HAL
open science

TCOWS - Canevas pour la composition de services web avec propriétés transactionnelles

Helga Duarte

To cite this version:

Helga Duarte. TCOWS - Canevas pour la composition de services web avec propriétés transactionnelles. Réseaux et télécommunications [cs.NI]. Université Joseph-Fourier - Grenoble I, 2007. Français. NNT : . tel-00188206v2

HAL Id: tel-00188206

https://theses.hal.science/tel-00188206v2

Submitted on 29 Nov 2007

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

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

UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ GRENOBLE - ALPES

GRENOBLE I

École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique - MSTII

Tcows

Canevas pour la composition de services web avec propriétés transactionnelles

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Discipline : Informatique

présentée et soutenue publiquement par

Helga DUARTE-AMAYA

le 13 novembre 2007

Directrice de thèse
Professeure Marie Christine FAUVET

Composition du Jury

Président :	Jacky Estublier - Chercheur au CNRS à Grenoble
Rapporteurs :	Michel Léonard - Professeur à l'Université de Genève
	Farouk Toumani - Professeur à l'Université de Clermont-Ferrand II
Examinateur :	Marlon Dumas - Associate-Professor à QUT, Brisbane - Australie
Directrice de thèse :	Marie Christine Fauvet - Professeure à l'Université Joseph Fourier

L I G Thèse préparée au sein du Laboratoire d'Informatique de Grenoble - LIG, dans l'Équipe MRIM - France

À la Providence Divine qui m'a tout donné

REMERCIEMENTS

Je suis arrivée à la fin de cette thèse, et je dois dire que je n'y suis pas arrivée toute seule. La liste de toutes les personnes qui d'une manière ou d'une autre m'ont aidée à arriver jusqu'ici est vraiment longue et j'essayerai de n'oublier personne :

Tout d'abord, tous mes remerciements au Programme Alßan[1], Programme de Bourses de Haut Niveau de l'Union Européenne pour l'Amérique Latine, pour son soutien financier. J'ai eu une bourse pendant trois ans[2] qui m'a permis de réaliser ce projet et sans laquelle il aurait été très difficile, voir impossible, d'accomplir.

Un grand merci à l'Université Nationale de Colombie[3], mon lieu de travail à Bogotá, qui m'a aussi donné son soutien financier sous la figure d'une *Comisión de Estudios*.

Je veux exprimer ma plus vive reconnaissance et gratitude à ma directrice, Marie Christine Fauvet, Professeur à l'université Joseph Fourier de Grenoble, d'abord pour tout ce qu'elle m'a appris pendant le développement de cette thèse, mais aussi pour ses encouragements et son aide à faire aboutir ce travail.

Je remercie tous les membres du laboratoire CLIPS[4], devenu plus tard le laboratoire LIG, pour m'avoir bien accueillie. Le temps de ma thèse s'est écoulé ici, où j'ai vécu une partie de ma vie, aussi.

Un grand merci à Bernard Cassagne par son aide continue à configurer ma machine, laquelle, des fois refusait de travailler avec moi. Heureusement, Bernard a été toujours là pour la faire marcher.

Je tiens à remercier M. Jacky Estublier, Directeur de Recherche au CNRS pour avoir accepté de présider ce jury. Je veux aussi le remercier de m'avoir mise en contact avec Marie-Christine Fauvet au moment de ma recherche d'un sujet de thèse.

Je tiens à remercier M. Michel Léonard, Professeur à l'université de Genève et M. Farouk Toumani Professeur à l'université de Clermont-Ferrand pour avoir accepté d'être rapporteurs de cette thèse.

[1] http ://www.programalban.org
[2] N° d'identification E03D13487CO
[3] http ://www.unal.edu.co
[4] Communication Langagière et Interaction Personne-Système

Je tiens à remercier aussi Marlon Dumas, *Associate-Professor* pour m'avoir fait l'honneur de faire partie du jury et qui m'a accueillie au sein de son équipe de recherche à *Queensland University of Technology* à Brisbane en Australie. Une expérience scientifique très enrichissante que je garderai toujours dans ma mémoire.

Je remercie Cécile Lefort et Sandra Michelet pour leurs relectures et leurs conseils au moment de la rédaction de ce manuscrit.

Un grand merci à la famille Villedieu, qui m'a offert son amitié, m'a aidée à m'installer à Saint Martin d'Hères et surtout, a gardé ma fille comme la sienne pendant mes voyages professionnels.

Je remercie Line Corbel pour toute l'aide qu'elle m'a apportée et pour avoir aussi gardé ma fille quand j'en ai eu tant besoin.

Je tiens à remercier Carlos et Liliana pour m'avoir aidée à emménager et à me faire connaître les premières démarches administratives à Grenoble.

Merci Edith et Pilar Villamil pour les bons moments que vous avez partagés avec moi et avec ma fille.

Merci Guillermo et Amelita, j'ai trouvé de l'amitié, de la compagnie et de l'aide auprès de vous.

Merci Cécile pour le temps que tu as dédié à faire des échanges linguistiques avec moi, cela m'a beaucoup aidée à améliorer mon français. Et merci surtout pour la confiance que tu m'as faite et l'amitié que tu m'as offerte.

Merci Helena pour tes prières et tes conseils.

Merci Gabriel pour toute ton aide.

Merci Face pour ta compagnie dans la distance.

Je remercie de tout mon cœur ma sœur et mon père pour m'avoir écouté pendant mes jours d'angoisse.

Je tiens à remercier Ricardo, qui m'a soutenu à sa façon et ma fille, pour avoir su supporter mes absences pendant le développement de ce travail.

TABLE DES MATIÈRES

1 Introduction **1**

1.1 Contexte de la thèse 2

1.2 Problématique 3

1.3 Scénario illustratif 4

1.4 Contributions de la thèse 7

1.5 Plan de la thèse 8

I Etude bibliographique **9**

2 Contexte de référence **11**

2.1 Transactions dans l'évolution des architectures logicielles 11

2.1.1 Transactions dans les bases de données 11

2.1.2 Architectures d'un système d'information 13

2.1.3 Intégration d'applications et transactions 14

2.2 Intégration d'applications et Internet 17

2.2.1 Intégration d'applications inter-entreprise 17

2.2.2 Services et services web 18

2.2.3 Composition de services web 21

2.3 Synthèse 24

3 Aspects transactionnels dans les services web **25**

3.1 Introduction 26

3.2 Caractéristiques consensuelles identifiées 26

3.2.1 Généralités 26
3.2.2 Propriétés ACID 28
3.2.3 Mécanismes de compensation 29
3.3 Standards pour le web 31
3.3.1 Protocoles spécifiques pour les transactions sur le web 31
3.3.2 Protocoles spécifiques pour la composition 33
3.4 Approches académiques pour la composition 39
3.4.1 Approches algorithmiques 39
3.4.2 Plates-formes pour la composition de services web 40
3.4.3 Approches dirigées par les modèles 42
3.5 Composition et aspects transactionnels 43
3.5.1 Séparation des préoccupations 43
3.5.2 Limitation de la portée des transactions 44
3.5.3 Prise en compte des besoins non fonctionnels des applications 45
3.5.4 Canevas et architectures 45
3.6 Synthèse 46
3.6.1 Protocoles standards 46
3.6.2 Approches spécifiques pour la composition 48
3.6.3 Contributions de Tcows 48

II Tcows - Transactional Compositions Of Web Services 51

4 Sélection de services 53
4.1 Motivations et travaux reliés 54
4.2 Modèle de communautés 56
4.2.1 Description informelle 57
4.2.2 Communautés et modèles de qualité 58
4.2.3 Sélection de services 60
4.3 Illustration 61
4.3.1 Interface abstraite vs. interface des services 61
4.3.2 Sélection de services et modèle de qualité 62
4.4 Conclusion 65

5 Tcows : application au scénario “Agence de Voyages” 67
5.1 Préambule 67
5.2 Conception de compositions transactionnelles 68

5.2.1 Propriétés transactionnelles des services composants 69
5.2.2 Modèle d'orchestration de la composition 70
5.2.3 Paramètres de la composition 72
5.3 Sélection des participants d'une composition 75
5.4 Exécution de composition transactionnelle 79
5.4.1 Services tous atomiques 79
5.4.2 Services tous quasi-atomiques 81
5.4.3 Services non-atomiques 84
5.5 Conclusion 84

6 Formalisation 87
6.1 Phases du processus mis en œuvre dans `Tcows` 87
6.2 Formalisation à base de types abstraits 89
6.2.1 Phase de conception 89
6.2.2 Phase de sélection 90
6.2.3 Phase d'exécution 94
6.3 Conclusion 95

7 Mise en œuvre du canevas `Tcows` 97
7.1 Architecture logicielle de `Tcows` 97
7.2 Détails de la mise en œuvre 100
7.2.1 Module de communication 100
7.2.2 Module de construction des options 100
7.2.3 Module de réécriture 102
7.3 Mise en œuvre du modèle de communauté 105
7.3.1 Modèle de données 105
7.3.2 L'application web 106
7.3.3 Le service web 106
7.4 Conclusion 107

III Bilan et perspectives 109

8 Conclusion et perspectives 111
8.1 Conclusion 111
8.2 Perspectives 112
8.2.1 Phase de conception 112
8.2.2 Phase de sélection 112

8.2.3 Phase d'exécution 114
8.2.4 Communautés de services web 114
8.2.5 Validation expérimentale 115

IV Annexes **117**

A WSDL de la communauté Vol **119**

B Détails spécifiques de la mise en œuvre **125**
B.1 Le modèle de composition `Tcows` 125
B.2 Les communautés de services web 126

C Publications **133**

Bibliographie **135**

TABLE DES FIGURES

1.1 Les capacités impliquées dans l'organisation d'un voyage 5
1.2 Différentes configurations pour la composition d'un voyage 5
1.3 Les paramètres de la composition 6
1.4 Les capacités et les instances de services 6

2.1 Protocole de validation à deux phases (diagramme de séquences UML) . . . 12
2.2 Les différentes architectures d'un Système d'Information [ACKM03] 14
2.3 Architecture d'un moniteur transactionnel 15
2.4 Interactions dans les architectures à base de services web 20
2.5 Modèle d'orchestration de l'application "Agence de voyage" 22
2.6 Modèle de chorégraphie de l'application "Agence de voyage" 23

3.1 Régions atomiques dans un modèle d'orchestration 30
3.2 Les différents protocoles pour la gestion des transactions 31
3.3 Langages de composition de services web (adapté de [JMS06]) 34

4.1 Éléments principaux des communautés 57
4.2 Modèle d'interactions entre un client et un service 62
4.3 Mise en correspondance entre un client et une communauté 63

5.1 Modèle d'orchestration du scénario "Agence de Voyages" (vision grossière) . 71
5.2 Le modèle de l'orchestration détaillée 73
5.3 La zone transactionnelle de l'orchestration 74
5.4 Mise en correspondance des opérations (services atomiques) 80
5.5 Mise en correspondance des opérations (services quasi-atomiques) 82

5.6 Un service quasi-atomique et un service non-atomique 83

6.1 Phases du processus proposé dans Tcows 88

7.1 Architecture logicielle de Tcows 98
7.2 Module de communication 99
7.3 Vue globale d'une communauté 105
7.4 Modélisation des informations d'une communauté 106

8.1 Schéma général du Canevas de composition et les différentes perspectives . 113

A.1 L'interface abstraite WSDL de la communauté Vol 119

B.1 Document XML décrivant les adresses urls des communautés 126
B.2 Les adresses de services web par communauté 127
B.3 Fichier en format XML de la liste initiale d'options 128
B.4 Relation qui implémente le modèle de qualité de la communauté 129
B.5 Fichier en format XML du modèle de qualité de la communauté Vol 129
B.6 Modèle relationnel de la base de données de la communauté Vol 130
B.7 Modèle de la base de données pour les communautés 130
B.8 L'"Application web" de la communauté Vol 131

LISTE DES TABLEAUX

3.1 Identification de propriétés transactionnelles selon [MTSM02] 30
3.2 Synthèse des approches spécifiques que pour la composition 49
3.3 Synthèse des approches : composition et transaction 50

4.1 Les valeurs des critères du modèle de qualité 64

5.1 Valeurs des critères de qualité des services sélectionnés 76
5.2 Prix et propriétés transactionnelles des services 77
5.3 Scores par option 78

7.1 Nombre de types d'options 101

B.1 Le modèle de qualité de la communauté Vol 126

CHAPITRE 1

INTRODUCTION

Le premier chapitre de cette thèse a par but d'introduire et de présenter, de manière générale, notre travail de recherche. Nous situons notre approche dans le contexte de la composition de services web, et nous identifions la problématique existante en ce qui concerne la gestion des transactions dans ce contexte. Ce chapitre montre les objectifs fixés et décrit, via un exemple, le problème que nous allons résoudre. Finalement les contributions de notre approche sont présentées.

Depuis l'invention du *World Wide Web* par Tim Berners-Lee et Robert Cailliau[1] et le développement des technologies associées, le web et l'internet sont devenus bien plus qu'un simple instrument de partage d'information. La manière dont les systèmes d'information interagissent au travers des réseaux, et la façon dont les applications sont développées ont été complètement remises en cause. En effet, l'internet fournit un moyen universel aux organisations pour composer leurs applications (on parle alors de *services*), partager leurs ressources et savoir-faire, afin de minimiser leurs coûts, d'offrir de nouvelles applications à valeur ajoutée, sans pour autant perdre de leur autonomie. Ainsi, l'évolution de ces systèmes d'information et le développement de processus métiers entre plusieurs entreprises ont fait du web le support idéal des interactions inter processus. Cependant, la mise en œuvre de processus métiers interagissant sur le web reste une tâche complexe. Le concept de service web, basé sur les standards de l'internet, vise à faciliter le développement de ce type de processus. Cependant, chaque entreprise a ses propres règles de gestion et donc ses propres services. Ces derniers, qui vont devoir interagir avec des services provenant d'autres entreprises, doivent être traités comme des boîtes noires, où seulement les interfaces qu'ils fournissent sont connues (dans le chapitre 2 nous donnons les définitions relatives à ces notions).

Les services web ont été créés pour faciliter les interactions entre plusieurs partenaires dans le but de produire un service à valeur ajoutée. Mais, paradoxalement, le développement de services créés par chaque entreprise de manière autonome a donné lieu à une hétérogénéité

[1] Au sein du CERN à Genève, en 1989

qui pose divers problèmes au moment de l'exécution de la composition obtenue, surtout lorsque celle-ci est munie de propriétés transactionnelles.

L'étude présentée dans ce document nous a permis d'identifier les problèmes liés d'une part, à la composition de services web, et d'autre part à l'association de propriétés transactionnelles à cette composition. C'est en nous intéressant à ces deux problématiques que nous avons conçu une plate-forme de composition de services dont le principal objectif est de maximiser les chances pour une exécution de réussir, tout en satisfaisant au mieux les besoins des clients[2] de la composition.

Dans la suite de chapitre, la section 1.1 introduit le contexte de notre approche et la section 1.2 présente la problématique dans cette thèse. La section 1.3 illustre, via un exemple, les problèmes que nous tentons de résoudre. Finalement, la section 1.4 présente les contributions de notre travail et la section 1.5 donne le plan de cette thèse.

1.1 Contexte de la thèse

Dans la perspective de réagir mieux et plus vite aux sollicitations des marchés, les entreprises doivent faire face à l'intégration et à l'automatisation de leurs différentes unités organisationnelles, ainsi qu'à la construction de partenariats. Il en découle une problématique d'intégration de systèmes d'information hétérogènes et répartis. Le problème a été posé dans un premier temps dans un contexte intra-entreprise. Des outils ont été proposés pour cela dans la classe des EAIs (*Intégration d'Applications d'Entreprise*). Ces outils sont dédiés à un domaine d'application particulier, non extensibles car basés sur des solutions propriétaires, et lourds à mettre en œuvre. Les systèmes de gestion de flot de tâches (*workflows*) et les intergiciels (*middleware*) sont des technologies qui ont été développées, entre autre, afin de masquer l'hétérogénéité des systèmes à intégrer et de décrire de manière explicite la logique d'exécution d'un processus métier qui s'appuie sur plusieurs applications intra-organisation [Eme00].

Comme dans le contexte des EAIs, le processus métier est ici considéré selon une modalité d'interaction dite d'applications vers applications (*Application-to-Application*, A2A). Cette modalité s'oppose à celle dite de métier à métier (*Business-to-Business*, B2B). L'intégration d'applications inter-organisation est alors abordée par le biais de la mise en place de "serveurs d'applications". Bien que les applications de type EAI et B2B partagent le même objectif, qui est de fournir des plates-formes pour l'intégration d'applications, les méthodes pour leur mise en place diffèrent. Dans le cadre de l'intégration intra-entreprise (EAI), la gestion des applications à intégrer est centralisée, les schémas d'interactions entre ces applications sont statiques car toutes les applications appartiennent à la même organisation. En revanche, dans le cadre de l'intégration inter-entreprise (B2B), les applications à intégrer appartiennent à des organisations différentes, il s'en suit que la gestion ne peut plus être centralisée, et que les schémas d'interactions évoluent dans le temps.

Les systèmes d'information distribués ont évolué vers des architectures à base de services (*Service Oriented Architecture - SOA* [DD04, BDDM05]). Selon ce modèle d'architecture, les processus métiers des entreprises sont encapsulés par des services qui interagissent les

[2]Dans le cadre de la composition de services, le client n'est pas, le plus souvent, une personne mais une application ou un autre service.

uns avec les autres par le biais d'échanges de messages. Ce schéma d'interactions faiblement couplées permet d'élargir le spectre des alliances possibles. Ces services sont appelés *services web* lorsqu'ils s'appuient sur les technologies du web pour interagir et communiquer les uns avec les autres. Un service web est un logiciel, répondant à un ensemble de besoins fonctionnels, indépendant de la plate-forme d'exécution, auto-descriptif et qui peut être découvert et exécuté via l'internet. Les efforts de standardisation effectués dans ce cadre ont permis de définir des protocoles standards qui permettent de masquer l'hétérogénéité des applications. Les technologies web et les standards qui en découlent permettent d'envisager l'émergence de solutions technologiques pour faciliter l'intégration d'applications accessibles par l'Internet [Kal02]. Une définition détaillée et argumentée des notions mentionnées ici est donnée dans le chapitre 2.

Ainsi, dans le domaine de l'intégration et de l'automatisation des interactions de processus métier inter-entreprise et intra-entreprise, les services web semblent constituer une solution prometteuse [BCC+02, BDM02, BCTH03]. Mais la transition entre les solutions existantes et les nouvelles technologies est progressive de telle sorte que les plates-formes des intergiciels existants sont étendues avec des extensions appropriées pour développer, déployer et maintenir des services web. De plus, les standards proposés ne couvrent qu'une partie des aspects des services : la description, la coordination, la sécurité, ou la gestion des transactions, etc. En particulier, lorsqu'il s'agit de composer des services, les modèles visent à standardiser la description des services et à automatiser leur coordination. Cependant, ces modèles se limitent à la prise en compte des besoins fonctionnels laissant de côté les aspects extra fonctionnels, en particulier ceux liés à la notion de qualité de services ou de propriétés transactionnelles. Le chapitre 3 discute de ces modèles de manière détaillée.

C'est dans ce contexte que se place la recherche présentée dans cette thèse. Dans la section suivante nous posons les problèmes dont l'étude est rapportée dans ce document.

1.2 Problématique

Une application obtenue par composition de services web possède des caractéristiques spécifiques. En premier lieu, les interactions entre les services sont de durées variables et s'inscrivent le plus souvent dans un contexte inter-organisationnel. Ensuite, les services impliqués ont été conçus et implantés indépendamment les uns des autres, et non dans la perspective de participer à une composition. En conséquence, lorsqu'il s'agit de munir une composition de services de propriétés extra fonctionnelles, en particulier de propriétés transactionnelles, les principes habituellement utilisés dans les systèmes distribués ne sont plus adaptés. Au niveau conceptuel, les propriétés ACID [Elm90] se révèlent être trop restrictives, voire inapplicables, car les interactions sont de longue durée, elles transitent par plusieurs entreprises, elle s'appuient sur différentes applications et requièrent des ressources qui peuvent être distribuées entre plusieurs partenaires. Il s'avère que dans ce contexte, seule la propriété d'atomicité est pertinente à condition de l'assouplir. Au niveau de l'implantation, les protocoles de validation de transactions distribuées (comme par exemple le protocole de validation à deux phases, *Two Phase Commit* - 2PC) ne sont plus applicables, soit parce que les partenaires entrant dans la composition ne disposent pas des opérations requises, soit parce qu'il n'est pas possible de définir, *a priori* de coordinateur central. Il peut aussi arriver que certains partenaires n'acceptent pas de verrouiller

des ressources. Une solution à ce dernier problème consiste à établir des contrats par une phase de négociation préalable. Cette pratique rend les applications rigides et difficiles à adapter aux évolutions des besoins des marchés. De plus, chaque fois qu'un partenaire joint (ou quitte) la composition, le contrat doit être renégocié.

Dans le domaine des services web, des outils ont été développés pour concevoir la composition des applications, leur implantation et leur exécution. Dans ce cadre, il existe de nombreux langages pour décrire des processus métiers, dans les chapitres 2 et 3 nous montrons les caractéristiques et les limites de ces spécifications. En général, ces langages permettent de modéliser les processus à très bas niveau par le biais d'appels à des opérations offertes par les services web partenaires. Concernant l'association de propriétés transactionnelles à une composition, les études sont moins nombreuses et les solutions qui en sont issues sont partielles. Les langages et outils disponibles permettant de programmer des transactions, qui s'appuient sur des services web, ne fournissent pas de concepts de haut niveau pour : (i) exprimer les propriétés transactionnelles désirées au niveau du service composé ; (ii) assurer ces propriétés de façon automatisée en exploitant les propriétés transactionnelles des services composants. Il s'avère en effet que certains services web, en particulier dans le domaine du commerce électronique, ont des propriétés transactionnelles inhérentes [BBCT04]. Ceci est le cas notamment des services associés à la gestion de ressources (au sens large), comme par exemple la réservation de chambres d'hôtel, de places de spectacle, de services professionnels, etc. Des standards ont été proposés (par exemple, *WS-Transaction* avec *WS-Coordination*), mais leur portée est limitée et les compositions obtenues sont peu flexibles. Dans le chapitre 3 nous détaillons ces approches ainsi que leurs apports et leurs limites.

Il est courant que de nombreux services répondent à un même ensemble de besoins fonctionnels. Ces services se distinguent les uns des autres par leurs propriétés extra fonctionnelles. La plupart des propositions existantes pour la composition de services web ne tirent pas partie de cette propriété : au moment de la conception, les services web participant à la composition sont fixés. Si, au moment de l'exécution, un des composants n'est pas disponible le processus doit s'arrêter et attendre l'intervention du concepteur qui peut, soit relancer l'exécution de la même composition, soit remplacer le service défaillant dans la composition avant de la relancer.

En résumé, les questions soulevées par la discussion ci-dessus sont :

- Comment spécifier parmi un ensemble de participants potentiels, les services qui entrent dans la composition de manière à maximiser les chances pour que la composition s'exécute avec succès ?
- Comment exprimer la propriété d'atomicité d'une composition de services ?
- Étant données les propriétés transactionnelles hétérogènes des services participants, comment exécuter la composition avec la propriété d'atomicité spécifiée ?

1.3 Scénario illustratif

Nous choisissons d'illustrer la problématique introduite ci-dessus par un scénario dans lequel une application est conçue pour l'organisation de voyages (voir figure 1.1).

L'organisation d'un voyage nécessite l'achat d'un billet d'avion, la réservation d'une

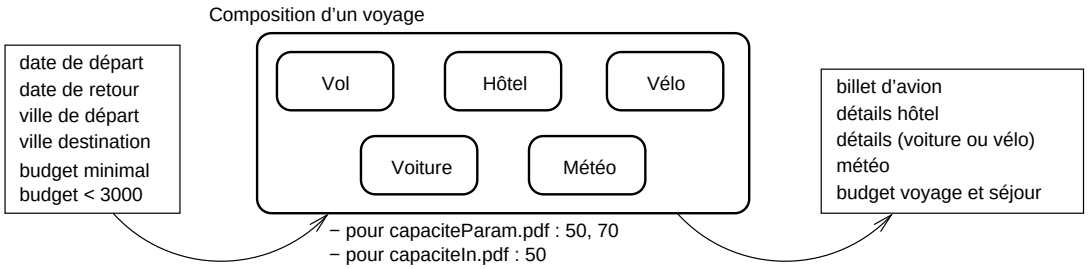


FIG. 1.1 – Les capacités impliquées dans l'organisation d'un voyage

chambre d'hôtel, le cas échéant la location d'une voiture ou d'un vélo et la consultation de la météo pour le lieu de la destination. Chacune de ces tâches s'appuie sur un ensemble d'opérations que nous regroupons, dans la perspective d'en faire abstraction, sous le terme de *capacité* de service. La figure 1.1 montre par un graphique les capacités entrant de la composition qui découle de cette analyse. Cette étape de la conception fait abstraction du service qui remplira effectivement la capacité identifiée.

L'étape suivante consiste à déterminer les capacités parmi celles décrites plus haut, qui sont indispensables à l'élaboration du voyage. En particulier, le voyage est possible même lorsque seuls le vol et la chambre d'hôtel sont réservés (voir figure 1.2(a)). D'autres configurations sont acceptables : le vol et l'hôtel avec vélo ou voiture (voir figures 1.2(b) et 1.2(c)), ou encore le vol et l'hôtel avec la météo, mais sans véhicule (voir figure 1.2(d)).

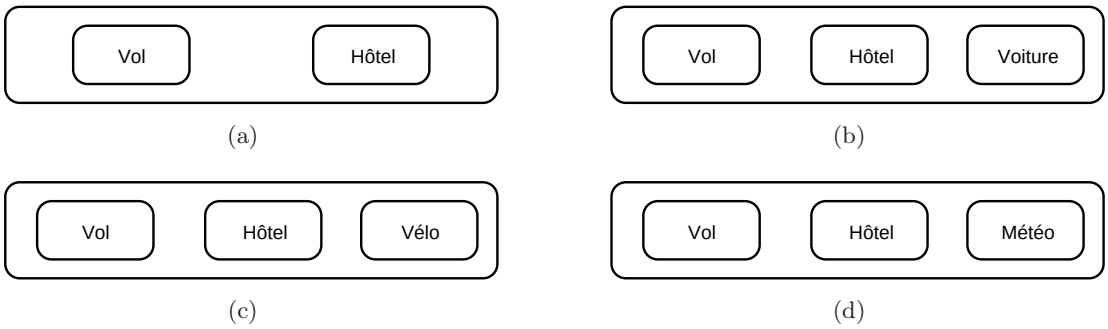


FIG. 1.2 – Différentes configurations pour la composition d'un voyage

La prise en compte de ces différentes configurations pour un voyage nécessite d'assouplir la propriété transactionnelle d'atomicité : les capacités impliquées dans la composition ne doivent pas toutes nécessairement s'exécuter avec succès. L'exécution de certaines peut échouer, par exemple parce qu'aucun service n'est disponible pour répondre aux besoins de la capacité, ou bien parce que la ressource désirée ne peut pas être acquise. Cet échec peut se produire sans pour autant avoir comme conséquence l'annulation de l'exécution en cours de la composition. Ainsi, un premier besoin qui se dégage est la possibilité pour le concepteur de l'application de déterminer, au moment de la conception, parmi les capacités entrant dans la composition, celles qui doivent s'exécuter avec succès, en d'autres termes, de fixer les capacités requises (obligatoires), les autres étant facultatives.

Des paramètres sont associés à la composition permettant ainsi de fixer les données en entrée et celles attendues en sortie (voir figure 1.3). Certains de ces paramètres sont

des fonctions de restriction (*le budget du voyage doit être inférieur à 3000*) d'autres expriment des préférences (*le budget du voyage doit être minimisé*). Les fonctions de restriction doivent être satisfaites pour que l'exécution se termine avec succès. Les fonctions de préférences doivent être maximisées.

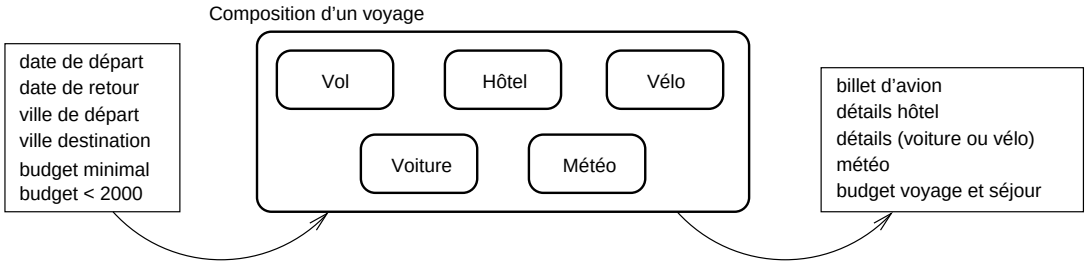


FIG. 1.3 – Les paramètres de la composition

L'étape suivante consiste à associer un service à chaque capacité identifiée. Puisqu'il existe plusieurs services web qui répondent aux même besoins fonctionnels, la question qui se pose alors est : où et comment trouver le service qui répond le mieux aux besoins requis par une capacité ? La notion de *référentiel* ou *communauté* de services nous fournit un cadre pour effectuer la sélection d'un service parmi ceux en concurrence pour une capacité donnée (voir figure 1.4).

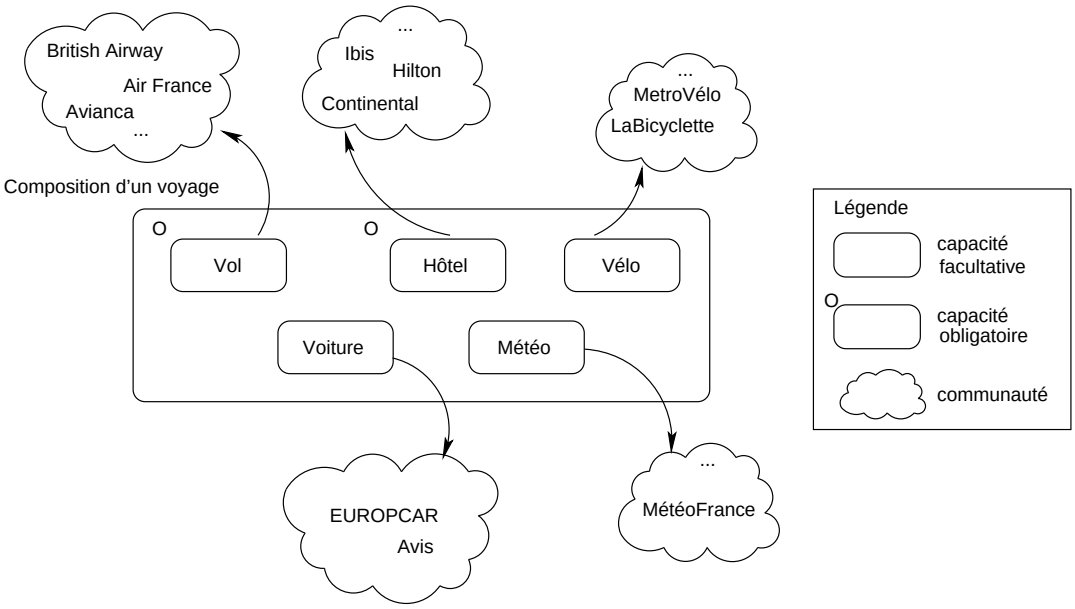


FIG. 1.4 – Les capacités et les instances de services

Le fait de disposer de référentiels où sélectionner plusieurs services spécifiques et appropriés à la composition, facilite la construction d'un ensemble d'instances pour chaque configurations possibles. Nous désignons par *option* chacune de ces instances. Par exemple, la composition du service Air France avec le service Hilton constitue une option. Une autre option est constituée des services Air France, Hilton et Avis. Chaque option vérifie les

fonctions de restriction, et elles peuvent être classées selon leur degré de satisfaction des fonctions de préférence.

La connaissance des propriétés transactionnelles de services à considérer permet de construire des options dont les chances de réussite sont maximales. Ainsi, la paramétrisation de la composition est utilisée pour guider la construction et la classification de ces options par rapport aux paramètres spécifiés.

En conclusion, pour augmenter les chances de réussite d'une composition, nous prônons (i) l'assouplissement de la propriété d'atomicité par la restriction de sa portée au minimum requis, (ii) la paramétrisation de la composition pour guider la sélection des participants potentiels, (iii) la sélection appropriée de services web avec des propriétés transactionnelles.

1.4 Contributions de la thèse

La thèse présentée ici vise à définir un modèle de "composition de services web munis de propriétés transactionnelles". Nous nous intéressons plus spécifiquement à des propriétés d'atomicité. Le modèle permet la prise en compte des propriétés d'atomicité des composants. Tirant partie de l'existence d'un grand nombre de services répondant aux mêmes besoins fonctionnels et afin d'augmenter les chances pour les compositions munies de propriétés transactionnelles de s'exécuter avec succès, l'étude s'appuie sur un principe de sélection de services, à l'exécution. Le concept de "communautés de services web" fournit un cadre à cette sélection.

Les contributions du travail, rassemblées dans le canevas `Tcows`, sont :

- Un modèle dans lequel la composition s'exprime en termes de capacités de services et non pas en termes de services web spécifiques. Une capacité de services est associée à un ensemble de besoins fonctionnels souhaités (par exemple la réservation d'une chambre d'hôtel). Le modèle permet au concepteur d'établir parmi les capacités entrant dans la composition, lesquelles sont obligatoires.
- Un modèle de composition avec des propriétés transactionnelles et permettant la prise en compte des propriétés transactionnelles hétérogènes des services composants.
- Un modèle dans lequel il est possible de paramétrer la composition par des contraintes et des préférences fixées par le concepteur et qui permettent de guider la sélection des services, cette sélection étant effectuée au moment de l'exécution de la composition.
- Une validation expérimentale du modèle proposé par le biais de la mise en œuvre d'un prototype.

Le canevas proposé possède les caractéristiques suivantes :

- *Modulaire* : les préoccupations concernant la conception de la composition sont séparées de celles concernant la spécification de ses propriétés transactionnelles. Cette spécification est faite indépendamment des propriétés transactionnelles hétérogènes des services web composants.
- *Flexible* : il est possible de déterminer, au moment de la conception, quelles capacités de services sont obligatoires et quelles sont optionnelles pour une composition donnée. Cette caractéristique permet de relâcher la propriété d'atomicité pour la transaction et permet d'obtenir des compositions différentes selon la disponibilité de composant dans un moment d'exécution donné.

– *Adaptable* : les services web sont choisis au moment de l'exécution. Cette sélection, s'appuyant sur le concept de communautés, permet de choisir les services les mieux adaptés au contexte de l'exécution en termes de leurs propriétés non fonctionnelles dont en particulier leurs caractéristiques transactionnelles.

1.5 Plan de la thèse

Ce chapitre a présenté une introduction à notre travail de recherche, suivi de la problématique que nous avons traitée, des objectifs que nous poursuivons et des contributions que nous avons apportées.

Notre document a été divisé en trois parties. La première partie est une étude bibliographique, elle-même organisée en deux chapitres : le chapitre 2, présente l'évolution des systèmes de transactions dans les architectures pour les systèmes d'information. Ce chapitre montre en quoi ces systèmes de transactions ne sont plus valides pour la gestion des interactions entre des applications sur internet. Ce chapitre introduit aussi les caractéristiques principales de la technologie des services web et sa problématique. Le chapitre 3 discute des divers travaux qui se sont attaqués au problèmes de la gestion des transactions dans les services web. Nous présentons les avantages et les limites des standards les plus répandus ainsi que les différents approches spécifiques proposées par la recherche académique et industrielle.

La deuxième partie du document présente notre proposition de canevas pour la composition de services web munis de propriétés transactionnelles. Cette partie est formée de quatre chapitres. Le chapitre 4 présente le concept de communauté sur lequel s'appuie notre modèle de composition. Il s'agit de formaliser et de mettre en œuvre un référentiel de services web permettant de pallier les limites et la fermeture d'UDDI. Cette approche nous permet aussi de donner un cadre à la sélection de services web. Le chapitre 5 illustre l'utilisation du canevas `Tcows` sur un exemple concret. Dans le chapitre 6, nous détaillons et formalisons les mécanismes proposés dans `Tcows`. Finalement, avec le chapitre 7, nous présentons les détails de la mise en œuvre du canevas proposé.

La troisième partie est constituée du seul chapitre 8 qui dresse le bilan de notre travail et qui donne les perspectives que notre travail a ouvert et effectue la liaison avec d'autres travaux de recherche en cours.

