant le réseau virtuel a changé. Il est basé sur les services abstraits comme décrit dans la figure 5.20a. SP3 interroge les pairs de sa communauté et choisit localement (onlinePay, F) comme implémentation du service abstrait *Paiement*.



(a) Exemple d'un réseau virtuel basé sur les services abstraits implémentés

Super-pair	*Services abstraits*
SP1	$\{RechercheCatalogue\}$
SP2	$\{SelectionProduit\}$
SP3	$\{Paiement\}$
SP4	$\{Livraison\}$

(a) Table d'état global

Pair		(Service, *Service Abstrait*)
P_{ID}	$St(P_i)$	
A	ON	{(payFriend, *Paiement*)}
F	ON	{(onlinePay, *Paiement*)}

(b) Table d'état local sur SP3

(b) Extraits de la base de connaissance

Figure 5.20: Exemple 2 : jeu de données pour une instanciation

En consultant la table d'état global, illustrée dans la figure 5.20b, SP3 contacte les autres super-pairs demandant leurs meilleures implémentations de *RechercheCatalogue*, de *SelectionProduit* et de *Livraison*. SP3 reçoit de SP1 : (freeCatalog, B), de SP2 : (checkInv, G) et de SP4 : (rapidShip, A). Sur SP3, l'instance de départ calculée est $i_d = \{$(freeCatalog, B), (checkInv, G), (onlinePay, F), (rapidShip, A)$\}$.

À la réception de la requête de SP3 les super-pairs SP1, SP2 et SP4 ont contacté tous les pairs de leur communauté. Le coût total des communications inter-communauté est égale à $3 + 1.5 + 1.5 + 2 = 8$ unités. SP3 a sollicité deux pairs de sa communauté générant ainsi un coût total de communications intra-communauté égale à 1 unité. La première mesure de performance a comme valeur 9 unités de temps. La deuxième mesure de performance indiquant le prix d'exécution de $i_d = \{$(freeCatalog, B), (checkInv, G), (onlinePay, F), (rapidShip, A)$\}$ a comme valeur 1.35 euros plus une constante représentant le prix de communications sur les liens entre les services pendant l'exécution de cette instance.

En comparaison avec une organisation du réseau virtuel basé sur la distance, une organisation basée sur les services abstraits implémentés génère plus de trafic durant le processus d'instanciation. Cette différence de performance est due à l'échange de messages entre les communautés. En effet, quand le réseau virtuel est basé sur les services abstraits implémentés, chaque service abstrait du GBP sera "matché" dans

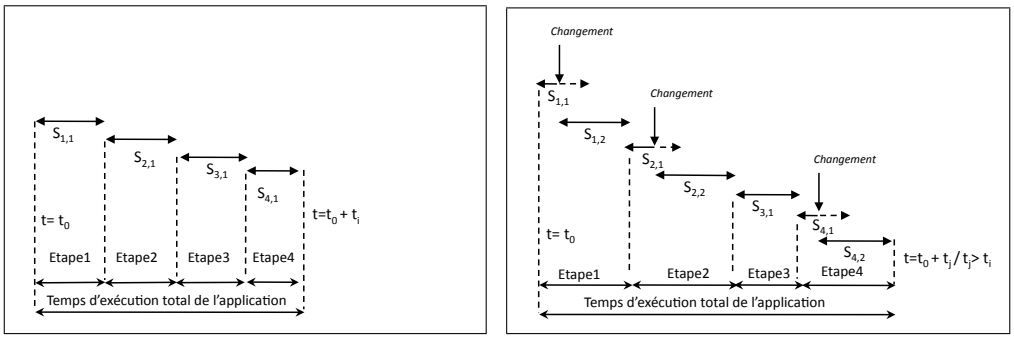
une communauté différente d'où un nombre élevé de messages inter-communautés. Quand le réseau virtuel est basé sur la distance, la communauté locale en charge de l'instanciation peut fournir des implémentations de plus d'un service abstrait, réduisant ainsi le nombre de messages inter-communautés nécessaires pour rechercher des implémentations à distance. Cependant, le prix total prévu pour l'instance de départ calculée dans un réseau virtuel basé sur les services abstraits implémentés est inférieur à celui d'une instance calculée dans un réseau virtuel basé sur la distance. En effet, le réseau virtuel étant basé sur les services abstraits implémentés, chaque super-pair fournira la meilleure implémentation du service abstrait représenté par sa communauté.

5.4 Analyse de l'exécution d'une application composite

Comme décrit en section 5.2.2.2, le module d'exécution permet de gérer l'exécution d'une application composite. Il est responsable du contrôle de l'exécution de l'instance de départ. En relation avec un pair, le module d'exécution simule l'exécution des services fournis par le pair. En relation avec un super-pair, le module d'exécution simule l'interaction entre le super-pair et les pairs impliqués dans l'instanciation.

5.4.1 Exécution d'une instance

L'exécution d'une instance dans DyCoSe correspond à l'orchestration des séquences d'invocations des services composant l'instance. Deux stratégies sont envisageables. Une première stratégie centralisée consiste à exécuter l'orchestration sur un super-pair. Ce dernier est responsable de coordonner l'activité des pairs. Il surveille les échanges de messages et l'exécution des services pour assurer le bon comportement de l'application composite instanciée. Une deuxième stratégie distribuée consiste à déléguer aux pairs fournissant les services de l'instance courante la surveillance et la gestion de l'exécution. L'orchestration est totalement gérée par les pairs interagissant. L'avantage de cette deuxième stratégie est l'indépendance totale de l'application composite vis-à-vis du réseau virtuel. Cependant, la détection des erreurs dues notamment au départ inattendu d'un pair devient difficile. DyCoSe adopte la première stratégie garantissant un meilleur contrôle sur l'exécution d'une instance.



(a) Exécution sans changements dans l'environnement

(b) Exécution avec trois changements ou pannes

Figure 5.21: Exemples d'exécution d'une instance

Un aspect majeur de l'écosystème de DyCoSe est le dynamisme qui se caractérise par la déconnexion de certains pairs membres et la connexion de nouveaux pairs entraînant la modification des services disponibles. Il est effectivement difficile de garantir que les services membres composant une instance de départ i_d soient disponibles tout au long de l'exécution de cette instance. Pair ailleurs, il est aussi difficile de garantir la non dégradation des propriétés de qualité de service ou toute autre propriété non fonctionnelle d'un service durant son exécution. Par exemple, il est difficile de garantir la réception de la réponse d'un service à l'instant prévu à cause des risques de surcharge du service, de goulets d'étranglement du réseau ou de pannes imprévues.

Dans un environnement où les changements sont fréquents et imprévisibles, le coût d'une instance calculé a priori perd rapidement en signification. En effet, suite aux changements susceptibles d'avoir lieu dans l'environnement, les coûts calculés en se basant sur des données rassemblées avant l'exécution seront modifiés. Nous prenons l'exemple d'un GBP ayant un chemin d'exécution composé de quatre services abstraits $\overline{V} = \{A_1, A_2, A_3, A_4\}$ associés séquentiellement tel que $\overline{E} = \{(A_1, A_2), (A_2, A_3), (A_3, A_4)\}$. Soit l'instance de départ i_d composée des quatre services membres suivants : $S_{1,1}$, $S_{2,1}$, $S_{3,1}$, $S_{4,1}$. L'exécution de i_d implique l'invocation de ces services membres selon le modèle d'orchestration décrit par le diagramme séquentiel du GBP. La figure 5.21a illustre les quatre étapes de l'exécution de cette instance. En supposant que cette exécution commence à l'instant $t = t_0$ et que le temps total calculé est $c^{temps}(i_d) = t_i$, l'exécution sera terminée avec succès à l'instant $t = t_0 + t_i$, si aucun des changements qui ont eu lieu dans l'écosystème

n'affectent les services composant le GBP instancié. Cependant, si nous considérons les trois changements dans l'environnement de la figure 5.21b, au niveau des services $S_{1,1}$, $S_{2,1}$ et $S_{4,1}$ impliquant leur substitution par les services $S_{1,2}$, $S_{2,2}$ et $S_{4,2}$ respectivement, nous observons clairement que le temps d'exécution effectif t_j de l'instance dépasse le temps prévu d'exécution t_i (calculé a priori).

Deux aspects critiques sont à adresser : (i) comment une instance de départ va réagir pendant son exécution face aux changements de l'environnement : la complétude de son exécution sera-t-elle compromise par le départ d'un fournisseur ou tout autre changement ? (ii) le cas échéant, comment choisir une instance alternative ?

Dans la section suivante nous proposons d'étudier la réussite de l'exécution d'un ensemble d'instances dans l'environnement simulé de DyCoSe. Le choix de l'instance alternative après une panne, s'inscrit dans les perspectives futures de nos travaux de recherche.