Première partie

Etude bibliographique

CHAPITRE 2

CONTEXTE DE RÉFÉRENCE

Ce chapitre présente l'évolution des notions liées aux transactions en regard de celle des différentes architectures logicielles jusqu'à l'émergence des architectures à base de services web. La section 2.1 montre, sans trop rentrer dans les détails, comment les systèmes de transactions ont évolué et se sont adaptés au fur et à mesure que les architectures logicielles ont évolué. Ensuite, dans la section 2.2, nous précisons les concepts liés aux architectures à base de services web en considérant les aspects de leur définition jusqu'à leur composition. Nous terminons par une synthèse (voir section 2.3) où nous identifions des problèmes posés par la mise en œuvre d'applications transactionnelles basées sur des services web.

2.1 Transactions dans l'évolution des architectures logicielles

Cette section présente la manière dont les systèmes d'information ont évolué sur le plan de leurs architectures. Les environnements principaux pour leurs mises en œuvre sont présentés. Cette étude nous permet de montrer comment les modèles de transactions ont été utilisés, et comment ils ont dû être adaptés au fil des évolutions des évolutions.

2.1.1 Transactions dans les bases de données

Les protocoles de transaction jouent un rôle très important dans le fonctionnement des applications qu'elles soient centralisées ou distribuées. Le protocole de transaction est l'un des éléments les plus importants pour réussir l'intégration des applications. Cette sous-section commence par donner les définitions classiques des transactions et en décrit l'utilisation à travers l'évolution des systèmes d'information et des intergiciels[1].

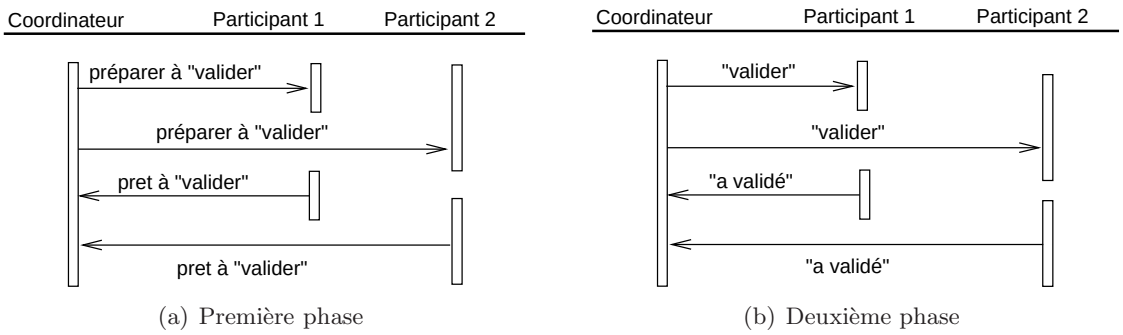
[1] Un intergiciel est un intermédiaire de communication entre des applications complexes, distribuées sur un réseau informatique.

Dans le domaine des bases de données, le concept de transaction a fait l'objet d'efforts de recherche, nombreux, importants et continus (voir par exemple [Elm90, GR93, WV02]). Une transaction est une séquence d'opérations qui visent à transformer un état d'une base de données (ou plus largement du contexte d'une application) en un autre état. Une transaction possède les propriétés suivantes :

- **Atomicité :** toutes les opérations de la transaction sont exécutées, ou aucune. La propriété d'atomicité est assurée par le biais d'un mécanisme qui défait les effets de la transaction lorsque celle-ci doit être annulée, soit parce qu'elle viole les contraintes d'intégrité soit parce qu'elle n'a pas pu être achevée en raison de défaillances du système.
- **Cohérence :** : une transaction est une transformation de la base de données (ou plus généralement du contexte de l'application) d'un état cohérent en un autre état cohérent. Si à l'issue de la transaction, un contrainte d'intégrité est violée, la transaction est annulée, sinon elle est validée.
- **Isolation :** les transactions bien qu'exécutées en concurrence, le sont indépendamment les unes des autres. Les effets non validés d'une transaction ne sont pas visibles des autres transactions.
- **Durabilité :** une fois que la transaction est validée les changements qu'elle a provoqués doivent survivre aux défaillances.

L'ensemble de ces propriétés est connu sous le nom de propriétés ACID [Gra81, GR93, Alo05]. Ces propriétés ont tout d'abord fait l'objet d'un consensus, puis des travaux ont porté sur le développement de techniques permettant de vérifier que les propriétés ACID sont respectées.

L'atomicité est garantie par des mécanismes d'annulation et de validation de la transaction et la durabilité par des mécanismes de stockage des informations sur des supports stables. Un système de contrôle de la concurrence permet de garantir l'isolation des transactions. Le maintien de la cohérence des données s'appuie d'une part sur le système de contrôle de la concurrence et d'autre part sur un ensemble de contraintes d'intégrité issues du modèle de données, du schéma de la base de données et des règles de gestion fournies par l'application. Dans le domaine des bases de données, les transactions sont le plus souvent de courte durée. Cette hypothèse facilite l'implantation des mécanismes assurant des transactions avec les propriétés ACID.



(a) Première phase (b) Deuxième phase

Fig. 2.1 – Protocole de validation à deux phases (diagramme de séquences UML)

Lorsque les systèmes d'information distribués sont apparus, les systèmes sous-jacents pour la gestion des transactions ont dû être adaptés en particulier pour permettre la gestion

de transactions dans des bases de données réparties. Les architectures à deux niveaux (un serveur et des clients) ont été le premier pas vers les systèmes modernes d'information distribués. De protocoles de programmation et de communication ont été introduits pour supporter les interactions entre le serveur et les clients (voir plus loin, figure 2.2(b)). Parmi ces protocoles, RPC *(Remote Procedure Call)* [Blo92] est sans doute le plus répandu. Le protocole de validation à deux phases (*two-phase commit protocol - 2PC*), s'appuyant sur le concept de RPC pour la communication, consiste à encapsuler une ou plusieurs opérations dans une enveloppe de manière à les exécuter comme une seule unité. Son principe est décrit dans la figure 2.1, il nécessite la désignation d'un coordinateur central.

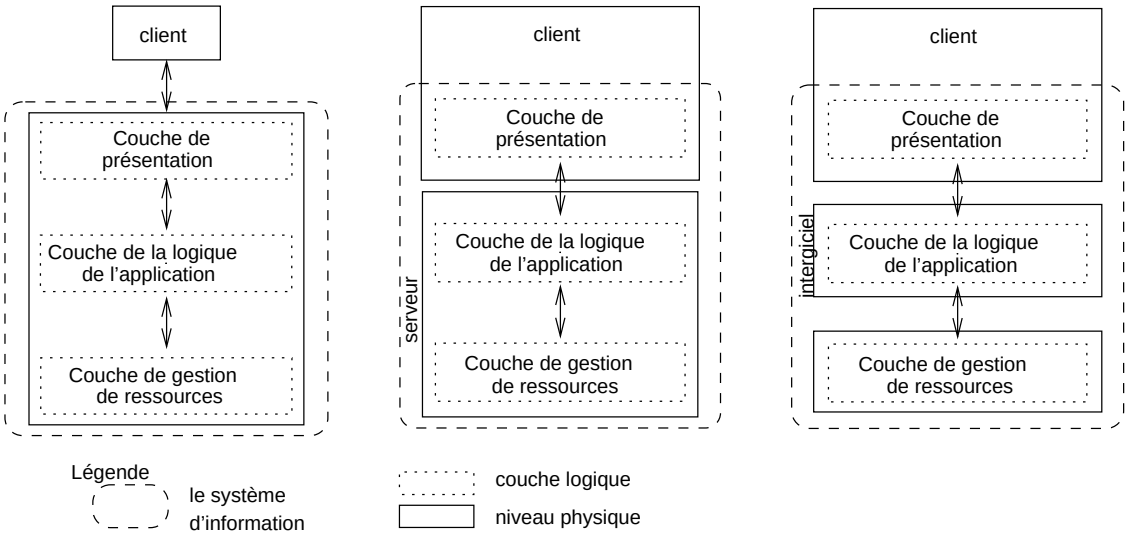
La première phase du protocole 2PC est une phase de consultation dans laquelle le coordinateur de la transaction envoie à tous les participants un message de préparation pour leur soumettre les opérations à valider. Cette phase est suivie par une deuxième phase de validation au cours de laquelle les participants informent le coordinateur qu'ils ont validé les opérations (voir les figures 2.1(a) et 2.1(b)). Ce protocole permet à tous les participants d'un système distribué, soit d'accepter les changements, soit de les rejeter. Si l'un des participants rejette ces changements, tous les autres participants en reçoivent la notification et procèdent à l'annulation de leur part de la transaction. Le protocole 2PC garantit le respect des propriétés ACID et assure la cohérence des opérations même si une défaillance survient. La mise en œuvre de ce protocole exige d'une part une connaissance *a priori* de l'ensemble des participants et d'autre part d'un coordinateur central. Ce protocole est la technique la plus répandue pour implanter des transactions dans les bases de données distribuées. Il a donné lieu à de nombreuses extensions (voir par exemple [Elm90]).

Aujourd'hui les systèmes d'information sont distribués dans des organisations différentes, pas nécessairement toutes connues *a priori*. Ces systèmes formalisent des processus métiers qui nécessitent d'être implantés sur la base de transactions ACID. La suite du texte vise à étudier cette dernière question en présentant tout d'abord les différentes architectures des systèmes d'information (voir section 2.1.2), puis en décrivant les modèles de transaction utilisés sur ces architectures (voir 2.1.3).

2.1.2 Architectures d'un système d'information

De manière générale, les architectures des systèmes d'information sont conçues en trois couches logiques dont les fonctionnalités respectives regroupent la *présentation*, la *logique de l'application* et la *gestion des ressources*. Ces architectures se distinguent selon le nombre de niveaux physiques selon lesquels elles sont structurées : un, deux, trois ou N niveaux (voir la figure 2.2). Le système de gestion des transactions utilisé dépend de l'architecture choisie. Les architectures à un niveau sont celles utilisées dans le cas des systèmes centralisés fondés sur le modèle *maître/esclave*. Ces systèmes sont monolithiques dans la mesure où ils intègrent au sein d'une même unité, toutes leurs composantes (la gestion des données, la logique métier et la couche de présentation). Les accès s'effectuent à partir d'un terminal passif sans capacité locale (voir la figure 2.2(a)).

Au fur et à mesure que les architectures des systèmes d'information ont évolué, les modèles de transaction ont dû être adaptés. Par exemple, dans le cas des systèmes à deux niveaux (voir figure 2.2(b)), s'appuyant sur le protocole RPC, le programmeur de l'application n'a pas besoin de traiter tous les aspects liés à la distribution car c'est le serveur associé



(a) Architecture à un niveau (b) Architecture à deux niveaux (c) Architecture à trois niveaux

FIG. 2.2 – Les différentes architectures d'un Système d'Information [ACKM03]

qui s'en charge. Cependant, dans ces architectures, lorsque les clients sont connectés à plusieurs serveurs, il faut installer sur le client autant d'interfaces de communication que de serveurs (en admettant qu'il n'y a pas deux serveurs qui partagent la même interface de communication, ce qui est assez réaliste !). Ceci entraîne une augmentation, d'une part de la charge sur les clients et d'autre par du coût de maintenance en particulier dans le cas de l'ajout ou de la modification d'un serveur.

Les systèmes à trois niveaux ont apporté une solution à ce problème : la couche de gestion des ressources encapsule l'accès aux données distribuées sur des serveurs et les traitements réalisés par le biais de l'intergiciel chargé de la gestion des interactions entre les clients et les serveurs (voir figure 2.2(c)). L'infrastructure des intergiciels fournit un cadre pour le développement de l'intégration logique et dote ce niveau intermédiaire d'autres fonctionnalités, telles que la gestion des transactions via différents gestionnaires de ressources, la réplication de données, la persistance ou encore l'équilibrage de charge. En outre, la distribution de ce niveau intermédiaire sur plusieurs nœuds augmente la fiabilité et facilite l'adaptation et la maintenance des applications. Cependant ces architectures posent problème lorsqu'il s'agit de couvrir les besoins en termes de gestion des transactions des systèmes d'information issus de l'intégration d'applications développées par différentes organisations. Ceci est discuté dans la section suivante.

2.1.3 Intégration d'applications et transactions

Les évolutions décrites précédemment (selon un point de vue physique) ont conduit à l'émergence d'intergiciels qui permettent de masquer l'hétérogénéité des applications, et ainsi de faciliter la standardisation et l'automatisation de processus inter entreprises (selon un point de vue logique). Le rôle de ces intergiciels est d'abstraire une collection de serveurs et d'applications par le biais d'une même interface de service.

Sur le plan de la gestion des transactions dans les architectures à deux niveaux, le protocole RPC a dû être étendu dans la perspective de communications asynchrones. C'est ainsi que le protocole *RPC Asynchrone* a été développé. Il constitue la base des intergiciels fondés sur l'envoi de messages [Dic98]. DCE (*Distributed Computing Environment*) est l'infrastructure la plus connue du protocole RPC Asynchrone [Ope93]. Proposée comme une extension de TCP/IP [Com05, CT06], cette infrastructure est adoptée par la quasi-totalité des fournisseurs. Ce standard permet l'interopérabilité et la distribution à grande échelle des applications client/serveur. Il offre une vision intermédiaire entre les systèmes client/serveur et les systèmes à trois niveaux. Une autre extension du protocole RPC a été proposée, elle consiste à encapsuler des appels RPC au sein d'une transaction (*Transactional RPC - TRPC*) [Elm90, BN97b]. L'infrastructure TRPC garantit l'atomicité en utilisant un module de gestion de transactions chargé de coordonner les interactions entre les clients et les serveurs. Ce module coordonne l'exécution du protocole de validation à deux phases (selon le protocole 2PC) entre les serveurs participants à la transaction. Le protocole TRPC a été standardisé par le consortium *The Open Group* dans la spécification X/Open [eXO98]. Le protocole TRPC est utilisé pour valider des transactions qui s'exécutent sur des systèmes différents. Les moniteurs transactionnels (*Transaction Processing Monitor - TP Monitor*) sont une implantation représentative de ce protocole [ACF96].

En ce qui concerne les aspects liés à la gestion des transactions, tous les moniteurs transactionnels s'appuient sur ces protocoles. Les moniteurs transactionnels sont l'une des formes les plus anciennes des intergiciels (voir par exemple [GR93, BN97a, WV02]). Aujourd'hui, ils sont à la base de presque tous les modèles transactionnels des architectures à N niveaux (voir figure 2.3). Leurs implantations et leurs fonctionnalités font toujours référence pour le développement de nouvelles formes d'intergiciels.

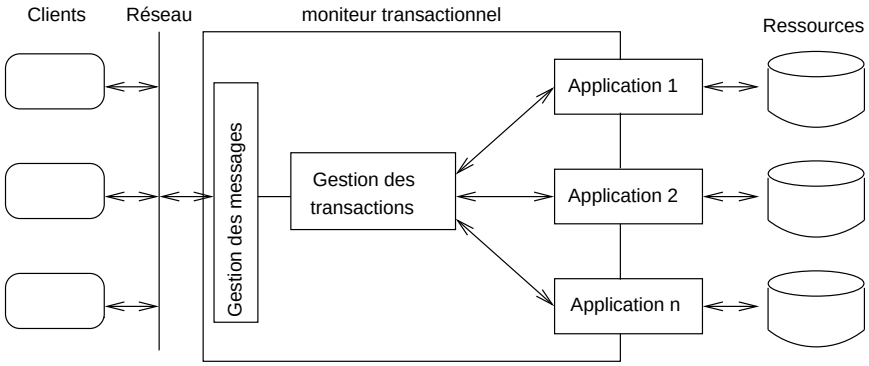


Fig. 2.3 – Architecture d'un moniteur transactionnel

La figure 2.3 schématise le fonctionnement d'un moniteur transactionnel. Ce dernier est chargé de faire communiquer, au travers d'un réseau d'entreprise, les clients avec les différentes applications qui se trouvent à l'intérieur de l'organisation. Le moniteur reçoit les messages envoyés sur le réseau, les traite via le module de gestion des messages qui les transmet aux module de gestion des transactions. Ce dernier coordonne l'exécution des transactions distribuées en s'appuyant sur le protocole 2PC de manière à garantir les propriétés ACID des processus exécutés par le biais du moniteur. Ce module fait office de coordinateur de la transaction, et les différentes applications en sont les participants.

Pendant de nombreuses années, les moniteurs transactionnels ont été la forme d'intergiciel

conventionnel la plus répandue. Ils sont à la base de nombreux systèmes et serveurs d'application [ACKM03]. Cependant, un moniteur transactionnel nécessite de connaître le fonctionnement interne des applications qu'il est censé faire communiquer ainsi que les types de requêtes envoyées par les clients. Ce besoin s'oppose aux principes des systèmes distribués sur Internet où les applications sont indépendantes les unes des autres, hétérogènes et organisées selon une typologie inconnue *a priori*.

Les organisations ont continué à se développer et à diversifier leurs activités de telle sorte que les fonctionnalités de leurs applications sont devenues si différentes les unes des autres que l'utilisation des intergiciels est devenue très complexe, voire même impossible dans de nombreux cas, y compris en leur sein. C'est ainsi qu'a émergé le domaine de l'intégration d'applications d'entreprise (*Enterprise Application Integration* - EAI). Des travaux dans ce domaine ont été conduits pour tenter d'étendre les capacités des intergiciels afin de réussir l'intégration des applications intra entreprise tout en réalisant l'automatisation des processus métier [Row00, Gav03]. Le concept d'EAI est un pas en avant dans l'évolution des intergiciels et il apparaît comme une alternative au développement de nouvelles applications. Ces extensions impliquent des changements importants dans la façon d'utiliser l'intergiciel, en particulier en ce qui concerne la prise en compte d'interactions asynchrones entre les applications à intégrer.

La frontière entre un intergiciel conventionnel et un logiciel de type EAI est très diffuse. D'une part, il s'agit de doter l'intergiciel de certaines caractéristiques spécifiques rendues nécessaires par les besoins d'intégration. D'autre part, les EAIs sont la réponse à la nécessité d'établir la communication entre les systèmes et de masquer leur hétérogénéité. Le principe des EAIs généralise l'idée d'intégrer des serveurs qui sont placés dans la couche de gestion des ressources et indique comment construire des couches correspondant à la logique métier des différents systèmes sous-jacents. Pour la mise en œuvre d'EAIs, la plate-forme la plus polyvalente est connue sous le nom de courtier de messages (*Message Brokers* [Red05]). Un courtier de messages est un système de gestion des échanges de messages entre applications, comprenant un dispositif de transport des messages, un moteur de règles et un moteur de formatage des données. Un courtier fournit une infrastructure de communication logique ou physique entre des applications dans le but de masquer l'hétérogénéité et la distribution des applications à intégrer.

Le choix d'un intergiciel, et surtout sa mise en place dans un environnement informatique hétérogène, est un problème complexe dont la solution passe le plus souvent par l'intégration de plusieurs produits d'origines et de caractéristiques différentes. Le type de l'intergiciel choisi détermine l'endroit où la logique de l'application à intégrer doit être placée. Par exemple, dans les systèmes de courtage la logique métier réside dans l'application elle-même [Red05]. En revanche, dans le cas des systèmes de gestion de flots de tâches (*Workflow Management Systems*) le point central de contrôle pour mettre en œuvre le processus et orchestrer son exécution est fourni par le système lui-même [Coa96, Icu96]. De la même manière, les systèmes de gestion de processus métier (*Business Process Manager*) peuvent être placés au-dessus d'un intergiciel de type courtage ce qui permet d'abstraire les applications qui participent au processus [DHL01].

Souvent, dans la perspective d'intégrer un même ensemble d'applications, il est intéressant de combiner ces types intergiciels car ils sont complémentaires. Malheureusement, chaque intergiciel s'appuie sur une infrastructure propriétaire qui le rend le plus souvent incompa-

tible avec les autres, même si les principes sous-jacents sont proches les uns des autres (ils sont dans la majorité des cas basés sur RPC). La section qui suit discute d'une part du rôle d'Internet dans cette problématique et d'autre part de la mise en œuvre d'un processus transactionnel, lorsqu'il s'appuie sur des applications diverses.

2.2 Intégration d'applications et Internet

L'émergence d'Internet a créé de nouvelles opportunités pour les entreprises à la recherche de partenariats. Les problèmes qui en découlent sont introduits dans la section 2.2.1. Dans le même temps, la notion de service est apparue comme nouveau paradigme pour les architectures logicielles. La section 2.2.2 introduit les définitions liées au concept de services et discute des spécificités des services web. Puis, nous présentons rapidement la mise en œuvre de services web par le biais des protocoles et leurs langages standards associés. Enfin, la section 2.2.3 montre les différentes manières d'aborder la composition de services web.

2.2.1 Intégration d'applications inter-entreprise

A l'origine, le web a constitué un support pour le partage de l'information sur Internet. Puis, il a rendu possible des interactions de type B2C (*Business to Customer*) au travers desquelles des clients accèdent à des applications (pour acheter des produits par exemple). Dans ce cadre, de nombreuses organisations mettent à disposition leurs applications sur Internet au moyen d'interfaces interactives ou de pages web plus ou moins sophistiquées. Aujourd'hui le web est aussi utilisé comme support d'interactions B2B (*Business to Business*) permettant la communication entre des applications accessibles via le web. Cette dernière orientation a rendu nécessaire de repenser les intergiciels existants pour l'intégration d'applications dans la perspective de franchir les frontières des entreprises.

L'intergiciel responsable de la gestion des interactions entre les applications doit être capable de prendre en compte les spécificités de chaque application à intégrer (gestion des ressources, modèle transactionnel, etc.). Dans les intergiciels conventionnels et les EAIs, les applications sont distribuées tandis que l'intergiciel implantant la logique de l'intégration est centralisé et contrôlé par une seule entreprise. Si le concept est étendu sur Internet, il est difficile de déterminer où localiser l'intergiciel tout en respectant les impératifs de sécurité, d'autonomie et de confidentialité des différents partenaires. En outre, il est nécessaire que tous les participants acceptent de coopérer avec l'intergiciel choisi pour l'intégration et les interactions, ce qui implique la connaissance *a priori* de tous les membres. De plus, les applications accessibles par Internet ont des caractéristiques qui font que les systèmes transactionnels utilisés auparavant ne sont plus applicables : les transactions sont de longue durée, transitent par des applications hétérogènes fournies par des organisations différentes et manipulent des ressources indépendantes les unes des autres et placées en des endroits distants les uns des autres. Le protocole 2PC ne convient plus car il n'est plus possible de verrouiller les ressources pour le temps nécessaire, d'autant plus que souvent, les entreprises qui participent à la transaction ne se connaissent pas les unes les autres. Dans ce contexte, la communication et donc l'intégration d'applications ne sont plus possibles via les protocoles utilisés par les intergiciels (conventionnels et de type EAI). En particulier,

les protocoles tels que RPC, RMI (*Remote Method Invocation de JAVA*), GIOP (*General Inter-ORB Protocol* de CORBA) ou même IIOP (*Internet Inter-ORB Protocol* pour traduire des appels GIOP en appels TCP/IP) deviennent incapables de traverser les pare-feu pour accéder aux applications placées hors de leur contexte organisationnel. La solution consiste à encapsuler, via un intermédiaire, le message original dans un document HTML ou XML en l'envoyant via HTTP. L'opération inverse est effectuée par le destinataire à la réception du message. Ce mécanisme ne fonctionne que si les applications à intégrer partagent un vocabulaire commun, c'est-à-dire les objets et leur signification.

2.2.2 Services et services web

C'est dans ce contexte qu'ont émergées les architectures à base de services (*Service Oriented Architecture* - SOA) dont l'objectif est d'abstraire les applications à intégrer par des services qui interagissent les uns avec les autres de manière faiblement couplée. Lorsque ces interactions s'effectuent en s'appuyant sur les technologies Internet, on parle de services web. S'en est suivi une prolifération des protocoles/logiciels pour concevoir, décrire, programmer, coordonner, composer les services web. Ces protocoles couvrent des ensembles de fonctionnalités aux intersections non vides. Le chapitre 3 détaille les principaux de ces protocoles et les comparent les uns aux autres.

Le terme service est doté de sens différents selon le contexte où il est utilisé. Dans [Dum05] les auteurs présentent la définition ci-dessous comme l'une de celles traditionnellement données dans le monde du marketing et du management. Seuls le service rendu et le client qui en tire profit sont pris en compte.

> Product involving a performance which results in added value in forms (such as convenience, amusement, timeliness, ...) which are essentially intangible to the first purchaser.

La définition suivante, prise dans le glossaire du consortium W3C[2], tout en ne considérant toujours pas les aspects liés à la mise en œuvre des services en donne un point de vue plus concret :

> A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities. To be used, a service must be realized by a concrete provider agent.

Un service est une mise en œuvre d'une fonctionnalité spécifique d'un processus métier et peut être exécuté indépendamment du langage et de la plate-forme d'exécution de l'application cliente [BCTH03]. L'ensemble des services - internes et externes à une organisation - compose une architecture à base de services, où "toutes les fonctionnalités sont présentées comme des services" [BEK+00].

Les premières architectures à base de services sont DCOM (*Distributed Component Object Model*) et ORB (*Object Request Brokers*) toutes les deux basées sur la spécification de CORBA [Obj92]. DCOM[3] est la technologie proposée par Microsoft pour réaliser des ap-

[2] http ://www.w3.org/2003/glossary/
[3] Maintenant connue sous le nom de .NET [Cor]

pels distants à des objets COM (*Component Object Microsoft*), dont une version améliorée est COM+ [Pla99]. CORBA est un modèle d'architecture dont le but est de faciliter l'interaction des composants, indépendamment de leur plate-forme d'origine et du langage dans lequel ils ont été développés. A cette fin, CORBA utilise un modèle d'interface appelé IDL (*Interface Definition Language*) qui permet à un composant d'exposer aux autres les fonctionnalités qu'il offre. CORBA a été défini pour fonctionner dans un environnement hétérogène, mais il s'est avéré très complexe à mettre en œuvre. L'OMG (*Object Management Group*[4]) a défini le protocole IIOP (Internet Inter ORB Protocol) pour communiquer avec des objets CORBA à travers le réseau.

Une caractérisation et une classification des services sont proposées dans [DOHE01]. Ainsi, dans le domaine des intergiciels [Ber96] et des systèmes de bases de données [Col00], un service est vu comme un ensemble des fonctionnalités logicielles conçues pour faciliter l'implantation des applications en ne les contraignant pas à adopter des services dont elles n'ont pas besoin. Ici, les services sont des composants de logiciels, avec une fonctionnalité précise dans le développement d'une application (par exemple, le service d'évaluation des requêtes ou celui de la persistance des données).

D'autres approches plus récentes ont émergé comme par exemple les bus de services (*Enterprise Service Bus*) et l'approche OSGi (connue aujourd'hui sous le nom *Open Services Gateway Alliance*, dont le nom original était *Open Services Gateway initiative*, voir le chapitre sept dans [Kra06]).

Un bus de services, aussi appelé plate-forme de distribution XML (*XML hub*), est définie au dessus d'une architecture distribuée combinant des services et des technologies traditionnelles pour l'intégration (courtiers, systèmes de routage et de transformation de données, etc.) [Cha04]. Un bus de services offre un point d'intégration à travers lequel des services d'application et des composants d'applications peuvent être appelés, dans le cadre de l'exécution d'un processus métier. Les bus de services constituent une alternative aux solutions traditionnelles de type EAI, qu'ils ne remplacent cependant pas complètement. Ils sont en effet très loin de proposer l'ensemble des fonctionnalités des solutions de type EAI. Par exemple, même s'ils facilitent le déploiement de services, ils ne prennent pas en charge les couches de gestion et de supervision des processus métier.

Dans le contexte du web, de nombreuses tentatives pour définir un service web ont conduit à différentes interprétations. La définition donnée ci-dessus proposée par le consortium W3C[5] fait l'objet d'un large consensus :

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL[6]). Other systems interact with the Web in a manner prescribed by its description using SOAP[7] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Ce que nous retenons essentiellement de cette définition et de celles des services présentés

[4] http ://www.omg.org
[5] http ://www.w3.org/TR/ws-arch/
[6] *Web Service Description Langage* [CCMW01].
[7] *Simple Object Acces Protocol* [BEK+00].

précédemment, est que chaque service fonctionne comme une entité indépendante, développée et déployée indépendamment des autres services. Dans le contexte d'Internet, un service web est une application pouvant interagir avec d'autres services ou applications en utilisant des protocoles d'échanges basés sur XML comme SOAP, XML. WSDL, fondé sur XML, est le standard incontournable pour la description de l'interface fournie d'un service web [New02, ACKM03]. Cette interface décrit les fonctions offertes par le service et les types des messages que ce dernier peut recevoir et envoyer. L'interface fournie d'un service est exposée afin d'être consultée par des machines ou des humains dans la perspective de développer des applications clientes capables de communiquer automatiquement sur Internet avec des services, et d'en utiliser les fonctionnalités offertes. Ces applications peuvent à leur tour être exposées comme des services web. Un service est une boite noire, son implantation et son interface requise en particulier, ne sont pas visibles de l'extérieur. Dans certains cas, il peut être attendu qu'un service expose aussi certaines de ses propriétés extra fonctionnelles, notamment celles reliées à la notion de qualité de services.

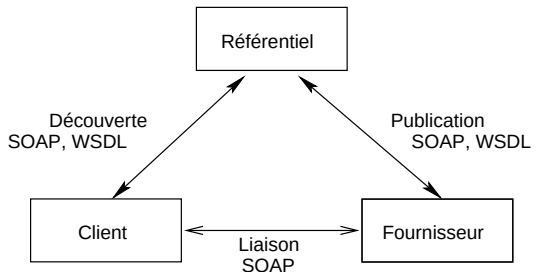


Fig. 2.4 – Interactions dans les architectures à base de services web

Quant aux technologies pour la mise en œuvre de services web, elles sont, le plus souvent, déclinées sur la base des deux standards WSDL et SOAP qui supportent respectivement, la description et la communication. La figure 2.4 montre les interactions entre ces protocoles : un fournisseur fournit la mise en œuvre d'un service et publie sa description dans un référentiel (par exemple structuré selon le standard UDDI[8]). Dans le chapitre 4 nous reviendrons sur les défauts de ce standard. Sur le plan opérationnel, les organisations clientes recherchent dans le référentiel le ou les services qui pourraient correspondre à leurs besoins, en obtiennent l'URI ainsi que la description de l'interface. Ensuite, elles conçoivent et mettent en œuvre une application cliente qui réalise les appels au(x) service(s) sélectionné(s). Toutes les interactions s'effectuent pas le biais d'échanges de messages XML structurés selon le standard SOAP.

Malheureusement, cette infrastructure n'est pas suffisante pour satisfaire les exigences des applications telles que celles relatives à la coordination, la sécurité, la gestion des transactions, la fiabilité et la qualité du service. Les spécifications de type WS-* (par exemple *WS-Coordination*, *WS-Transaction*, *WS-Security*, *WS-ReliableMessaging*, *WS-SecureConversation*, *WS-PolicyAttachment*, etc.)[9] sont des efforts pour satisfaire les besoins d'infrastructure des applications web.

[8] *Universal Description and Discovery and Integration* [OAS05].
[9] WS-Coordination et WS-Transaction sont détaillés dans le chapitre 3.

2.2.3 Composition de services web

Le développement de nouveaux services faisant référence à d'autres services déjà existants (fournis ou non par des organisations différentes) est une manière d'envisager l'intégration d'applications hétérogènes. Nous parlons alors de services *composés*. Une composition de services web spécifie quels services ont besoin d'être exécutés, dans quel ordre et comment gérer les conditions d'exception. Les composants restent indépendants du service composé. L'intergiciel responsable de l'exécution d'une composition s'appuie sur un protocole de *coordination* pour gérer de façon appropriée l'ordre et le type des messages échangés.

La composition de services web peut être étudiée selon deux points de vue complémentaires : (1) un point de vue global dans lequel l'ensemble des partenaires entrant dans la composition sont considérés ; le modèle qui en découle est appelé *chorégraphie*. (2) un point de vue local ou privé dans lequel seul le processus interne des services est modélisé ; le terme employé pour cela est *orchestration*.

La coordination d'une orchestration d'un service est un processus centralisé qui contrôle et qui coordonne l'ordre et l'exécution des interactions du service avec les partenaires impliqués dans la composition (voir figure 2.5). Ces derniers ne savent pas nécessairement qu'ils forment une partie d'un processus de plus haut niveau, ni ne connaissent les autres partenaires [JMS06]. Une orchestration décrit, en plus des interactions entre les partenaires, les actions que le service exécute en interne.

La figure 2.5 montre un diagramme d'activités UML qui modélise le processus du scénario "Agence de voyage". La spécification comporte des activités de communication entre le service client (l'agence de voyage) et les services sur lesquels s'appuie le processus. Le processus est coordonné par le client qui connaît à chaque instant l'état de la composition. Les services composants interagissent avec le client sans avoir de connaissance sur le processus métier dans lequel ils forment une partie.

A contrario dans une chorégraphie, la logique qui contrôle les interactions entre les services composants est distribuée entre les participants (voir figure 2.6). Il n'y a pas de coordinateur central. Chaque service impliqué dans la chorégraphie connaît exactement avec qui interagir et quand exécuter les opérations dont il est responsable (voir par exemple [FDBP01, BDSN02, JMS06]). La composition par chorégraphie, aussi appelée *collaboration* dans la littérature, décrit d'une part les interactions entre les services et d'autre part les relations qui existent entre ces interactions. Les opérations internes des participants ne sont pas considérées. La figure 2.6 montre une chorégraphie qui réalise une partie du scénario "Agence de voyage" (la figure ne montre qu'une partie des interactions mises en jeu). Dans cette figure, chaque fournisseur de service connaît à l'avance avec quel service il va interagir. Par exemple, le service Vol a été conçu pour interagir avec le service Hôtel auquel il transmet les détails concernant le vol. Puis le service Hôtel réalise la réservation d'une chambre selon les données transmises et retourne au client le message validationVoyage qui contient les détails du voyage concernant l'hôtel et le vol.

Une composition de services web peut aussi être décrite en termes d'un processus exécutable ou abstrait [JMS06] :

- *Un processus métier exécutable* spécifie tous les détails en tenant compte des opérations internes et externes des composants. Ce type de processus est modélisé par une orchestration.
- *Un processus métier abstrait* spécifie seulement les messages échangés entre les composants. Les détails des opérations internes ne sont pas inclus. Ce type de processus est

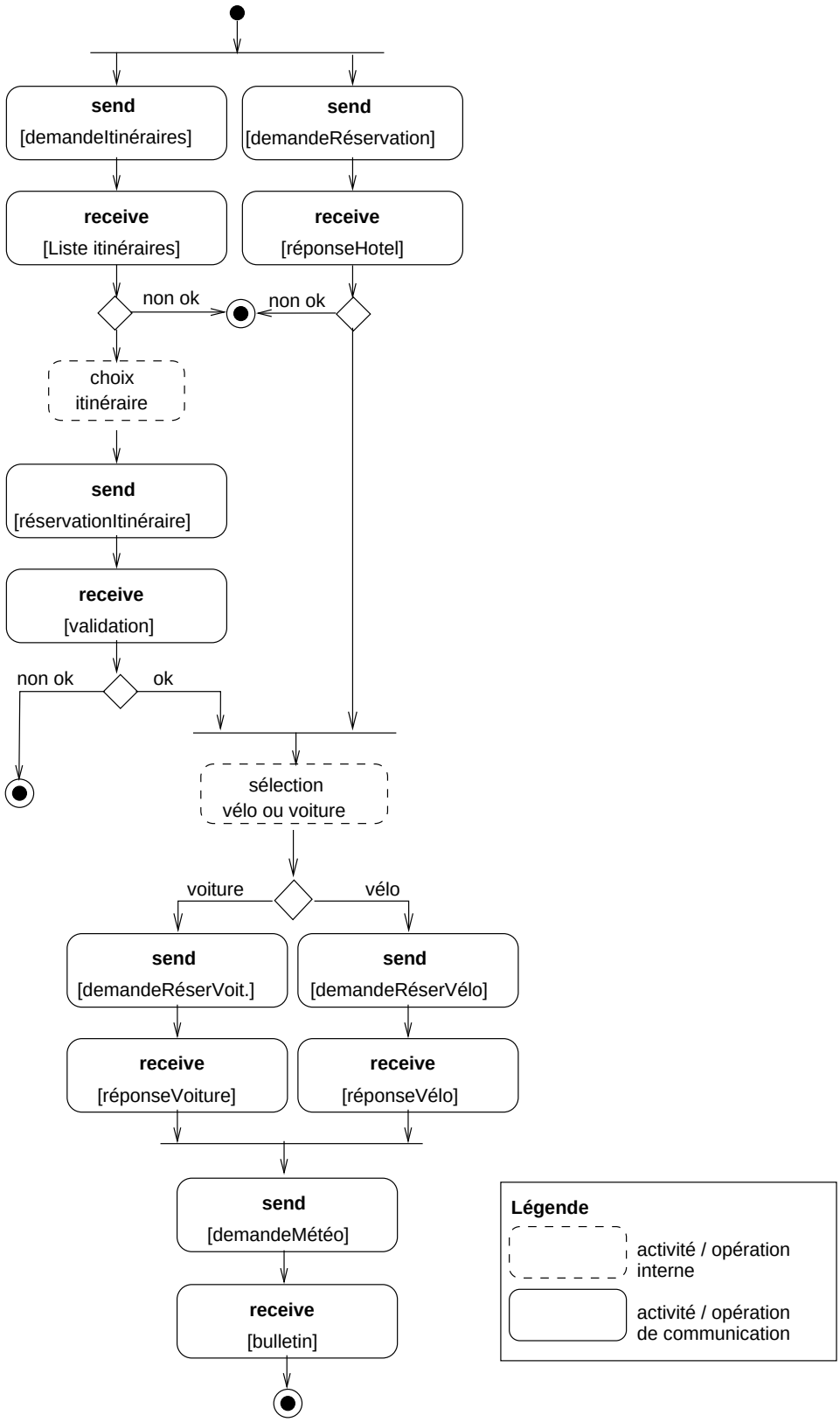


FIG. 2.5 – Modèle d'orchestration de l'application "Agence de voyage"

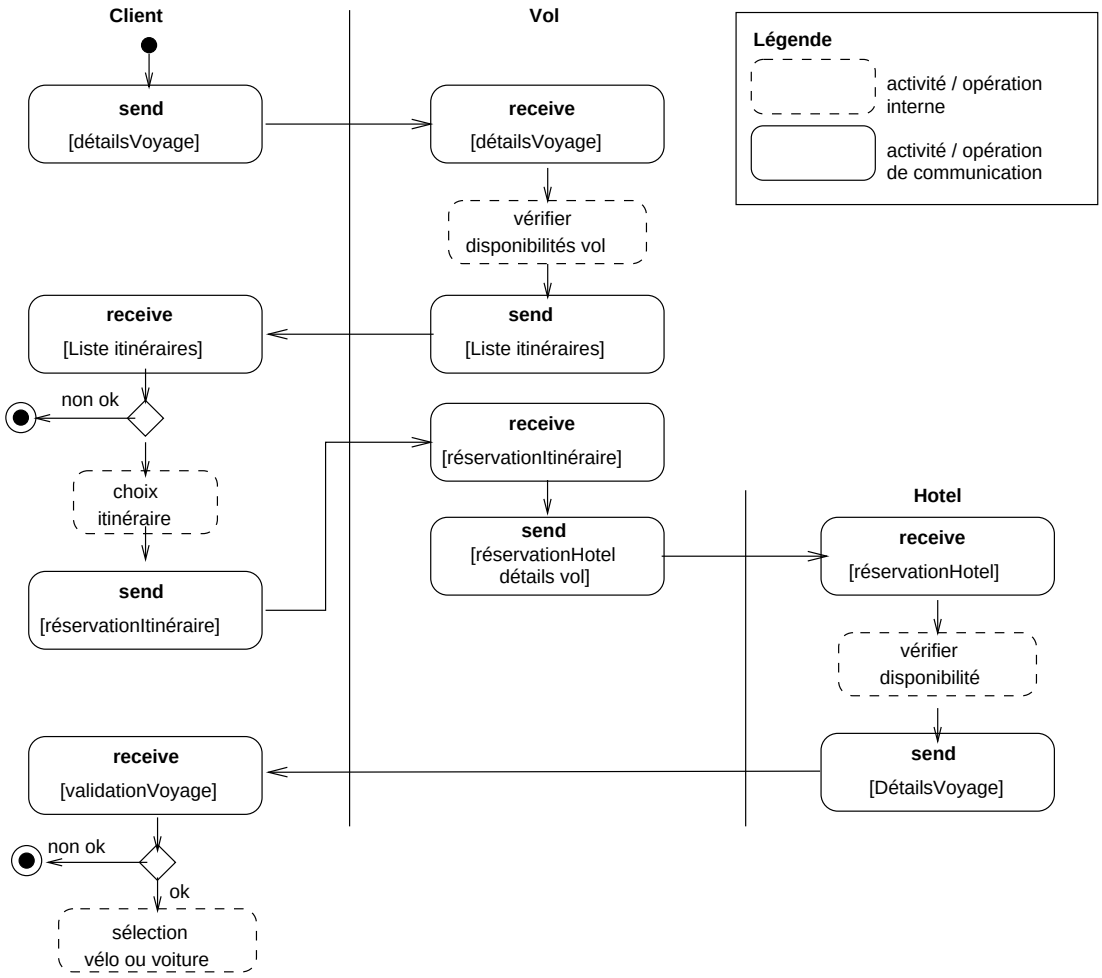


FIG. 2.6 – Modèle de chorégraphie de l'application "Agence de voyage"

modélisé par une chorégraphie.

Un processus métier exécutable compose un nouveau service web à partir d'un ensemble de services web existants. Un processus métier abstrait n'est pas exécutable, il est utilisé principalement pour deux raisons : (i) décrire le comportement d'un service sans savoir exactement dans quel processus métier il va intervenir et (ii) définir les protocoles de collaboration entre plusieurs partenaires et décrire le comportement externe de chacun. Les processus abstraits sont généralement utilisés comme des modèles pour aider à la définition des processus exécutables.

Les modèles de chorégraphie ou d'orchestration qui ont été étudiés jusque là ne font pas référence à un modèle transactionnel en particulier. La raison est qu'habituellement les propriétés transactionnelles d'un composant ou d'un service font parties de ses aspects extra fonctionnels, d'autre part, comme nous le verrons dans le chapitre 3 qui suit, il n'existe pas de modèle de composition qui permette d'exprimer la sémantique transactionnelle de la composition. Dans la plupart des cas, les participants à une composition doivent se mettre d'accord préalablement (via des contrats par exemple) sur la façon dont les ressources seront gérées.

2.3 Synthèse

Dans ce chapitre nous avons présenté l'évolution des architectures sur lesquelles reposent les systèmes d'information. En partant des architectures centralisées, nous avons considéré les architectures distribuées structurées en différents niveaux. Dans cette évolution nous avons montré les différents systèmes de gestions des transactions utilisés à chaque étape et nous avons montré leurs insuffisances.

Nous avons discuté des plates-formes spécifiques pour les interactions sur Internet et nous avons donné les définitions nécessaires à la lecture de la suite de ce document. Reprenant les éléments de la problématique introduite dans le chapitre 1, nous sommes maintenant en mesure d'en préciser les contours. Les problèmes qui restent posés pour composer des services web munis de propriétés transactionnelles sont listés ci-après :

- Les participants appartiennent à différentes organisations, ils ne se connaissent donc pas entre eux, et il est difficile d'établir un consensus sur le choix d'un coordinateur pour la gestion des transactions.
- Chaque composant a été implanté selon une logique et des caractéristiques qui lui sont propres indépendamment bien sûr des autres partenaires de la composition à laquelle il participe. Les objectifs individuels des composants peuvent se révéler incompatibles avec ceux du tout que forme la composition.
- La connaissance qu'un client (une application) a des services web qu'il utilise est limitée à l'interface qu'ils fournissent. Même si quelques services exposent le point de vue comportemental de leur interface (c'est-à-dire le modèle d'interaction défini sur les opérations exposées), il n'est pas possible de connaître la manière dont ils sont implantés.
- La topologie du réseau sur lequel s'appuie la composition n'est pas connue *a priori* : un service peut être momentanément indisponible, de nouveaux services peuvent apparaître.

La section 2.1 a décrit des systèmes transactionnels génériques qu'il est possible d'utiliser pour réaliser l'intégration d'applications. La section 2.2.1, dédiée à l'exposé des problèmes posés par l'intégration des applications sur Internet et plus particulièrement en s'appuyant sur des services web, a montré que les limites de ces systèmes génériques. Dans le chapitre qui suit, nous étudions en détails les différentes approches qui ont été proposées pour résoudre les problèmes que nous venons de soulever. Une synthèse sera proposée afin de mettre en évidence les apports de la thèse présentée dans ce document.

CHAPITRE 3

ASPECTS TRANSACTIONNELS DANS LES SERVICES WEB

De très nombreux modèles de transactions ont déjà été définis, notamment dans le domaine des bases de données, des systèmes distribués et des environnements coopératifs (voir par exemple [Elm90, GR93, Pap03]). En même temps que les systèmes d'information ont évolué, le modèle transactionnel classique a été adapté afin de répondre aux besoins transactionnels des nouvelles applications qui demandent plus de flexibilité. Les modèles transactionnels existants ont résolu les problèmes liés aux interactions entre des applications sur des bases de données réparties. Cependant, comme nous l'avons vu au chapitre 2, ces modèles ne sont pas appropriés pour la gestion et le traitement des interactions entre des processus métiers qui s'effectuent au dessus d'internet. Il est donc nécessaire de définir un nouveau modèle transactionnel, ou d'adapter les modèles existants, afin d'intégrer différents services hétérogènes et d'incorporer différents patrons d'interactions au sein de la même transaction. Ce modèle doit tenir compte de la distribution inhérente aux services web dont les caractéristiques individuelles sont hétérogènes. Ainsi, en plus de la description des opérations offertes, un service web doit inclure la description des propriétés extra fonctionnelles qu'il supporte, en particulier ses propriétés transactionnelles.

Dans la suite de ce chapitre, la section 3.1 positionne la discussion sur les transactions dans les services web par rapport au chapitre 2 précédent. Puis, la section 3.2 présente les caractéristiques transactionnelles nécessaires à un processus métier telles qu'elles ont été établies dans la littérature. La section 3.3 discute des efforts de standardisation effectués aussi bien pour la définition de protocoles que pour celle de langages. Cette section discute de la manière dont la gestion des transactions est abordée. Dans la section 3.4 nous présentons des contributions spécifiques de la recherche aussi bien industrielle qu'académique et finalement la section 3.6 compare les contributions étudiées dans ce chapitre les unes par rapport aux autres, et les situe par rapport à l'approche que nous proposons dans cette thèse.

3.1 Introduction

La technologie des services web présente des caractéristiques différentes par rapport à celles des intergiciels et des EAIs, en particulier en ce qui concerne l'architecture des applications et la gestion des interactions entre les partenaires fournissant ces applications. Ainsi, un service web peut être vu comme une application distribuée qui s'appuie sur d'autres services hébergés par des organisations différentes, indépendantes les unes des autres, et qui ne partagent pas les mêmes règles de gestion. Il s'en suit que dans l'environnement des services web, les transactions impliquent différentes parties, franchissent plusieurs organisations et peuvent se dérouler sur une longue durée. Une autre caractéristique importante à considérer est que le choix d'un service qui entre dans une composition peut être effectué dynamiquement au moment de l'exécution, ainsi l'ensemble des partenaires d'une même transaction n'est pas connu *a priori* ce qui engendre un environnement très instable.

Des extensions des standards pour la composition de services web avec des propriétés transactionnelles essentielles ont été proposées :

- Par l'introduction de constructions algorithmiques spécifiques exprimant des propriétés transactionnelles [Alo05].
- Par l'extension des propriétés ACID à l'environnement transactionnel d'internet en distinguant le modèle classique des transactions du domaine des bases de données de celui spécifique aux processus métiers [Pap03, vdHA02] (voir la section 3.4).
- Par la prise en compte de transactions compensatrices qui permettent de terminer correctement une action lorsqu'un processus ou un utilisateur l'annule [GMS87].

Les aspects transactionnels ne peuvent pas être considérés de manière isolée. En règle générale, les modèles transactionnels sont liés à la coordination et au développement des processus métiers basés sur des services web. C'est ainsi que le développement d'outils pour la composition de services s'est diversifié. Actuellement, la recherche, aussi bien académique qu'industrielle, vise à produire des outils pour automatiser et standardiser la composition de services dans le but de produire des applications extensibles, ouvertes et développées en un minimum de temps. La première contribution de l'ensemble de ces travaux est l'obtention d'un consensus sur un ensemble de caractéristiques que doivent vérifier les processus métiers transactionnels mis en œuvre sur internet. Nous détaillons ces caractéristiques dans la section suivante.

3.2 Caractéristiques consensuelles identifiées

Cette section s'appuie, principalement sur les travaux rapportés dans [MTSM02, Pap03, Alo05] et qui ont porté sur la définition des propriétés et des caractéristiques d'un modèle transactionnel adapté aux besoins des services web. Il est intéressant de remarquer qu'en général leurs conclusions sont très proches les unes des autres.

3.2.1 Généralités

Les transactions pour les processus métiers se distinguent selon qu'elles sont : (i) des transactions atomiques, de courte durée qui respectent les propriétés ACID, ces transactions

utilisent le protocole 2PC (classiquement utilisé dans le domaine de bases de données) ; (ii) des transactions de longue durée, définies comme un ensemble de transactions atomiques [Pap03]. Elles se comportent alors comme des transactions emboîtées [Elm90]. Ce type de transactions permet la confirmation sélective par certains participants ou des annulations par d'autres. En fonction de la situation, l'issue de la transaction n'est donc pas le même.

Quelque soit le modèle, notons que le comportement global d'une transaction associée à un processus métier, dépend des capacités transactionnelles des services web participant au processus. Il n'est pas possible d'en changer et la définition des propriétés transactionnelles du processus global doit prendre en compte cette contrainte. La conception des modèles transactionnels pour les processus métiers peut être menée selon trois phases [Pap03] :

1. La *phase de pré-transaction* qui correspond à l'échange des informations critiques entre les participants (prix, conditions de livraison, garanties, etc). Le but est de diminuer le nombres des annulations. Le protocole Tentative de Réservation de Ressources (*Tentative Hold Protocol - THP* [RS01, RCMS01]) est utilisé dans cette phase. L'objectif du protocole THP est de faciliter la coordination automatisée de transactions inter-organisation. Il s'agit d'un protocole ouvert, avec une architecture faiblement couplée à base de messages pour l'échange d'informations entre différents processus avant la transaction elle-même. Son principe est de permettre à un processus de s'abonner à l'information transmise par un autre sur la disponibilité d'une ressource gérée par ce dernier dès que celle-ci change de statut. Dès lors que la ressource est définitivement acquise par un des processus, un message est envoyé à tous les autres processus s'étant abonnés afin de les informer que la ressource n'est plus disponible. Ce protocole permet ainsi aux processus de se mettre à jour quand l'état des ressources change, augmentant ainsi les chances de validation d'une transaction.

2. La *phase de transaction* proprement dite ou d'exécution du processus métier. Cette phase fournit les protocoles et l'infrastructure pour coordonner l'exécution des opérations distribuées dans un environnement de services web. Les principales caractéristiques de cette phase sont : (i) la possibilité de développer des applications distribuées de façon automatisée et de gérer la coordination des transactions multi-processus et des interactions non transactionnelles, (ii) des extensions du protocole 2PC (par exemple TIP - *Transaction Internet Protocol* [LEK98]) et (iii) des capacités de restauration et de gestion des fautes.
3. La *phase de post-transaction* qui est responsable de la vérification des contraintes devant être satisfaites par la transaction.

Cette conception en trois phases ne peut être conduite que si les partenaires de la transaction se connaissent, ce qui en restreint le champ d'application.

Par ailleurs, dans le cadre des processus métiers, la propriété d'atomicité peut être caractérisée selon le niveau d'abstraction auquel elle s'applique [Pap03] :

- Atomicité au niveau système : ceci implique que chaque fournisseur offre des services et des opérations qui seront exécutés de manière atomique.
- Atomicité au niveau des interactions entre les partenaires : la transaction doit alors être précédée par une une séquence d'interactions.

– Atomicité au niveau opérationnel : la terminaison de l'ensemble des tâches opérationnelles (par exemple le paiement effectif de la facture et l'envoi de la marchandise sont deux tâches à réaliser, ou aucune).

La mise en place des trois phases ci-dessus, tout en prenant en compte ces niveaux d'atomicité, nécessite de :

– Mettre en place un contrat transactionnel entre les participants.
– Relâcher les propriétés ACID et forcer les mécanismes de compensation de façon similaire à celle proposée dans l'approche des Sagas [GMS87], ce qui résout les problèmes de verrouillage sur des ressources par des transactions de longue durée.
– Mettre en place un coordinateur pour synchroniser et orchestrer le bon déroulement d'une transaction et la gestion des fautes.
– Séparer finalement le niveau de la mise en œuvre interne de celui de la description externe du protocole en utilisant des outils appropriés.

Un canevas pour la gestion de processus métiers transactionnels (*Business Transaction Framework* - BTF) a été proposé dans la perspective de tenir compte de ces caractéristiques. Ce canevas orchestre de manière faiblement couplée des services web au sein d'un processus en offrant un support transactionnel en termes de coordination distribuée et en garantissant le résultat convenu préalablement par les participants du processus [Pap03].

3.2.2 Propriétés ACID

Lorsqu'un service web intervient dans une composition, la transaction inhérente à cette composition n'a pas la même signification que dans le domaine des bases de données. Dans le domaine de la composition de services web, l'objectif visé est plutôt de "supporter les processus transactionnels" qui dérivent de toutes les formes d'intégrations d'applications [Bus03]. Nous revenons plus loin sur la distinction entre modèle transactionnel pour les bases de données, et modèle transactionnel pour processus métiers. Les processus métiers interagissent de multiples façons et parfois en dehors du système où est exécuté le processus. Chacun possède sa propre stratégie de gestion des transactions. La remise en cause des propriétés ACID est nécessaire.

Il n'est pas possible d'isoler les processus les uns par rapport aux autres. Pour cette raison, la propriété d'isolation n'est pas pertinente pour assurer le fonctionnement correct du processus. La cohérence est la propriété la plus difficile à considérer. Les processus peuvent finir correctement de plusieurs façons et le maintien de la cohérence relève de la responsabilité du processus lui-même, c'est une caractéristique qui doit être programmée par le concepteur du processus, ce n'est plus une propriété du système. La durabilité dans l'environnement des processus transactionnels fait référence à la persistance et à la trace du processus qu'il est légalement nécessaire de conserver dans la plupart du temps. De même que pour l'isolation et la cohérence, le problème de la durabilité n'est pas traité par la plupart des modèles de processus métiers transactionnels.

La seule propriété qui reste pertinente dans l'environnement des processus métiers transactionnels est l'atomicité, mais pour des raisons différentes de celles mises en avant dans contexte des bases de données. Dans le contexte des transactions distribuées, l'atomicité est assurée par le protocole de validation à deux phases (protocole 2PC, voir chapitre 2),

mais, quand il s'agit de transactions inter-processus, les mécanismes pour assurer l'atomicité sont nécessairement des mécanismes algorithmiques. Par ailleurs, il y a des situations dans lesquelles une partie seulement du processus est atomique. Il est aussi quelques fois souhaitable de garder la trace du processus même s'il ne réussit pas. La meilleure façon de résoudre ce problème est de fournir aux programmeurs des constructions algorithmiques appropriées. Parmi ces techniques de programmation l'approche à base de Sagas, basée sur des mécanismes de compensation, est sans doute la plus représentative [GMS87]. Dans cette approche, un processus est composé de plusieurs pas où chaque pas est une transaction indépendante qui peut être exécutée séquentiellement ou en parallèle par rapport aux autres. Pour garantir l'atomicité, chaque pas dans le processus est associé à une action de compensation, et pour défaire tous les effets du processus, les actions de compensation sont exécutées dans l'ordre inverse de l'ordre dans lequel les actions ont été exécutées. Une autre solution à mentionner est la semi-atomicité qui s'appuie sur des chemins alternatifs qui peuvent être suivis si le chemin choisi échoue [ZNBB94].

La structure plate des transactions, bien adaptée à l'environnent conventionnel des processus distribués, n'est pas suffisante dans celui des services web [MTSM02]. Plus précisément, les transactions dans ce contexte présentent les caractéristiques suivantes :

- Les interactions entre les services sont faiblement couplées : chaque service peut mettre en œuvre un processus selon, nous l'avons déjà souligné, sa propre stratégie transactionnelle. Comme les transactions commencent et finissent dans chaque service, il n'est pas possible d'utiliser le modèle plat de transactions dans une application composée par plusieurs services web.
- Les transactions peuvent être de longue durée : la propriété d'isolation doit être relaxée pour que les transactions dans le processus métier puissent durer des heures, des jours ou des semaines. Pour cette raison les processus doivent imposer une limite de temps sur certains types de transactions (par exemple, le nombre de jours durant lesquels la réservation d'un billet d'avion est valable). Ceci peut s'appuyer sur le protocole THP [RS01]. Le protocole transactionnel doit permettre la négociation entre les participants.
- Les transactions peuvent être structurées en sous-transactions : dans certains processus métiers, un sous-ensemble des activités peut être optionnel. Cette caractéristique relaxe la propriété d'atomicité, mais il est indispensable que la logique de l'application décrive l'ensemble des activités qui doivent être considérées atomiques et génère les transactions de compensation.
- Le contexte de la transaction doit être géré : une application qui combine plusieurs services web de différentes entreprises exprime des transactions qui s'appuient sur ces services. Ainsi, le contexte d'exécution de la transaction doit être propagé aux différents services participants.

La tableau 3.1 résume la discussion ci-dessus et donnent les différences entre le modèle de transactions conventionnelles et le modèle de transactions pour des processus métiers.

3.2.3 Mécanismes de compensation

La notion de *régions atomiques* dans un schéma d'orchestration est très répandue. La figure 3.1 montre un service web avec deux régions atomiques, chacune décrite par une

Propriété	Transactions conventionnelles	Transactions dans les processus métier
Atomicité	Requise	quelques fois souhaitable, quelques fois applicable à un sous-ensemble seulement des fonctions
Cohérence	Requise	Requise
Isolation	Requise	Relaxée ; chaque service contrôle le degré de visibilité des ressources qu'il manipule
Durabilité	Requise	Requise ; mais basée sur la propriété d'atomicité. Quelque parties peuvent être volatiles
Propagation du contexte	Relaxée, non requise	Requise

TAB. 3.1 – Identification de propriétés transactionnelles selon [MTSM02]

séquence d'activités. La région 1 contient les activités A et B qui doivent être exécutées en séquence, avant de passer à la deuxième région. La région 2 contient les activités C, D et E, et toutes ces activités doivent être exécutées à leur tour pour que le service se termine conformément à sa spécification. Mais si l'activité B ou C échoue, que se passe t-il ? Dans ce cas le concepteur doit prévoir une issue pour spécifier un comportement correct du service. Actuellement, la solution la plus adaptée est d'établir des mécanismes de compensation pour chaque activité qui échoue. Le problème est que l'introduction des activités compensatrices est une tâche très complexe à concevoir et difficile à mettre en œuvre. Cela demande beaucoup d'efforts et une connaissance profonde des mécanismes de fonctionnement de chaque composant [ACKM03].

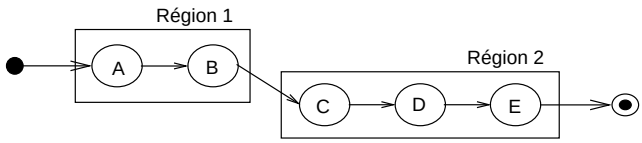


FIG. 3.1 – Régions atomiques dans un modèle d'orchestration

L'atomicité de ces régions peut être mise en œuvre via le protocole de validation à deux phases, mais au prix d'une certaine adaptation. Il faut en effet prendre en compte le couplage faible entre les services et la durée des transactions. Il existe deux types d'approches pour cela :

- Le moteur qui assure l'exécution de la composition des services web, s'appuyant sur *WS-Transaction*, prend en charge le protocole de compensation, plus approprié pour relaxer la propriété d'atomicité (*WS-Transaction* est présenté plus loin, voir section 3.3).
- Le programmeur du service est responsable de la mise en œuvre de sa propre logique de compensation.

La dernière solution prend comme hypothèse le fait que les services ne supportent pas tous *WS-Transaction* [CCC+02b]. Quand une défaillance se produit, le moteur d'exécution stoppe l'opération active et exécute la logique de compensation définie par le concepteur [ACKM03]. Quelques modèles et langages de composition comme BPEL, XLANG et BPML incluent des sémantiques transactionnelles qui permettent aux concepteurs de définir des mécanismes de compensation (voir plus loin, section 3.3.2).

3.3 Standards pour le web

Cette section présente deux types de standards : (i) ceux qui sont spécifiques aux transactions entre les services web et (ii) ceux qui sont conçus pour la composition de services web.

3.3.1 Protocoles spécifiques pour les transactions sur le web

La figure 3.2 montre la chronologie d'apparition des différents protocoles qui ont pour but de gérer les transactions pour des applications distribuées (via le web ou non).

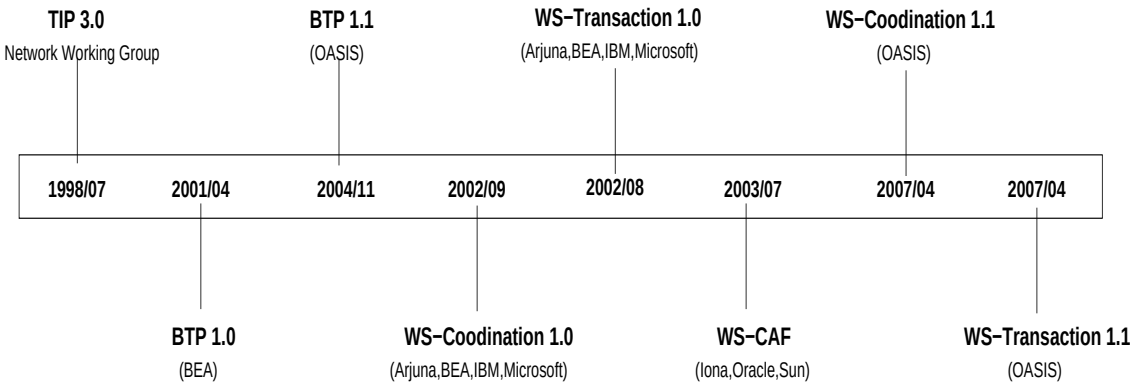


FIG. 3.2 – Les différents protocoles pour la gestion des transactions

Cette section présente les caractéristiques les plus importantes de chacun de ces protocoles.

Transaction Internet Protocol - TIP

Le protocole TIP, le premier protocole basé sur internet, a été développé pour remédier aux insuffisances du protocole 2PC dans les systèmes distribués hétérogènes. Ainsi, le protocole TIP vise à assurer la coordination entre les gestionnaires de transactions hétérogènes qui interagissent sur internet [LEK98]. L'objectif est de définir un protocole de validation à deux phases, plus léger que le protocole 2PC dans lequel la manière dont les participants acceptent le résultat d'une transaction n'est pas spécifiée *a priori*. Il permet aux données de la transaction d'être transmises par l'intermédiaire de différents protocoles de communication. Contrairement au protocole 2PC, le protocole TIP est indépendant de la couche de communication.

Business Transactions Protocol - BTP

La notion de transactions métiers introduite dans le protocole BTP [OAS02] vise à adapter

les concepts traditionnels de systèmes de transactions à l'environnement d'exécution des processus métiers. Ce protocole permet la coordination des applications qui s'appuient sur plusieurs participants appartenant à différentes organisations autonomes. Le protocole BTP n'a pas été conçu spécifiquement pour les transactions dans les services web.

L'objectif est de coordonner les relations entre les participants par le biais de contrats et non pas par une autorité centrale. De cette façon, une transaction peut avoir plusieurs résultats valides, et l'isolation et l'atomicité peuvent être relâchées, selon les termes du contrat. Le protocole est aussi conçu pour manipuler des transactions longues, pour permettre des résultats "flexibles"[1] pour les activités transactionnelles, et pour offrir un concept de transaction qui va au delà des transactions traditionnelles sur des données centralisées. Ce dernier point est particulièrement important parce que les interactions mises en œuvre dans un processus métier au-dessus d'internet, comme c'est le cas des transactions impliquant des services web, doivent assurer l'uniformité du processus aussi bien que la condition standard pour l'uniformité de données. De plus, les activités du processus sont presque toujours de longue durée, ce qui les rend difficiles à gérer par des protocoles de transactions ACID.

BTP s'appuie sur une variante du protocole de validation à deux phases pour garantir l'atomicité des mises à jour. Comme le protocole 2PC n'impose pas de limite de temps entre le deux phases, BTP utilise cette facilité pour introduire, entre les deux phases, des actions qui font référence aux points de contrôle définis dans la logique du processus métier. Malheureusement, ce comportement compromet l'indépendance entre la conception et l'exécution d'un processus métier.

La seule mise en œuvre de protocole a été celle proposée par Hewlett Packard : *Hewlett Packard Web Services Transaction* - HP-WST, abandonnée aujourd'hui.

Web Service Coordination - WS-C

WS-C décrit une structure extensible pour coordonner activités réalisées par des applications distribuées.

Cette spécification définit la manière d'utiliser conjointement différents protocoles, par exemple *WS-Transaction* et le protocole BTP, pour coordonner les appels de services qui composent un processus métier. De plus, cette spécification décrit la structure du contexte et les conditions selon lesquelles propager le contexte entre les services participants [CCC+02a].

WS-Coordination dans sa version 1.1 proposée par OASIS est devenu un standard le 16 avril 2007.

Web Service Transaction - WS-T

Dans le domaine des services web, *WS-Transaction* [CCC+02b] est un ensemble de spécifications construites au dessus *WS-Coordination*. *WS-Transaction* suppose l'existence d'un ensemble de services web qui participent à une transaction, ainsi que d'un ou de plusieurs coordinateurs qui coordonnent la transaction, soit de façon centralisée, soit de façon pair-à-pair.

WS-T propose deux modèles de transactions qui se traduisent par deux spécifications :

– *WS-Atomic Transaction* pour gérer des transactions selon la sémantique ACID. Par

[1] Chacun des participants dans un processus métier peut voir le résultat de la transaction.

conséquent cette spécification suppose que les ressources peuvent être verrouillées pendant la durée de la transaction.
– *WS-Business Activity*, conçue pour des transactions de longue durée, nécessaires au cours de l'exécution de processus métiers : toutes les mises à jour dans un système sont faites immédiatement, ce qui réduit de manière significative la période pendant laquelle les verrous doivent être tenus.

WS-Transaction ne peut pas être utilisé de manière indépendante, il s'appuie sur *WS-Coordination* pour les transactions de durées courtes et longues. Il définit plusieurs protocoles de coordination tels que la validation à deux phases pour les transactions courtes ou un protocole basé sur le principe d'accord mutuel (*Business Agreement Protocol* - BAP) pour les transactions longues.

WS-AtomicTransaction et *WS-BusinessActivity* dans leur version 1.1 proposées par OASIS sont devenus des standards le 16 April 2007.

Web Service Composite Application Framework - WS-CAF

Le canevas WS-CAF prend en compte des services web de différentes origines (.Net, J2EE, CORBA, etc.) [AFI+03, New04]. WS-CAF fournit les spécifications pour résoudre les problèmes liés à la fourniture d'une infrastructure commune où il est nécessaire de gérer les différents contextes liés à la diversité des données, comme c'est le cas des différents services web appartenant à différentes entreprises mais qui sont composés pour produire un résultat commun. WS-CAF offre un gestionnaire de contextes, un coordinateur générique et un support de type *plug* pour les modèles de transactions.

WS-CAF inclut trois spécifications :
– *Web Service Context - WS-CTX*, qui vise à simplifier la gestion du contexte et de l'état d'une transaction afin de permettre à tous les services web participant au même processus de partager un contexte commun.
– *Web Service Coordination - WS-CF* qui vise à gérer l'enrichissement du contexte, de son état à un instant donné et de son cycle de vie et assure la notification des résultats aux services web impliqués dans une transaction.
– *Web Service Transaction Management - WS-TXM* qui permet à plusieurs gestionnaires de transactions hétérogènes de partager le même contexte transactionnel.

Cette proposition est équivalente au couple *WS-Coordination* et *WS-Transaction*.

WS-CAF est une proposition qui sépare nettement le contexte, le coordinateur et le protocole transactionnel d'un processus métier. Mais, ces caractéristiques sont aussi proposées par BPEL que nous étudions dans la section suivante.

3.3.2 Protocoles spécifiques pour la composition

Plusieurs langages ont été proposés dans la perspective d'améliorer le travail de conception et de mise en œuvre des processus métiers, en particulier des processus de type B2B ainsi que toutes ses déclinaisons. L'éventail de ces spécifications va des outils pour la conception graphique des processus jusqu'à la conception de plates-formes pour la composition en utilisant diverses approches : algorithmiques, dirigées par les modèles, etc.

Cette section présente tout d'abord les principaux protocoles standardisés que visent la composition de services web.

Pour la modélisation de services, BPMI (*Business Process Modeling Notation*) décrit une notation pour la conception de processus métiers [BPM04]. Le but de cette spécification est d'unifier les notations existantes et de rendre plus facile la conception de processus métier en BPEL4WS[2] [ACD+03]. Similaire à UML, la spécification permet plusieurs types de diagrammes. Quelques propriétés transactionnelles sont aussi supportées.

UN-CEFACT est une méthodologie de modélisation qui s'appuie sur UML pour la modélisation des processus métiers [UC01]. Cet outil supporte quatre hiérarchies organisées comme des vues : les domaines de processus, les requis, les transactions et les services. En utilisant ces vues, un processus métier peut être modélisé selon une démarche descendante. Malheureusement, la modélisation graphique des propriétés transactionnelles n'est pas abordée. Le langage graphique ISDL - *Interaction System Design Language* a été proposé pour la modélisation de services web, mais il est très peu utilisé [QDvS04].

Le développement de services web a fait exploser le nombre de langages qui ont pour but d'implanter, de déployer, et de composer des services web. Malheureusement, ces propositions, par la diversité de leurs approches et des techniques utilisées, manquent d'orthogonalité. La figure 3.3 montre la chronologie d'apparition des différents langages pour décrire des orchestrations et des chorégraphies. UDDI, SOAP et WSDL sont dans la figure à titre de référence.

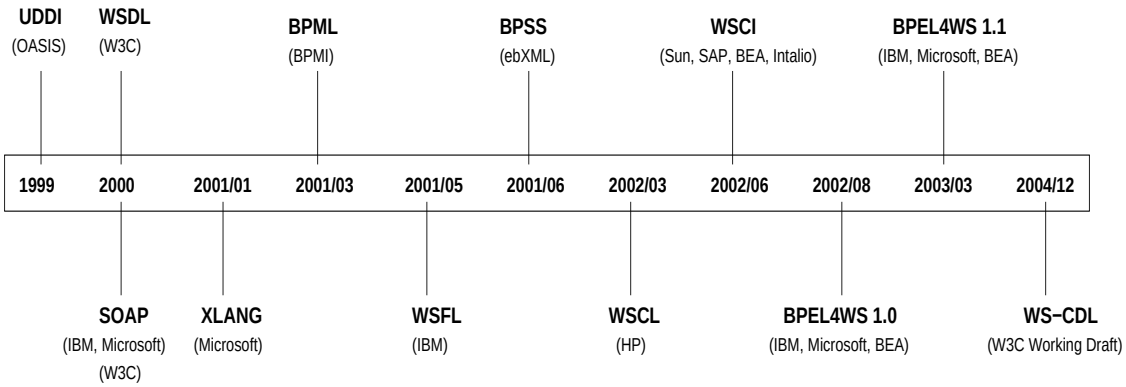


Fig. 3.3 – Langages de composition de services web (adapté de [JMS06])

Certains de ces langages permettent de concevoir la composition de services web centrée sur le paradigme de l'orchestration et d'autres, sur celui de la chorégraphie (voir le chapitre 2, section 2.2.3). Nous décrivons ci-dessous les langages les plus connus en soulignant leurs principales caractéristiques.

XML business process language - XLANG

XLANG a été créé par Microsoft au début de l'année 2001 comme une extension du langage de description de services web, WSDL [Tha01]. Il ajoute aux quatre types d'opérations de WSDL (requête/réponse, sollicitation de réponse, requête unilatérale, et notification) deux autres types d'action : arrêts (délai et durée) et exceptions. De plus, il permet de décrire en XML des processus collaboratifs. Le processus complet ne s'exécute pas sur un moteur unique, il est constitué de la collaboration entre plusieurs sous processus éventuellement répartis sur des environnements d'exécutions hétérogènes.

[2]Par la suite, BPEL seulement

Microsoft a exploité le langage XLANG dans son environnement BizTalk [Mic01]. BizTalk contient un environnement de développement complet : *BizTalk Orchestration Designer*, permettant de générer des ordonnancements XLANG. Le serveur BizTalk utilise ensuite ces ordonnancements XLANG pour instancier les processus à exécuter. XLANG est implémenté dans le produit BizTalk Server 2002 [Mic04a].

Web Services Flow Language - WSFL

En mai 2001, IBM a proposé le langage WSFL concernant la composition de services web [Lay01]. Il permet de décrire deux types de compositions : (i) un premier type décrit l'enchaînement des appels d'opérations de services web, comme avec un langage de programmation impératif. (ii) un second type décrit les interactions entre deux services web. Ce langage permet donc de spécifier des orchestrations et des chorégraphies.