5.4.2 Réussite de l'exécution d'une application composite

Nous proposons d'étudier dans la suite, l'impact du taux de départ des pairs fournisseurs sur la réussite de l'exécution des instances. La figure 5.16 illustre les résultats des expériences conduites avec les paramètres suivants :

- Nombre de pairs = 5000.
- Nombre de services abstraits dans l'écosystème = 40.
- Nombre maximal de services fournis par un pair = 1.
- Nombre de GBP à instancier = 100.
- Nombre maximal de services abstraits dans un GBP = 5.
- Variables : μ désignant le taux de départ des pairs et n.
- Modèle de panne adopté : la "mort subite", c'est-à-dire si un pair se déconnecte, il ne se reconnecte plus durant la simulation.
- Mesures de performances : nombre de GBPs non instanciés et nombre d'instances dont l'exécution a échoué.
- Durée de la simulation : 1000 cycles.

Nous avons mené une série d'expériences visant à instancier et à exécuter l'instance de départ en variant le taux de départ des pairs entre deux expériences. μ prend des valeurs entre 0.5 et 13. Durant chaque expérience, 100 GBP sont injectés aléatoirement sur les super-pairs à raison d'un GBP tous les 6 cycles de sorte à distribuer les demandes d'instanciation uniformément sur la période de simulation.

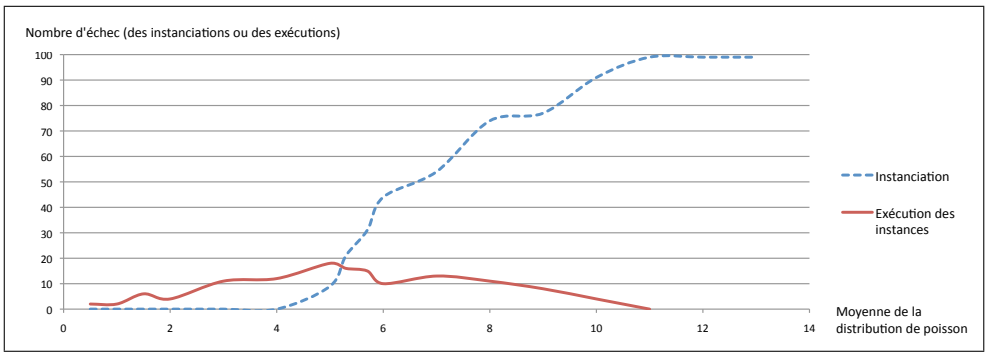


Figure 5.22: Résultats combinés d'instanciation et d'exécution (100 GBP)

Durant la simulation, un super-pair ayant un GBP à instancier, exécute le module d'instanciation comme décrit dans la section 5.3.3.2. À la fin de l'instanciation, le super-pair s'il a réussi à calculer une instance de départ, lance le module d'exécution d'instances. Le super-pair initialise l'exécution de l'instance courante en envoyant un message au pair fournissant le premier service de l'instance. Le super-pair orchestre et surveille l'exécution de l'instance comme décrit en section 5.4.1. À chaque cycle, le super-pair vérifie l'état de l'exécution de toutes les instances dont il est responsable. Si l'instance est en état de panne, le super-pair marque l'instance comme non exécutable et incrémente un compteur partagé.

Sur la figure 5.22, la courbe des instanciations indique le nombre de GBP dont l'instanciation a échoué car un des services abstraits n'est pas implémenté au moment de l'exécution de l'instanciation. Elle permet de vérifier l'analyse menée en section 5.3.3.2. En effet, nous retrouvons le taux d'échec qui croit avec la valeur de μ entre entre 5 et 12. Pour $\mu > 12$, le taux d'échec se stabilise dépassant toujours 95%. La courbe des exécutions indique le nombre d'instance dont l'exécution à échouée.

Sur la figure 5.22, la courbe des exécutions indique le nombre d'instances qui ont échoué pendant leur exécution ou qui n'ont pas pu entamé l'exécution à cause du départ du pair fournisseur du premier service durant le cycle suivant l'instanciation.

Quand $\mu < 5$, nous constatons que le nombre d'échec des exécutions est supérieur au nombre d'échec des instanciations. En effet, nous avons vérifié en section 5.3.3.2, que le processus d'instanciation est toujours réalisable pour $\mu < 5$ dans une configuration de paramètres similaire à la configuration de cette expérience.

Cependant, quand $5 < \mu < 11$, nous observons que bien que le nombre d'échec des instanciations augmente, le nombre d'échec des exécutions diminue. En effet,

avec l'augmentation le nombre d'échec des instanciations, le nombre d'instances de départ générées diminue. En d'autres termes, il y a moins d'échec d'exécution car il y a moins d'instances à exécuter.

Finalement, quand $\mu > 11$, la courbe d'exécution s'arrête car en effet le taux d'échec de l'instanciation a atteint 100%, donc il n'y a plus d'instances de départ à exécuter.

Chapitre 6

Conclusion

Sommaire

6.1 Discussion **196**

6.2 Perspectives futures **199**

Dans ce chapitre nous présentons d'abord une discussion du travail décrit dans les chapitres précédents en soulignant les contributions majeures. Ensuite, nous dégageons les perspectives futures en rapport avec les problèmes étudiés.

6.1 Discussion

Dans cette thèse nous avons proposé l'environnement DyCoSe pour la conception et le développement d'applications par composition dynamique de services. Une application est décrite par un diagramme d'interaction entre composants de haut niveau. Sa description fonctionnelle est raffinée en une composition de services abstraits par une démarche en trois étapes. Au moment de l'exécution, un processus d'instanciation remplace les services abstraits par des services membres fournis dans l'environnement de partage, générant ainsi une orchestration de services réalisant la logique applicative. Le processus d'instanciation permet de tenir compte des préférences de l'utilisateur exprimées en terme de contraintes sur les propriétés non fonctionnelles des services composant l'application. Il permet aussi, de reporter le choix des services jusqu'à l'exécution pour tenir compte des changements susceptibles d'avoir lieu entre le moment de description de la composition et le moment de son exécution dans l'environnement de partage de services.

Une analyse de la littérature sur les services permet de constater que (*i*) l'aspect fonctionnel est largement adressé par la recherche sur la description, la découverte et la composition de services. (*ii*) L'aspect non fonctionnel est souvent présenté comme une extension à la description et à la découverte. (*iii*) Les modèles de composition ne tiennent pas explicitement compte des propriétés de l'environnement où les services sont fournis. DyCoSe s'appuie sur des concepts abordés dans la littérature traitant la description, la découverte et la composition de services mais il propose de les réorganiser de façon à étudier l'aspect dynamique des applications composites. Les contributions de DyCoSe sont :

- Un écosystème coopératif où les entreprises membres partagent un consensus global décrivant les fonctionnalités métier récurrentes et les propriétés non fonctionnelles partagées. Cet écosystème propose, premièrement, une solution aux conflits sémantiques entre membres participants. En effet, les services abstraits modélisent les fonctionnalités métiers traditionnelles et le consensus global propose une terminologie unifiée des propriétés significatives aux membres. Il propose deuxièmement, une solution aux défis de la découverte de

services. D'abord, il limite l'étendue de la découverte, au niveau physique, aux membres de l'écosystème et au niveau logique, aux fonctionnalités reconnues. Ensuite, il simplifie la découverte pour l'utilisateur qui manipule seulement des services abstraits, les correspondances entre les services abstraits et les services membres étant gérées par les fournisseurs de services.

- Une architecture de composition d'applications à trois niveaux qui supporte une approche de composition combinant à la fois une démarche descendante et une autre ascendante. La démarche descendante permet de modéliser la logique applicative par une interaction de haut niveau, de la raffiner en une orchestration de services abstraits et de propager les contraintes de réalisation et les préférences de l'utilisateur en vue de choisir une composition de départ. La démarche ascendante projette les caractéristiques du réseau sous-jacent au niveau contenant l'instance du processus décrivant l'application, permettant ainsi d'évaluer les propriétés et les contraintes étudiées.

- Une description de l'instanciation d'une application comme un problème d'optimisation de coûts utilisant un ensemble de variables décisionnelles indiquant les services membres retenus dans l'instance. Une instance possible a un ensemble de coûts d'exécution exprimés en fonction des propriétés non fonctionnelles des services et des liens qui la composent. L'objectif de l'instanciation est de retenir une instance de départ composée de services proposant le meilleur score et respectant les contraintes de l'utilisateur. Deux types de solutions au problème de l'instanciation sont étudiées. D'abord une solution globale qui considère l'ensemble des services disponibles dans l'écosystème. Ensuite, une solution locale qui favorise les services fournis par un ensemble de communautés choisies par l'utilisateur. Pour chaque solution nous avons proposé un protocole décrivant le rôle des pairs et des super-pairs dans le calcul.

- Une implémentation de l'instanciation initiale d'une application composite. Nous avons proposé un algorithme génétique permettant de calculer une solution approchée au problème d'instanciation globale. L'objectif de cette implémentation est de calculer une instance de départ en un temps "raisonnable" quand le nombre de services abstraits et le nombre de services membres sont élevés. L'expérimentation a montré que l'exécution de l'algorithme propose rapidement une instance approchée, à 20% en moyenne, de l'instance optimale.

– Un prototype réalisant l'environnement de partage de services. Il permet de simuler l'organisation du réseau virtuel décrit dans le chapitre 3, l'exécution du processus d'instanciation initiale dans un environnement dynamique et l'exécution d'une instance de départ. Nous avons analysé le taux de réussite du processus d'instanciation en simulant le départ des pairs fournisseurs. Nous avons décrit une série d'expériences visant à observer le taux de réussite de l'exécution d'une application composite, soulignant ainsi le besoin d'adaptabilité.

La démarche de raffinement décrite dans DyCoSe peut être réalisée avec un outil de conception et de composition de services compatible avec les spécifications de la notation BPMN et du langage BPEL. Nous citons BPMN-Layouter proposé par Effinger *et al.* [41], BPMN Modeler [1] qui est un plugin eclipse, le plugin SOA [2] de Netbeans permettant de décrire des compositions de services en BPEL ou BPEL Project [3] qui est un plugin eclipse.

Plusieurs travaux ont abordé la composition dynamique de services. Casati *et al.*, proposent E-Flow qui est un environnement de composition de services implémentant un moteur de composition dynamique [26]. Ils décrivent un ensemble de règles de sélection permettant d'analyser la réquête de l'utilisateur et d'en extraire les informations qui seront utilisées lors de la découverte des services. Cependant, ils n'adressent pas les propriétés non fonctionnelles des services. Sun *et al.* présentent StarWSCoP dans [137], comme un environnement de composition de services qui repose sur un moteur de composition dynamique. Les auteurs proposent une extension de UDDI pour ajouter des paramètres de QoS. Cependant, le prise en charge des propriétés non fonctionnelles au moment de la sélection consiste à comparer les services individuellement. Agarwal *et al.* [2] proposent un environnement de création de services basé sur la composition. Ils affirment que leur outil de composition "facilitera l'automatisation du processus de composition partant de la définition d'une fonctionnalité métier et allant jusqu'à l'exécution et le déploiement d'un workflow composite de fonctionnalités". Cependant, ils abordent rapidement l'aspect non fonctionnel à travers un bref état de l'art. L'environnement de composition de services web, Self-Serv est présenté dans [15, 131]. Les auteurs étudient le problème d'instanciation par une approche de programmation linéaire. Cependant, ils n'abordent pas l'aspect dynamique qui conditionne la réussite de l'instanciation.

1. BPMN Modeler est disponible à l'adresse suivante : `http://www.eclipse.org/bpmn/`.
2. Plugin SOA pour Netbeans : `http://soa.netbeans.org/#compapp`.
3. BPEL Project est disponible à l'adresse suivante : `http://www.eclipse.org/bpel/`.

6.2 Perspectives futures

À la suite de cette thèse, nous dégageons un ensemble de perspectives futures répondant à deux objectifs principaux : (*i*) compléter les travaux sur l'instanciation et (*ii*) adresser le problème d'adaptabilité dynamique d'une instance.

Concernant le processus d'instanciation, il serait intéressant de :

- Étudier d'autres possibilités d'implémentation de l'intanciation globale, comme par exemple le tabu search [52, 53] et de l'instanciation locale, comme par exemple le "branch and bound" [75, 94].
- Étudier sur les possibilités de construction des communautés sur une ou plusieurs propriétés non fonctionnelles contenues dans la fonction de coût de sorte à simplifier le calcul de l'instanciation (une ou plusieurs variables seront "cachées" dans la structure du réseau virtuel).
- Compléter le prototype de simulation pour prendre en compte tous les aspects décrits par DyCoSe et pousser l'expérimentation :
  - Implémenter l'instanciation locale pour la comparer avec l'instanciation globale au niveau (*i*) du taux de reussite et (*ii*) du coût de l'instance retenue.
  - Comparer la performance de l'algorithme génétique proposé avec d'autres types d'algorithmes de calcul de l'instance de départ proposés dans la littérature.
  - Expérimenter avec différentes topologies de super-pairs. Nous rappelons que dans le réseau virtuel décrit actuellement dans DyCoSe, nous supposons l'inter-connectivité totale entre super-pairs.

Généralement, l'adaptabilité dynamique d'un système se manifeste par la reconfiguration de ce système durant l'exécution. Dans la littérature, la reconfiguration est adressée à deux niveaux, à savoir au niveau physique et au niveau logique [32]. La reconfiguration physique des composants électroniques d'un système vise à augmenter sa performance. Les circuits FPGA modulables sont un exemple classique de systèmes adaptables au niveau physique. La reconfiguration logique des composants applicatifs d'un système vise la modification d'une ou de plusieurs fonctionnalités accomplies par le système.

Casati *et al.* désignent par l'adaptabilité d'un processus de services, "sa capacité de se reconfigurer pour tenir compte des changements dans son environ-

nement d'exécution avec une intervention manuelle minimale, voire sans interventions" [26]. Plus généralement en génie logiciel, l'adaptabilité "permet d'améliorer la réutilisation d'un logiciel mais elle peut parfois compromettre sa performance" [33]. Deux approches pour implémenter l'adaptabilité d'un logiciel sont envisageables [91] : *parameter adaptation* et *compositional adaptation*. La première approche consiste à modifier certaines variables ou paramètres définissant le comportement d'un logiciel. La deuxième approche consiste à recomposer un logiciel afin de lui permettre d'accomplir une nouvelle fonctionnalité jusqu'alors non disponible. Selon les auteurs de [33] et de [79], l'adaptabilité d'un logiciel peut être (*i*) automatique c'est-à-dire elle est accomplie par le logiciel sans intervention humaine ; par opposition à une adaptabilité manuelle nécessitant l'intervention d'un expert humain. Elle peut être aussi (*ii*) dynamique c'est-à-dire elle modifie le comportement d'un logiciel durant son exécution ; par opposition à une adaptabilité statique modifiant le code source d'un logiciel. Elle peut être finalement (*iii*) réactive c'est-à-dire elle génère une reconfiguration d'un logiciel après un événement déclencheur ; par opposition à une adaptabilité proactive prévoyant a priori les reconfigurations possibles d'un logiciel.

Dans le cadre des applications composites, l'adaptabilité dynamique se manifeste par la "récupération" de l'application après une panne ou un autre changement dans l'environnement affectant son comportement. Il serait intéressant d'étudier l'adaptabilité dynamique d'une application composite dans le double but de :

- Comparer les deux méthodes d'instanciation proposées afin d'identifier des paramètres permettant de favoriser une méthode par rapport à l'autre.
- Comparer le coût d'une instance calculé a priori avec le coût réel obtenu après exécution.

Il serait aussi intéressant de comparer deux stratégies d'adaptabilité. Une première stratégie proactive qui calcule pendant l'instanciation une ou plusieurs instances "de secours" et une deuxième stratégie réactive qui substitue un service membre ou une séquence de services membres par leur équivalent durant l'exécution.

Annexe A

Exemple d'interfaces WSDL

Cet Annexe fournit deux exemples d'interfaces WSDL.