La plate-forme *MQ Series Workflow* de IBM, connue aujourd'hui sous le nom de *WebSphere Process Manager* [IBM], a pris en charge la spécification WSFL afin d'automatiser des flots de processus métiers. IBM, Microsoft et BEA ont ensuite réalisé ensemble la spécification BPEL4WS (connu aujourd'hui sous le nom de BPEL), qui étend et remplace les précédentes spécifications, XLANG de Microsoft, et WSFL d'IBM.

Business Process Modeling Language - BPML

Développé par l'organisation BPMI.org[3], BPML a été publié en mars 2001. Initialement, BPML a été conçu pour modéliser et exécuter des processus industriels. Les dernières versions de BPML ont incorporé aussi un support pour WSCI (voir page suivante).

BPML gère la coordination de toutes sortes de participants, non seulement des services web. Il utilise un processus centralisé pour en assurer la coordination.

Un processus métier BPML est un enchaînement d'activités primitives ou structurées, et de processus incluant une interaction entre les participants dans le but de réaliser un objectif commun. Les conditions portent sur les variables globales du processus ou sur les données échangées entre participants. Les actions déclenchent d'autres tâches BPML.

BPML fournit des constructions de processus et des activités semblables aux activités BPEL telles que les activités primitives pour envoyer, recevoir, et appeler des services et des activités structurées pour manipuler des compositions conditionnelles, séquentielles, parallèles, des unions et des itérations. BPML permet également le déclenchement d'actions à des instants spécifiques. Le langage a été conçu pour contrôler des processus de longue durée et il inclut aussi des dispositifs pour gérer la persistance. Les interactions entre les participants sont gérées par des constructions similaires à celles proposées dans BPEL. BPML permet également la composition récursive pour définir des processus globaux composés de sous-processus. BPML supporte des transactions de durée courte et longue, en utilisant une notion de portée semblable à celle de BPEL pour contrôler les règles de compensation. Les concepteurs de processus peuvent emboîter des processus et des transactions et utiliser un mécanisme de gestion des exceptions.

La société Intalio a développé le produit *Intalion3* qui est un système de gestion de processus métiers basé sur le standard BPML et indépendant de la plate-forme d'exécution [pro]. *Intalion3* contient cinq composants permettant de concevoir, déployer, exécuter, maintenir, et optimiser les processus. Le cœur de ce système est une machine virtuelle de processus

[3]Business Process Management Initiative, http ://www.bpmi.org

permettant d'exécuter les processus métiers conformément à la spécification BPML. L'outil de conception, dans *Intalion3*, permet de générer plus de 80 pour cent du code requis pour construire un processus métier exécutable, ce qui réduit le temps de développement. L'environnement de gestion de *Intalion3* permet de gérer, configurer, et superviser les processus déployés. Les utilisateurs peuvent interagir avec le processus déployé à travers une application web.

Web Services Choreography Interface - WSCI

Sun, SAP, BEA, et Intalio ont proposé le langage WSCI devenu une note dans le W3C le 8 août 2002 [AAF+02]. WSCI permet de décrire les interfaces de services web afin de réaliser une chorégraphie entre eux. La collaboration est coordonnée de manière décentralisée sans utiliser de processus principal, mais plutôt plusieurs processus distribués.

L'objectif consiste à prendre en compte des interactions d'application à application. Le but de WSCI est de décrire les détails du comportement d'un service web en terme de dépendances temporelles et logiques entre les messages que ce service web échange avec d'autres services web. Un service web peut avoir plusieurs interfaces qui lui permettent de jouer différents rôles. Les interfaces statiques des services web sont décrites en WSDL et la chorégraphie entre eux est décrite en WSCI.

WSCI décrit seulement le comportement observable entre les services web et il ne permet pas la définition de processus métiers exécutables comme le fait BPEL. Il faut remarquer qu'aujourd'hui il n'existe aucune plate-forme permettant d'exécuter des chorégraphies basées sur les interfaces décrites en WSCI. Ceci est certainement dû à la difficulté de réaliser une coopération de services web de manière décentralisée qui ne permet pas aux participants d'avoir une vision globale de la coordination.

Une interface WSCI décrit uniquement la participation des partenaires dans le cadre d'échanges de messages. Une chorégraphie doit inclure un ensemble d'interfaces WSCI, une pour chaque partenaire qui intervient dans l'interaction. Hors, dans WSCI, aucun processus ne contrôle l'interaction : chaque action représente une unité de travail qui correspond à une opération spécifique définie dans l'interface WSDL. WSCI étend WSDL par la définition de chorégraphies basées sur les opérations spécifiées dans l'interface WSDL. En d'autres termes, WSDL décrit les points d'entrée pour chaque service disponible, et WSCI décrit les interactions en incluant les opérations décrites en WSDL.

WSCI supporte des activités primitives et des activités structurées. Comme BPEL, WSCI supporte une grande variété d'activités structurées : des compositions conditionnelles, séquentielles, parallèles et itératives. Les échanges de messages sont décrits par des étiquettes d'action. Une étiquette d'action définit un message de base destiné à émettre une requête ou à recevoir une réponse. Chaque activité indique l'opération WSDL impliquée et le participant spécifique qui doit l'exécuter. Une chorégraphie peut alors appeler des services externes par le biais d'une étiquette d'appel. Sun Microsystems propose un éditeur d'interfaces WSCI appelé *Sun ONE WSCI Editor*. Une version initiale de cet éditeur a été publiée en accès gratuit. Cette version de l'éditeur est assez légère, mais elle fournit les fonctionnalités de base pour construire des descriptions d'interfaces WSCI à partir de descriptions WSDL. Il s'agit d'interfaces qui décrivent des processus abstraits, non exécutables [Pel03]. Ici encore, la difficulté de réaliser une coopération de services web de manière décentralisée limite la portée de l'utilisation du langage.

Web Services Conversation Language - WSCL

Hewlett-Packard a soumis au W3C, le 14 mars 2002, le langage de description de conversation WSCL [BBC+02]. L'objectif consiste à décrire la séquence des interactions possibles entre deux services web. WSCL est donc moins ambitieux sur ce point que WSCI qui décrit des conversations entre plusieurs services web.

Comme WSCI, WSCL s'appuie sur WSDL pour la description des opérations offertes par les services. Ainsi, la description de l'interface des services est séparée de celle de leur comportement, ce qui en facilite la réutilisation. Par exemple, une même conversation WSCL peut avoir lieu entre différentes paires de services web décrits en WSDL, chaque service pouvant participer à plusieurs conversations WSCL. Ceci différencie WSCL des autres approches comme XLANG et WSFL. Un fournisseur de service peut, soit fournir la définition de conversations WSCL directement à l'utilisateur du service, soit enregistrer la description de son interface dans un annuaire UDDI afin de publier le service correspondant.

Bien que WSCL offre une vision globale du processus (contrairement à WSCI) pour décrire les interactions qui sont coordonnées de manière décentralisée, il n'existe aucun système permettant d'exécuter une conversation basée sur WSCL. Ce qui confirme la difficulté de réaliser une coordination décentralisée de services web.

Puisque BPML et WSCI partagent le même modèle de processus d'exécution, les concepteurs peuvent employer WSCI pour décrire des interactions publiques entre les processus métiers et réserver BPML pour développer des mises en œuvres privées des processus.

Web Services Choreography Description Language - WS-CDL

Le langage de description de chorégraphies de services web, WS-CDL est un langage qui décrit des collaborations *pair-à-pair* des partenaires en définissant, selon un point de vue global, leur comportement observable commun et complémentaire, dont le résultat est l'accomplissement d'un processus métier [KBR+04].

En WS-CDL, seules les interactions entre deux services web sont considérées. La description qui en découle prend la forme d'un processus dont l'exécution est effectuée de manière décentralisée, comme une chorégraphie. La description des interactions est indépendante des environnements d'exécution des services impliqués. Une définition globale des conditions et des contraintes selon lesquelles les messages sont échangés est donné par le biais d'un contrat que chaque partenaire doit respecter.

Le niveau d'abstraction de WS-CDL se situe au dessus de celui des standards, notamment WSDL et WSDL-MEPs[4], la description de mises en correspondance de WS-CDL vers les standards est donc nécessaire.

Les principaux problèmes que WS-CDL présente sont, le manque de séparation entre le méta-modèle et la syntaxe du langage. WS-CDL essaie de définir simultanément un méta-modèle pour décrire les chorégraphies de service et une syntaxe basée sur XML. En conséquence, il n'y a aucune séparation stricte entre les aspects sémantiques et syntaxiques. Un méta-modèle de chorégraphie de services devrait être développé indépendamment d'un format particulier d'échange.

[4] WSDL-MEPs - Message Exchange Patterns, http ://dev.w3.org/cvsweb/2002/ws/desc/wsdl12/Attic/wsdl12-meps.html ?rev=1.2

Business Process Execution Language for Web services - BPEL

Le langage BPEL [ACD+03] est une spécification proposée conjointement par IBM, Microsoft, et BEA et qui fusionne et remplace les précédentes spécifications XLANG de Microsoft, et WSFL d'IBM. BPEL est un effort pour standardiser la composition de services web [WvdADtH03, ACD+03]. BPEL est un langage pour définir et gérer des activités d'un processus métier. Ce langage permet de décrire des protocoles d'interactions et de collaborations entre les services web sur lesquels s'appuie le processus. BPEL utilise le modèle de contrôle classique des flots de tâches (*workflows*) pour décrire des processus métiers.

Les processus BPEL interagissent en invoquant d'autres services web et en recevant des invocations de ces services web [Pap03]. La relation entre le processus et un partenaire est une relation *pair-à-pair*. Le partenaire est en même temps le consommateur d'un service que le processus produit, et le producteur d'un service que le processus consomme. Le lien de partenariat définit le rôle qui joue chacun des deux partenaires dans une conversation. Ainsi, ce langage définit des chorégraphies lorsqu'il s'agit de décrire le contrat d'interactions entre deux services web, mais aussi des orchestrations lorsqu'il s'agit de décrire l'enchaînement des appels d'opérations offertes par des services web. Avec BPEL il est possible de concevoir deux types de processus :

- Le *Processus abstrait* : spécifie les échanges de messages entre les différentes parties, sans spécifier le comportement interne de chacun d'eux.
- Le *Processus exécutable* : spécifie l'ordre d'exécution des activités constituant le processus, les partenaires impliqués dans le processus, les messages échangés entre ces partenaires, et le traitement des fautes et des exceptions.

Le processus dans BPEL est constitué d'activités et d'un flot de contrôle. Les activités peuvent être primitives ou structurées. BPEL a les caractéristiques d'un langage structuré en blocs (caractéristique issue de XLANG), ainsi que celles d'un flot de tâches (caractéristique issue de WSFL). L'interface d'un processus BPEL est décrite en WSDL. *WS-Coordination* et *WS-transaction* peuvent être utilisés pour étendre un processus BPEL.

BPEL est très largement utilisé. De nombreuses mises en œuvre ont été réalisées (voir [Jam05]) :

IBM a même remplacé, dans son logiciel *WebSphere* WSFL par BPEL. Oracle[5], a aussi intégré BPEL dans son serveur d'applications ainsi que IBM, qui a développé *IBM WebSphere Business Integration Server Foundation*[6], et *IBM alphaWorks BPWS4J*[7].

- "Collaxa Inc." est une société, fondée en décembre 2000, dont la vision est de construire les applications sur les standards d'orchestration. Cette société était la première à proposer un serveur d'orchestration de services web basé sur BPEL. Les partenaires de Collaxa sont BEA Oracle, et Sun Microsystems [Sky02, Pel03].
- En juin 2004, Oracle a racheté Collaxa, afin de l'intégrer dans son serveur d'applications "Oracle Application Server 10g" [Ora04].
- Le laboratoire *alphaWorks* d'IBM a produit une mise en œuvre spécifique BPWS4J (*Business Process Execution Language for Web Services Java Run Time*). BPWS4J

[5] http ://www.oracle.com/technology/products/ias/bpel/index.html
[6] http ://www-306.ibm.com/software/integration/wbisf/features/
[7] http ://www.alphaworks.ibm.com/tech/bpws4j

permet de décrire et d'exécuter des processus métiers conformément à la spécification BPEL [alp02, Pel03].
- Dans le cadre du projet "Jupiter" Microsoft a créé le langage XLANG, et l'a intégré dans son produit *BizTalk Server 2002*. Le logiciel *WebSphere Process Manager* d'IBM était fondé au départ sur la spécification WSFL, remplacée ensuite par BPEL [KCH+04]. Microsoft a également introduit BPEL dans son produit *BizTalk Server 2004* et l'a rendu capable d'importer et d'exporter des processus métiers écrits en BPEL [Mic04b].
- La société Momentum[8] a également choisi BPEL parmi les standards d'orchestration de services web, pour son produit d'automatisation de processus *ChoreoServer for .NET* (aujourd'hui disponible aussi pour Java/J2EE).

3.4 Approches académiques pour la composition

La composition de service est un secteur très actif de recherche et de développement. Cette section étudie les propositions selon qu'elles sont :
- des approches algorithmiques,
- des plates-formes pour la composition de services,
- des approches dirigées par les modèles

Bien que la plupart de ces différentes propositions n'intègrent pas les aspects liés à la gestion des transactions, nous les présentons dans la perspective de les comparer au modèle de composition sous-jacent au canevas que nous proposons.

3.4.1 Approches algorithmiques

Dans les architectures actuelles les intergiciels à base de composants occupent une place importante. Ces intergiciels sont appropriés pour l'intégration d'un petit nombre de services fortement couplés entre eux et munis d'interfaces stables. Ils s'appuient sur des structures d'objets distribués comme CORBA, DCOM, EJB, ou d'autres technologies comme celles à base d'EAI (par exemple, *IBM MQSeries*) ou d'ERP (par exemple, *SAP* ou *R/3*), ou encore des bases de données et des moniteurs de transactions.

Un intergiciel à base de composant fournit des structures de données standards et des outils pour l'intégration d'applications hétérogènes. Dans cette approche, des services composites peuvent être assemblés à partir de composants développés indépendamment. Le défaut est que le code pour la composition est développé de manière *ad-hoc*. De plus, une composition peut impliquer un grand nombre de partenaires, organisés selon une typologie inconnue *a priori* et très évolutive [BDS05].

Une autre approche consiste à étendre les concepts de programmation (autant que possible) à la conception de services web dans la perspective de faire face à la complexité inhérente des grands systèmes des logiciels et de fournir des concepts pour permettre une meilleure réutilisation des objets déjà existant [Loe03]. La plate-forme proposée offre des services d'infrastructure pour la sécurité, la persistance, et la gestion des transactions. Au temps d'exécution, ces services sont fournis par l'environnement d'exécution de l'application. Cependant, la logique des processus métiers et les services d'infrastructure doivent être

[8] http ://www.momentumsi.com/, http ://www.perfectxml.com/411/DirDetails.asp ?ID=193

intégrés par la suite. Ceci peut être fait de façon programmée en insérant des ordres de contrôle dans le code source des composants ou de façon déclarative par des annotations au format prédéfini associées aux opérations fournies par les composants. Les annotations sont alors interprétées au temps d'exécution par la plate-forme pour activer les services requis. Des développements *ad-hoc* posent problème ici aussi.

Le service de gestion des transactions décrit le comportement transactionnel pour chaque application. Ce comportement implique la délimitation de la transaction, la définition des dépendances entre les transactions et la spécification du comportement de l'application lors de l'occurrence de fautes.

Dans [DDR05], les auteurs proposent un langage de programmation pour supporter le développement compositions de services web. Ce langage combine des fonctionnalités pour le développement de services web avec celles conventionnelles des langages de programmation. Le langage se caractérise par la manipulation de données XML, la gestion des messages échangés, le contrôle de la concurrence et de la corrélation de messages. Comme le langage est nouveau il s'avère nécessaire de mettre en correspondance ses constructions avec celles des standards comme WSDL, BPEL et WS-CDL. Malheureusement ce qui se gagne avec l'utilisation de ce nouveau langage, se perd avec ce besoin intégration. Les propriétés transactionnelles des compositions obtenues ne sont pas étudiées.

3.4.2 Plates-formes pour la composition de services web

Dans ce domaine, la recherche et le développement ont été particulièrement riches. La nécessité d'automatiser, de standardiser et de rendre les applications extensibles a suscité la proposition de nombreuses approches. Celles-ci proposent de modéliser les compositions de services par le biais de chorégraphies ou d'orchestrations. Dans la majorité des cas, les modèles traitent des aspects liés à la logique du processus métier (le flot du processus, les aspects fonctionnels), en laissant de coté les aspects extra fonctionnels de la composition. Par exemple, dans [OYP03] les auteurs proposent d'exprimer la composition d'un processus par le biais de règles de planification. La méthodologie s'appuie sur UML, en particulier, les règles de composition sont exprimées en OCL (*Object Constraint Language* [Gro03]). Dans cette approche les services web à contacter sont choisis au cours de la phase de planification, il n'existe pas d'action prévue si un de ces services n'est pas disponible au moment de l'exécution. Les aspects transactionnels ne sont pas traités.

SELF-SERV [FDBP01, BSD03, BDS05] et eFlow [CIJ+00] sont sans doute les propositions les plus significatives de plates-formes pour la composition de services web. Dans SELF-SERV l'orchestration de services est modélisée par le biais de diagrammes d'états [Har87], tandis que dans eFlow le formalisme choisi est celui des réseaux de Petri.

SELF-SERV fournit des outils pour spécifier la composition et des règles de conversion de données entre les services impliqués dans la composition. SELF-SERV comme eFlow permettent la sélection de services par divers critères. La coordination des exécutions d'une composition est, dans SELF-SERV, distribuée sur l'ensemble des partenaires alors qu'elle centralisé dans eFlow. Ces deux plates-formes se concentrent sur la découverte et la composition de services, laissant de côté les aspects transactionnels.

CLF (*Coordination Language Facility* [APR00]) est une autre plate-forme dont les caractéristiques sont proches de la nôtre, même si l'objectif est l'intégration de protocoles

tels que Java/Jini, CORBA et EJB, et non pas la composition de services. CLF a été conçu pour l'exécution de transactions distribuées à travers plusieurs composants hétérogènes et tirant profit des caractéristiques transactionnelles que chaque composant fournit. L'architecture proposée dans CLF offre deux avantages. Tout d'abord, il n'est pas nécessaire de modifier le composant et les systèmes existants et les applications peuvent être ainsi plus facilement réutilisées. En second lieu, il est possible, avec l'aide de la bibliothèque de Mekano[9], d'étendre quelques propriétés ACID d'un composant particulier quant il s'agit de gérer une ressource. Pour éviter les problèmes liés à la concurrence des transactions, il est possible d'employer les systèmes transactionnels natifs de l'environnement de chaque composant intervenant dans la transaction.

CLF propose de coordonner les composants provenant de différents environnements d'exécution dans une transaction distribuée. Dans ce sens, il ressemble à la proposition de WS-CAF (voir section 3.3.1, page 33). CLF est un coordinateur capable d'activer des plans qui décrivent les interactions entre les différents composants hétérogènes. Le plan est exécuté en deux phases : la première qui consiste à construire une solution au problème associé au plan et la seconde qui exécute de façon atomique les actions proposées par les composants à la première phase (cette phase s'appuie sur le protocole 2PC).

Dans [BCTH03], les auteurs proposent un canevas construit sur la spécification WSDL qui permet d'enrichir la description des caractéristiques des services web par la modélisation des conversations dans lesquelles les services peuvent s'impliquer. Cette modélisation s'appuie sur le formalisme des diagrammes d'états. Le canevas traite des aspects transactionnels par la prise en compte, en particulier, de mécanismes de compensation. Cependant, la propriété atomique d'une transaction n'est pas relâchée.

Let's Dance est un langage de description de chorégraphies proposé récemment [ZBtHA06]. Ce langage est un langage visuel destiné à la modélisation des interactions de services web, et cela, à deux niveaux d'abstraction : (i) les interactions au niveau global pour décrire le modèle global de la chorégraphie et (ii) les interactions au niveau local, pour décrire un ou plusieurs modèles locaux appartenant au modèle global. A la différence des langages comme WS-CDL ou BPEL, cités précédemment, qui ont une structure de langage de programmation, *Let's Dance* est un langage dédié à l'analyse et à la conception de processus métiers. Une chorégraphie dans *Let's Dance* est décrite par un ensemble d'interactions entre services. Ces interactions se traduisent par des échanges de messages entre les services participants. Maestro est la mise en œuvre ce langage [DKZD06]. Il permet la modélisation des interactions des services rentrant dans une chorégraphie et il supporte aussi l'analyses statique des modèles globaux et la génération des modèles locaux à partir du modèle global. Les interactions entre ces modèles peuvent être obtenues par simulation.

Ce langage peut, dans une certaine manière, remplacer WS-CDL, car il est plus expressif et surmonte les insuffisances de WS-CDL. Cependant, le langage reste dans le domaine de la composition et ne traite pas de la problématique transactionnelle.

[9] CLF/Mekano est un système d'exploitation ouvert, voir les détails dans [AAP+99]

3.4.3 Approches dirigées par les modèles

L'approche dirigée par les modèles a été appliquée à la conception de services web composés depuis quelques années, mais dans ce contexte, la modélisation des propriétés transactionnelles n'a pas fait l'objet de nombreuses recherches.

Dans cette classe d'approches, les auteurs dans [SD04] ont développé une solution dans laquelle la phase de la conception dédiée à exprimer des propriétés transactionnelles pour la composition est clairement séparée des autres préoccupations. Cette approche peut être utilisée pour spécifier d'autres propriétés de services web telles que la sécurité ou la qualité. Les auteurs proposent l'utilisation de deux couches. La modélisation, à chaque niveau de préoccupation s'appuie sur UML et OCL est utilisé pour l'expression des liens entre les niveaux. Ce genre d'approche réduit la complexité et facilite la maintenance. Le modèle permet de définir plusieurs types de transactions : (i) des transactions de longue durée où sont considérés les mécanismes de compensation et si nécessaire, l'intervention humaine ; (ii) des transactions alternatives programmées au moment de la conception ; (iii) des transactions deux phases, dont une phase de pré-transaction suivie d'une phase de transaction. Dans la de pré-transaction l'approche de [LZ04] basée le protocole THP est utilisée pour connaître les ressources qui deviennent indisponibles avant de rentrer dans la phase de transaction.

Une autre approche dirigée par des modèles est celle proposée dans [BBCT04]. Le canevas supporte le développement des services composés en commençant par la spécification externe de la composition (interface et spécifications du protocole). Le modèle génère automatiquement des services à travers des patrons de comportement, ce qui permet de simplifier le travail de développement.

Généralement, les plates-formes pour la composition des services web proposent des méthodes de conception ascendantes ou descendantes, mais aucune ne facilite vraiment le développement de services conformes aux besoins. Le programmeur doit mettre en œuvre la logique de manipulation du protocole et vérifier que le service se comporte conformément aux spécifications externes. Ce travail prend bien sûr beaucoup de temps et peut générer de nombreuses erreurs. Dans [BBCT04], les auteurs essayent de résoudre cette problématique en proposant une solution qui offre une méthode de conception dirigée par des modèles et utilisée conjointement avec une technique de conception descendante : il s'agit de considérer les spécifications de l'interface du service et de générer à partir de là l'interface en Java par exemple. A partir des spécifications externes, il est possible de générer le patron d'un service composé. Cette idée peut être appliquée à la logique du métier ce qui permet de compléter la spécification. Ce type d'approche décharge le programmeur d'une partie du travail de codage et facilite la validation du service obtenu. La question de la gestion des transactions n'est pas abordée.

Le canevas *QBroker* est destiné à faciliter la sélection des services web selon des critères de qualité dans le cadre d'une composition spécifique [YZL07]. Comme il peut y avoir plusieurs services web offrant la même fonctionnalité mais des qualités de services différentes il est possible de sélectionner services les plus appropriés, en termes de qualité de services, pour la composition. Cette sélection est effectuée au moment de la conception. Le problème a été abordé selon un point de vue combinatoire et selon le point de vue de la théorie des graphes. Ainsi le point de vue combinatoire présente le problème comme un problème

multidimensionnel et multicritère ; le point de vue des graphes consiste à trouver la chemin optimal en prenant en compte les multiples restrictions. La découverte de services web, la sélection, la planification et l'adaptation sont les principales fonctionnalités offertes par le canevas.

3.5 Composition et aspects transactionnels

La recherche sur la problématique transactionnelle n'est pas riche par rapport à celle effectuée par exemple sur la composition de services. Tous les services n'ont pas le même comportement transactionnel et il est difficile de construire un environnement où ils puissent réagir de manière homogène. De plus, la problématique transactionnelle ne fait pas partie des aspects fonctionnels d'une composition, en conséquence, dès lors que la composition est conçue les aspects transactionnels qui la concernent sont dans la majorité des cas traités par le biais de contrats ou d'accords préalables qui fixent la manière de conduire les interactions entre les partenaires.

Cette section présente les travaux traitant de la composition de services web en considérant leurs aspects transactionnels. Nous avons mis en avant un ensemble de critères qui facilitent leur comparaison. Ces critères sont :

- La *séparation des préoccupation*, qui fait référence à la séparation de la logique de la composition de la logique transactionnelle sous-jacente à cette composition. Concernant les avantages de ce mécanisme, il y a la clarté de la conception et l'identification facile des aspects liés à chaque aspect, ce qui facilitent la mise en œuvre et la mise à jour de chacun d'eux.
- La *limitation de la portée d'une transaction* qui se traduit par la spécification explicite de l'ensemble des opérations qui doivent être exécutées avec la sémantique du tout ou rien. Les opérations qui s'exécutent dans cette limite sont généralement appelées zones atomiques ou sphères atomiques. Cette technique est utilisée dans le but d'assouplir la propriété d'atomicité dans le sens où minimiser le nombre d'opérations devant être obligatoirement exécutées avec succès, maximise la probabilité de réussite de la transaction.
- La *prise en compte des besoins extra fonctionnels* d'une application. Ces besoins extra fonctionnels (l'expression *besoins non fonctionnels* est utilisée aussi dans la littérature) représentent des propriétés de l'application qui se placent en dehors de la portée de la logique d'un processus métier mis en œuvre par l'application (par exemple, la qualité de service).
- La *proposition d'un canevas, d'une méthodologie*. La problématique liée à la prise en compte des transactions dans la composition de services web est posée depuis la conception de la composition jusqu'à la coordination de ses exécutions. Les propositions étudiées sont plus ou moins complètes à cet égard, nous proposons de les comparer selon ce critère.

3.5.1 Séparation des préoccupations

Ainsi, une des premières approches qui a traité de manière formelle la séparation de la logique du processus de la gestion d'exceptions a été celle proposée dans [HA00]. Jusque là, il est courant de mélanger la logique du processus avec celle de la gestion des exceptions, ce

qui rend difficile les tâches de maintenance. La réutilisation du processus est aussi limitée. Ces remarques s'appliquent aussi aux compositions de services web. Les auteurs proposent une solution qui s'appuie sur la notion de sphères atomiques [Dav78] en associant une sphère atomique à une tâche ou à un ensemble de tâches du processus métier. De cette façon, si une tâche dans la sphère échoue la procédure associée à la sphère est exécutée pour gérer l'exception. Cette approche propose un algorithme pour valider la conformité des sphères. Ce mécanisme de validation est une contribution importante dans la perspective de développer des processus tolérants aux fautes.

Ce travail de recherche est à la base de plusieurs approches (voir par exemple [VV03, Bhi04, BGP05]).

3.5.2 Limitation de la portée des transactions

Une transaction peut être étudiée selon le point de vue des ressources qu'elle acquiert. Limiter la portée d'une transaction revient à déterminer le nombre minimal requis que la transaction doit acquérir. Le concept de *sphère* a été introduit pour cela [vdHA02]. Le modèle de transactions proposé introduit des sphères d'infrastructure et des sphères d'exécution basée sur la notion des sphères atomiques [Dav78]. Les sphères d'infrastructure représentent la partie statique et celles d'exécution correspondent à la partie dynamique. Cette approche permet aussi de distinguer entre la définition fonctionnelle d'une transaction métier de sa définition extra fonctionnelle. Les auteurs prônent la nécessité de munir les services web de propriétés transactionnelles pour mettre en œuvre des processus métiers fiables et robustes. Ils proposent un canevas dans la perspective d'exécuter des transactions avec les propriétés ACID. L'idée est d'étendre les langages de composition par des propriétés transactionnelles essentielles (comme proposé dans [Alo05]).

Les sphères atomiques sont des collections d'actions qui peuvent être représentées à plusieurs niveaux de granularité. Une sphère (et ses sous-sphères) est atomique si ses actions sont toutes exécutées, ou aucune. Avec ce mécanisme, il est possible de faire des transactions emboîtées, très utiles dans l'environnent de services web.

Le canevas propose d'autre part d'étendre WSFL[10] par les propriétés ACID étendues. Les propriétés transactionnelles peuvent être décrites en utilisant une approche déclarative ou fonctionnelle. L'approche déclarative est la déclaration des propriétés transactionnelles désirées par le biais de méta-données, alors que l'approche fonctionnelle décrit les orchestrations de services web par la définition d'une séquence de tâches à exécuter, des ressources externes et de la gestion des exceptions. La version ainsi étendue de WSFL inclut des constructeurs pour définir des exécutions parallèles et concurrentes. Des opérateurs de compensation sont aussi proposés pour traiter les défaillances des transactions, ainsi que la possibilité de défaire et refaire une transaction.

Ce travail a été l'un des premiers à étudier les transactions dans le domaine de services web. D'une certaine façon, cette proposition avec celle de [HA00] constituent les bases de la partie transactionnelle de notre canevas.

[10] WSFL - *Web Services Flow Language*, remplacé par BPEL [Lay01]

3.5.3 Prise en compte des besoins non fonctionnels des applications

Les approches que nous étudions ici visent à associer, à un processus métier, des caractéristiques extra fonctionnelles, les séparant ainsi la logique de l'application.

Le canevas *WSTx* (*Web Services Transaction* [MTR02]) introduit la notion de *comportement transactionnel* (*Transactional Attitudes*). Il s'agit d'un intergiciel centralisé qui gère et contrôle les transactions, telles qu'attendues par les clients en accord avec les propriétés transactionnelles exposées par le fournisseur du service. Cette notion est proche de celle d'*intermédiaire* introduite dans [KGM98]. Selon l'approche proposée par le biais de ce canevas, les fournisseurs exposent leurs comportements transactionnels et les clients exposent leurs exigences en ce qui concerne la gestion des transactions. Le canevas *WSTx* inclut un module qui est responsable de la gestion des interactions entre les clients et les fournisseurs, de manière cohérente d'une part avec les attentes des clients et d'autre part avec les possibilités des fournisseurs.. *WSTx* s'appuie sur WSDL pour la description des interfaces.

Dans [VV03], les auteurs présentent un modèle de composition de services multi niveaux. Un service est spécifié comme processus à plusieurs niveaux d'abstraction. Le modèle permet de faire la description des spécifications des propriétés transactionnelles de la composition comme l'atomicité et la garantie de terminaison à chaque niveau. Puis une mise en correspondance est définie entre l'abstraction de bas niveau qui est basée sur les transactions classiques [WV02], un modèle qui généralise des garanties transactionnelles au niveau de processus [SBS02], et l'abstraction de haut niveau comme celui utilisé dans PARIDE [MPP02] qui orchestre des services par l'intermédiaire des réseaux de Petri. Dans ce modèle de composition les services sont distingués selon qu'ils sont compensables, pivots et rejouables. Le but du modèle est de déterminer les terminaisons correctes du processus en s'appuyant sur les propriétés transactionnelles des composants.

Une des premières approches qui a traité de la problématique transactionnelle dans le domaine du commerce électronique est proposée dans [KGM98]. Cette approche prend en compte le problème du manque de confiance et propose une solution basée sur des techniques multi agents : un agent *intermédiaire de confiance* est désigné parmi les participants de la transaction. Celui-ci est chargé de surveiller que la transaction se déroule en toute équité pour les participants.

3.5.4 Canevas et architectures

Dans cette partie, nous présentons les approches dont l'objectif est de proposer un environnement pour la composition de services web. Cet environnement prend le plus souvent la forme d'un canevas.

Le canevas WebTP définit une architecture pour le traitement de transactions sur le web, basées sur des services web. WebTP s'appuie sur le modèle X/Open DTP (*X/Open Distributed Transaction Processing*) [QRTY04]. Les services considérés dans WebTP concernent la gestion des accès à différentes bases de données disponibles sur le web. Le protocole X/Open DTP permet à plusieurs applications de partager ces ressources et d'en coordonner la gestion par le biais de transactions globales. La majorité des SGBDR (par exemple, Oracle, Sybase, DB2) supportent ce modèle. Cette architecture est principalement adaptée

à la gestion des accès à des bases de données sur internet.

Dans [LZ04] les auteurs présentent une architecture de composition basée. Le formalisme utilisé pour l'expression des compositions est celui des diagrammes d'états [Har87]. Des mécanismes de compensation *a posteriori* remplace le protocole 2PC, inapplicable dans le contexte du web. Le protocole de tentative de réservation (*Tentative Hold Protocol* - THP [RS01, RCMS01], voir page 27) afin d'éviter de devoir défaire des transactions qui échouent.

Un autre canevas pour la composition et l'orchestration de services transactionnels est proposé dans [Bhi05]. L'objectif majeur de l'étude est permettre au concepteur de décrire des compositions fiables en indiquant : (i) la structure globale de la transaction et (ii) les états de terminaison corrects (comme dans [VV03]). Le canevas est conçu en trois niveaux : niveaux des processus de service, niveau des processus coopératifs inter entreprise et niveau de la gestion des conversations. Pour la définition des types de terminaison l'étude reprend les notions d'activités compensables, pivots et rejouable déjà introduites [VV03]. L'approche hérite très fortement de l'acquis du domaine des flots de tâches. En effet, le concept d'état du processus est utilisé pour déterminer si une tâche en situation d'échec, peut être relancée ou non. La propriété d'atomicité n'est pas relâchée, bien au contraire, elle prône pour une exécution de type "tout au rien". Les services entrant dans la composition sont choisis au moment de la conception.

3.6 Synthèse

Nous venons de présenter un ensemble de travaux dont les contributions pour la prise en compte des aspects transactionnels dans le contexte de la composition de services web, si mêmes si elles ne couvrent pas l'ensemble de la problématique, nous semblent significatives. Tout au long de ce chapitre, nous avons décrit soit des approches dédiées aux mécanismes transactionnels qui ne tiennent pas vraiment compte des aspects liés à la composition (c'est le cas des standards spécifiques aux aspects transactionnels que nous avons présentés dans la section 3.3.1), soit des approches qui se concentrent sur les problèmes liés à la composition de services sans proposer de fonctionnalités de gestion des transactions (c'est le cas des langages dédiés à la composition et présentés dans la section 3.3.2). La même remarque s'applique à la majorité des approches étudiées dans la section 3.4.

La synthèse proposée dans ce chapitre est structurée en deux parties. La première partie (3.6.1) discute des apports des standards, et la seconde partie (3.6.2) fait référence aux approches spécifiques.

3.6.1 Protocoles standards

Dans les protocoles dédiés à la gestion des transactions dans les services web, les fonctions concernant les transactions sont proposées au même niveau de quelles liées à la composition, sans véritable séparation. L'objectif poursuivi a été de définir un modèle générique et paramétrable, applicable à n'importe quel type de transactions dans n'importe quel contexte applicatif, comme celui proposé conventionnellement dans le domaine des bases

de données. Malheureusement, les types de transactions dans le domaine des services web sont bien différentes et les contours d'un modèle unique et générique sont difficiles à identifier.

Dans le protocole précurseur TIP, le protocole 2PC a été adapté aux services web. Mais rapidement il a été abandonne et il lui a succédé le protocole BTP, mieux adapté aux caractéristiques des processus métiers transactionnels, puis ont suivi de *WS-Transaction* et *WS-Coordination*.

BTP a un impact sur l'implémentation du processus, tant sur le plan transactionnel que sur le plan du service. BTP, *WS-Transaction* et WS-CAF ont des caractéristiques très proches. Ce qui rend difficile la tâche du conception du processus métiers quand il s'agit d'en choisir un. Par ailleurs, la pression exercée par les principaux acteurs du marché a produit deux "camps" bien séparés : d'un côté BEA, IBM et Microsoft et de l'autre côté Iona, Oracle et Sun Microsystems. Les premiers soumettent les propositions de leurs spécifications au groupe OASIS[11] et les seconds les proposent au groupe W3C. La différence est que le groupe W3C[12] prône pour des spécifications de libre diffusion tandis qu'OASIS prône pour un système de licences payantes.