A.1 Interface d'un service d'information météo

Service de météo

```
1
2  <?xml version="1.0"?>
3  <definitions name="temperature"
4
5  targetNamespace="http://example.com/temperature.wsdl"
6            xmlns:tns="http://example.com/temperature.wsdl"
7            xmlns:xsd1="http://example.com/temperature.xsd"
8            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
9            xmlns="http://schemas.xmlsoap.org/wsdl/">
10
11     <types>
12        <xs:schema targetNamespace="http://example.com/temperature.xsd"
13               xmlns="http://www.w3.org/2000/10/XMLSchema">
14
15                  <xs:element name="TemperatureRequest">
16               <xs:complexType>
17                              <xs:sequence>
18                                     <xs:element name="location" type="xs:string"/>
19                                     <xs:element name="date" type="xs:date"/>
20                           </xs:sequence>
21               </xs:complexType>
22            </xs:element>
```

```
23
24                  <xs:element name="TemperatureValue" type="float" />
25
26          </xs:schema>
27      </types>
28
29      <message name="GetTemperatureInput">
30          <part name="body" element="TemperatureRequest"/>
31      </message>
32
33      <message name="GetTemperatureOutput">
34          <part name="body" element="TemperatureValue"/>
35      </message>
36
37      <portType name="TemperaturePortType">
38          <operation name="GetTemperature">
39             <input message="tns:GetTemperatureInput"/>
40             <output message="tns:GetTemperatureOutput"/>
41          </operation>
42      </portType>
43
44      <binding name="TemperatureSoapBinding" type="tns:TemperaturePortType">
45          <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
46          <operation name="GetTemperature">
47             <soap:operation soapAction="http://example.com/GetTemperature"/>
48             <input> <soap:body use="literal"/> </input>
49             <output> <soap:body use="literal"/> </output>
50          </operation>
51      </binding>
52
53      <service name="TemperatureService">
54          <documentation>Exemple d'un service simplifié de météo</documentation>
55          <port name="TemperaturePort" binding="tns:TemperatureSoapBinding">
56             <soap:address location="http://example.com/temperature"/>
57          </port>
58      </service>
59
60  </definitions>
61
```

A.2 Interface d'un service de sécrétariat médical

Service de sécrétariat médical

```
1
2  <? xml version = "1.0" ?>
3
4  <definitions name = "Medical Secretary"
5     targetNamespace = "http://example.com/medical/MedicalSecretary"
6     xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
7     xmlns:tns = "http://example.com/medical/MedicalSecretary"
8     xmlns = "http://schemas.xmlsoap.org/wsdl/">
9
10 <!-- ************************************************************ -->
11 <!-- ****************** COMPLEX TYPES ********************** -->
12 <!-- ************************************************************ -->
13
14 <types>
15      <schema xmlns = "http://www.w3.org/2000/10/XMLSchema">
16
17         <complexType name = "patient">
18         <sequence>
19           <element name = "name" type = "xsd:string" />
20           <element name = "patientID" type = "xsd:string" />
21         </sequence>
22         </complexType>
23
24         <complexType name = "proposedAppointement">
25         <sequence>
26           <element name = "startDate1" type = "date" />
27           <element name = "startDate2" type = "date" />
28           <element name = "comments" type = "xsd:string" />
29         </sequence>
30         </complexType>
31
32         <complexType name = "attributedAppointement">
33         <sequence>
34           <element name = "appointmentID" type = "xsd:string" />
35           <element name = "startDate" type = "date" />
36           <element name = "endDate" type = "date" />
37           <element name = "patientID" type = "xsd:string" />
38           <element name = "comments" type = "xsd:string" />
39         </sequence>
40         </complexType>
41
42         <complexType name = "prescription">
43         <sequence>
44           <element name = "prescriptionID" type = "xsd:string" />
```

```
45          <element name = "prescriptionURL" type = "xsd:string" />
46        </sequence>
47        </complexType>
48
49      </schema>
50    </types>
51
52  <!-- ************************************************************ -->
53  <!-- ****************** MESSAGES ******************************** -->
54  <!-- ************************************************************ -->
55    <message name = "appointmentRequest">
56      <part name = "patient" type = "tns:patient"/>
57      <part name = proposedAppointementtype = "tns:proposedAppointement"/>
58    </message>
59
60    <message name = "appointmentAcknowledgementRequest">
61      <part name = "patient" type = "tns:patient"/>
62      <part name = "attributedAppointment" type = "tns:attributedAppointment"/>
63    </message>
64
65
66    <message name = "appointmentAcknowledgement">
67      <part name = "patient" type = "tns:patient"/>
68      <part name = "attributedAppointment" type = "tns:attributedAppointment"/>
69    </message>
70
71    <message name = "prescriptionDispatch">
72      <part name = "patient" type = "tns:patient"/>
73      <part name = "prescription" type = "tns:prescription"/>
74    </message>
75
76  <-- ************************************************************ -->
77  <-- ***** MEDICAL SECRETARY SERVICE PORT TYPES *** -->
78  <-- ************************************************************ -->
79
80  <portType name = "MedicalSecretarytoPatient">
81      <documentation>
82         This port type references the operations the secretary office
83         performs with the medical secretary service
84      </documentation>
85
86      <operation name = "getAppointment">
87         <input message = "tns:appointmentRequest"/>
88         <output message = "tns:appointmentAcknowledgementRequest"/>
```

```
89      </operation>
90
91      <operation name = "confirmAppointment">
92         <input message = "tns:appointmentRequest"/>
93         <output message = "tns:appointmentAcknowledgementRequest"/>
94      </operation>
95
96      <operation name = "getPrescription">
97         <output message = "tns:prescriptionDispatch"/>
98      </operation>
99   </portType>
100 </definitions>
101
```

Annexe B

Exemple d'une description comportementale d'un service

Interface WSCI d'un service de sécrétariat médical

```
1
2  <? xml version = "1.0" ?>
3  <wsdl:definitions name = "Medical Secretary Dynamic Interface"
4     targetNamespace = "http://example.com/consumer/MedicalSecretary"
5     xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
6     xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
7     xmlns:tns = "http://example.com/consumer/MedicalSecretary"
8     xmlns = "http://www.w3.org/2002/07/wsci10">
9     <!-- WSDL complex types -->
10    <!-- WSDL message definitions -->
11    <!-- WSDL operations and port types -->
12    <!ó- selectors -->
13
14    <correlation name = "appointmentCorrelation"
15             property = "tns:appointmentID">
16    </correlation>
17
18    <interface name = "MedicalSecretary">
19      <process name = "assignAppointment"
20           instantiation = "message">
21
22         <sequence>
23         <action name = "ReceiveAppointmentRequest"
24             role = "tns:MedicalSecretary"
25             operation = "MedicalSecretarytoPatient/getAppointment">
```

```
26          </action>
27
28          <action name = "ReceiveDateConfirmation"
29              role = "tns:MedicalSecretary"
30              operation = "MedicalSecretarytoPatient/confirmAppointment">
31             <correlate correlation="tns:appointmentCorrelation"/>
32          </action>
33
34     </interface>
35  </wsdl:definitions>
36
```

Annexe C

Exemple d'ontologie non fonctionnelle

Exemple d'ontologie

```

<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://www.owl-ontologies.com/Ontology1255615917.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.owl-ontologies.com/Ontology1255615917.owl">

<owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="non_QoS">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Non_Fonctionnelles"/>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:ID="Temps_reponse">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="QoS"/>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:Class rdf:about="#QoS">
    <rdfs:subClassOf rdf:resource="#Non_Fonctionnelles"/>
  </owl:Class>

```

```
29  <owl:Class rdf:ID="Temps_transfert">
30    <rdfs:subClassOf rdf:resource="#Temps_reponse"/>
31  </owl:Class>
32
33  <owl:Class rdf:ID="Temps_execution">
34    <rdfs:subClassOf rdf:resource="#Temps_reponse"/>
35  </owl:Class>
36
37  </rdf:RDF>
38
39
```

Annexe D

Exemple d'une interface d'un service abstrait

Billing Abstract Service

```
1
2  <? xml version = "1.0" ?>
3
4  <definitions name = "Billing"
5     targetNamespace = "http://example.com/Billing"
6     xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
7     xmlns:tns = "http://example.com/Billing"
8     xmlns = "http://schemas.xmlsoap.org/wsdl/">
9
10
11 <!-- ************************************************************ -->
12 <!-- ****************** COMPLEX TYPES **************************** -->
13 <!-- ************************************************************ -->
14
15    <types>
16      <schema xmlns = "http://www.w3.org/2000/10/XMLSchema">
17
18         <complexType name = "paymentOrder">
19         <sequence>
20           <element name = "paymentOrderID" type = "xsd:string" />
21           <element name = "purchaseID" type = "xsd:string" />
22           <element name = "paymentOrderURL" type = "xsd:string" />
23         </sequence>
24         </complexType
25
```

```
26          <complexType name = "receipt">
27          <sequence>
28            <element name = "receiptID" type = "xsd:string" />
29            <element name = "receiptURL" type = "xsd:string" />
30          </sequence>
31          </complexType>
32
33          <complexType name = "CCInfo">
34          <sequence>
35            <element name = "number" type = "xsd:string" />
36            <element name = "issuer" type = "xsd:string" />
37            <element name = "expiryDate" type = "month" />
38            <element name = "holderName" type = "xsd:string" />
39            <element name = "securityNumber" type = "integer" />
40          </sequence>
41          </complexType>
42
43        </schema>
44      </types>
45
46  <!-- ************************************************************ -->
47  <!-- ****************** MESSAGES ********************************** -->
48  <!-- ************************************************************ -->
49
50      <message name = "paymentOrderRequest">
51        <part name = "purchaseID" = "xsd:string"/>
52      </message>
53
54      <message name = "paymentOrderDispatch">
55        <part name = "paymentOrder" = "xsd:paymentOrder" />
56      </message>
57
58      <message name = "receivePayment">
59        <part name = "paymentOrderID" = "xsd:string"/>
60        <part name = "CCInfo" type = "tns:CCInfo"/>
61      </message>
62
63      <message name = "receiptDispatch">
64        <part name = "purchaseID" = "xsd:string"/>
65        <part name = "paymentOrderID" = "xsd:string"/>
66        <part name = "receipt" type = "tns:receipt"/>
67      </message>
68
69  <-- ************************************************************ -->
```

```
70  <-- *************** BILLING SERVICE PORT TYPES ****************** -->
71  <-- ************************************************************** -->
72
73  <portType name = "BillingtoClient">
74       <documentation>
75       </documentation>
76
77       <operation name = "getpaymentOrder">
78          <input message = "tns:paymentOrderRequest"/>
79          <output message = "tns:paymentOrderDispatch"/>
80       </operation>
81
82       <operation name = "validatePayment">
83
84          <input message = "tns:receivePayment" sawsdl:modelReference=
85                "http://lsdis.cs.uga.edu/projects/meteor-s/
86                   wsdl-s/ontologies/LSDIS_FInance.owl#InternalTransfer"/>
87
88          <output message = "tns:receiptDispatch" sawsdl:modelReference=
89                 "http://lsdis.cs.uga.edu/projects/meteor-s/
90                  wsdl-s/ontologies/LSDIS_FInance.owl#Receipt"/>
91
92       </operation>
93
94     </portType>
95  </definitions>
```

Annexe E

Extraits du prototype

E.1 Extrait du code d'un pair

Les pairs

```
1
2  public class EALNode extends GeneralNode {
3      private double onlineTime;
4      private double offlineTime;
5      private double currentOnlineTime;
6      private ArrayList<Service> hostedServices;
7      public boolean isSuperPeer = false;
8      public ArrayList<EALNode> localState; // used only if selected as superPeer
9      public ArrayList<EALNode> mySuperPeers; // used only if selected as superPeer
10     public ArrayList GBPQueue;
11     public ArrayList InstancesQueue;
12     public ArrayList ToDo;  // Services queue
13     public ArrayList<AbstractService> implementedAbstractServices;
14
15     public EALNode(String prefix) {
16         super("eal-" + prefix); }
17  ...
18  } // end EALNode
19
20 public class GeneralNode implements Node {
21    protected Protocol[] protocol = null; //The protocols on this node.
22    private int index; // index in the network array
23    protected int failstate = Fallible.OK; //The fail state of the node.
24    private long ID; //ID
25
```

```
26     public GeneralNode(String prefix) { // PeerSim code to initialize a node
27         String[] names = Configuration.getNames(PAR_PROT);
28         CommonState.setNode(this);
29         ID=nextID();
30         protocol = new Protocol[names.length];
31         for (int i=0; i < names.length; i++) {
32                 CommonState.setPid(i);
33                 Protocol p = (Protocol)
34                         Configuration.getInstance(names[i]);
35                 protocol[i] = p;
36         }
37 } // end constructor
38 ...
39 } // end GeneralNode
40
```

E.2 Extrait du code du générateur de réseau physique

Générateur de réseaux physiques

```
1
2  public class physicalInit implements Control {
3
4      private final String name;
5      private ExtendedRandom myRd;
6
7      public physicalInit(String name) {
8          this.name = name;
9          myRd = new ExtendedRandom(220481);
10     }
11
12     private void initAbstractServices(int nAbstractServices) {
13         EALparam.tabAbstractServices = new AbstractService[nAbstractServices];
14
15         for (int i = 0; i < nAbstractServices; i++) {
16             EALparam.tabAbstractServices[i] = new AbstractService();
17             EALparam.tabAbstractServices[i].setLabel("AbstractService-" + i);
18             EALparam.tabAbstractServices[i].setID(i);
19         }
20
21     }
22
23     private void initPeersProperties() {
24         EALparam.peersHolder = new ArrayList();
```

```
25          EALparam.tabServices = new ArrayList();
26          int t = 0; // compteur de services generés, pour avoir un ID unique
27          int tempTime = 0; // temp var to store a service's exec time
28          int servicesNumber;
29          ArrayList tempPeerServices;
30          Service tempService;//= new Service();
31          AbstractService tempAbstractService;
32          for (int i = 0; i < Network.size(); i++) {
33              // fill the tabPeers
34              EALparam.peersHolder.add((EALNode) Network.get(i));
35              // set peer properties
36                  ...
37              // associate services to peers
38                  ...
39              //// initialiser la file d'attente de chque pair
40              ((EALNode) Network.get(i)).ToDo = new ArrayList();//(ArrayList)tempToDo.clone();
41              ((EALNode) Network.get(i)).mySuperPeers = new ArrayList();
42
43              // Create services for the current peer
44              if (servicesNumber > 0) {
45                   ...
46                  } // fin remplir les services d'un pair
47              } // fin if
48              ((EALNode) Network.get(i)).setHostedServices((ArrayList) tempPeerServices.clone());
49              tempPeerServices.clear();
50      } // fin initPeersProperties
51
52
53      private void generateCostMatrix() {
54           EALparam.distanceMatrix = new double[Network.size()][Network.size()];
55          double tempRandom = 0;
56          for (int i = 0; i < Network.size(); i++) {
57              for (int j = 0; j < Network.size(); j++) {
58                  if (i == j) {
59                     EALparam.distanceMatrix[i][j] = 0;
60                  } else {
61                      while (tempRandom == 0) {
62                          tempRandom = Math.round(CommonState.r.nextDouble() * 100) / 100.0;
63                      }
64                     EALparam.distanceMatrix[i][j] = tempRandom * 2;
65                      EALparam.distanceMatrix[j][i] = EALparam.distanceMatrix[i][j];
66                     // assigns a double between 0 and 2
67                      tempRandom = 0;
68                  }
```

```
69              }
70          }
71      }
72
73      public boolean execute() {
74
75          initAbstractServices(EALparam.nAbstractServices);
76          initPeersProperties();
77          generateCostMatrix();
78
79          return false;
80      }
81  }
82
83
```

E.3 Extrait du code du générateur de réseaux virtuels

Générateur de réseaux virtuels

```
1
2   public class overlayInit implements Control {
3
4       private final String name;
5       private ExtendedRandom myRd;
6
7       public overlayInit(String name) {
8           this.name = name;
9           myRd = new ExtendedRandom(220481);
10      }
11
12      private void createCommunitiesNEW(double vDensite)
13      // densite = 10 donc pour 100 pairs 10 communautés
14      {
15          int densite = (int) (Configuration.getInt("network.size") * (vDensite / 100));
16          // densite veut dire nodesPerComm
17
18          int nSps = Configuration.getInt("network.size") / densite;
19          densite = densite - 1;
20          int peerNumber = 0; // to fill a community
21          EALparam.superPeersHolder = new EALNode[nSps];
22
23          // select sps
24          EALNode tempNode;
```

```
25          for (int i = 0; i < EALparam.superPeersHolder.length; i++) {
26              ...
27          }
28          // fill sps communities
29          ...
30
31          // select peers in community
32          for (int j = 0; j < densite; j++) {
33
34              //choose a random peer number
35                  ...
36              // check if it is not a SP
37              while (EALparam.peersHolder.get(peerNumber).isSuperPeer) {
38                ...
39              }
40
41              // add to the current sp community
42              EALparam.superPeersHolder[i].localState.add(EALparam.peersHolder.get(peerNumber));
43              EALparam.peersHolder.get(peerNumber).mySuperPeers.add(EALparam.superPeersHolder[i]);
44              ...
45
46          }
47      }
48  }
49
50  public boolean execute() {
51
52          EALparam.communityDirectoryHolder = new CommunityDirectory();
53          createCommunitiesNEW(10);
54          returnfalse;
55      }
```

E.4 Fichier sortie

Dans la suite nous fournissons un ensemble d'images "imprimé écran" décrivant le fichier HTML en sortie de la simulation après la phase d'initialisation.

```
######### CYCLE: 0, Peers ON :10, Peers OFF : 0 ##########################

Pair:7 - voisins : [ (Pair:0,0.42), (Pair:1,1.5), (Pair:3,0.68), (Pair:4,1.08), (Pair:9,0.1), ]

Pair:9 - voisins : [ (Pair:0,0.32), (Pair:1,1.22), (Pair:7,0.1), (Pair:5,1.04), (Pair:8,1.78), ]

Pair:5 - voisins : [ (Pair:2,0.76), (Pair:4,1.9), (Pair:3,0.94), (Pair:1,0.94), (Pair:6,1.76), (Pair:8,1.86), (Pair:9,1.04), ]

Pair:1 - voisins : [ (Pair:9,1.22), (Pair:7,1.5), (Pair:3,0.34), (Pair:5,0.94), (Pair:6,0.94), ]

Pair:6 - voisins : [ (Pair:5,1.76), (Pair:1,0.94), ]

Pair:4 - voisins : [ (Pair:5,1.9), (Pair:7,1.08), (Pair:8,1.92), ]

Pair:3 - voisins : [ (Pair:2,1.32), (Pair:1,0.34), (Pair:7,0.68), (Pair:5,0.94), ]

Pair:2 - voisins : [ (Pair:5,0.76), (Pair:3,1.32), ]

Pair:0 - voisins : [ (Pair:7,0.42), (Pair:9,0.32), ]

Pair:8 - voisins : [ (Pair:4,1.92), (Pair:5,1.86), (Pair:9,1.78), ]
```