En conséquence, la spécification WS-CAF avait peu de chances de s'imposer face à l'ensemble des spécifications regroupées maintenant sous le standard *WS-Transaction version 1.1*. Une des raisons a été le nombre très faible de mises en œuvres du standard WS-CAF, et l'autre le soutien très fort des compagnies BEA, IBM et Microsoft au profit du standard *WS-Transaction*. Cependant, la spécification WS-CTX, présente dans WS-CAF et concurrente directe de *WS-Coordination*, est devenue un standard d'OASIS. Les deux standards visent la coordination de contextes pour les transactions de services web dans la composition d'une application. La différence est que WS-CTX définit un contexte de gestion plus générique pour les services web.

Par rapport aux langages standards pour les transactions, si nous examinons le cas de WSCI, nous constatons qu'il supporte des transactions et des manipulations d'exceptions. Avec WSCI, un concepteur de processus peut décrire des contextes transactionnels spécifiques. A un contexte est associé un ensemble d'activités qui sont exécutées selon la sémantique du tout ou rien. La transaction peut contenir un ensemble d'activités de compensation et défaire s'il y a lieu la transaction après qu'elle se soit terminée. Les transactions sont soit atomiques, soit emboîtées.

WSCI ne permet pas de limiter la portée de la transaction (la même remarque s'applique à son successeur WS-CDL). Selon ce standard, les interactions possibles se déroulent entre des partenaires connus à l'avance.

WS-Coordination et *WS-Transaction* sont des protocoles qui ne peuvent pas agir de manière autonome : ils constituent des briques de base à intégrer dans le cadre d'un langage de composition pour faciliter la coordination des composants et guider la transaction entre-eux. La seule exigence est que tous les services rentrant dans la transaction doivent supporter ces deux protocoles, ce qui est très restrictif. Nous pensons en effet qu'il faut éviter de placer des contraintes trop fortes sur les services participants.

[11] Organization for the Advancement of Structured Information Standards, http ://www.oasis-open.org

[12] Le rôle joué par le W3C pour la définition et le maintien du même web pour tous, est sans doute fondamental.

Pour conclure, malgré la quantité des approches existantes dans ce domaine, où nous venons de référencer les plus connus, parmi cet ensemble de standards, les spécifications *WS-Coordination* et *WS-Transaction* sont devenues très populaires. Les développements les concernant sont menés de manière très étroite. Pour illustrer l'intérêt de séparer les notions liées à la coordination de celles liées à la gestion des transactions, Steven VanRoekel[13] de Microsoft effectue une comparaison avec la construction automobile : pour fabriquer un moteur, il faut commander des pièces auprès de divers fournisseurs. C'est là que *WS-Coordination* intervient, sa tâche est de s'assurer que les commandes sont bien parvenues aux fournisseurs. Chaque fournisseur doit envoyer un message pour confirmer qu'il en mesure d'honorer la commande qui le concerne. Si l'un de ces fournisseurs n'est pas en mesure de confirmer sa participation la totalité de la commande sera annulée. C'est ce rôle que joue *WS-Transaction*.

Le langage BPEL est sans doute le langage le plus utilisé pour la programmation de compositions de services web, il s'est imposé en face aux autres standards comme BPML et WS-CDL. BPEL fait en effet l'objet d'un grand nombre de mise en œuvre ; c'est la référence obligée pour la composition de services. Il lui manque cependant certaines caractéristiques nécessaires dans la perspective de la découverte et de la sélection de services au moment de l'exécution. En effet, ces fonctionnalités, avec la possibilité de paramétrer la composition, sont très utiles pour obtenir des compositions aussi adaptables que possible, d'une part aux besoins de l'application tels qu'ils sont au moment de l'exécution, et d'autre part aux caractéristiques des services effectivement utilisés à ce moment là. En outre, BPEL ne fournit ni les mécanismes de coordination ni les mécanismes de transaction. Il s'appuie pour cela sur *WS-Coordination* et *WS-Transaction*.

3.6.2 Approches spécifiques pour la composition

La section 3.4 a présenté des approches issues de la recherche aussi bien industrielle qu'académique. Ici aussi, force est de constater que certaines traitent les aspects liés à la composition de services web en laissant à côté ceux liés à la gestion des transactions et *vice versa*.

Le tableau 3.2 récapitule les approches dédiées à la composition de services web. Pour chaque approche donnée en ligne, le tableau indique par une croix (X) si elle remplit les critères, donnés en colonne, que nous avons mis en avant dans la section 3.5. De manière similaire, le tableau 3.3 montre les approches qui traitent la composition des services avec des propriétés transactionnelles.

3.6.3 Contributions de `Tcows`

Le canevas `Tcows`[14] issu du travail présenté dans ce document, a été conçu pour répondre aux six critères auxquels l'analyse de la littérature a conduit et selon lesquels nous avons établi la comparaison des approches détaillées dans ce chapitre. Nous introduisons ci-dessous la manière dont `Tcows` répond à ces critères :

[13] http ://www.microsoft.com/presspass/features/2005/sep05/09-06Infrastructure.mspx
[14] Transactional Composition of Web Services

Approche	1	2	3	4
eFlow	X			
SELF-SERV	X		X	X
[**Loe03**]	X			
[**OYP03**]	X			
[**BBCT04**]	X			
[**SD04**]	X			
[**DDR05**]	X			
Let's Dance	X			
QBroker	X		X	X

1 = Permet la séparation des préoccupations
2 = N'impose pas de négociation préalable
3 = Prend en compte les aspects non fonctionnels
4 = Permet la sélection des services à l'exécution

Tab. 3.2 – Synthèse des approches spécifiques que pour la composition

1. Permet la séparation des préoccupations.

   La logique du processus métier est conçue indépendamment de la spécification de ses propriétés transactionnelles. La portée de la transaction dans la composition est définie comme un paramètre de la composition.

2. N'impose pas de négociation préalable.

   La composition est conçue à partir de l'identification de capacités de services faisant référence à des communautés qui rassemblent des services ayant les mêmes propriétés fonctionnelles et se distinguant par leurs propriétés non fonctionnelles. La logique du processus est exprimée en terme des interfaces dites abstraites de ces communautés qui permettent ainsi de faire abstraction des participants. L'exécution du processus ne nécessite pas de négociation préalable car elle est conduite en fonction des propriétés transactionnelles des services sélectionnés.

3. Prend en compte les aspects non fonctionnels.

   La composition est munie de paramètres permettant de fixer des propriétés qu'elle doit vérifier au moment de son exécution. Ces propriétés portent sur la qualité de service souhaitée (par exemple, la réputation minimale attendue du service choisi pour le vol) et sur des variables du contexte de l'application (par exemple, le coût global du voyage ou la température moyenne au lieu de destination au moment du voyage).

4. Permet la sélection des services à l'exécution.

   La notion de communautés sur laquelle s'appuie `Tcows` fournit un cadre à cette sélection. La sélection est guidée par les paramètres de la composition.

5. Prend en compte les propriétés transactionnelles hétérogènes des services.

   La logique du processus est décrite par le concepteur sous forme d'une orchestration conçue en termes des interfaces abstraites exposées par les communautés et

Approche	**1**	**2**	**3**	**4**	**5**	**6**
CLF	X		X		X	
[**HA00**]	X				X	
WSTx	X		X		X	
WebTP	X					
[**VV03**]	X					
[**vdHA02**]	X					
[**LZ04**]	X					
[**SD04**]	X				X	X
[**Bhi05**]	X				X	

1 = Permet la séparation des préoccupations
2 = N'impose pas de négociation préalable
3 = Prend en compte les aspects non fonctionnels
4 = Permet la sélection des services à l'exécution
5 = Prend en compte les propriétés transactionnelles hétérogènes des services
6 = Permet la définition de la portée de la transaction

Tab. 3.3 – Synthèse des approches spécifiques pour la composition de services avec des propriétés transactionnelles

paramétrée, entre autre, par la propriété transactionnelle attendue. Cette orchestration est réécrite en une autre orchestration exprimée en termes des interfaces des services effectivement sélectionnés au moment de l'exécution. Cette réécriture est effectuée par `Tcows` sur la base, d'une part des propriétés transactionnelles des services sélectionnés et de celle de l'orchestration initiale, et d'autre part d'annotations données par le concepteur et qui guident le processus de réécriture.

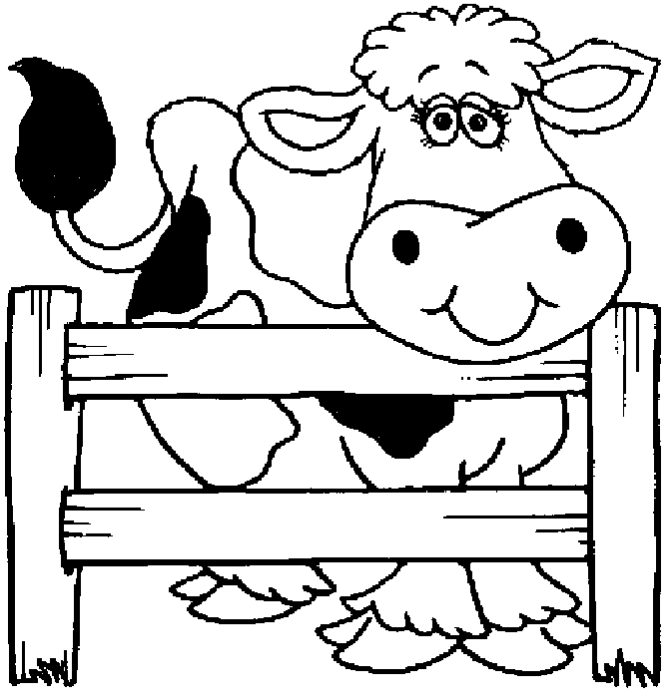
6. Permet la définition de la portée de la transaction

   La notion de zone transactionnelle est introduite. Elle est considérée ici comme l'un des paramètres de la composition. Déterminée sur la base de la spécification *obligatoire* ou *optionnelle* de chaque capacité identifiée au début du processus de conception de l'application, elle englobe les opérations à considérer dans la transaction.

Deuxième partie

Tcows : Transactional Compositions Of Web Services

Canevas pour la composition transactionnelle de services web



CHAPITRE 4

SÉLECTION DE SERVICES

Il arrive très fréquemment que de nombreux services web répondent à un même ensemble de besoins fonctionnels : pour réaliser la réservation d'un vol, un client peut mettre en concurrence plusieurs compagnies aériennes. Ces services se distinguent les uns des autres par leurs propriétés extra fonctionnelles. Nous nous intéressons ici à les sélectionner selon ses propriétés en nous concentrant sur les critères de qualités (réputation, fiabilité, etc.).

Notre objectif est d'étudier la notion de *communauté* faisant référence à un ensemble de services de même fonctionnalité. Une communauté[1] offre à ses utilisateurs (développeurs d'application, fournisseurs de services, applications clientes) des fonctions pour connaître des informations liées à la communauté, pour sélectionner, puis utiliser un ou plusieurs des services qui y sont enregistrés. Ainsi, ce chapitre présente un modèle de communauté formalisé par le biais d'un ensemble de types abstraits, dont les opérations associées permettent en particulier d'effectuer, au sein d'une communauté, la sélection du service (ou de plusieurs) répondant le mieux à un ensemble de critères de qualité [FDTB07]. Nous prenons l'hypothèse que ce sont les fournisseurs, intéressés à offrir leurs services, qui sont à l'origine des différentes communautés de services. Nous supposons aussi que les communautés peuvent être découvertes via un moteur de recherche quelconque en fournissant des mots clés correspondant aux fonctionnalités recherchées.

Le travail rapporté dans ce chapitre s'intègre dans le cadre d'une collaboration menée dans le contexte du projet Web Intelligence[2] (d'autres travaux s'y rapportent, voir par exemple [FAB06, TBFM06, ABF07, BMT+07]).

La suite de chapitre est organisée comme suit : la section 4.1 présente d'une part les raisons pour lesquelles nous introduisons la notion de communautés et d'autre part situe notre approche par rapport à celles proposées dans la littérature. Le modèle associé est décrit et formalisé dans la section 4.2. Dans la section 4.3, nous illustrons l'application de notre proposition au scénario "Agence de Voyages". Enfin, la section 4.4 conclut le chapitre.

[1] Dans la suite, le mot "communauté" est utilisé pour "communauté de services web"

[2] Cluster Informatique Systèmes Logiciels Embarqués - Région Rhône-Alpes, 2007 - 2010.

4.1 Motivations et travaux reliés

La mise en œuvre des interactions entre des services web nécessite la découverte, la localisation et l'accès à ces services. Les acteurs participant à ces interactions sont : les applications clientes (qui peuvent être des services) qui cherchent à interagir avec d'autres services et les fournisseurs qui exposent les services qu'ils offrent afin de les rendre disponibles aux clients.

Selon l'approche UDDI, les fournisseurs de services publient les interfaces des services qu'ils offrent (le plus souvent selon une description WSDL) dans un annuaire. Un client soumet ensuite à cet annuaire des requêtes de recherche de services. L'annuaire répond en envoyant l'adresse (une URL) du service correspondant, s'il en existe. Le client utilise ensuite cette URL pour l'envoi des messages traduisant l'exécution des opérations offertes par le service.

Un annuaire UDDI contient les informations sur les entreprises et les services qu'elles ont développés et publiés. Il est structuré de la manière suivante :

- Les pages blanches recensent des informations sur l'identité des fournisseurs.
- Les pages jaunes comprennent les descriptions au format WSDL des services web déployés par les fournisseurs.
- Les pages vertes qui fournissent des informations techniques détaillées sur les services fournis.

Les requêtes qu'il est possible de soumettre auprès d'un annuaire UDDI ont une portée limitée aux aspects fonctionnels des services recherchés. Relativement au scénario introduit au chapitre 1, il est possible de chercher des services de capacité Vol par exemple, mais il est par contre impossible de retrouver ceux les *plus fiables*.

Les services référencés dans un annuaire exposent leurs propres interfaces, indépendamment les uns des autres. Aucune uniformité n'est offerte. Ainsi, lorsqu'il arrive qu'un client dont l'interface requise a été implantée en fonction de celle fournie par un service S1 sélectionné à un moment donné, doive s'appuyer sur un autre service S2 issu du même annuaire et répondant à la même requête, il y a toutes les chances pour l'interface fournie par S2 ne corresponde pas à celle fournie par S1. Le client doit en conséquence modifier son interface afin de la rendre conforme à celle de S2.

De plus, étant donnée l'existence de moteurs de recherche sophistiqués aussi bien pour des intranets qu'au niveau de l'internet tout entier, et d'autres technologies pour la gestion de répertoires tels que LDAP [HKY95], la valeur ajoutée apportée par UDDI est difficile à identifier. Peu d'architectures à base de services s'appuient aujourd'hui sur UDDI et le futur de ce standard est incertain[3].

De nombreux travaux ont été menés dans la perspective de pallier les insuffisances d'UDDI. Par exemple, dans [ZM05] les auteurs s'intéressent à la sélection dynamique d'un service afin d'adapter la composition à laquelle il participe au profil de l'utilisateur qui en a demandé l'exécution. Ce profil décrit en particulier le dispositif physique sur lequel la composition s'exécute. Contrairement au modèle de communautés que nous présentons plus loin dans ce chapitre, l'adaptation de l'interface fournie par le service est à la charge de l'utilisateur final qui a demandé la sélection.

[3] *UDDI Business Registry* tenu par IBM, Microsoft et SAP a été fermé le 12 janvier 2006.

D'autres approches décrites ci-dessous se distinguent par le fait qu'elles s'appuient sur la notion de communauté ou sur des ontologies.

Les communautés. La notion de communauté n'est pas propre au contexte des services web. Dans le domaine des grilles, par exemple, des solutions pour le partage de ressources sur une grille s'appuient sur des communautés [CFK05, FFK+06]. A chaque ressource est associé un modèle de qualité contrôlé par le fournisseur de la ressource. Ce principe s'oppose à notre approche où le modèle de qualité est défini par la communauté (les services peuvent le satisfaire, tout ou en partie) et où l'évaluation de la valeur d'un critère peut être effectuée de trois manières (par le biais d'une opération exposée par le service concerné, par la communauté sur la base d'une valeur stockée fournie par le service, ou encore calculée dynamiquement par la communauté à partir de traces collectées ; voir la fonction Value spécifiée plus loin dans section 4.2 de ce chapitre). Les services sont liés à la communauté par des contrats qui spécifient le niveau de qualité qu'ils s'engagent à offrir. Cette notion est proche de celle modélisée par la fonction Value lorsque son évaluation fournie une valeur donnée par le service et stockée par la communauté. La solution que nous avons choisie pour l'évaluation de la satisfaction d'un service pour un critère est moins restrictive.

WS-catalogNet est un canevas pour l'organisation et l'intégration d'un grand nombre de catalogues [PBT04]. Ici, les catalogues sont rassemblés en un container, appelé aussi communauté, associé à un domaine spécifique et qui expose une catégorie et une description des produits (via des attributs) référencés dans les catalogues. Les fournisseurs, pour rendre disponibles leurs produits, doivent enregistrer leurs catalogues auprès d'une communauté. *WS-catalogNet* est un service web qui permet la création des communautés, l'adhésion d'un catalogue auprès d'une communauté et la définition des relations entre ces catalogues. Le but est de fournir un modèle pour interroger (en ligne) la communauté au travers des relations entre les différents catalogues de la communauté. Ce canevas permet d'implanter des portails web destinés à des utilisateurs finaux, mais ne permet pas, comme dans l'approche que nous proposons ici, de mettre en relation une application cliente et les services qui correspondent à ses besoins.

Dans [LL05] une communauté de services web est un moyen offert pour partager, utiliser et gérer des ressources en s'appuyant sur les services qui sont fournis par la communauté. L'objectif des fournisseurs est d'offrir un cadre pour la qualité de service, conformément aux demandes des clients. Une communauté de services est un ensemble d'utilisateurs et de fournisseurs qui interagissent et collaborent les uns avec les autres dans le but de fournir des services et d'échanger des ressources formant ainsi des réseaux. Contrairement à notre solution, une communauté n'est pas associée à un domaine spécifique de services, mais vise plutôt à fournir un cadre pour l'utilisation des applications choisies en fonction de la qualité offerte.

Notre approche est inspirée, en partie, du concept de communauté de services introduit dans [BDS05] selon lequel un service web peut rejoindre la communauté, même s'il ne répond pas à l'ensemble des besoins couverts par la communauté. Lorsqu'une application cliente désire utiliser un service, celui-ci est sélectionné puis son exécution est déléguée par la communauté auprès du fournisseur de service. *A contrario*, nous proposons ici de retourner l'adresse du service (ou de plusieurs services) au client et c'est à lui de contacter

le service directement auprès du fournisseur. En outre, dans [BDS05], les valeurs associées au modèle de qualité de chaque service sont fixées de manière statique. Il n'est pas possible, comme dans notre solution, d'en calculer la valeur au moment de la sélection.

Les ontologies. Dans cette classe de solutions, les services sont munis chacun d'une description conceptuelle appelée ontologie. Dans [PKPS02] ces descriptions sont exploitées par un algorithme de mise en correspondance, selon un certain degré d'incertitude, entre la requête qui décrit le service requis et les ontologies des services offerts. Il s'agit de sélectionner des services web qui répondent le mieux à une requête formulée en termes des signatures des opérations, mais qui ne prend pas en compte les propriétés extra fonctionnelles des services, comme nous le proposons ici.

Une autre approche à base d'ontologies est celle de [BHRT03] où la solution s'appuie sur le langage *DAML-S* (DARPA Agent Markup Language for Web Services[4]). Les auteurs proposent un algorithme de mise en correspondance entre une requête et une ontologie DAML-S pour trouver la combinaison de services Web qui satisfait au mieux les sorties de la requête et qui exige le moins possible d'entrées qui ne sont pas fournies dans la description de la requête. Comme dans [PKPS02], les propriétés extra fonctionnelles des services ne sont pas prises en compte.

Dans [MM03], les auteurs mettent en avant la spécification conceptuelle, sous forme d'une ontologie, du processus métier à implanter afin d'effectuer automatiquement (1) la découverte des services Web qui peuvent participer au processus métier, et (2) la réalisation des interactions entre les partenaires du processus. Les services eux aussi sont décrits par des ontologies. Un service participe à un processus métier lorsque sa description correspond à une partie de l'interaction. La sélection de services est ici opérée à la conception et non pas dynamiquement à l'exécution comme c'est le cas dans notre approche.

Enfin, dans [GdT06], les auteurs visent à permettre la sélection d'un service sur des critères fonctionnels aussi bien que non fonctionnels. L'approche s'appuie d'une part, sur une extension de *WS-policy* par une ontologie *OWL* (*Web Ontology Language*[5]) pour la modélisation des modèles de qualité des services, et d'autre part sur une extension d'UDDI pour la publication et la découverte de services. Cependant, les auteurs ne discutent pas du moment de la sélection (à la conception vs. à l'exécution), ni de la mise en correspondance de l'interface fournie par le service sélectionné avec celle requise par le client.

4.2 Modèle de communautés

Nous proposons maintenant un modèle de communautés qui satisfait les besoins énoncés ci-dessus et tente de pallier les défauts des approches étudiées. Nous commençons, dans la section 4.2.1, par donner une description informelle de ce modèle que nous formalisons ensuite dans les sections 4.2.2 et 4.2.3. Une illustration est ensuite donnée dans la section 4.3 sous la forme d'une application au scénario "Agence de Voyages".

[4]http ://www.daml.org
[5]http ://www.w3c.org/2004/OWL/

4.2.1 Description informelle

Le principal objectif d'une communauté de services web est d'offrir un cadre à la recherche et à la sélection dynamique de services web et ainsi de remédier aux déficiences des annuaires UDDI [BDS05, LL05]. C'est aussi un moyen pour supporter la composition dynamique de services [FDDB05] ou encore de permettre la substitution (à la conception ou à l'exécution) de services [TBFM06]. Contrairement à un annuaire UDDI, une communauté est spécialisée dans un domaine spécifique. Les services d'une communauté diffèrent entre-eux sur des propriétés extra fonctionnelles comme par exemple, des critères de qualité (disponibilité, prix d'exécution, fiabilité, sécurité, tolérance aux fautes, réputation, propriétés transactionnelles, etc.) [BSD03]. Ces critères de qualité sont définis via un *modèle de qualité* spécifique à une communauté donnée.

La figure 4.1 présente un diagramme de classes UML décrivant les éléments principaux d'une communauté et les services associés [Mul97]. Dans ce diagramme, toutes les associations sont de cardinalité 1-1, sauf spécification contraire. La description des opérations associées aux fonctionnalités offertes par la communauté est exposée dans ce que nous appelons l'Interface Abstraite de la communauté. Une communauté possède de plus, deux interfaces dites concrètes, car elles sont implantées par la communauté, contrairement à son interface abstraite. L'une est à destination des clients, l'autre à destination des fournisseurs de services. La première (Interface Client), pour les clients, décrit des opérations qui sont utilisées à la conception de l'application afin d'obtenir de l'information sur la communauté (par exemple, le modèle de qualité de la communauté, ou bien la description WSDL de son interface abstraite). Pour faciliter la lecture du diagramme nous avons fait apparaître deux classes pour modéliser les descriptions WSDL des interfaces, mais il s'agit bien du même concept. D'autres opérations, utilisées à l'exécution permettent de sélectionner des services et de réaliser les appels aux opérations exposées par l'interface fournie par les services sélectionnés. La seconde interface (Interface Fournisseur), à destination des fournisseurs de service, leur permet d'enregistrer un service, ou encore d'obtenir des informations sur la communauté. Dans la figure 4.1 nous avons fait abstraction de la description détaillée de toutes ces opérations.

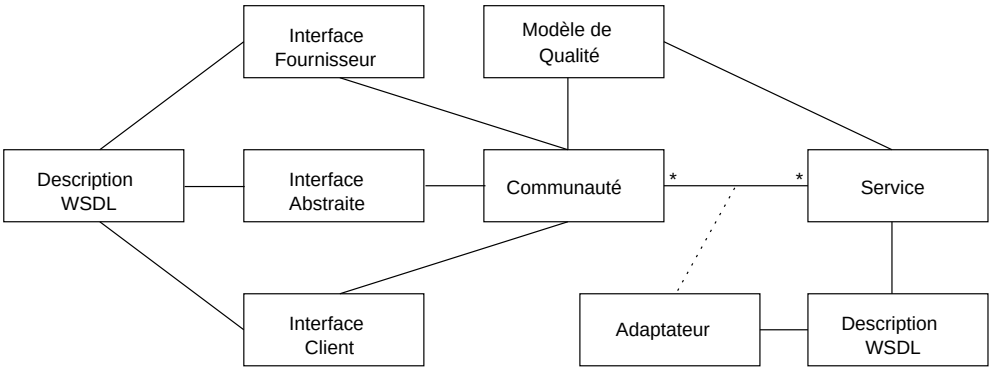


FIG. 4.1 – Éléments principaux des communautés

Les services participant à une communauté ne fournissent pas nécessairement une interface identique à celle de l'interface abstraite de la communauté. Le cas échéant, chaque service doit être muni d'un adaptateur permettant d'établir la mise en correspondance entre les

opérations de l'interface abstraite de la communauté avec celles de l'interface concrète que le service fournit. L'étude de l'adaptation d'interface sort du champ couvert par la thèse décrite ici ; cette étude est généralement menée selon le point de vue structurel des interfaces (voir par exemple [TBFM06, BMT+07]) ou selon le point de vue comportemental (voir par exemple [FAB06, ABF07]). Nous prenons pour hypothèse que cet adaptateur est fourni, pour chaque service de la communauté, par son fournisseur.

Un client désirant rechercher un service parmi ceux référencés par une communauté, soumet une requête de sélection munie d'un certain nombre de paramètres (voir plus loin, section 4.2.3). Le client obtient en retour les URLs des services sélectionnés ainsi que les URLs de leurs adaptateurs respectifs.

4.2.2 Communautés et modèles de qualité

Nous détaillons et formalisons le modèle de communautés introduit plus haut. Nous avons choisi d'utiliser un style de formalisation fonctionnel car il fournit un cadre convenable pour décrire, de manière concise, les types associés à la notion de communauté et leurs opérations en faisant abstraction des aspects syntaxiques.

Les notations ci-après sont utilisées : type T est une expression qui introduit un type de nom T ; T1 $\longrightarrow$ T2 décrit le type des fonctions dont le domaine est T1 et le codomaine T2. {T} dénote le type ensemble de T, [T] dénote le type liste de T, et <L1 : T1, L2 : T2, . . . , Ln : Tn> désigne le type des nuplets dont le i^{e} composant est de type Ti ($1 \leq i \leq n$). Si t est un nuplet de type <L1 : T1, L2 : T2, . . . , Ln : Tn>, alors la notation qualifiée t.Li ($1 \leq i \leq n$) désigne la valeur du composant Li de t.

Les types Service, Client et Fournisseur sont introduits ci-dessous. Nous faisons ici abstraction de leurs détails qui seront donnés plus loin (voir chapitre 6, section 6.2).

type Service
{ *Le type Service permet de modéliser une offre d'un ensemble d'opérations (par exemple, achat de billets d'avion). Une instance du type Service correspond à une entité logicielle à laquelle un client peut accéder au travers des opérations qu'elle propose.* }
type Client
{ *Les instances du type Client représentent les clients (ou des applications ou des services) qui utilisent les opérations offertes par les services et qui sont identifiés.* }
type Provider
{ *Le type Provider modélise une organisation ou une entreprise qui fournit et expose des services.* }
type Community
{ *Le type Community modélise la notion de communauté comme indiqué dans la figure 4.1.* }
Name : Community $\longrightarrow$ string
{ *Name (co) est le nom donné à la communauté co (par exemple, 'Vol').* }

Le type Criterion décrit la notion de critère de qualité, il est paramétré par le type des valeurs du critère :

type Criterion $\langle$ T $\rangle$

Les sélecteurs associés au type Criterion sont :

Name : Criterion $\langle$ T $\rangle$ $\longrightarrow$ string
{ *Name (c) est le nom donné au critère c (par exemple, "temps de réponse").* }
OrderingMode : Criterion $\langle$ T $\rangle$ $\longrightarrow$ {ascendant | descendant}
{ *OrderingMode (c) permet de fixer la comparaison d'un ensemble de services selon les valeurs associées au critère de qualité c donné. Par exemple, le meilleur service selon le critère "temps de réponse" est le premier dans l'ordre croissant des valeurs, si c est ce critère, alors OrderingMode (c) = ascendant.* }
Nature : Criterion $\langle$ T $\rangle$ $\longrightarrow$ $\langle$string, string$\rangle$
{ *Soit $\langle s, u\rangle$ = Nature (c) : s donne la sémantique des valeurs du critère (par exemple, "durée") et u donne l'unité dans laquelle la valeur doit être fournie (par exemple, "seconde"). Modéliser cette information permettra dans le futur d'effectuer des comparaisons et des conversions de valeurs exprimées dans des unités diverses pour des critères de même nature.* }

Chaque communauté est associée à un modèle de qualité formalisé par la fonction QualityModel qui appliquée à une communauté retourne l'ensemble des critères définis dans son modèle de qualité :

QualityModel : Community $\longrightarrow$ {Criterion $\langle$ T_1 $\rangle$, Criterion $\langle$ T_2 $\rangle$,... Criterion $\langle$ T_n $\rangle$}
{ *QualityModel(co) est l'ensemble de critères de qualité associés à la communauté co.* }

Les services qui ont été enregistrés dans une communauté sont modélisés par le type RegisteredService :

type RegisteredService
{ *Le type RegisteredService est une représentation que les communautés ont des services qui sont enregistrés.* }

Les sélecteurs qui en découlent sont :

RegisteredServices : Community $\longrightarrow$ {RegisteredService}
{ *RegisteredServices (co) est l'ensemble des représentations des services enregistrés dans co.* }
ServiceURL, AdapterURL : Community, RegisteredService $\longrightarrow$ URL
{ *ServiceURL (co, rs) (respectivement AdapterURL (co, rs)) est l'URL du service rs (respectivement de l'adaptateur du service rs) enregistré dans la communauté co.*
Pré condition : rs $\in$ RegisteredServices (co). }
QoS : Community, RegisteredService $\longrightarrow$ {Criterion $\langle$ T_1 $\rangle$, Criterion $\langle$ T_2 $\rangle$,... Criterion $\langle$ T_n $\rangle$}
{ *QoS (co, rs) est l'ensemble de critères de qualité associés au service rs enregistré dans co.*
Pré condition : rs $\in$ RegisteredServices (co). }

La communauté dispose d'une opération appelée Value. Cette opération permet de calculer la valeur d'un critère de qualité pour un service donné ;

Value : Community, Criterion $\langle$ T $\rangle$, RegisteredService $\longrightarrow$ T

Cette fonction admet trois implantations différentes qui traduisent trois modes d'évaluation :

– Les valeurs sont gérées par la communauté. Les valeurs des critères sont calculées par la communauté sur la base de traces. La conservation de traces d'exécution par une communautés implique une communication entre la communauté et les clients des services dont les exécutions sont observées.

– Les valeurs sont gérées statiquement par les fournisseurs de services. Chaque fournisseur détermine à l'enregistrement la valeur de chaque critère pour chaque service qu'il a enregistré. Si ces valeurs changent, le fournisseur doit mettre à jour les données correspondantes.
– Les valeurs sont gérées dynamiquement. Ce mode d'évaluation nécessite une opération spécifique offerte par les services de telle sorte qu'au moment de l'évaluation de la fonction Value pour un service donné, la communauté appelle cette opération exposée par le service et détermine ainsi la valeur du critère.

Dans un premier temps, nous ne considérons que le deuxième mode d'évaluation. La constitution de traces n'est pas discutée ici. Cette étude fait partie des perspectives faisant suite au travail décrit dans ce texte.

Les opérations ci-dessous sont destinées aux fournisseurs de services :

AbstractInterface : Community $\longrightarrow$ WSDL-description
{ *AbstractInterface (co) est la description WSDL de l'interface abstraite de la communauté co.* }
Register : Community, Service, URL, URL, string $\longrightarrow$ boolean
{ *Register (co, s, urlS, urlA, pro) est vrai* $\Longleftrightarrow$ *l'inscription auprès de la communauté co du service s d'URL urlS a réussi. Ce service est muni de l'adaptateur accessible à l'URL urlA. Cet enregistrement est fait par le fournisseur pro et rend accessible le service s via la communauté co, c'est-à-dire :*
$\exists$ *rs* $\in$ *RegisteredServices (co)* | *urlS* = *ServiceURL (co, rs) et urlA* = *Adapter (co, rs).*
Il n'y a pas de contrainte sur le modèle de qualité de s. }
Quit : Community, RegisteredService, string $\longrightarrow$ boolean
{ *Quit(co, rs, pro) est vrai* $\Longleftrightarrow$ *le service rs, fournie par pro, n'est plus inscrit à la communauté co (c'est-à-dire rs* $\notin$ *RegisteredServices (co)).*
Pré condition : rs $\in$ *RegisteredServices (co)* }

4.2.3 Sélection de services

Nous introduisons ci-dessous la fonction Select, à destination des clients, qui leur permet de sélectionner des services selon un ensemble de critères de qualité. L'ensemble des critères fournis à la sélection est un sous-ensemble des critères du modèle de qualité de la communauté. Les services qui ne répondent pas à tous les critères fixés dans l'expression de sélection ne sont pas considérés. Nous ramenons le choix multicritère à un choix monocritère en associant à chaque critère un poids [BC06]. Finalement, une fonction booléenne est donnée en paramètre et est associée à chaque critère. Cette dernière permet d'exprimer une restriction sur les services sélectionnés (par exemple, le temps de réponse doit être inférieur à 10 millisecondes). Des exemples concrets sont montrés dans la sous-section 4.3. Le type Restriction décrit des fonctions à valeurs booléennes, et le type Weight celui des valeurs réelles définies dans l'intervalle [0..1]. Ces types sont utilisés plus loin dans la spécification de la fonction de sélection de services dans une communauté :

type Restriction : (type $\longrightarrow$ boolean)
type Weight : real in [0..1]
RequiredQuality : $\langle$ c : Criterion$\langle$T$\rangle$, p : Weight, r : Restriction $\rangle$
{ *Soit nq donné dans RequiredQuality. nq décrit un niveau de qualité relativement au critère nq.c. La restriction nq.r exprime une contrainte requise sur ce critère et nq.p est le poids donné à ce critère.* }

La fonction de sélection est maintenant spécifiée comme suit :

Select : Community, integer, {RequiredQuality} $\longrightarrow$ [RegisteredService]

Soit L = Select (co, nb, {nq_1, ...,nq_n})

L est la liste des nb services accessibles via la communauté co, qui maximisent l'ensemble des niveaux de qualité (nq_i, i = 1..n). Chaque nq_i, est pondéré par le poids nq_i.p (nous imposons sans perte de généralité que $\sum_{i=1}^{n} nq_i.p = 1$). Chaque service sélectionné vérifie la restriction nq_i.r associée à chaque nq_i (i = 1..n). L est ordonnée en ordre décroissant des satisfactions des services à l'ensemble des niveaux de qualité nq_i (i = 1..n). La pondération est utilisée pour ramener le choix multicritère à un choix monocritère. La pré condition ci-après s'applique : $\forall$ i = 1..n, nq_i.c $\in$ QualityModel (co).

4.3 Illustration

Cette partie présente l'application du modèle présenté ci-dessus au scénario "Agence de Voyages". La section 4.3.1 montre les mécanismes d'adaptation mis en œuvre par les fournisseurs de services afin de réaliser la mise en correspondance entre les envois de messages réalisés en termes de l'interface abstraite de la communautés et ceux attendus par les services. La section 4.3.2 illustre par quelques exemples l'utilisation de la fonction de sélection.

4.3.1 Interface abstraite vs. interface des services

L'interface abstraite de la communauté Vol décrit en particulier les deux opérations demandeItinéraires et réservationItinéraire spécifiées ci-après. Ces opérations correspondent respectivement aux messages détailsVoyage et réservationItinéraire que les services de la communauté Vol peuvent recevoir (voir l'interface sur service Vol utilisée figure 2.6, page 23). Nous faisons abstraction ici des types Itinéraire et Billet utilisés.

demandeItinéraires : date, date, string, string $\longrightarrow$ {Itinéraire}
{ *demandeItinéraires (dDép, dRet, vDép, vDes) retourne une liste d'itinéraires possibles ayant dDép comme date pour le départ et dRet comme date pour le retour, vDép comme ville de départ et vDes comme ville de destination.* }

réservationItinéraire : Itinéraire, string $\longrightarrow$ Billet
{ *réservationItinéraire (i, c) retourne le billet d'avion correspondant à l'itinéraire i, pour le client c.* }

La figure 4.2 montre un diagramme d'activités décrivant les messages échangés entre un client et un service selon l'interface abstraite définie par la communauté Vol. Dans cette figure nous ne mentionnons que les opérations de communication.