(a) Réseau physique (par le composant WireKOut)

Table d'etat global	
Super-pair	**Services abstrait**
Pair-5	, AbstractService-0, AbstractService-2
Pair-6	, AbstractService-3, AbstractService-1, AbstractService-0

Etat local de la communaute geree par Pair-5		
Pair	**Etat (O/F)**	**Services fournis**
Peer 5	0	ConcreteService-0.1 -
Peer 9	0	
Peer 7	0	ConcreteService-2.4 - ConcreteService-0.5 -
Peer 1	0	
Peer 3	0	

Etat local de la communaute geree par Pair-6		
Pair	**Etat (O/F)**	**Services fournis**
Peer 6	0	ConcreteService-3.2 - ConcreteService-1.3 -
Peer 0	0	
Peer 8	0	ConcreteService-0.6 - ConcreteService-1.7 -
Peer 2	0	ConcreteService-1.0 -
Peer 4	0	

(b) Réseau virtuel

Figure E.1: Réseaux simples générés par simulation

Bibliographie

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE : Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, December 2002.

[2] Vikas Agarwal, Koustuv Dasgupta, Neeran Karnik, Arun Kumar, Ashish Kundu, Sumit Mittal, and Biplav Srivastava. A service creation environment based on end to end composition of web services. In *WWW '05 : Proceedings of the 14th international conference on World Wide Web*, pages 128–137, New York, NY, USA, 2005. ACM.

[3] Matteo Agosti, Francesco Zanichelli, Michele Amoretti, and Gianni Conte. P2pam : a framework for peer-to-peer architectural modeling based on peersim. In *Simutools '08 : Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems and workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[4] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s, November 2005.

[5] Sheth Amit, Miller John, Arpinar I. Budak, and Verma Kunal. Meteor-s : Semantic web services and processes, 2005.

[6] Anupriya Ankolekar, Frank Huch, and Katia Sycara. Concurrent semantics for the Web services specification language DAML-S. *Lecture Notes in Computer Science*, 2315, 2002.

[7] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. Web services architecture requirements, 2004.

[8] Alistair P. Barros, Marlon Dumas, and Peter D. Bruza. The move to web service ecosystems. dec 2005.

[9] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language, 2004.

[10] Tom Bellwood, Douglas Bryan, Vadim Draluk, Dave Ehnebuske, Tom Glover, Andrew Hately, Yin Leng Husband, Alan Karp, Keisuke Kibakura, Chris Kurt, Jeff Lancelle, Sam Lee, Sean MacRoibeaird, Anne Thomas Manes, Barbara McKee, Joel Munter, Tammy Nordan, Chuck Reeves, Dan Rogers, Christine Tomlinson, Cafer Tosun, Claus von Riegen, and Prasad Yendluri. Uddi version 2.04, July 2002.

[11] Tom Bellwood, Steve Capell, Luc Clement, John Colgrave, Matthew J. Dovey Daniel Feygin, Andrew Hately, Rob Kochman, Paul Macias, Mirek Novotny, Massimo Paolucci, Claus von Riegen, Tony Rogers, Katia Sycara, Pete Wenzel, and Zhe Wu. Uddi version 3.0.2, October 2004.

[12] Tom Bellwood, Luc Clément, David Ehnebuske, Andrew Hately Maryann Hondo, Yin Leng Husband, Karsten Januszewski, Sam Lee, Barbara McKee, Joel Munter, and Claus von Riegen. Uddi version 3.0, July 2002.

[13] Boualem Benatallah, Mohand-Said Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. On automating web services discovery. *The VLDB Journal*, 14(1) :84–96, 2005.

[14] Boualem Benatallah, Said Hacid, Christophe Rey, and Farouk Toumani. Request rewriting-based web service discovery, 2003.

[15] Boualem Benatallah, Quan Z. Sheng, and Marlon Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1) :40–48, 2003.

[16] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture, 2004.

[17] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are two web services compatible ? In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *TES*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2004.

[18] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1, 2000.

[19] D. Bunting, M.C.O. Hurley, M. Little, J. Mischkinsky, E. Newcomer, J. Webber, and K. Swenson. Web services composite application framework (ws-caf) ver1.0, 2003.

[20] Rey C, Toumani Fand Hacid M-S, and Leger A. An algorithm and a prototype for the dynamic discovery of e-services. Technical report, Technical Report RR05-03, LIMOS, Clemont-Ferrand, France, 2003.

[21] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Specification : Web services transaction (ws-transaction), 2002.

[22] F. et al. Cabrera. Web services coordination (ws-coordination), 2005.

[23] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3) :281–308, 2004.

[24] Jorge Cardoso. Quality of service and semantic composition of workflows, 2002.

[25] Michael J. Carey. SOA what ? *IEEE Computer*, 41(3) :92–94, 2008.

[26] Fabio Casati, Ski Ilnicki, Li-jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in eflow. In *CAiSE '00 : Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 13–31, London, UK, 2000. Springer-Verlag.

[27] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream : high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 298–313, New York, October 19–22 2003. ACM Press.

[28] Michelene T. H. Chi. Quantifying qualitative analyses of verbal data : A practical guide, february 1997.

[29] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0, June 2007.

[30] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) version 1.1, March 2001.

[31] The Digital Ecosystems cluster members. The digital ecosystems research vision :2010 and beyond. July 2005.

[32] Katherine Compton and Scott Hauck. Reconfigurable computing : a survey of systems and software. *ACM Comput. Surv.*, 34(2) :171–210, 2002.

[33] Carine Courbis and Anthony Finkelstein. Towards aspect weaving applications. In Gruia-Catalin Roman, William G. Griswold, and Bashar Nuseibeh, editors, *ICSE*, pages 69–77. ACM, 2005.

[34] B. Craenen, A. E. Eiben, E. Marchiori, and A. Steenbeek. Combining local search and fitness function adaptation in a GA for solving binary constraint satisfaction problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2000.

[35] DAML-S-Coalition. Daml-s : Semantic markup for web services.

[36] Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta Kenig-Ries, Jacek Kopecky, Ruben Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web service modeling ontology (wsmo) - w3c member submission, June 2005.

[37] Jos de Bruijn, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu. Web service modeling language (wsml) - w3c member submission, June 2005.

[38] Marcin Detyniecki. Numerical aggregation operators : State of the art. In *International Summer School on Aggregation Operators and their Applications*, Asturias, Spain, July 2001. (invited).

[39] John Domingue, Liliana Cabral, Stefania Galizia, Vlad Tanasescu, Alessio Gugliotta, Barry Norton, and Carlos Pedrinaci. Irs-iii : A broker-based approach to semantic web services. *Web Semant.*, 6(2) :109–132, 2008.

[40] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 372–383, Toronto, Canada, September 2004. ACM Press.

[41] Philip Effinger, Martin Siebenhaller, and Michael Kaufmann. An interactive layout tool for bpmn. *E-Commerce Technology, IEEE International Conference on*, 0 :399–406, 2009.

[42] Wolfgang Emmerich. Distributed component technologies and their software engineering implications. In *ICSE '02 : Proceedings of the 24th International Conference on Software Engineering*, pages 537–546, New York, NY, USA, 2002. ACM.

[43] Michael D. Ernst, Raimondas Lencevicius, and Jeff H. Perkins. Detection of web service substitutability and composability, 2006.

[44] A. Agrawal et al. Web services human task (ws-humantask), version 1.0, 2007.

[45] Joel Farrell and Holger Lausen. Semantic annotations for wsdl and xml schema, August 2007.

[46] Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2) :113–137, 2002.

[47] Organization for the Advancement of Structured Information Standards. Oasis website.

[48] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Compatibility verification for web service choreography. In *ICWS '04 : Proceedings of the IEEE International Conference on Web Services*, page 738, Washington, DC, USA, 2004. IEEE Computer Society.

[49] T. Freund and T. Storey. Transactions in the world of web services, part 1. an overview of ws-transaction and ws-coordination, 2002.

[50] T. Freund and T. Storey. Transactions in the world of web services, part 2. an overview of ws-transaction and ws-coordination, 2002.