Le service Aviatour.com est enregistré dans la communauté Vol, mais l'interface qu'il fournit est différente de l'interface abstraite de Vol. La figure 4.3 décrit les interactions entre un client dont l'interface requise a été implantée selon l'interface abstraite de la communauté Vol et le service Aviatour.com via l'adaptateur que le fournisseur du service a dû mettre à la disposition des clients.

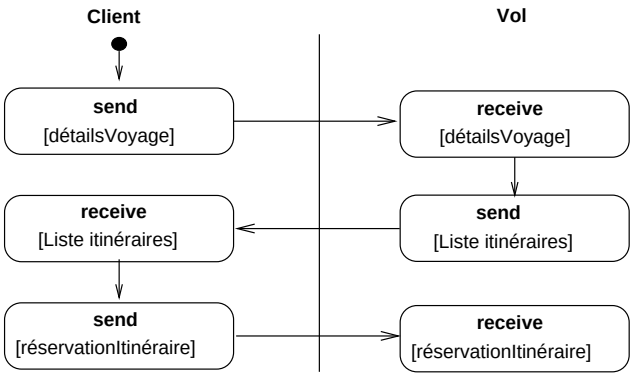


Fig. 4.2 – Modèle d'interactions entre un client et un service, selon l'interface abstraite de la communauté Vol

La figure décrit une situation où l'interface fournie par le service Aviatour.com attend deux messages : la premier contient la donnée de la ville de départ et de la destination et un second qui fournit les dates du voyage. Le service Aviatour.com retourne, en réponse au premier, un message indiquant s'il existe des vols entre les villes indiquées, et en réponse au second, une liste éventuellement vide, des itinéraires satisfaisant à la demande. La figure ne montre pas, ici encore, les opérations internes des services considérés (comme par exemple, l'opération de recherche de connexions entre deux villes, ou encore la construction d'une liste vide d'itinéraires).

Cette figure ne détaille pas le processus d'adaptation qui peut se révéler complexe à mettre en œuvre. Lorsque l'adaptation requise porte uniquement sur la structure des messages échangés, des techniques issues des travaux effectués sur le thème de l'intégration de données hétérogènes sont utilisées (voir exemple [TBFM06]). Lorsque l'adaptation porte aussi sur les aspects comportementaux des interfaces, alors d'autres méthodes de réconciliation de conversations doivent être utilisées aussi (voir par exemple [FAB06, ABF07]). Nous rappelons que la proposition d'une solution pour l'adaptation d'interfaces sort du champ de l'étude rapportée dans ce texte.

4.3.2 Sélection de services et modèle de qualité

La communauté Vol est associée à un modèle de qualité dont les critères sont le temps de réponse, la fiabilité et la réputation. L'application du modèle formalisé dans la section 4.2.2 à la définition de la communauté Vol et de son modèle de qualité, est donnée ci-dessous. L'expression Name (c1) = 'Réputation' signifie que la valeur retournée par le sélecteur Name, appliqué au critère c1 est 'Réputation'.

c1 : Criterion ⟨ integer in [1.0, 20.0] ⟩
  Name (c1) = 'Réputation', OrderingMode (c1) = descendant, Nature (c1) = ⟨ nil, nil ⟩
c2 : Criterion ⟨ float ⟩
  Name (c2) = 'Temps de réponse', OrderingMode (c2) = ascendant,
  Nature (c2) = ⟨ 'durée', 'millisecondes' ⟩
c3 : Criterion ⟨ integer ⟩
  Name (c3) = 'Fiabilité', OrderingMode (c3) = descendant, Nature (c3) = ⟨ nil, nil ⟩

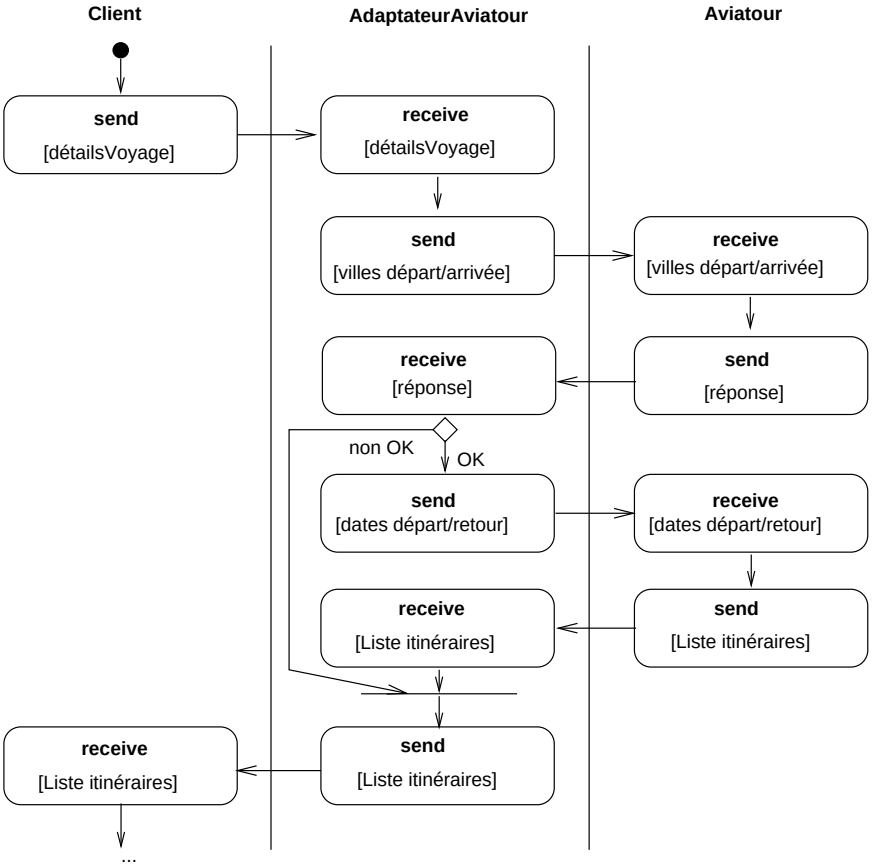


FIG. 4.3 – Une mise en correspondance entre un client selon l'interface abstraite de la communauté Vol, et Aviatour.com

La communauté co de nom Vol est définie comme suit :

co : Community
Name (co) = 'Vol', QualityModel (co) = {c1, c2, c3}, InterfaceAbstraite (co) = fichierWSDL

Des services, par exemple s1, s2, s3 et s4, appartiennent à la communauté Vol. Les descriptions ci-dessous sont limitées aux noms et aux modèles de qualité de ces services. Les modèles de qualité de s2 et s3 répondent exactement à celui de la communauté co de nom Vol, alors que ceux de s1 et s4 n'y répondent que partiellement.

s1, s2, s3, s4 : Service
Name (s1) = 'VirginBlue', Qos (s1, co) = {c2, c3}
Name (s2) = 'Qantas', Qos (s2, co) = {c1, c2, c3}
Name (s3) = 'Aviatour', Qos (s3, co) = {c1, c2, c3}
Name (s4) = 'EasyJet', Qos (s4, co) = {c2, c3}

Pour chaque critère de chacun de ces services, la valeur associée est calculée sur la base des évaluations de la fonction Value fournies dans le tableau 4.1. Nous faisons ici abstraction de son mode d'évaluation. Pour chaque critère, le tableau donne pour chaque service la valeur retournée et le score associé.

Service		Réputation (c1)		Temps de réponse (c2)		Fiabilité (c3)	
nom	id	valeur	score	valeur	score	valeur	score
VirginBlue	s1		-	0.02	4	1	1
Qantas	s2	10.5	2	0.04	3	4	4
Aviatour	s3	11.8	1	0.05	2	4	4
EasyJet	s4		-	0.08	1	3	2

TAB. 4.1 – Les valeurs des critères du modèle de qualité des services web, relativement à la communauté Vol

A titre d'exemple, les expressions suivantes montrent la façon d'utiliser l'opération Select. L'expression ci-dessous permet de sélectionner parmi les services enregistrés dans la communauté Vol, celui dont la valeur du critère Temps de réponse (d'identifiant c2) est minimale, tout en étant inférieure à 1.0. Conformément au tableau 4.1, s1 est le seul service sélectionné. Ici le poids fourni ne joue aucun rôle, puisque la sélection porte sur un seul critère. Lors de l'évaluation de la sélection les quatre services de la communautés ont été considérés car ils répondent tous les quatre au critère Temps de réponse.

$$\text{Select (Vol, 1, } \{\langle \text{ c2, 1, } \lambda x \bullet x < 1.0 \rangle\}) = [\text{s1}]$$

La sélection d'une liste des deux meilleurs services parmi ceux accessibles via Vol, selon le critère Réputation (identifié par c1) est donnée ci-après. Selon ce critère, les services sont donnés dans la liste [s2, s3]. Cette liste est fournie en ordre décroissant des valeurs du critère (OrderingMode (c1) = descendant) ; ni s1 ni s4 ne sont considérés dans la sélection car le critère Réputation n'est pas défini dans leurs modèles de qualité. En conséquence la sélection retourne seulement la liste [s2, s3].

$$\text{Select (Vol, 2, } \{\langle \text{ c1, 1, } \lambda x \bullet \text{vrai} \rangle\}) = [\text{s2, s3}]$$

L'expression donnée ci-après s'appuie sur deux critères Fiabilité (d'identifiant c3) et Temps de réponse (d'identifiant c2) :

$$\text{Select (Vol, 1, } \{\langle \text{ c2, 0.2, } \lambda x \bullet x < 1.0 \rangle\}, \langle \text{ c3, 0.8, } \lambda x \bullet \text{vrai} \rangle) = [\text{s1}]$$

Dans cette sélection, le critère Fiabilité est pondéré par 0.8, tandis que le critère Temps de réponse l'est par 0.2. Selon les valeurs données dans le tableau 4.1 pour le critère Fiabilité, le classement est [s2, s3, s4, s1], le score obtenu par chaque service est en conséquence 4 pour s2 et s3, car ils sont en tête d'une liste de quatre, 2 pour s4 car il est second après deux services *ex æquo* ; enfin, s1 obtient un score de 1. Selon le critère TempsdeRéponse le classement est [s1, s2, s3, s4], est le score pour chaque service est de 4, 3, 2 et 1 respectivement.

Associer un score à chaque service pour chaque critère permet de ramener la sélection multicritère à une sélection monocritère tenant compte de la pondération fournie. Ainsi, le service sélectionné est s2 car il obtient le meilleur score, comme indiqué ci-dessous :

$$\text{s1} = 4 * 0.2 + 1 * 0.8 = 1.6 \qquad \text{s2} = 3 * 0.2 + 4 * 0.8 = 3.8$$
$$\text{s3} = 2 * 0.2 + 4 * 0.8 = 3.4 \qquad \text{s4} = 1 * 0.2 + 2 * 0.8 = 1.8$$

Enfin, ci-dessous est illustrée une situation dans laquelle aucun service n'est sélectionné : les modèles de qualité de s1 et de s4 n'incluent pas le critère Réputation (dont l'identifiant est c1), quant à s2 et à s3 leurs valeurs pour ce critère ne vérifient par la condition $\lambda x \bullet x < 5$.

$$\mathsf{Select}\ (\mathsf{Vol},\ 1,\ \{\langle\ \mathsf{c1},\ 1,\ \lambda x \bullet x < 5\ \rangle\ \}) = [\ ]$$

4.4 Conclusion

Ce chapitre a présenté un modèle de communautés de services qui introduit un niveau d'abstraction intermédiaire entre les applications clientes et les services qui les composent. Ainsi, ces applications sont exprimées en termes d'interfaces abstraites de communautés et non plus en termes des interfaces des services spécifiques.

Plus particulièrement, le modèle proposé permet de regrouper un nombre potentiellement grand de services répondant au même ensemble de besoins fonctionnels et se distinguant par des propriétés non fonctionnelles. Ceci fournit un cadre pour la sélection d'un ou de plusieurs services portant sur leurs caractéristiques non fonctionnelles. Cette sélection peut aussi bien être effectuée au moment de la conception de l'application qu'au moment de son exécution. Le modèle de composition de services que nous présentons dans les deux chapitres qui suivent s'appuie sur cette dernière caractéristique. L'objectif est de permettre au concepteur d'associer des propriétés à une composition au moment de sa conception, puis de sélectionner, à l'exécution, les services participant afin de choisir ceux qui répondent le mieux aux besoins exprimés par les propriétés.

L'étude rapportée ici ouvre plusieurs perspectives de recherche. La sélection proposée s'appuie sur un modèle de qualité qui doit être enrichi, par exemple pour couvrir un éventail plus large de critères de qualité ou bien pour permettre de réaliser des conversions entre des valeurs d'un même critère exprimées dans différentes unités (monétaires, temporelles, etc.). Dans l'approche que nous avons proposée, si certains services offrent un modèle de qualité qui n'inclut pas tous les critères utilisés dans une sélection, ils ne sont pas considérés dans cette sélection. Cette contrainte doit être relâchée, surtout lorsqu'aucun service enregistré dans la communauté n'est muni d'un modèle de qualité correspondant à tous les critères utilisés dans la sélection. De façon plus générale, nous avons adopté une vision simplifiée du choix multicritère en nous ramenant, par l'introduction de pondération, à un choix monocritère. Une perspective consiste à intégrer des résultats obtenus dans ce domaine (voir par exemple [BC06, YZL07]). Enfin, un outil doit être mis à la disposition des administrateurs de communautés afin qu'ils puissent modifier le modèle de qualité de la communauté, répondant ainsi aux évolutions des besoins des clients d'une part, et des concepteurs de services d'autre part.

Par ailleurs, nous avons pris pour hypothèse que l'adaptation entre l'interface abstraite exposée par la communauté et celles fournies par les services de la communauté était une tâche à la charge des fournisseurs de services. Une interface peut être décrite selon la dimension structurelle, comportementale ou encore extra fonctionnelles des services. Ainsi l'adaptation doit être étudiée selon chacune de ces dimensions (voir par exemple [ABF07] pour l'adaptation comportementale). Un travail en cours vise à proposer un outil dans la perspective d'automatiser ou, au moins, d'assister le fournisseur lors de la définition des règles de mise en correspondance entre deux interfaces selon leur dimension structurelle.

A terme, ce jeu de règles sera utilisé par un composant appelé OSC *Open Service Connectivity* [BMT+07], pour la réécriture, en termes de l'interface fournie du service sélectionné, des appels exprimés dans le code du client en terme de l'interface abstraite de la communauté. Ce composant peut être vu comme une bibliothèque de fonctions permettant aux applications clientes d'exécuter les opérations offertes par les communautés.

CHAPITRE 5

TCOWS : APPLICATION AU SCÉNARIO "AGENCE DE VOYAGES"

Dans ce chapitre l'objectif est, en s'appuyant sur le scénario "Agence de Voyages", de montrer les différentes étapes définies dans le canevas Tcows : la conception d'une composition (voir section 5.2), la sélection de ses participants (voir section 5.3), et finalement son exécution (voir section 5.4). Nous commençons dans la section 5.1 par rappeler les objectifs de l'approche que nous proposons.

5.1 Préambule

L'objectif initial de l'étude effectuée dans la thèse présentée ici est de construire des compositions transactionnelles de services web qui, à l'exécution, tiennent compte de propriétés extra fonctionnelles des composants impliqués : leur caractéristique transactionnelle et leur score par rapport à des critères donnés pris dans un modèle de qualité de services (par exemple *la réputation*) ou dans le modèle du service lui-même (par exemple *le prix de la chambre double pour trois nuits*). L'idée est d'identifier, *a priori*, les exécutions qui risquent de ne pas aboutir ou qui donneront des résultats incohérents par rapport aux besoins de l'application : *le voyage ne peut se faire que si le vol et l'hôtel ont été réservés*, *le service utilisé pour le vol doit avoir une réputation au moins égale à 10*, ou encore *le budget du voyage doit être inférieur à 3000*). La première assertion est liée à la portée de la propriété d'atomicité dans la transaction alors que les deux dernières sont spécifiques à des valeurs prises dans le contexte de l'exécution (par exemple, *la réputation du service Air France est 20*, *pour ce voyage, le prix du billet d'avion est de 1600, celui de l'hôtel est de 540, et celui de la location de la voiture est de 230*). Tenant compte du fait qu'il existe un grand nombre de services répondant aux mêmes besoins fonctionnels nous proposons de les mettre en concurrence au moment de l'exécution. Ceci permet de choisir, au moment de l'exécution, les meilleurs services par rapport aux requis exprimés dans la composition.

Pour atteindre cet objectif, nous avons défini le canevas Tcows (*Transactional Compositions Of Web Services*) dans lequel les compositions sont munies de paramètres permettant de fixer d'une part les propriétés que la composition doit vérifier au moment de son exécution, et d'autre part la portée de la transaction dans la composition.

Le modèle qui en découle permet en particulier le relâchement de la propriété d'atomicité. Rappelons que l'atomicité est une des propriétés transactionnelles (avec la cohérence, l'isolation et la durabilité) que l'exécution de la séquence des opérations qui constituent une transaction conventionnelle doit vérifier. Pour que cette propriété soit respectée, toutes les opérations de la transaction doivent s'exécuter, ou aucune. S'il y a au moins une de ces opérations que n'a pas pu s'exécuter alors la transaction a échoué et tous les effets des opérations exécutées doivent être défaits. Dans le cas d'une composition de services web, la transaction considérée selon le point de vue de l'application cliente, est constituée des opérations définies dans son modèle d'orchestration ainsi que de celles exposées par les partenaires entrant dans la composition. Garantir l'atomicité d'une telle transaction implique de pouvoir annuler les effets de toutes les opérations déjà exécutées lorsque l'une d'entre elles a échoué, y compris lorsque l'opération à annuler a été exécutée par un service composant. Ce qui pose problème lorsque le service concerné n'expose pas d'opération pour cela. Étant donnée une composition, au moment de sa conception, les capacités qui la composent ont été déclarées obligatoires ou facultatives (voir par exemple la figure 1.1, page 5). Le but est de définir une zone d'atomicité réduisant la portée de la transaction. Ainsi, ayant limité au strict nécessaire les opérations devant être exécutées avec succès, la probabilité de réussite de la transaction est plus grande.

Nous appuyons notre approche sur le concept de communauté de services web détaillé dans le chapitre 4. Ce concept nous permet d'exprimer des compositions en termes de capacités et non en termes de services. Ainsi, au moment de l'exécution d'une composition, un ou plusieurs services sont sélectionnés dans chaque communauté associée. Les diverses combinaisons qui en découlent constituent autant d'options d'exécution pour la même composition, ce qui augmente encore les chances d'obtenir une exécution correcte de la composition tout en satisfaisant les besoins de l'application exprimés au moment de sa conception.

5.2 Conception de compositions avec propriétés transactionnelles

La phase de conception est dédiée à l'élaboration du modèle d'orchestration de compositions transactionnelles, c'est-à-dire à :

- l'expression des besoins fonctionnels en termes de capacités de services : pour organiser un voyage il faut un billet d'avion et un hôtel, un véhicule et des données météorologiques sur la destination, et les interactions entre les services abstraits par les capacités,
- la spécification des données en entrée (les dates du voyage, etc.) et en sortie (le billet d'avion, etc.),
- l'expression de besoins extra fonctionnels (le budget doit être minimisé tout en étant inférieur à 3000, la réputation du service utilisé pour le vol doit être au moins égale à 10),

– et finalement à l'identification des capacités obligatoires (le billet d'avion et la chambre d'hôtel sont indispensables) et de celles facultatives (il n'est pas indispensable de disposer d'un véhicule sur place).

La section 5.2.1 commence par préciser les propriétés transactionnelles des services web entrant dans une composition. La section 5.2.2, reprenant le scénario "Agence de voyage", décrit la conception du modèle d'orchestration de l'application correspondante. Enfin, dans la section 5.2.3 discute de la définition des paramètres de la transaction.

5.2.1 Propriétés transactionnelles des services composants

Nous nous intéressons ici aux services web qui permettent l'acquisition de ressource(s) par un client (une application, un intergiciel, etc.). Cette acquisition ne peut s'opérer que par le biais d'une transaction entre le client et le fournisseur du service. Ainsi, le fournisseur doit doter ses services web d'opérations pour permettre la cession de la ressource au client. Selon la nature de ces opérations, il est possible de caractériser le comportement d'un service web par rapport à la propriété d'atomicité de la transaction dans laquelle il intervient [HA00]. Le but est de caractériser le comportement transactionnel de la composition en fonction de celui de ses participants :

– Un service est dit **atomique** lorsqu'il offre les opérations nécessaires pour permettre à la transaction qui l'utilise d'annuler si besoin les opérations qu'il a exécutées pour le compte de cette dernière. Ces fonctions sont la réservation, puis la validation ou l'annulation de la réservation de la ressource.
– Un service **quasi-atomique** est un service qui peut simuler un comportement "atomique" si le fournisseur de ce service a prévu un processus pour défaire les effets des opérations qu'il a exécutées pour le compte de la transaction. Ce processus est connu sous le nom de mécanisme de compensation [Elm90]. Ainsi, le service offre une opération pour acquérir (si possible) définitivement une ressource, et il offre aussi une opération de compensation.
– Un service est dit **non-atomique** lorsque l'opération qu'il offre pour l'acquisition de la ressource ne peut pas être annulée. Il fournit uniquement une opération pour acquérir définitivement une ressource. Il n'y a pas d'opération de réservation *a priori* ni de compensation *a posteriori*.

Prenons par exemple l'acquisition d'un billet d'avion. Si le service a un comportement atomique, il est possible de réserver le billet, en général pour un temps déterminé. Le fait de réserver le billet le rend indisponible aux autres clients, la ressource est donc verrouillée. C'est la garantie qu'au moment de valider la transaction la ressource sera toujours disponible. Avant que le délai ait expiré, le client peut soit annuler sa réservation, soit acheter le billet réservé. Dans le premier cas, la transaction est annulée et la ressource est rendue disponible à nouveau. Dans le second cas la transaction a réussi. L'expiration du délai provoque l'annulation de la transaction et libère la ressource. Si le service est quasi-atomique, le client peut acquérir le billet d'avion s'il est disponible, la compagnie peut cependant offrir des conditions particulières pour le report du voyage. L'opération de compensation consiste ici à modifier les dates du voyage par exemple, il s'agit de libérer la ressource acquise puis d'en acquérir une nouvelle. Finalement, dans le cas d'un service non-atomique le client peut acheter un billet d'avion mais, s'il change d'avis, il ne dispose

d’aucune opération, ni d’annulation ni de compensation.

Pour que les applications clientes puissent tenir compte de ce comportement transactionnel, les services web doivent exposer la propriété selon laquelle ils gèrent leurs ressources. Le comportement transactionnel d’un service est une de ses propriétés non fonctionnelles, il est donc prévisible de le trouver comme l’un des critères de son modèle de qualité.

La possibilité de garantir l’atomicité de l’exécution d’une composition dépend du comportement transactionnel des services participants à la composition. L’atomicité de l’exécution d’une composition est garantie lorsque tous les services participants sont atomiques. Pour cela, il suffit de coordonner correctement les opérations de réservation offertes par les participants, puis celles permettant la validation ou si nécessaire l’annulation. Si certains participants ont un comportement quasi-atomique l’atomicité peut être garantie aussi. Simplement, pour ces derniers, la coordination doit considérer leur comportement spécifique lors des phases de réservation et de validation ; l’annulation, en ce qui les concerne, s’appuie sur les mécanismes de compensation offerts. La présence de services non-atomiques pose problème : l’atomicité peut être garantie à la condition de pouvoir coordonner l’acquisition de la ressource correspondante avec les réservations des autres participants (atomiques ou quasi-atomiques). Cette discussion est détaillée et illustrée dans la suite de ce chapitre. Les principes donnés sont ensuite formalisés dans le chapitre 6 qui suit.

5.2.2 Modèle d’orchestration de la composition

La phase de conception de la composition conduit à l’identification des capacités entrant dans la composition, à la spécification d’un processus métier et à la définition de ses paramètres qui incluent les types de données utilisées en entrée et en sortie du processus, une fonction de restriction et une fonction de préférence. Les capacités étant déterminées, chacune associée à une communauté et les paramètres spécifiés (voir les figures 1.3 et 1.4 page 6), il est possible de donner une vision grossière du processus (voir la figure 5.1).

La figure 5.1 décrit le modèle d’orchestration du scénario “Agence de Voyages” qui a été introduit dans le chapitre 1. Cette figure décrit le processus de construction d’un voyage en faisant abstraction des détails des services utilisés : l’application cliente est conçue de manière à : (i) rechercher en parallèle un billet d’avion et une chambre d’hôtel, puis (ii) selon la localisation de l’hôtel, réserver une voiture ou une bicyclette, enfin (iii) interroger les conditions climatiques du lieu de séjour.

Nous prenons pour hypothèse qu’à chaque capacité associée à la conception de l’application “Agence de Voyages” une communauté a pu être identifiée (Vol, Hôtel, Voiture, Vélo et Météo). L’interface abstraite de chacune est formalisée ci-après. Chaque opération bilatérale (c’est-à-dire qui retourne un résultat) est abstraite par une fonction dont le domaine décrit le message qui lorsqu’il est envoyé déclenche l’exécution de l’opération. Le codomaine de la fonction décrit le message retourné à la suite de l’exécution de l’opération. Les opérations unilatérales sont abstraites par des actions dont le profil décrit le contenu du message envoyé.

Notre objectif n’est pas de définir un nouveau langage de description de services, mais plutôt d’abstraire les concepts généralement admis dans de nombreuses approches (voir par exemple [BDM02, MTR02, DTL+03]) et standards (voir par exemple [OAS02, CCC+02b, ACD+03]), et de dériver à partir de là, une notation abstraite permettant de raisonner sur

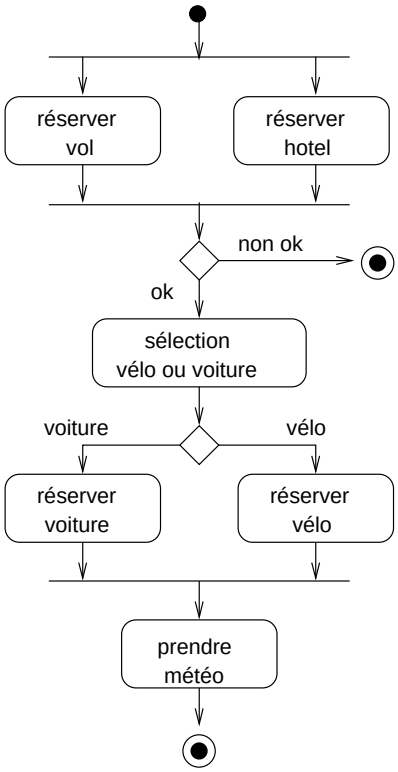


FIG. 5.1 – Modèle d'orchestration du scénario "Agence de Voyages" (vision grossière)

les propriétés des services.

A titre d'exemple, l'annexe A fournit la description en WSDL de l'interface exposée par la communauté Vol dont les opérations offertes sont :

demandeItinéraires : date, date, string, string $\longrightarrow$ {Itinéraire}
{ *demandeItinéraires (dDép, dRet, vDép, vDes) retourne l'ensemble des itinéraires possibles ayant dDép comme date pour le départ et dRet comme date pour le retour, vDép comme ville de départ et vDes comme ville de destination.* }

réservationItinéraire : Itinéraire, string $\longrightarrow$ Billet
{ *réservationItinéraire (i, c) retourne le billet d'avion correspondant à l'itinéraire i pour le client c.* }

La communauté Hôtel expose une seule opération de réservation d'une chambre :

réservationChambre : date, date, string, string, string $\longrightarrow$ boolean
{ *réservationChambre (dDéb, dFin, lieu, type, c) est vrai si la réservation d'une chambre depuis la dDép jusqu'à la date dFin, dans la ville lieu, pour le client c a pu être effectuée. Le paramètre type fait référence au type de la chambre : individuel, double, etc.* }

La communauté Voiture expose une seule opération de réservation d'une voiture :

réservationVoiture : date, date, string, string, string $\longrightarrow$ boolean
{ *réservationVoiture (dDéb, dFin, lieu, type, c) est vrai si la réservation d'une voiture depuis la dDép jusqu'à la date dFin, dans la ville lieu, pour le client c a pu être effectuée. Le paramètre type fait référence à la capacité de la voiture : 4 ou 5 ou 7 places, etc.* }

La communauté Vélo expose, elle aussi, une seule opération de réservation :

réservationVélo : date, date, string, string, string $\longrightarrow$ boolean
{ *réservationVélo (dDéb, dFin, lieu, type, c) est vrai si la réservation d'un vélo depuis la dDép jusqu'à la date dFin, dans la ville lieu, pour le client c a pu être effectuée. Le paramètre type fait référence au type de vélo : TT, TV, etc.* }

Finalement la communauté Météo expose une opération qui permet de connaître les conditions climatiques d'une ville pendant une période donnée :

demandeMétéo : date, date, string, $\longrightarrow$ text
{ *demandeMétéo (dDéb, dFin, lieu) donne les conditions climatiques à l'endroit lieu entre les dates dDép et dFin.* }

En considérant les interfaces exposées par chaque communauté, il est possible de raffiner le modèle décrit dans la figure 5.1 afin de donner un modèle d'orchestration en termes d'envois et de réceptions de messages et en termes d'opérations internes (voir figure 5.2).

Dans la figure 5.2, les activités sont associées soit à des envois ou à des réceptions de messages, elles sont alors étiquetées send pour l'envoi ou receive pour la réception, soit à des opérations internes.

La description des données en entrée et en sortie de cette orchestration est issue de l'analyse qui a conduit à la définition des paramètres de la composition. Pour ce qui concerne le scénario “Agence de Voyages” ces données sont (voir figure 1.3, page 6) :

{ *Les données en entrée :* }	{ *Les données en sortie :* }
dateDépart, dateRetour : date	billetAvion, détailsHôtel, détailsVéhicule, météo : string
villeDépart, villeDestination : string	budget : real

A partir de ces données, il est possible de construire le message contenant les détails du voyage. Ce message est envoyé au service sélectionné dans la communauté Vol afin d'obtenir une liste d'itinéraires possibles. En parallèle, un message est transmis au service de la communauté Hôtel pour la réservation d'une chambre. Si la liste d'itinéraires est vide, ou si la réservation de l'hôtel n'a pas abouti, le processus s'arrête. Dans le cas contraire, le client choisit dans la liste d'itinéraires celui qui correspond le mieux à ses besoins puis en demande la réservation. Lorsque le billet d'avion et la chambre d'hôtels le processus se continue par le choix du moyen de transport sur place (vélo ou voiture), puis par la demande des conditions météorologiques locales. Le processus se termine avec succès même si l'une parmi ces deux dernières opérations échouent.

Cette étape a été menée de manière à séparer les préoccupations liées à la conception du processus de celles liées à aux besoins extra fonctionnels, que nous avons décomposés en des propriétés transactionnelles, une restriction et une préférence. Ces derniers points sont abordés dans la section suivante.

5.2.3 Paramètres de la composition

Afin de limiter la portée de cette transaction aux opérations nécessaires uniquement, nous utilisons la notion de *zone transactionnelle*. Nous définissons une zone transactionnelle

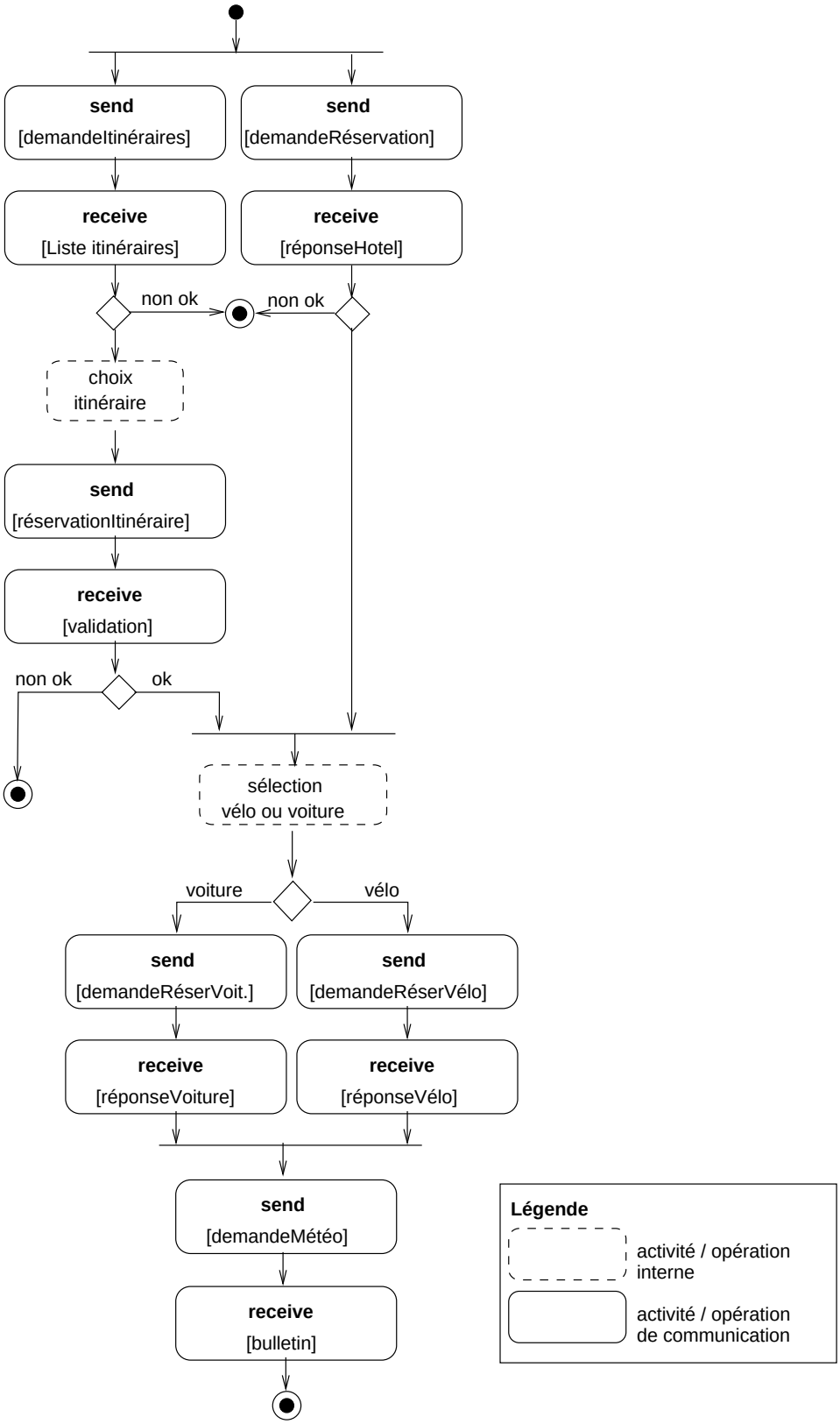


FIG. 5.2 – Le modèle de l'orchestration détaillée

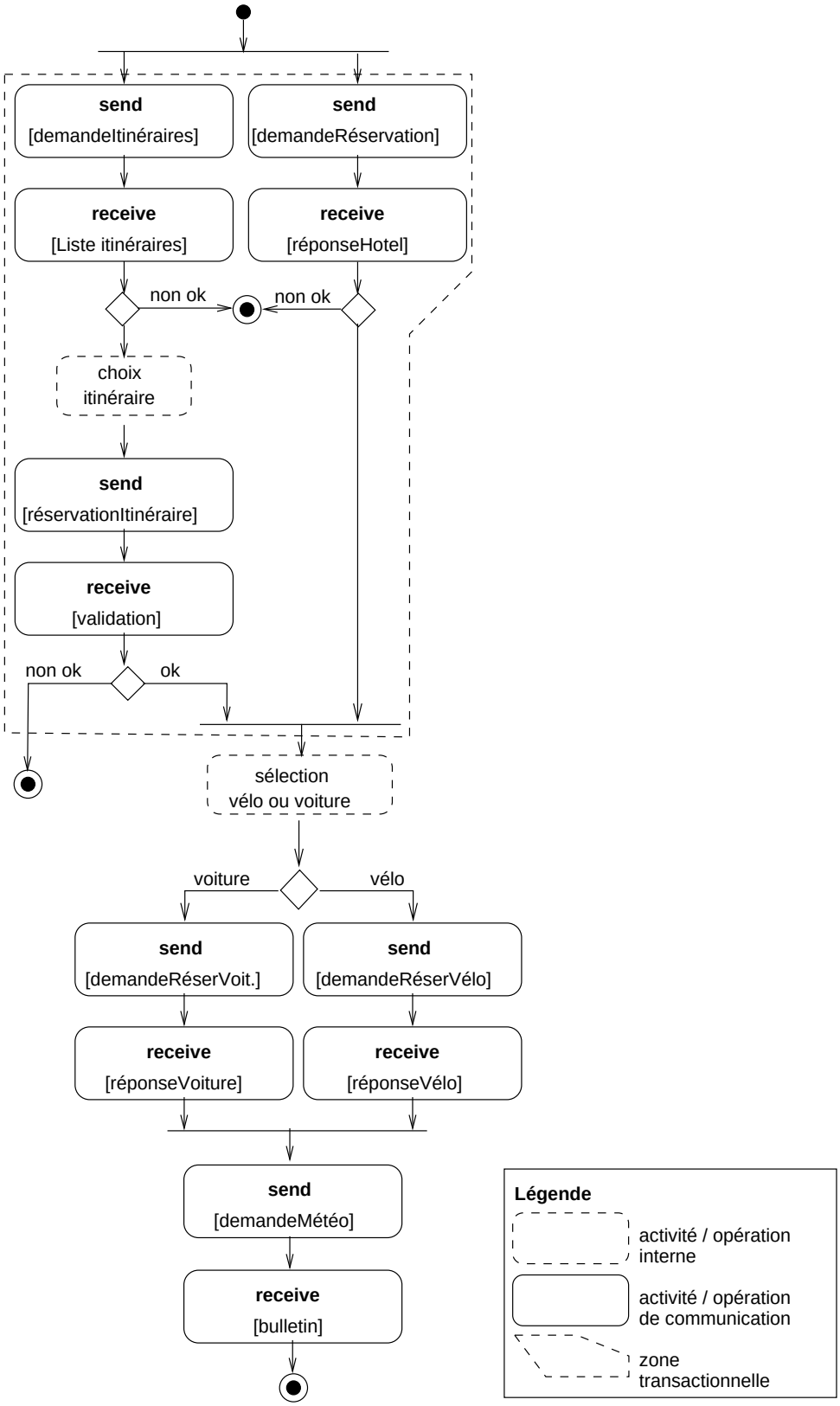


FIG. 5.3 – La zone transactionnelle de l’orchestration

comme un sous-diagramme d'activités, engendré par un sous-ensemble des activités du diagramme spécifiant le modèle d'orchestration de l'application[1]. Ainsi, la propriété d'atomicité est limitée aux opérations incluses dans la zone transactionnelle. Nous considérons ici qu'il n'y a qu'une seule zone transactionnelle et qu'elle est connexe. Dans le chapitre 8, nous revenons sur cette restriction.