[51] Marc Friedman, Alon Levy, and Todd Millstein. Navigational plans for data integration. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI*, pages 67–73. AAAI Press/The MIT Press, 1999.

[52] Fred Glover. Tabu search–Part I. *ORSA Journal on Computing*, 1(3) :190–206, 1989.

[53] Fred Glover. Tabu search : A tutorial. *Interfaces*, 20(4) :74–94, 1990.

[54] Nathan Griffiths and Michael Luck. Coalition formation through motivation and trust. In *AAMAS '03 : Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 17–24, New York, NY, USA, 2003. ACM.

[55] Object Management Group. Business process modeling notation.

[56] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2) :199–220, 1993.

[57] Mohand-Said Hacid and Chantal Reynaud. L'integration de sources de donnees. *Revue Information - Interaction et Intelligence (I3) Une Revue en Sciences du Traitement de l'I*, 4(2), Juin 2004.

[58] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, and Bharat Bhargava. PROMISE : peer-to-peer media streaming using collectcast. In *Proceedings of the 11th ACM International Conference on Multimedia (MM-03)*, pages 45–54, November 4–6.

[59] F. Huch. Verifcation of erlang programs using abstract interpretation and model checking — extended version. Technical Report AIB-1999-02, RWTH Aachen, 1999.

[60] IBM. Business process execution language for web services.

[61] ISO. Iso 8402 quality management and quality assurance vocabulary, 1994.

[62] I.Stoica, R.Morris, D.Karger, M. F.Kaashoek, and H.Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM*, pages 149–160, 2001.

[63] ITU. Tu-t recommendation e.800, terms and definitions related to quality of service and network performance including dependability, 1994.

[64] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *ICDCS*, pages 102–109. IEEE Computer Society, 2004.

[65] Márk Jelasity, Alberto Montresor, and Özalp Babaoglu. T-man : Gossip-based fast overlay topology construction. *Computer Networks*, 53(13) :2321–2339, 2009.

[66] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http ://peersim.sf.net`.

[67] Howard Karloff. *Linear Programming*. Birkhäuser-Verlag, Boston, 1991.

[68] Uwe Keller, Rubén Lara, Holger Lausen, Axel Polleres, and Dieter Fensel. Automatic location of services. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[69] Uwe Keller, Rubén Lara, Axel Polleres, Ioan Toma, Michel Kifer, and Dieter Fensel. Wsmo web service discovery, November 2004.

[70] Mujtaba Khambatti, Kyung Dong Ryu, and Partha Dasgupta. Peer-to-peer communities : Formation and discovery. In *International Conference on Parallel and Distributed Computing Systems, PDCS 2002, November 4-6, 2002, Cambridge, USA, Proceedings*, pages 161–166, 2002.

[71] Matthias Klusch and Patrick Kapahnke. Semantic web service selection with SAWSDL-MX. In Rubén Lara Hernandez, Tommaso Di Noia, and Ioan Toma, editors, *SMRR*, volume 416 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[72] Matthias Klusch and Katia Sycara. Brokering and matchmaking for coordination of agent societies : a survey. 2001.

[73] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet : high bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 282–297, New York, October 19–22 2003. ACM Press.

[74] Kyriakos Kritikos and Dimitris Plexousakis. A semantic qos-based web service discovery algorithm for over-constrained demands. *Next Generation Web Services Practices, International Conference on*, 0 :49–54, 2007.

[75] Vipin Kumar. Branch-and bound search. In Stuart C. Shapiro, editor, *Encyclopaedia of Artificial Intelligence*, volume 2, pages 1000–1004. John Wiley and Sons, Inc., New York, 1987.

[76] Wilson Lau. Ecosystem for virtual enterprise. In Luis M. Camarinha-Matos and Hamideh Afsarmanesh, editors, *PRO-VE*, volume 262 of *IFIP Conference Proceedings*, pages 111–120. Kluwer, 2003.

[77] KangChan Lee, JongHong Jeon, WonSeok Lee, Seong-Ho Jeong, and Sang-Won Park. Qos for web services : Requirements and possible approaches, 2003.

[78] Maurizio Lenzerini. Data integration : a theoretical perspective. In *PODS '02 : Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM.

[79] Fernando Antônio Aires Lins, José Carlos dos Santos Júnior, and Nelson Souto Rosa. Improving transparent adaptability in web service composition. In *SOCA*, pages 80–87. IEEE Computer Society, 2007.

[80] LSDIS. Lsdis finance ontology.

[81] Zakaria Maamar, Djamal Benslimane, and Nanjangud C. Narendra. What can context do for web services ? *Commun. ACM*, 49(12) :98–103, 2006.

[82] Matteo Magnani and Danilo Montesi. BPMN : How much does it cost ? an incremental approach. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 80–87. Springer, 2007.

[83] Daniel J. Mandell and Sheila A. McIlraith. Automating web service discovery, customization, and semantic translation with a semantic discovery service. In *Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW'03)*, 2003.

[84] David Martin and al. Owl-s : Semantic markup for web services.

[85] David Martin and al. Owl-s : Semantic markup for web services - w3c member submission, 2004.

[86] David Martin, Massimo Paolucci, Sheila Mcilraith, Mark Burstein, Drew Mcdermott, Deborah Mcguinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing semantics to web services : The owl-s approach. pages 26–42. Springer, 2004.

[87] Klusch Matthias and Kapahnke Patrick. Sawsdl-tc, 2008.

[88] E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *SIGMOD Rec.*, 31(4) :36–41, 2002.

[89] Drew V. McDermott. Estimated-regression planning for interactions with web services. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *AIPS*, pages 204–211. AAAI, 2002.

[90] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In Dieter Fensel, Fausto Giunchiglia, Deborah McGuinness, and Mary-Anne Williams, editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, April 22–25 2002. Morgan Kaufmann Publishers.

[91] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *IEEE Computer*, 37(7) :56–64, 2004.

[92] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *VLDB J*, 12(4) :333–351, 2003.

[93] J. Miller, K. Verma, P. Rajasekaran, A. Sheth, R. Aggarwal, and K. Sivashanmugam. Wsdl-s : Adding semantics to wsdl, 2004.

[94] L. G. Mitten. Branch-and-bound methods : General formulation and properties. *Operations Research*, 18 :24–34, 1970.

[95] Alberto Montresor. A robust protocol for building superpeer overlay topologies. In Germano Caronni, Nathalie Weiler, and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, pages 202–209. IEEE Computer Society, 2004.

[96] Alberto Montresor, Mark Jelasity, and Ozalp Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Proceedings 2004 International Conference on Dependable Systems and Networks (DSN 2004), Dependable Computing and Communications Symposium*, pages 19–28, Florence, Italy, June-July 2004. IEEE Computer Society.

[97] Michael Mrissa, Chirine Ghedira, Djamal Benslimane, and Zakaria Maamar. A context model for semantic mediation in web services composition. In David W. Embley, Antoni Olivé, and Sudha Ram, editors, *ER*, volume 4215 of *Lecture Notes in Computer Science*, pages 12–25. Springer, 2006.

[98] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02 : Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM Press.

[99] Ngan, Wallach, and Druschel. Enforcing fair sharing of peer-to-peer resources. In *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS*, volume 2, 2003.

[100] OASIS. Web services business process execution language version 2.0.

[101] OASIS. Reference model for service oriented architecture 1.0, October 2006. http ://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[102] OASIS. Web services context specification (ws-context) version 1.0, 2007.

[103] Justin O'Sullivan, David Edmond, and Arthur Ter Hofstede. What's in a service ? towards accurate description of non-functional service properties. *Distrib. Parallel Databases*, 12(2-3) :117–133, 2002.

[104] Aris M. Ouksel. In-context peer-to-peer information filtering on the web. *SIGMOD Rec.*, 32(3) :65–70, 2003.

[105] C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H. M. W. Verbeek. Formal semantics and analysis of control flow in WS-BPEL (revised version). BPM Center Report BPM-05-15, BPMcenter.org, 2005.

[106] Chun Ouyang, van der Aalst, Wil M. P., Marlon Dumas, ter Hofstede, and Arthur H. M. Translating BPMN to BPEL, January 01 2006.

[107] Kasim Oztoprak and Gozde Bozdagi Akar. Two-way/hybrid clustering architecture for peer to peer systems. In *International Conference on Internet and Web Applications and Services (ICIW 2007), May 13-19, 2007, Le Morne, Mauritius, Proceedings*, pages 11–16, 2007.

[108] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Semantic matching of web services capabilities. 2002.

[109] Massimo Paolucci, Katia Sycara, Takuya Nishimura, and Naveen Srinivasan. Using daml-s for p2p discovery. In *International Conference on Web Services (ICWS03)*, 2003.

[110] Ioannis V. Papaioannou, Dimitrios T. Tsesmetzis, Ioanna Roussaki, and Miltiades E. Anagnostou. A QoS ontology language for web-services. In *AINA*, pages 101–106. IEEE Computer Society, 2006.

[111] M. P. Papazoglou and D. Georgakopoulos. Introduction : Service-oriented computing. *Communications of the ACM*, 46(10) :24–28, October 2003.

[112] Michael P. Papazoglou and Jean-jacques Dubray. A survey of web service technologies. Technical report, 2004.

[113] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *WWW*, pages 553–562, 2004.

[114] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10) :46–52, 2003.

[115] S. L. Peyton Jones, A. Gordon, and S. Finne. Concurrent haskell. In *POPL'96 — Symposium on Principles of Programming Languages*, pages 295–308, St Petersburg, Florida, January 1996. ACM.

[116] Shankar R. Ponnekanti and Armando Fox. SWORD : A developer toolkit for web service composition. January 01 2002.

[117] Preeda Rajasekaran, John A. Miller, Kunal Verma, and Amit P. Sheth. Enhancing web services description and discovery to facilitate composition. In *SWSWPC*, pages 55–68, 2004.

[118] Lakshmish Ramaswamy, Bugra Gedik, and Ling Liu. Connectivity based node clustering in decentralized peer-to-peer networks. *p2p*, 00 :66, 2003.

[119] Shuping Ran. A framework for discovering web services with desired quality of services attributes. In Liang-Jie Zhang, editor, *Proceedings of the International Conference on Web Services, ICWS '03, June 23 - 26, 2003, Las Vegas, Nevada, USA*, pages 208–213. CSREA Press, 2003.

[120] Shuping Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1) :1–10, 2003.

[121] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01 : Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[122] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings.*, pages 1190– 1199. IEEE Computer Society, 2002.

[123] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Appl. Ontol.*, 1(1) :77–106, 2005.

[124] Dumitru Roman, James Scicluna, Cristina Feier, Michael Stollberg, and Dieter Fensel. Ontology-based choreography and orchestration of wsmo services, March 2006.

[125] N. C. Romano, C. Bauer, H. Chen, and J. F. Nunamaker. Quantifying qualitative data for electronic commerce attitude assessment and visualization. january 2000.

[126] Pornpong Rompothong and Twittie Senivongse. A query federation of uddi registries. In *ISICT '03 : Proceedings of the 1st international symposium on Information and communication technologies*, pages 561–566. Trinity College Dublin, 2003.

[127] Antony Rowstron and Peter Druschel. Pastry : scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[128] Daml s Coalition, Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew Mcdermott, David Martin, Sheila A. Mcilraith, Massimo Paolucci, Terry Payne, and Katia Sycara. Daml-s : Web service description for the semantic web. pages 348–363, 2002.

[129] Marta Sabou and Jeff Pan. Towards semantically enhanced web service repositories. *Web Semant.*, 5(2) :142–150, 2007.

[130] Cristina Schmidt and Manish Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2) :211–229, 2004.

[131] Quan Z. Sheng, Boualem Benatallah, Marlon Dumas, and Eileen Oi-Yan Mak. SELF-SERV : A platform for rapid composition of web services in a peer-to-peer environment. In *VLDB*, pages 1051–1054. Morgan Kaufmann, 2002.

[132] Quan Z. Sheng, Boualem Benatallah, Zakaria Maamar, and Anne H.H. Ngu. Configurable composition and adaptive provisioning of web services. *IEEE Transactions on Services Computing*, 2(1) :34–49, 2009.

[133] Kaarthik Sivashanmugam, Kunal Verma, Ranjit Mulye, Zhenyu Zhong, and Amit Sheth. Speed-r : Semantic p2p environment for diverse web service registries, 2004.

[134] Kaarthik Sivashanmugam, Kunal Verma, and Amit Sheth. Discovery of web services in a federated registry environment. In *ICWS '04 : Proceedings of the IEEE International Conference on Web Services*, page 270, Washington, DC, USA, 2004. IEEE Computer Society.

[135] Kaarthik Sivashanmugam, Kunal Verma, Amit Sheth, and John Miller. Adding semantics to web services standards. In Liang-Jie Zhang, editor, *ICWS*, pages 395–401. CSREA Press, 2003.

[136] N. Srinivasan, M. Paolucci, and K. Sycara. Adding owl-s to uddi, implementation and throughput service. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004) 6-9, 2004, San Diego, California, USA*, 2004.

[137] H. Sun, B. Wang, X.and Zhou, and P. Zou. Research and implementation of dynamic web services composition. In *APPT 2003*, pages 457–466. Springer-Verlag Berlin Heidelberg, 2003.

[138] Katia Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Rec.*, 28(1) :47–53, 1999.

[139] Yehia Taher, Djamal Benslimane, Marie-Christine Fauvet, and Zakaria Maamar. Towards an approach forweb services substitution. In *IDEAS*, pages 166–173. IEEE Computer Society, 2006.

[140] Ioan Toma, Dieter Fensel, Matthew Moran, Kashif Iqbal, Thomas Strang, and Dumitru Roman. An Evaluation of Discovery approaches in Grid and Web services Environments. In *The 2nd International Conference on Grid Service Engineering and Management*, Erfurt, Germany, September 2005.

[141] Duc A. Tran, Kien A. Hua, and Tai T. Do. ZIGZAG : An efficient peer-to-peer scheme for media streaming. In *INFOCOM*, 2003.

[142] Mike Uschold and Michael Grüninger. Ontologies : Principles, methods, and applications. *Knowledge Engineering Review*, 11(2) :93–136, 1986.

[143] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. Meteor-s wsdi : A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Information Technology and Management*, 6(1) :17–39, January 2005.

[144] Iris Vessey and Ajay Paul Sravanapudi. Case tools as collaborative support technologies. *Commun. ACM*, 38(1) :83–95, 1995.

[145] W3C. Web services choreography description language version 1.0.

[146] W3C. Web service choreography interface (wsci) 1.0, 2002.

[147] W3C. Web services conversation language (wscl) 1.0, 2002.

[148] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qoS-aware selection model for semantic web services. In Asit Dan and Winfried Lamersdorf, editors, *Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Proceedings*, volume 4294 of *Lecture Notes in Computer Science*, pages 390–401. Springer, 2006.

[149] Yao Wang and Julita Vassileva. Trust-based community formation in peer-to-peer file sharing networks. In *WI '04 : Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 341–348, Washington, DC, USA, 2004. IEEE Computer Society.

[150] S. White. Using BPMN to model a BPEL process. *BPTrends*, 3(3) :1–18, March 2005.

[151] S.A. White. Introduction to bpmn. Technical report, 2004.

[152] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4 :65–85, 1994.

[153] WSMO working group. Web service modeling ontology, 2005.

[154] Li Xiao, Zhichen Xu, and Xiaodong Zhang. Low-cost and reliable mutual anonymity protocols in peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, PDS-14(9) :829–840, September 2003.

[155] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering, March 5-8, 2003, Bangalore, India*, pages 49–60. IEEE Computer Society, 2003.

[156] Xiaochuan Yi and Krys Kochut. A CP-nets-based design and verification framework for web services composition. In *ICWS*, pages 756–760. IEEE Computer Society, 2004.

[157] M. Zeleny. *Multiple Criteria Decision Making.* McGraw-Hill, 1982.

[158] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.

[159] Rongmei Zhang, Ali Raza Butt, and Y. Charlie Hu. Topology-Aware Peer-to-Peer On-demand Streaming. In *NETWORKING 2005 : Networking Technologies, Services, and Protocols ; Performance of Computer and Communication Networks ; Mobile and Wireless Communication Systems, 4th International IFIP-TC6 Networking Conference, Waterloo, Canada, May 2-6, 2005, Proceedings*, volume 3462 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2005.

[160] Xin Y. Zhang, Qian Zhang, Zhensheng Zhang, Gang Song, and Wenwu Zhu. A Construction of Locality-Aware Overlay Network : mOverlay and Its Performance. *IEEE Journal on Selected Areas in Communications*, 22(1) :18–28, 2004.

[161] Wenbing Zhao, Louise E. Moser, and P. M. Melliar-Smith. A reservation-based extended transaction protocol. *IEEE Transactions on Parallel and Distributed Systems*, 19(2) :188–203, 2008.

[162] Wei Zheng, Sheng Zhang, Yi Ouyang, Fillia Makedon, and James Ford. Node clustering based on link delay in p2p networks. In *SAC '05 : Proceedings of*

the 2005 ACM symposium on Applied computing, pages 744–749, New York, NY, USA, 2005. ACM.

[163] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee. Daml-qos ontology for web services. *Web Services, IEEE International Conference on*, 0 :472, 2004.