La figure 5.3 illustre la définition de la zone transactionnelle sur le modèle d'orchestration du scénario décrit en termes des opérations offertes par les interfaces des communautés Vol, Hôtel, Voiture, Vélo et Météo. La zone transactionnelle, considérée ici comme l'un des paramètres de la composition, est déterminée sur la base des capacités obligatoires telles qu'elles ont été spécifiées au début du processus de conception de l'application (voir chapitre 1). La transaction commence lorsque le client reçoit les itinéraires proposés pour le vol et se termine au moment de la synchronisation entre la réception du billet et de la confirmation de la réservation de l'hôtel (voir figure 5.3).

Les trois autres paramètres de la composition concernent, le premier l'expression de la restriction (*le budget du voyage doit être inférieur à 3000*), le second l'expression de la préférence (*le budget du voyage doit être minimisé*) ; enfin le troisième, décrit le niveau de qualité requis pour les services participants. La restriction est donnée sous forme d'une fonction à valeurs booléennes qui doit être satisfaite, et la préférence est données sous forme d'une fonction à valeurs réelles qui formalise un score. Le niveau de qualité est donné sous la forme d'une variable du type RequiredQuality défini dans le chapitre 4.

5.3 Sélection des participants d'une composition

La phase de sélection des services effectifs entrant dans la composition s'appuie sur les possibilités de sélection offertes par les communautés associées aux capacités identifiées dans la composition (ceci a été discuté dans la section 4.2.3 du chapitre 4).

A chaque communauté référencée dans le modèle d'orchestration obtenu est associé un nombre potentiellement important de services, il est donc possible de construire différentes configurations différentes d'exécution. Chaque configuration de services ainsi obtenue est appelée *option*, dans le sens où elle décrit une composition potentielle. L'ensemble des options est ensuite réduit à celles vérifiant la fonction de restriction et le niveau de qualité requis. Enfin, à chaque option est associé son score traduisant son degré de satisfaction à la préférence et au niveau de qualité. Il peut arriver qu'il n'existe aucune option satisfaisant la restriction et le niveau de qualité requis ; dans ce cas la transaction échoue. Le nombre de services sélectionnés peut être extrêmement variable. Une étude est nécessaire pour fixer des règles de sélection : il faut sélectionner suffisamment de services, mais pas trop (voir chapitre 8).

En considérant le scénario "Agence de Voyages", nous illustrons maintenant la construction des options. Dans un premier temps, nous montrons comment formuler les sélections pour chaque communauté impliquée. L'expression ci-dessous permet de sélectionner dans la communauté Vol deux services selon les critères réputation (c1) et fiabilité (c3), pondérés respectivement par 0.8 et 0.2 :

[1]Pour donner cette définition nous avons procédé par analogie avec un sous-graphe, d'un graphe donné G, engendré par un sous-ensemble des nœuds de G.

Select (Vol, 2, {⟨ c1, 0.8, λx • x $\geq$ 20 ⟩, ⟨ c3, 0.2, λx • vrai ⟩})
{ *Cette expression retourne une liste de deux services pris parmi les services enregistrés dans la communauté Vol. Pour chacun d'eux, la valeur du critère Réputation (d'identifiant c1) est supérieure ou égale à 20, et la valeur du critère Fiabilité (d'identifiant c3) est quelconque. Les deux services retournés ont obtenu le meilleur score compte tenu de leurs scores aux deux critères pondérés respectivement par 0.8 et 0.2* }

c1 et c3 sont des critères définis dans le modèle de qualité de la communauté Vol (voir la section 4.3.2). Pour simplifier nous considérons les mêmes critères dans les communautés Hôtel, Vélo et Météo.

De manière similaire, les sélections dans les communautés Hôtel, Véhicule et Météo se traduisent par les expressions ci-dessous.

Select (Hôtel, 3, {⟨ d1, 0.8, λx • x > 10 ⟩, ⟨ d4, 0.2, λx • vrai ⟩})
{ *d1 et d4 correspondent aux critères Réputation et Fiabilité, du modèle de qualité de la communauté Hôtel* }
Select (Vélo, 2, {⟨ e1, 0.8, λx • x > 10 ⟩, ⟨ e2, 0.2, λx • vrai ⟩})
{ *e1 et e2 correspondent aux critères Réputation et Fiabilité, du modèle de qualité de la communauté Vélo* }
Select (Météo, 1, {⟨ f2, 0.8, λx • x > 10 ⟩, ⟨ f3, 0.2, λx • vrai ⟩})
{ *f2 et f2 correspondent aux critères Réputation et Fiabilité, du modèle de qualité de la communauté Météo* }

Le tableau 5.1 donne les valeurs des critères de qualité correspondant aux références des services web considérés dans chaque communauté. Par exemple, dans ce tableau le service Vauban a une Réputation de 20 qui est la valeur maximale, son score est donc de 2, alors que le service WalkAbout obtient un score de 1. Le service Formule 1 est disqualifié car sa réputation ne vérifie pas la fonction λx • x > 10. Ce dernier service ne sera pas retourné dans le résultat de la sélection. Le score global obtenu par chaque service exprime son degré de satisfaction à la qualité selon les deux critères considérés.

		Réputation		Fiabilité		Score
Communauté	Service web	valeur	score	valeur	score	global
Hôtel	Vauban	20	2	4	2	4
	WalkAbout	15	1	4	2	3
	Formule 1	10	-	2		-
Vol	Air France	30	2	4	2	4
	Avianca	20	1	4	2	3
Vélo	Bike solutions	10	2	2	1	3
	Birds	10	2	4	2	4
Météo	MétéoFrance	10	2	2	2	4
	BOM	10	2	2	2	4

Tab. 5.1 – Valeurs des critères de qualité des services sélectionnés

Une fois que pour chaque communauté, les services potentiels sont connus il est nécessaire de les contacter pour connaître leurs propriétés transactionnelles. Nous prenons pour hypothèse que chacun expose une opération via laquelle il est possible de connaître sa propriété transactionnelle. Dans chaque option, il ne doit pas y avoir plus d'un service atomique parmi ceux correspondant aux capacités obligatoires car alors, il serait impossible de garantir la propriété d'atomicité lors de l'exécution de l'option : les services non-atomiques

ne fournissent ni opération de réservation ni opération de compensation (si un service non-atomique est exécuté avec succès, alors la ressource acquise l'est définitivement).

Pour chacun il faut aussi connaître son prix afin d'évaluer le budget global du voyage. Le but est de composer les options qui satisfont la restriction et le niveau de qualité requis, puis de les ordonner par rapport à la fonction de préférence à la satisfaction au niveau de qualité. Nous faisons ici abstraction de la manière dont le prix est calculé pour chaque service. Dans le cadre du scénario "Agence de Voyages", nous prenons pour hypothèse que pour chaque service considéré il est possible de connaître le prix de la ressource qu'il fournit. Il s'agit par exemple du prix d'un itinéraire, celui de la chambre d'hôtel ou encore celui de la location de la voiture. Pour simplifier, nous ne considérons qu'un seul prix pour tous les itinéraires proposés pour chaque service de la communauté Vol. L'extraction du prix retourné par chaque service est abstraite par la fonction Prix spécifiée ci-après :

Prix : Service, Resource $\longrightarrow$ réel
{ *Prix (s, r) est le prix qui doit être payé pour acquérir la ressource r offerte par le service s. r modélise les données du processus.* }

Nous étudierons plus loin, dans le chapitre 6, le type abstrait Option qui modélise la notion d'option. Ainsi, il est possible de calculer le prix de chaque option et d'exprimer la restriction et la préférence comme suit :

o : Option
{ *o est une option définie par une composition. Si C est l'une des capacités définie dans la composition, alors o.C désigne le service sélectionné dans o pour cette capacité (voir chapitre 6).* }

La fonction de restriction peut ainsi être exprimée de la manière suivante (o dénote une option définie par une composition) :

R : Restriction
R (o) = λo $\bullet$ LePrix (o.Vol) + LePrix (o.Hôtel) + LePrix (o.Vélo) + LePrix (o.Météo) $<$ 3000
{ *R (o) est vrai si le budget de O est inférieur à 3000.* }

Nous faisons de même pour exprimer la fonction de préférence :

P : Préférence
P (o) = λo $\bullet$ $-$ LePrix (o.Vol) + LePrix (o.Hôtel) + LePrix (o.Vélo) + LePrix (o.Météo)
{ *P (o) est le score obtenu par l'option o. Ici, c'est l'opposé du coût global de l'option (pour minimiser le coût global, il convient de maximiser son opposé).* }

Communauté	Service web	Prix valeur	Prix score	Niv. qualité score	Propriété transactionnelle
Hôtel	Vauban	2000	2	4	A
	WalkAbout	100	1	3	NA
Vol	Air France	1200	2	4	QA
	Avianca	850	1	3	NA
Vélo	Bike solutions	10	1	3	NA
	Birds	150	2	4	QA
Météo	MétéoFrance	0	1	4	NA
	BOM	0	1	4	NA

Tab. 5.2 – Prix et propriétés transactionnelles des services

Le tableau 5.2 montre les valeurs obtenues par les services sélectionnés dans chaque communauté. Pour chacun, le tableau indique la valeur retournée pour le prix (lorsqu'il a été fourni) et le score obtenu. Il donne aussi sa propriété transactionnelle et rappelle le score obtenu lors de la sélection sur les critères de qualité.

Op	Hôtel	Vol	Vélo	Météo	prix		niv. qual.		final	Nb.
					val.	sco.	val.	sco.		NA
-	WalkA.	Avianca	Birds	BOM	1100	16	14	16	32	2
-	WalkA.	Avianca	Birds	MétéoFr.	1100	16	14	16	32	2
-	WalkA.	Avianca	Bike sol.	MétéoFr.	1100	16	13	12	28	2
-	WalkA.	Avianca	Bike sol.	BOM	1100	16	13	12	28	2
1	WalkA.	Air Fr.	Birds	BOM	1450	8	15	18	26	1
2	WalkA.	Air Fr.	Birds	MétéoFr.	1450	8	15	18	26	1
3	WalkA.	Air Fr.	Bike sol.	MétéoFr.	1450	8	14	16	24	1
4	WalkA.	Air Fr.	Bike sol.	BOM	1450	8	14	16	24	1
-	WalkA.	Avianca		MétéoFr.	950	18	10	5	23	2
-	WalkA.	Avianca		BOM	950	18	10	5	23	2
-	WalkA.	Avianca	Birds		1100	16	10	5	21	2
5	WalkA.	Air Fr.		BOM	1300	10	11	10	20	1
6	WalkA.	Air Fr.		MétéoFr.	1300	10	11	10	20	1
7	WalkA.	Air Fr.	Birds		1450	8	11	10	18	1
-	WalkA.	Avianca	Bike sol.		1100	16	9	1	17	2
8	WalkA.	Air Fr.	Bike sol.		1450	8	10	5	13	1
9	Vauban	Avianca		BOM	2850	2	11	10	12	1
10	Vauban	Avianca		MétéoFr.	2850	2	11	10	12	1
dis	Vauban	Avianca	Bike sol.	BOM	3000					1
dis	Vauban	Avianca	Bike sol.		3000					1
dis	Vauban	Avianca	Bike sol.	MétéoFr.	3000					1
dis	Vauban	Avianca	Birds	MétéoFr.	3000					1
dis	Vauban	Avianca	Birds	BOM	3000					1
dis	Vauban	Avianca	Birds		3000					1
dis	Vauban	Air Fr.		MétéoFr.	3200					0
dis	Vauban	Air Fr.		BOM	3200					0
dis	Vauban	Air Fr.	Bike sol.	MétéoFr.	3350					0
dis	Vauban	Air Fr.	Bike sol.		3350					0
dis	Vauban	Air Fr.	Bike sol.	BOM	3350					0
dis	Vauban	Air Fr.	Birds	MétéoFr.	3350					0
dis	Vauban	Air Fr.	Birds	BOM	3350					0
dis	Vauban	Air Fr.	Birds		3350					0

Tab. 5.3 – Scores par option

Le tableau 5.3 récapitule la liste des options construites à partir des services sélectionnés. Chaque ligne du tableau décrit une option. Par exemple, la ligne indexée par 6 (dans la colonne “Op”) décrit l'option composée des services WalkAbout pour la capacité Hôtel, Air France pour la capacité Vol et MétéoFrance pour la capacité Météo. Aucun service n'a été sélectionné pour la capacité Véhicule. Le budget total de cette option est de 1300

(somme des prix de chaque participant à l'option). Cette option a obtenu un score de 10 suite à son classement par rapport à son prix total, et un score de 10 par rapport à son niveau de qualité. Cette option est classée en sixième position. Les options marquées "dis" dans la colonne "Op" ne satisfont pas la restriction sur le budget global du voyage. La colonne "Nb. NA" indique le nombre de services non-atomiques (associés aux capacités obligatoires uniquement), les options pour lesquelles ce nombre est au moins égal à deux sont marquées "-". Elles ne seront plus considérées dans la suite du processus.

5.4 Exécution de composition transactionnelle

Dans cette section nous détaillons le comportement des options correspondant à la composition. La liste constituée dans la phase de sélection est ordonnée en ordre décroissant des scores finalement obtenus par les options. La phase discutée dans cette section tente l'exécution et la validation de chaque option, l'une après l'autre. Cette itération s'arrête dès qu'une exécution a réussi, ou lorsque la liste a été épuisée. Ce dernier cas conduit à l'échec de l'exécution.

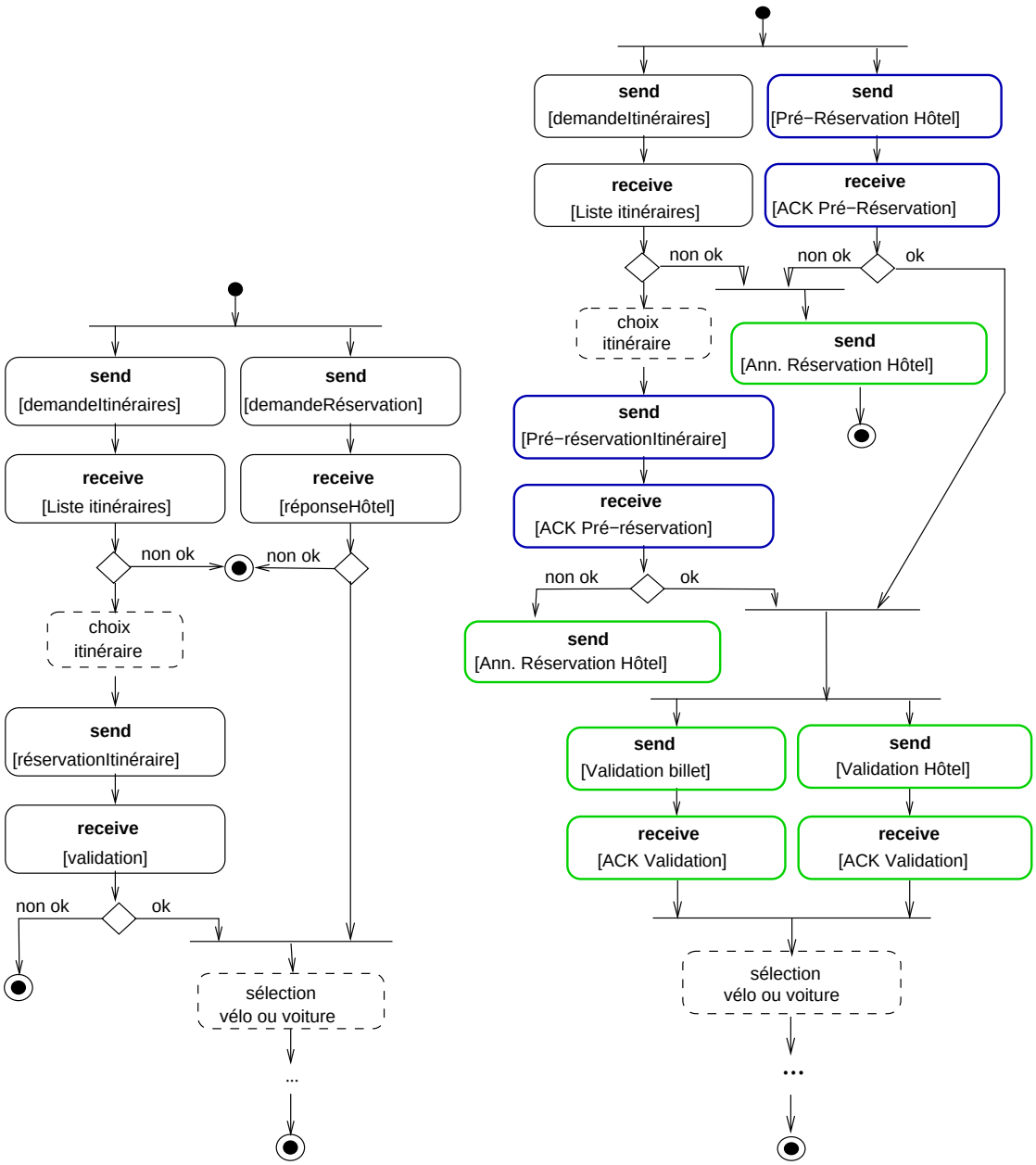
L'objet de cette section est de discuter de la mise en correspondance entre l'orchestration de la composition, conçue en termes des interfaces abstraites exposées par les communautés et sa coordination en termes des interfaces des services effectivement sélectionnés au moment de l'exécution. L'idée est d'ignorer les adaptateurs fournis par les services et de coordonner l'exécution en termes des opérations des services effectivement sélectionnés. En effet, la vision unifiée donnée par les interfaces abstraites des communautés ne permet pas de tenir compte des spécificités des services en ce qui concerne leur comportement transactionnel. Les mises en correspondance discutées ici tiennent compte de ces spécificités, elles sont donc effectuées selon que les services sélectionnés sont atomiques, quasi-atomiques ou non-atomiques. Nous considérons dans chaque cas le modèle d'orchestration défini dans la figure 5.2. Chaque service met à disposition la description WSDL de son interface. Ce qui permet de connaître les opérations spécifiques de chacun indépendamment de celles exposées par l'interface abstraite de la communauté.

La discussion est limitée à la zone transactionnelle car la mise en correspondance ne porte que sur les opérations incluses dans cette zone. En effet, la partie de l'orchestration qui se trouve en dehors de la zone transactionnelle est coordonnée en termes des interfaces des communautés et les accès aux services s'effectuent via leurs adaptateurs.

Pour simplifier le discours, nous nous limitons ci-après à décrire trois situations caractéristiques : tous les services sont atomiques (voir section 5.4.1), tous sont quasi-atomiques (voir figure 5.4.2), certains sont non-atomiques (voir figure 5.4.3). Le cas général où les services quelconques est étudié dans le chapitre 6.

5.4.1 Services tous atomiques

Un service atomique expose les opérations nécessaires à la réservation d'une ressource dans la perspective de son acquisition par un client, puis à la validation ou à l'annulation de cette acquisition. La figure 5.4 met en correspondance l'orchestration de la composition exprimée en termes de l'interface de la communauté (voir figure 5.4(a)) avec celle exprimée



(a) L’orchestration selon les interfaces des communautés

(b) L’orchestration selon les interfaces des services

FIG. 5.4 – Mise en correspondance des opérations (services atomiques)

en termes des opérations exposées par les services de l'option considérée (voir figure 5.4(b)).

Puisque les services considérés sont atomiques, les phases d'acquisition de ressources dans l'orchestration originale doivent être remplacées, dans l'orchestration réécrite, par des séquences de réservations/validations. Les validations devant être reportées le plus tard possible.

L'orchestration réécrite, donnée dans la figure 5.4(b) a été obtenue à partir de la figure 5.4(a), dans laquelle les activités send [réservationItinéraire] et receive [validation] ont été respectivement remplacées par les activités send [Pré-réservationItinéraire] et receive [ACK Pré-réservation]. Les activités send [Validation billet] et receive [ACK Validation] ont été ajoutées après la synchronisation des flux issus de la zone transactionnelle. Seront ajoutées à cet endroit toutes les activités correspondant à des appels aux opérations de validation d'acquisition de ressources.

De même, les activités send [demandeRéservation] et receive [réponseHôtel] traduisant les interactions avec le service sélectionné dans la communauté Hôtel sont respectivement remplacées par send [Pré-réservation Hôtel] et receive [ACK Pré-réservation]. Les activités send [Validation Hôtel] et receive [ACK Validation] ont été ajoutées après la synchronisation.

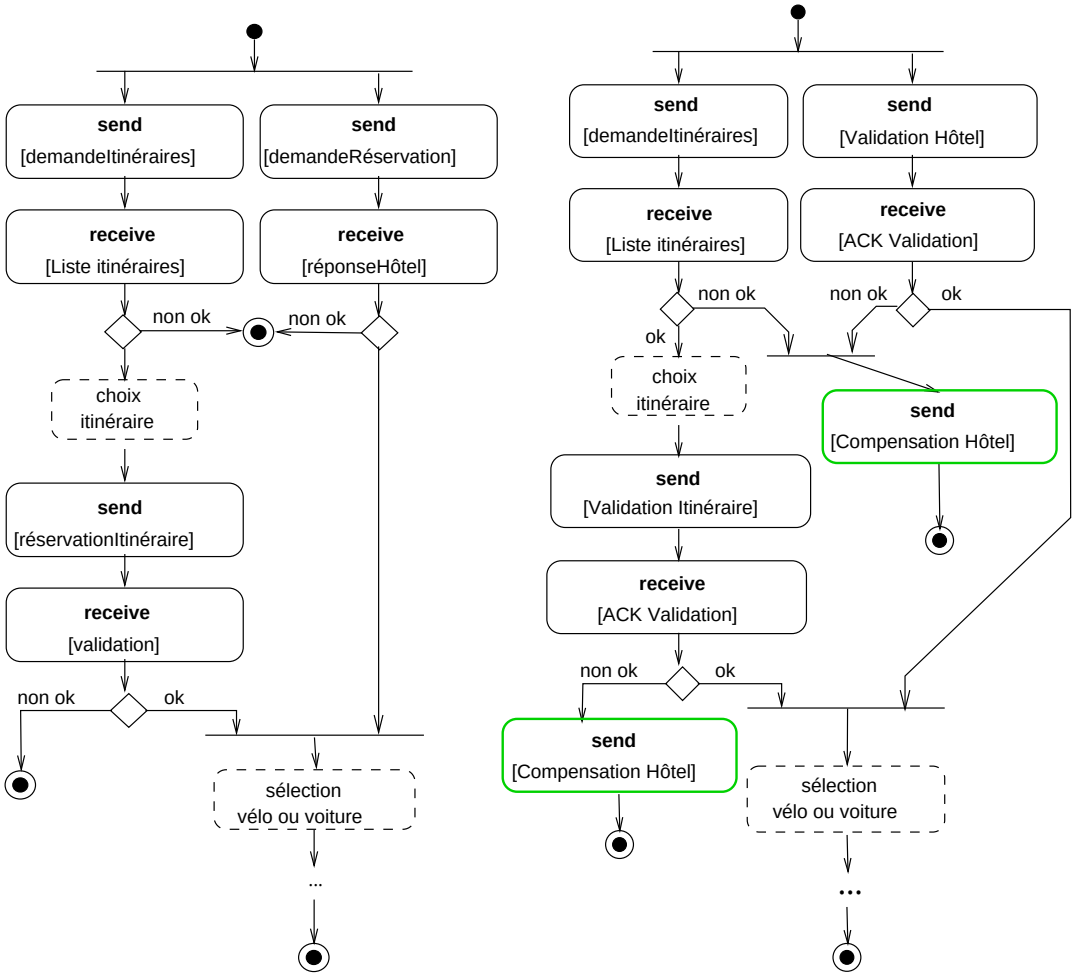
Des activités correspondant aux appels aux opérations d'annulation ont été ajoutées à la suite des flux relatifs aux conditions d'échec.

5.4.2 Services tous quasi-atomiques

Comme dans la section précédente, la figure 5.5(a) montre l'orchestration de la composition basée sur l'interface des communautés et la figure 5.5(b) montre la réécriture de cette orchestration quand les services composants exposent un comportement transactionnel quasi-atomique.

L'orchestration réécrite, donnée dans la figure 5.5(b) a été obtenue à partir de la figure 5.5(a), dans laquelle les activités liées à la réservation ou à l'acquisition de ressources ont été remplacées par des activités d'acquisition de ressources correspondant aux opérations offertes par des services quasi-atomiques. Ainsi, dans les interactions avec le service sélectionné dans la communauté Vol, les activités send [réservationItinéraire] et receive [validation] ont été respectivement remplacées par les activités send [Validation Itinéraire] et receive [ACK Validation]. De même, dans les interactions avec le service de la communauté Hôtel, les activités send [demanderRéservation] et receive [réponseHôtel] traduisant les interactions avec le service sélectionné ont été respectivement remplacées par les activités send [Validation Hôtel] et receive [ACK Validation].

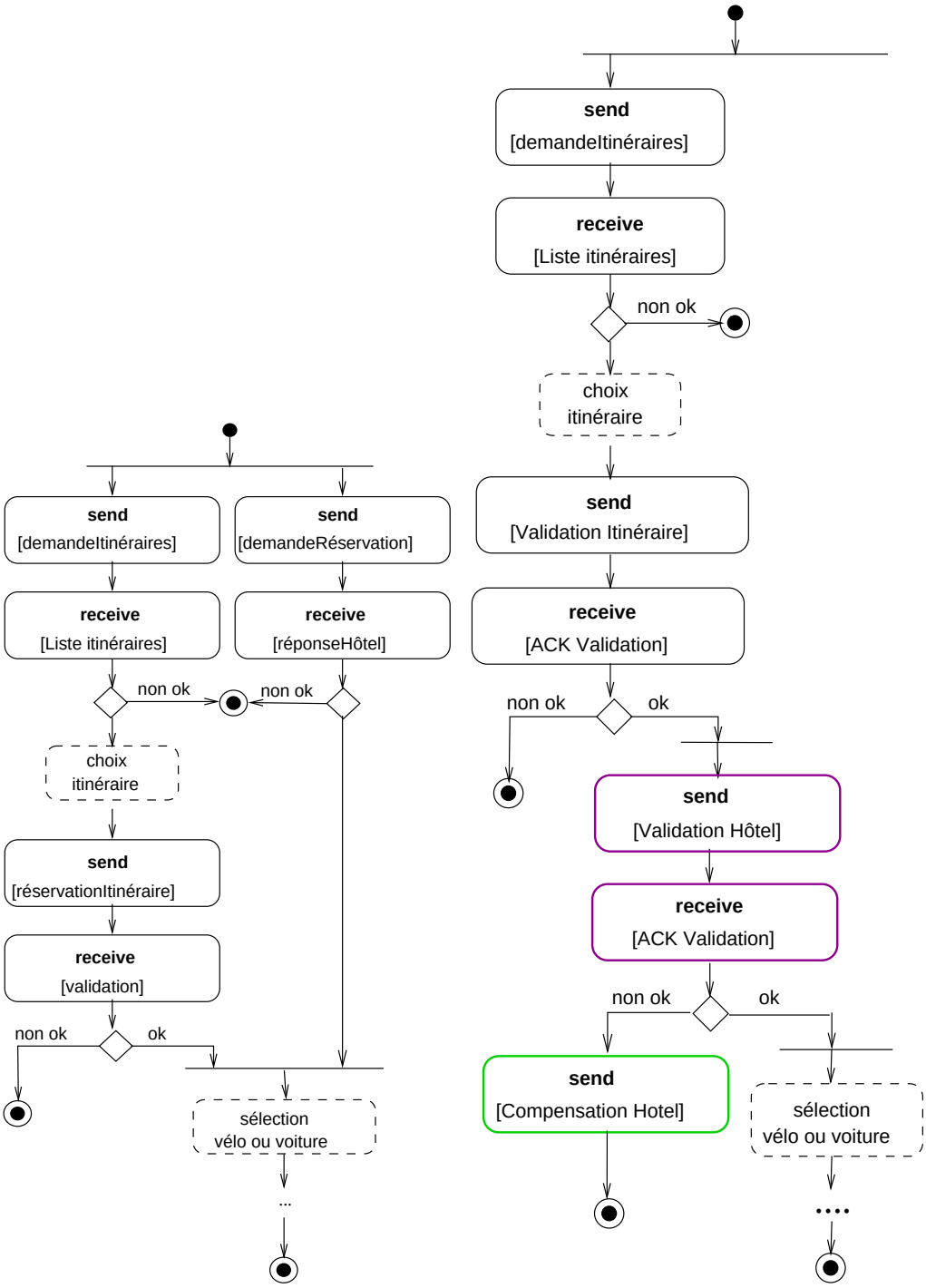
Des activités correspondant aux appels aux opérations de compensation ont été ajoutées à la suite des flux associés aux conditions d'échec. Par exemple, prenons par hypothèse que la demande d'itinéraires échoue alors que la chambre d'hôtel a été réservée. Comme le service sélectionné dans la communauté Hôtel est quasi-atomique, il est possible d'exécuter l'action de compensation pour annuler la chambre réservée et stopper la composition. Si l'échec se produit au moment de l'acquisition du billet d'avion, il est possible de faire de même. De cette façon, la transaction se termine correctement par rapport à la spécification de la composition.



(a) L’orchestration selon les interfaces des communautés

(b) L’orchestration selon les interfaces des services

FIG. 5.5 – Mise en correspondance des opérations (services quasi-atomiques)



(a) L'orchestration selon l'interface des communautés

(b) L'orchestration selon l'interface des services

FIG. 5.6 – Un service quasi-atomique et un service non-atomique

5.4.3 Services non-atomiques

Dans le cas de figure où au moins deux des services sélectionnés sont non-atomiques, il n'est pas possible d'assurer la propriété d'atomicité : si l'un des services, celui de la communauté Vol ou celui de la communauté Hôtel, échoue alors que l'autre a acquis la ressource, aucune annulation n'est possible. Dans cette situation, l'exécution de l'option n'est pas tentée.

Prenons pour hypothèse que dans cette composition il y a un service quasi-atomique, l'autre étant non-atomique. Supposons que le service sélectionné dans la communauté Vol est quasi-atomique et que le service sélectionné dans la communauté Hôtel est non-atomique. La figure 5.6(b) montre l'orchestration réécrite à partir de celle basée sur les interfaces des communautés (voir figure 5.6(a)). L'exécution des opérations du service non-atomique sont reportées le plus tard possible de manière à assurer l'acquisition de la ressource offerte par le service quasi-atomique. En cas d'échec (par exemple si la réservation de l'hôtel a échoué), l'activité de compensation pour annuler l'acquisition du billet d'avion est exécutée.

Les interactions avec le service sélectionné dans la communauté Vol ont été réécrites de la même manière que dans la figure 5.5. Comme il n'y a pas de dépendance entre le service de la capacité Vol avec celui de la capacité Hôtel, les interactions avec le service de la capacité Hôtel sont reportées après la validation de l'acquisition du billet d'avion. Ainsi, si la réservation de l'hôtel échoue, l'activité correspondant à la compensation du billet d'avion est exécutée. Si la réservation du billet d'avion échoue, aucune interaction avec le service de la capacité Hôtel n'est initiée.

5.5 Conclusion

Ce chapitre a présenté, par le biais d'une illustration, les principaux aspects liés au processus proposé dans le canevas Tcows pour la conception et l'exécution de compositions transactionnelles qui tiennent compte des propriétés transactionnelles hétérogènes des services composants. Nous avons caractérisé le comportement transactionnel des services en nous appuyant sur la définition proposée dans [HA00]. Nous avons montré les différentes phases proposées par le canevas, depuis la conception à gros grains de la composition, jusqu'à son exécution en passant par la définition des paramètres qui la caractérisent et la sélection des services qui entrent potentiellement dans la composition.

La notion de communauté présentée dans le chapitre 4 fournit un cadre à la sélection de services sur des critères variés. Grâce à cela, nous pouvons construire plusieurs options possibles pour la composition et les mettre en concurrence. Les options sont comparées les unes aux autres sur la base de leur adéquation aux critères définis à la conception de la composition. Ceux-ci concernent aussi bien des aspects liés au niveau de qualité des services, leurs propriétés transactions ou encore d'autres critères spécifiques à l'application.

Nous avons ensuite décrit comment nous obtenons l'orchestration à exécuter par réécriture de l'orchestration fournie par le concepteur. Cette réécriture, spécifique à chaque option, s'appuie sur les propriétés transactionnelles des services participant à l'option. La combinaison de ces propriétés joue un rôle décisif dans la réussite de la transaction. Par exemple, il peut arriver qu'une option mieux placée par rapport à la préférence exprimée dans la

composition ne puisse pas être exécutée car le nombre de services non-atomiques qu'elle contient est trop élevé. C'est le cas en particulier lorsque le nombre des capacités de service obligatoires est supérieur à un.

Dans ce chapitre, nous nous sommes attachés à illustrer les concepts et les mécanismes associés définis dans le modèle de compositions transactionnelles proposé dans le canevas `Tcows`. Le chapitre suivant (chapitre 6) est dédié à la formalisation de ce modèle.

CHAPITRE 6

FORMALISATION

Dans le chapitre 5 nous avons illustré l'utilisation de la méthode proposée dans `Tcows`. Cette méthode a pour perspective d'exécuter des compositions transactionnelles basées sur des services web. Rappelons que les trois phases proposées dans le canevas `Tcows` sont : (i) une phase de conception qui fixent les propriétés fonctionnelles et extra fonctionnelles de l'application à concevoir, (ii) une phase d'identification des communautés sur lesquelles appuyer la composition et pour chacune la sélection des services pouvant participer à l'exécution de la composition. Cette phase conduit à la fourniture d'un ensemble d'options, et enfin (iii) une phase d'exécution qui tente les options, l'une après l'autre. Si l'une des options est exécutée avec succès, alors la transaction associée à la composition a réussi (toutes ressources liées aux capacités obligatoires ont pu être acquises), sinon la transaction a échoué (aucune ressource liée aux capacités obligatoires n'a été acquise). Les phases (ii) et (iii) sont effectuées au moment de l'exécution de la composition conçue à la phase (i).

Ce chapitre présente, dans la section 6.1, le principe général proposé par le canevas de composition `Tcows`. La section 6.2 présente la formalisation de chacune des phases du canevas et la section 6.3 fait la conclusion.

6.1 Phases du processus mis en œuvre dans `Tcows`

La figure 6.1 formalise la méthode proposée dans `Tcows` par le biais d'un diagramme d'activités. Cette figure montre les trois phases de la méthode, chacune décrite par des activités, des données générées par ces activités et des données consommées. Parmi ces activités, certaines sont entièrement manuelles, sous la responsabilité du concepteur, certaines sont exécutées par le canevas.

La phase de conception permet d'exprimer les besoins fonctionnels et non fonctionnels de la composition. Les informations produites par cette phase sont celles issues de l'expression des besoins fonctionnels, c'est-à-dire : le modèle d'orchestration à gros grains, les capacités de services et les données d'entrée et de sortie du processus décrit par la composition. De

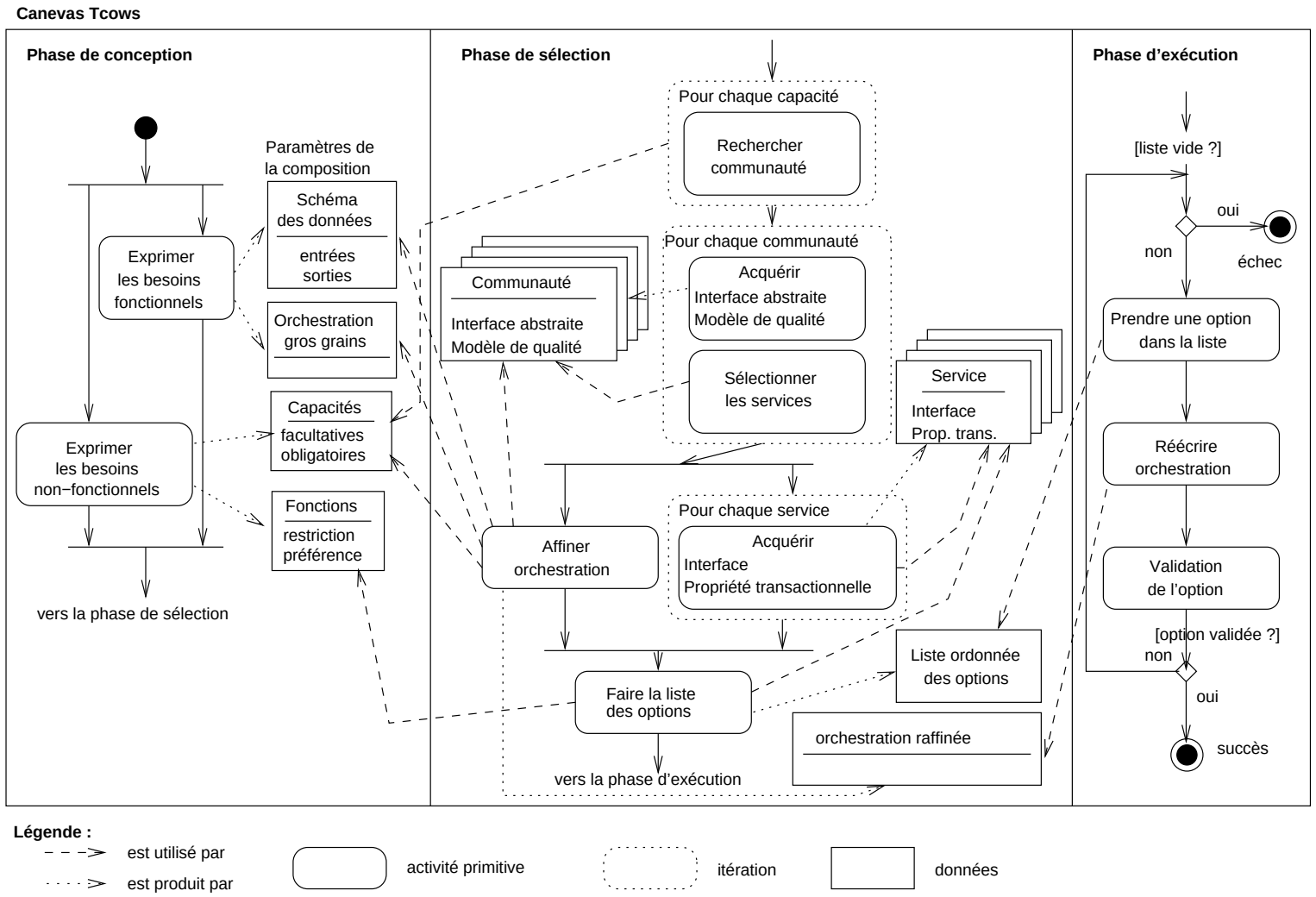


FIG. 6.1 – Phases du processus proposé dans TCOWS

plus, l'expression des besoins extra fonctionnels fixe pour chaque capacité son caractère obligatoire ou facultatif, et les fonctions de préférence et de restriction ainsi le niveau de qualité de service requis. L'ensemble de toutes ces informations constituent les paramètres de la composition.

Dans la phase de sélection, il s'agit de rechercher pour chaque capacité identifiée à la phase précédente, la communauté de services qu'il est possible d'associer. Puis, pour chacune d'entre elles, il s'agit d'obtenir son interface abstraite et son modèle de qualité. Il est alors possible de produire le modèle de l'orchestration raffinée de la composition, exprimé en termes des opérations décrites dans les interfaces abstraites des communautés. L'information générée est un processus BPEL dont les entrées et les sorties sont respectivement les entrées et les sorties du modèle d'orchestration. Pour chaque capacité de l'orchestration est défini un partenaire dans le processus BPEL.

D'autre part, cette phase est responsable, pour chaque capacité, de la sélection des services web pouvant participer à l'exécution de la composition. Cette sélection opérée auprès de la communauté associée à la capacité est paramétrée par des critères pris dans le modèle de qualité spécifique à la communauté. Considérant les services sélectionnés, cette phase va ensuite construire les différentes options possibles de la composition. Cette construction est guidée à la fois par les fonctions de restriction et de préférence et par la qualité des services pour les critères utilisés à la sélection. Chaque option fait référence à exactement un service web pour chaque capacité obligatoire et à un ou zéro service pour les capacités facultatives. Chaque option, décrivant une composition potentielle de services, satisfait la fonction de restriction et atteint le niveau requis de qualité de service.

La liste d'options est ordonnée par rapport à la moyenne des scores obtenus par chaque option à la fonction de préférence d'une part et à l'évaluation de sa qualité de service d'autre part.

La troisième phase, dite d'exécution, consiste à tenter l'exécution d'au moins une des options de la liste constituée précédemment. Chaque option de la liste est traitée l'une après l'autre. Chacune est exécutée selon une orchestration déduite du modèle de l'orchestration raffinée de la composition et qui tient compte de l'interface concrète et des propriétés transactionnelles des services référencés dans l'option.

6.2 Formalisation à base de types abstraits

Cette section présente les types et leurs opérations, qui formalisent chaque phase présentée dans la figure 6.1. Nous utilisons le même style de formalisation fonctionnel que dans chapitre 4 pour décrire les types associés à la notion de service et leurs opérations.

6.2.1 Phase de conception

Une composition de services est décrite, nous l'avons déjà vu, par son modèle d'orchestration à gros grains, ses capacités de services, obligatoires et facultatives, les données d'entrée et de sortie du processus associé, et enfin par les fonctions de préférence et de restriction. Les types décrits ci-après formalisent une composition.

type Composition
{ *Le type Composition modélise les éléments issus de la phase d'analyse des besoins : l'orchestration à gros grains, le schéma des données en entrée et en sortie, les capacités obligatoires et facultatives, et les fonctions de restriction et de préférence.* }

CoarseOrchestration : Composition $\longrightarrow$ BpelProcess
{ *CoarseOrchestration (cp) est le processus BPEL décrivant la composition cp. Le processus BPEL décrit les données d'entrée et de sortie. Chaque capacité de cp est associé à un partenaire dans le processus.* }

Capacités facultatives et obligatoires

type Capacity
{ *Le type Capacity modélise les capacités de services entrant dans une composition.* }
MandatoryCapacities : Composition $\longrightarrow$ {Capacity}
{ *MandatoryCapacities (cp) est l'ensemble de capacités obligatoires associées à la composition co.* }
OptionalCapacities : Composition $\longrightarrow$ {Capacity}
{ *OptionalCapacities (cp) est l'ensemble de capacités facultatives associées à la composition co.* }
{ *A chaque capacité de MandatoryCapacities (cp) $\cup$ OptionalCapacities (cp) est associé un partenaire dans CoarseOrchestration (cp).* }

Fonctions de restriction et de préférence

type Preference : (type $\longrightarrow$ real)
{ *Preference est le type des fonctions à valeurs réelles.* }
type Restriction : (type $\longrightarrow$ boolean)
{ *Restriction est le type des fonctions à valeurs booléennes.* }
CompositionPreference : Composition $\longrightarrow$ (type $\longrightarrow$ real)
{ *CompositionPreference (cp) est la fonction de préférence associée à la composition cp.* }
CompositionRestriction : Composition $\longrightarrow$ Restriction
{ *CompositionRestriction (cp) est la fonction de restriction associée à la composition cp.* }

6.2.2 Phase de sélection

La phase de sélection fait référence à :

- La recherche des communautés les plus appropriées pour chaque capacité définie dans la composition. Pour chaque communauté il est alors possible d'obtenir son modèle de qualité et son interface abstraite.
- Le raffinement de la composition afin d'obtenir une orchestration basée sur les interfaces abstraites des communautés.
- La définition du niveau de qualité requis par la composition.
- La sélection d'un nombre quelconque de services dans chaque communauté.
- L'établissement d'une liste d'options, dont chacune est une composition potentielle.

Qualité de service

La fonction QoS ci-dessous modélise le niveau de qualité requis par la composition, pour chaque capacité :

QoS : Composition, Capacity $\longrightarrow$ {RequiredQuality}
{ *Soit QoS (cp, ca) est un ensemble de spécifications de qualité de service requis pour la capacité ca dans la composition cp.*
Pré condition : $\forall$ *nq* $\in$ *QoS (cp, ca), nq.c* $\in$ *QualityModel (CommunityForCapacity (cp, ca)). nq.c dénote le critère de qualité associé à nq (voir chapitre 4, page 60).* }

Services et communautés

Nous détaillons ci-dessous le type Service (voir section 4.2).

type Service
{ *Le type Service permet de modéliser une offre d'un ensemble d'opérations (par exemple, achat de billets d'avion). Une instance du type Service correspond à une entité logicielle à laquelle un client peut accéder au travers des opérations qu'elle propose.* }
ServiceAtUrl : string $\longrightarrow$ Service
{ *ServiceAtUrl (s) est le service accessible à l'URL s.* }
ServiceCapacity : Service, Composition $\longrightarrow$ Capacity
{ *ServiceCapacity (s, cp) est la capacité à laquelle est associé le service s dans la composition co.* }

Les services exposent leur propriété transactionnelle, selon cette dernière, le jeu d'opérations qu'ils fournissent est différents.

TransactionalProperty : Service $\longrightarrow$ enum {atomique, quasi-atomique, non-atomique}
{ *TransactionalProperty (s) est la propriété transactionnelle du service s.* }

Pour chaque type de service (atomique, quasi-atomique ou non-atomique, voir section 5.2.1) nous définissons les opérations ci-dessous, en faisant abstraction de leur signature effective telle qu'elle est définie dans leur interface.

Reserve : Service, Resource $\longrightarrow$ boolean
{ *Reserve (s, r)* $\Longleftrightarrow$ *la ressource r fournie par le service s a pu être réservée.*
Pré condition : TransactionalProperty (s) = atomique. }
Cancel : Service, Resource $\longrightarrow$ void
{ *Cancel (s, r) annule la réservation de la ressource r fournie par le service.*
Pré condition : TransactionalProperty (s) = atomique $\wedge$ *Reserve (s, r).* }
Validate : Service, Resource $\longrightarrow$ boolean
{ *Validate (s, r)* $\Longleftrightarrow$ *l'acquisition de la ressource r fournie par le service s est validée.* }

L'opération de compensation offerte par les services quasi-atomiques :

Compensate : Service, Resource $\longrightarrow$ void
{ *CompensateR (s, r) compense l'acquisition de la ressource r fournie par le service s.*
Pré condition : TransactionalProperty (s) = quasi-atomique }

Les services non-atomiques offrent quant à eux la seule opération de validation (Validate).

Ci-dessous sont définies les relations entre les compositions, les capacités, les communautés, et les services :

type Community
{ *Le type Community modélise la notion de communauté comme indiqué dans la figure 4.1 donnée à la page 57 du chapitre 4.* }

CommunityForCapacity : Composition, Capacity $\longrightarrow$ Community
{ *CommunityForCapacity (cp, ca) est la communauté associée à la capacité ca dans le contexte de la composition cp. Nous considérons qu'il existe au moins une communauté pour chaque capacité. S'il en existe plusieurs, une seule sera choisie de manière arbitraire.* }
ServicesForCapacity : Composition, Capacity $\longrightarrow$ [Service]
{ *ServicesForCapacity (cp, ca) est la liste des services accessibles via la communauté CommunityForCapacity (cp, ca), qui satisfont et maximisent l'ensemble des critères donnés dans QoS (cp, ca)* }
Pré condition : $\forall$ nq $\in$ QoS (cp, ca), nq.c $\in$ QualityModel (CommunityForCapacity (cp, ca)).
Propriété : soit co $\longleftarrow$ CommunityForCapacity (cp, ca).
ServicesForCapacity (cp, ca) =
Map (Select (co, length (ServicesForCapacity (cp, ca)), Qos (cp, ca)),
λrs $\bullet$ ServiceAtUrl (ServiceUrl (co, rs)))

Les fonctions utilisées dans cette expression sont :

- Select est fournie par les communautés, elle est définie dans la section 4.2.3, page 60.
- length qui retourne le nombre d'éléments d'une liste.
- Map (L, F) applique la fonction F à chaque élément de la liste L. Le résultat est une liste d'éléments du type défini par le codomaine de F, ici le type Service.
- ServiceUrl (co, rs) est l'url du service rs enregistré dans la communauté co (cette fonction est offerte par la communauté).

Raffinement de l'orchestration

Dans cette phase nous raffinons l'orchestration à gros grains produite dans la phase de conception. La fonction FineOrchestration produit une orchestration de type BPEL dont les activités sont annotées par le concepteur selon une forme fixée. Les annotations seront utilisées pour la réécriture.

FineOrchestration : Composition $\longrightarrow$ AnnotatedBpelProcess
{ *FineOrchestration (cp) est un processus BPEL construit par le concepteur à partir de CoarseOrchestration (cp) et les interfaces des communautés co pour chacune des capacités ca $\in$ MandatoryCapacities (cp) $\cup$ OptionalCapacities (cp). Le processus BPEL est muni d'annotations qui vont guider la réécriture.* }

A partir des interfaces des différentes communautés définies pour la composition, le concepteur raffine et annote l'orchestration à gros grains issue de la phase de conception. Les annotations caractérisent le rôle de chaque activité dans le processus. Ces rôles sont les suivants :

- Demande d'information : il s'agit seulement d'une consultation (par exemple, demander un devis). Les activités de ce type n'engendre aucune réservation ni aucune acquisition de ressources.
- Acquisition de ressource : il s'agit d'activités dont la fonction est d'acquérir une ressource. Si l'opération associée est bilatérale alors il y a une activité de type acquittement associée pour attendre la réponse.
- Point de synchronisation : il s'agit d'un point dans le processus où il faut déterminer si toutes les activités précédentes se sont exécutées sans erreur. En cas d'échec, l'exécution doit être interrompue.
- Point de sélection : il s'agit de choisir l'exécution d'un service parmi un ou plusieurs services disponibles.

– Zone transactionnelle : ceci correspond à la fourniture de la zone transactionnelle telle que nous l'avons définie dans la section 5.2.3.

Un travail de validation est nécessaire pour étudier le complétude de cet ensemble d'annotations (voir le chapitre 8).

Liste d'options

Pour établir la liste d'options d'une composition (soit cp cette composition) il s'agit, au niveau le plus abstrait, de raisonner sur des nuplets de services. Le type Option formalise des nuplets de longueurs et de types variables et qui sont de la forme :

$\langle$ S_1, ..., S_m, S_{m+1}..., S_{m+n} $\rangle$, avec $\forall$ i = 1..m+n, S_i : Service

où $m = \|$ MandatoryCapacities (cp) $\|$[1] et $0 \leq n \leq \|$ OptionalCapacities (cp) $\|$

Option : $\langle$ s_1 : Service, ..., s_m : Service, s_{m+1} : Service, ..., s_{m+n} : Service $\rangle$
{ *Pour une composition cp, n variant de 0 à* $\|$ *OptionalCapacities (cp)*$\|$. }
ListOfOptions : Composition $\longrightarrow$ [Option]
{ *ListOfOptions (cp) est la liste ordonnée des options pour la composition cp.* }
Propriété : soit R = CompositionRestriction (cp)
ListOfOptions (cp) =
{o $\in$ O, R (o) $\wedge$ $\|$ Map (OptionCapacities (o, cp) $-$ OptionalCapacities (cp),
λo $\bullet$ .TransactionalProperty (ServiceForOption (o, cp, c))) $\| \leq 1$}
OptionCapacities : Option, Composition $\longrightarrow$ {Capacity}
{ *OptionCapacities (o, cp) est l'ensemble des capacités de services référencées dans o définie pour la composition cp.*
Propriétés : MandatoryCapacities (cp) $\subset$ *OptionCapacities (o, cp)*
OptionCapacities (o, cp) $-$ *MandatoryCapacities (cp)* $\subseteq$ *OptionalCapacities (cp)* }
ServiceForOption : Option, Composition, Capacity $\longrightarrow$ Service
{ *ServiceForOption (o, cp, c) est le service compris dans l'option o pour la capacité c dans la composition cp.*
Pré condition : c $\in$ *OptionCapacities (o, cp).* }

Les nuplets sont construits de la manière suivante : les m premiers composants de chaque nuplet sont associés aux m capacités obligatoires. Un premier ensemble, soit O, de nuplets de dimension m est obtenu par le produit cartésien des ensembles de services pris dans chaque capacité obligatoire :

$$O \longleftarrow \prod_{ca \in \text{MandatoryCapacities (cp)}} \text{ServicesForCapacity (cp, ca)}$$

Puis, pour chaque o dans l'ensemble O et pour toutes les valeurs de n variant de 0 à $\|$ OptionalCapacities (cp) $\|$, n services sont ajoutés à la suite de o pour construire des nuplets de dimension m+n, selon la forme ci-dessus.

Ces services sont tels que :

(1) $\|\{$ServiceCapacity (S_1, cp), ServiceCapacity (S_2, cp), ..., ServiceCapacity (S_p, cp)$\}\| = p$,
m $< p \leq$ (m+n) $\wedge$
(2) {ServiceCapacity (S_1, cp), ServiceCapacity (S_2, cp), ..., ServiceCapacity (S_p, cp)} $\subset$
OptionalCapacities (cp)

L'assertion (1) exprime que les capacités des services participants sont distinctes deux à deux, et l'assertion (2) que chaque capacité est défini dans l'une des capacités OptionalCapacities (cp).

[1] $\|$ E $\|$ dénote le cardinal de E.

L'ensemble d'options est ensuite réduit aux options vérifiant la fonction de restriction et dont le nombre de services non atomiques associés aux capacités obligatoires est inférieur au égal à un. Finalement les options de O_{final} sont ordonnées dans une liste. Le tri est effectué selon le score de chaque option obtenu par l'application de la fonction CompositionPreference (cp).

6.2.3 Phase d'exécution

Dans cette phase, il s'agit de tenter de valider une option parmi celles contenues dans la liste d'options obtenue à la phase précédente. En fonction des propriétés transactionnelles de chacun des services d'une option donnée, l'orchestration modélisée par FineOrchestration (cp) est réécrite en une orchestration exécutable. **Réécriture de l'orchestration**

Pour formaliser la réécriture nous utilisons la fonction décrite ci-dessous.

FinalOrchestration : Composition, Option $\longrightarrow$ BpelProcess
{ *FinalOrchestration(cp, o) est le processus BPEL réécrit à partir du processus BPEL annoté FineOrchestration(cp) de la composition cp et qui tient compte de l'option o.* }

Les activités de l'orchestration à réécrire ont été annotées. L'algorithme suivant formalise le processus de réécriture. Il admet en entrée une composition cp dont le processus FineOrchestration(cp) est traité comme un graphe. Dans cette sous-section cet algorithme présente les étapes principales de la réécriture d'une manière succinte en attendant le chapitre dedié à la mise en œuvre, (voir la sous-section 7.2.2 où cet algorithme est décrit de manière détaillée).

```
FinalOrchestration(cp, o) :
Begin Réécriture
   ANN : Annotation
   AC, ACfinal : Activity                                   { activité courante }
   BPR : BPELProcess                                        { le processus BPEL réécrit }
   BPA : BPELProcess                                        { le processus BPEL annoté }
   BPA ⟵ FineOrchestration(cp)
   Initialiser (BPR)
   AC ⟵ première (BPA)                                      { AC est la première activité de BPA. }
   Pour toutes les activités AC qui ∉ ZoneTransactionnelle (BPA)
      Ajouter (AC, BPR) ;
      AC ⟵ suivante (BPA)
      { recopie les activités non réécrites placées avant la zone transactionnelle }
   endBoucle
   Pour toutes les activités AC ∈ ZoneTransactionnelle (BPA)
      Analyser le type d'annotation ANN d'AC ;
         Ajouter (AC, BPR) ;
         Ajouter (PointSynchro, BPR) ;
         AC ⟵ suivante (BPA)
   endBoucle
   Pour toutes les activités AC qui ∉ ZoneTransactionnelle (BPA)
      Ajouter (AC, BPR) ;
      AC ⟵ suivante (BPA) ;
      { recopie les activités non réécrites placées après la zone transactionnelle }
   endBoucle
End Réécriture
```

Exécution des options

L'algorithme ci-dessous fournit les principes d'exécution de la liste des options.

```
LO ⟵ ListOfOptions (cp)
flag ⟵ 0
Begin CompositionExécution
   While LO ≠ ∅ ∧ flag = 0
      o ⟵ premier (LO)                                  { o est la première option de la liste. }
      LO ⟵ − {o}                                        { LO devient LO − {o} }
      if (MoteurBPEL (FinalOrchestration(cp, o)) = OK)
         flag ⟵ 1
      endIf
   ednWhile
   If flag = 1 then
      Composition(cp) réussie
   else
      Composition(cp) échouée
   endIf
End Begin
```

6.3 Conclusion

Dans ce chapitre, nous avons proposé une formalisation des éléments principaux du canevas `Tcows` sous forme fonctionnelle. La phase de conception fournit les données de base de la composition. La phase de sélection prend en charge la sélection des communautés et dans chacune, des services web. Une liste d'options de composition est produite. Chacune des options obtenues représente une exécution potentielle de la composition ; elles respectent toutes d'une part, le niveau de qualité requis et d'autre part la restriction associée à la composition. Les grandes lignes d'un algorithme de réécriture sont données dans la perspective de produire pour chaque option, prise une après l'autre, un processus BPEL exécutable. L'algorithme de réécriture s'appuie sur des annotations portées par le concepteur sur le processus de l'orchestration raffinée. L'exécution de la transaction a réussi si une au moins parmi ces options est exécutée avec succès.

Le chapitre 7 fournit les détails d'implantation du canevas.

CHAPITRE 7

MISE EN ŒUVRE DU CANEVAS TCOWS

Ce chapitre présente le prototype que nous avons développé pour valider l'approche que nous proposons au travers du canevas Tcows. Nous avons décliné la mise en œuvre de ce prototype en deux parties : (i) une qui correspond au modèle de composition de services web Tcows (celle-ci est introduite dans la section 7.1 puis détaillée dans la section 7.2) et (ii) une autre partie qui correspond à la mise en œuvre du modèle de communautés de services web (voir section 7.3). Finalement la section 7.4 conclut en discutant en particulier des points dont il faut tenir compte dans la perspective d'une utilisation du prototype dans le contexte d'une application réelle.

7.1 Architecture logicielle de Tcows

La figure 7.1 montre l'architecture logicielle du canevas de composition, constituée de quatre modules qui communiquent entre eux. Cette architecture doit être mise en correspondance avec les différentes phases proposées dans le canevas (voir la figure 6.1, page 88). Ainsi, le module de composition est utilisé dans la phase de conception, alors que ceux responsables de la constitution des options de composition et de la réécriture du processus BPEL annoté le sont dans la phase d'exécution. Le module de communication avec les communautés offre quant à lui des fonctions utilisées au moment de la conception et d'autres utilisées au moment de l'exécution.

Les communautés de services web et le moteur d'exécution BPEL sont des entités en dehors de l'architecture mais avec lesquelles Tcows interagit.

Dans l'état actuel d'avancement du développement, en plus de la mise en œuvre du modèle de communautés, seuls les modules de communication[1], de constitution des options et de réécriture ont été réalisés. Dans la suite, nous introduisons chacun de ces modules. Ceux réalisés sont détaillés plus loin.

[1] A l'exception de la fonction qui correspond au composant de Recherche

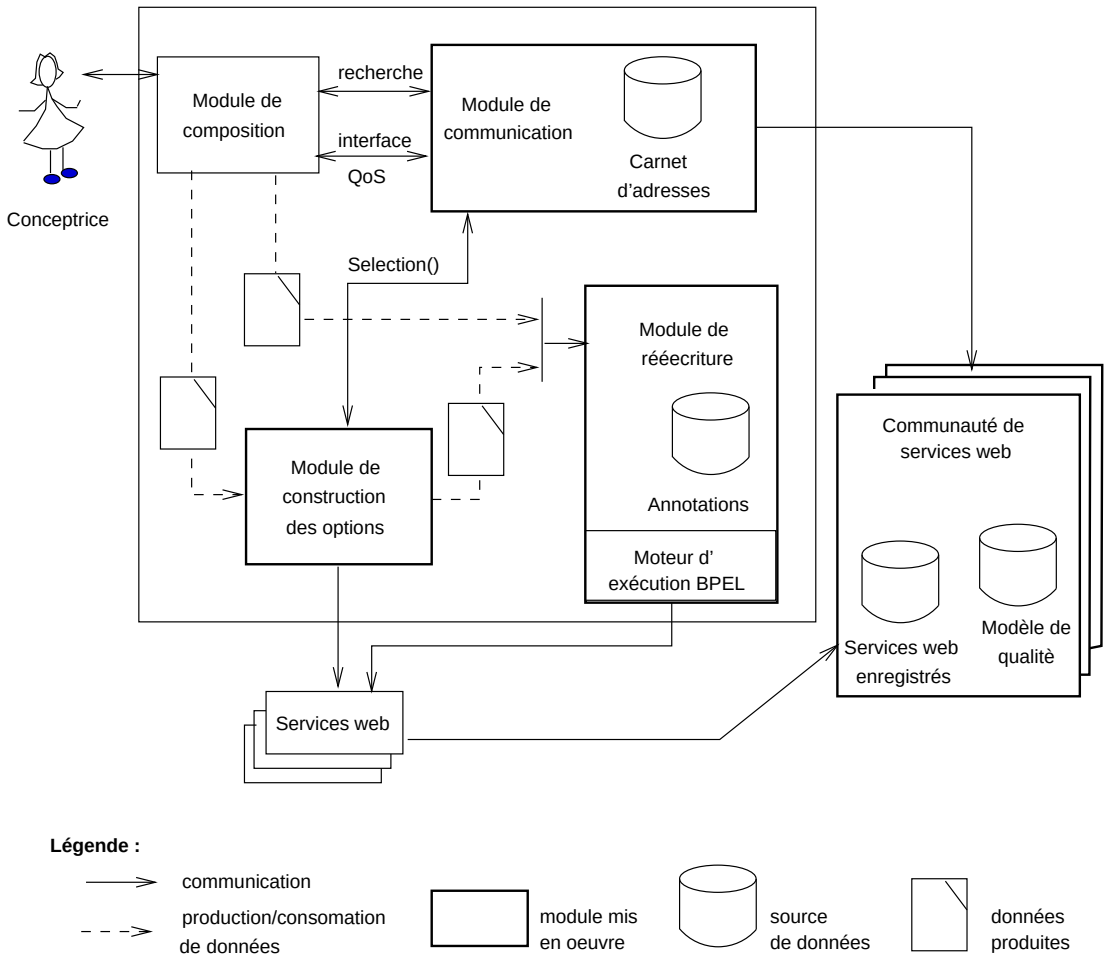


FIG. 7.1 – Architecture logicielle de Tcows

Le module de composition Ce module est utilisé par le concepteur de la composition pour développer les activités correspondant à la phase de conception de la composition (voir la figure 6.1 page 88). Par le biais d'interactions avec ce module, le concepteur fixe les données de la composition que nous avons détaillée dans le chapitre 6, à savoir :

- Les fonctions de restriction et de préférence,
- La structure des données nécessaires à la composition. Ces données seront instanciées lors de l'exécution de la composition.

Ce module interagit avec le module de communication afin de rechercher les communautés les mieux adaptées à la composition en cours, ainsi que leur interface et leur modèle de qualité. Ce module permet aussi au concepteur de raffiner le diagramme d'orchestration à gros grain généré au cours de la phase de conception (voir la figure 5.1, page 71), par la prise en compte des opérations de l'interface abstraite des communautés. Pour ce faire, le concepteur s'appuie sur un ensemble d'annotations prédéfinies qui qualifient chaque activité du diagramme. Cette procédure produit comme résultat un processus BPEL raffiné et annoté qui sert de base au module de réécriture.

Le module de communication Ce module centralise les communications avec les communautés. Les fonctions qu'il offre concernent :

- La recherche de communautés pour une capacité donnée,
- Le télé-chargement de l'interface abstraite ou du modèle de qualité d'une communauté donnée,
- La soumission, à une communauté, de la sélection de services.

Les deux premières de ces fonctions sont utilisées lors d'interactions avec le module de composition et sont respectivement implantées dans les composants Recherche et Chargement décrits dans la figure 7.2. La troisième fonction est utilisée lors d'interactions avec le module de construction des options (voir le composant Soumission sélection).

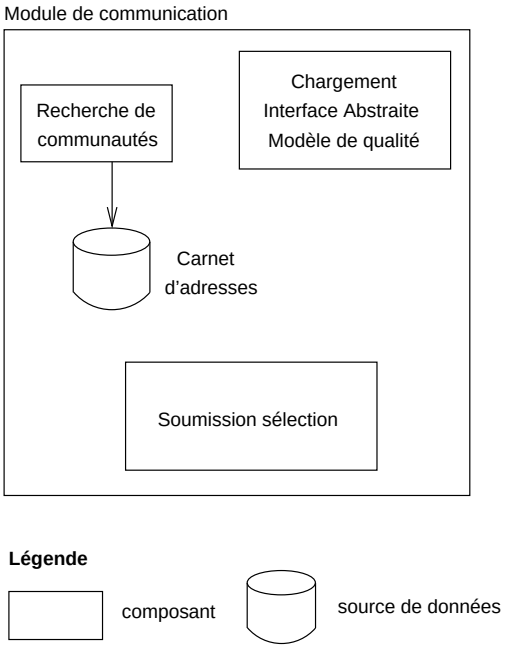


Fig. 7.2 – Module de communication

Le module de construction des options Ce module construit la liste des différentes options de composition. Chaque option décrit une composition potentielle. Dans cette liste, toutes les options vérifient la fonction de restriction et le niveau de qualité requis. Enfin, à chaque option est associé un score traduisant son degré de satisfaction à la préférence et au niveau de qualité. Rappelons que dans chaque option, il n'y a pas plus d'un service atomique parmi ceux correspondant aux capacités obligatoires car sinon, il serait impossible de garantir la propriété d'atomicité lors de l'exécution de l'option. Le tableau 5.3 du chapitre 5 (voir page 78) illustre sur un exemple le principe de fonctionnement de ce module.

Le module de réécriture Ce module est chargé, pour chaque option à valider, de produire un processus exécutable par réécriture du processus BPEL en considérant les propriétés transactionnelles des services web qui composent l'option. Il considère chaque

option prise l'une après l'autre ; pour chacune, il réalise la réécriture du processus BPEL raffiné et annoté dans la perspective de produire un processus exécutable ensuite soumis au moteur d'exécution BPEL.

7.2 Détails de la mise en œuvre

Dans cette section nous décrivons les détails de l'implantation des modules mis en œuvre.

7.2.1 Module de communication

Comme indiqué dans la figure 7.2 ce module, centralisant la gestion des interactions avec les communautés, s'appuie sur trois composants. Nous n'avons pas développé le composant Recherche de communautés chargé de trouver une communauté répondant aux besoins associés à une capacité donnée. Il s'agit en effet d'un sujet en dehors de la portée de cette thèse. En outre, il existe des nombreux outils spécialisés pour réaliser cette tâche, comme des moteurs de recherche par exemple.

Les fonctions de télé-chargement mises en œuvre dans le composant Chargement permettent à un utilisateur d'obtenir l'interface abstraite d'une communauté donnée aussi bien que son modèle de qualité. L'interface est fournie sous forme d'une description WSDL et le modèle de qualité sous forme d'un document XML (voir l'annexe B).

Le composant Soumission sélection, suite à une requête de sélection visant une communauté et reçue du module de construction des options, transmet à la communauté les paramètres de la sélection et reçoit en retour les services satisfaisant les critères de la sélection donnés en termes de niveau de qualité.

7.2.2 Module de construction des options

Ce module construit la liste des différentes options de composition. Nous décrivons ici l'algorithme qui en constitue le cœur. Cet algorithme s'articule en deux phases : la première phase vise à produire les différentes combinaisons de capacités spécifiées dans la composition, ce que nous appelons types d'options. La seconde phase vise à instancier ces types d'options en considérant les services sélectionnés par les communautés associées à chaque capacité de la composition.

Phase de construction des types d'options A partir de la spécification des capacités obligatoires et facultatives, il s'agit de construire les types d'option, chacun défini par un sous-ensemble des capacités spécifiées dans la composition. Toutes les capacités obligatoires de la composition font partie de ce sous-ensemble. Le type d'option qui ne contient que les capacités obligatoires est qualifié de minimal. Les autres types d'options sont construits à partir de ce type minimal auquel ont été ajoutées toutes les combinaisons possibles de capacités facultatives. Par exemple, considérons les capacités de notre scénario exemple : soit F pour la capacité Vol, H pour Hôtel, C pour Voiture, V pour Vélo et M pour Météo. Le type d'option minimal dans ce cas est {F,H}. A partir de ce type nous déduisons le types suivants :

{ *Le type minimal est complété par toutes les combinaisons à une capacité parmi les N capacités facultatives :* }
{F,H,C}
{F,H,V}
{F,H,M}
{ *Le type minimal est complété par toutes les combinaisons à deux capacités parmi les N capacités facultatives :* }
{F,H,C,V}
{F,H,C,M}
{F,H,V,M}
{ *Le type minimal est complété par toutes les combinaisons à trois capacités parmi les N capacités facultatives :* }
{F,H,C,V,M}

Capacités facultatives	0	1	2	...	i	...	N	total
Nombre de types	0	2	$\binom{N}{2}$	$\binom{N}{...}$	$\binom{N}{i}$	$\binom{N}{..}$	$\binom{N}{N}$	$\sum_{i=0}^{N}\binom{N}{i}$

TAB. 7.1 – Nombre de types d'options

L'algorithme que nous avons réalisé pour le calcul des types d'options s'appuie sur N pas d'itération où N est le nombre de capacités facultatives. Le tableau 7.1[2] montre le nombre de types d'options générés à chaque pas d'itération, en plus du type minimal. Par exemple, à l'étape i (c'est-à-dire avec i capacités facultatives choisies parmi les N) le nombre de types générés est $\binom{N}{i}$. Le nombre de types d'options finalement généré est $1 + \sum_{i=0}^{N}\binom{N}{i}$.

Phase de construction des options Posons $\mathsf{X}=\{\mathsf{C}_1, ... \mathsf{C}_n\}$ l'un de ces types. Soit $\mathsf{S}(\mathsf{C}_i)$ l'ensemble des services sélectionnés par la communauté associée à la capacité C_i ($\mathsf{C}_i \in \mathsf{X}$). Le produit cartésien : $\prod_{i=1}^{n} \mathsf{S}(\mathsf{C}_i)$ contient toutes les options potentielles correspondant au type X.

La deuxième étape de cet algorithme consiste à construire toutes les options potentielles issues des produits cartésiens calculés de manière similaire pour chacun des types d'options générés. La liste des options ainsi obtenue est ensuite réduite aux options qui vérifient la fonction de restriction. Les options qui contiennent plus d'un service dont la propriété transactionnelle est non-atomique sont supprimées de la liste. Cette liste est finalement triées selon le score associé à chaque option.

Algorithme L'algorithme proposé ici formalise le processus décrit ci-dessus (les types et les opérations associées sont définis dans le chapitre 6). Le principe de l'algorithme est de parcourir l'ensemble des parties de l'ensemble des capacités facultatives. Chaque partie est codée par un vecteur de bits 0 ou 1 : si la capacité de rang i dans la liste représentant l'ensemble des capacités est présente dans la combinaison alors la valeur indexée par i dans le vecteur vaut 1 sinon elle vaut 0. Une valeur du vecteur est aussi la représentation binaire d'un entier compris entre 0 et 2^{nbOC} (où nbOC est le nombre de capacités facultatives).

[2] L'expression $\binom{N}{p}$ dénote le nombre de combinaisons à p éléments dans un ensemble de N éléments.

En itérant sur ces entiers, on parcourt toutes les combinaisons à p capacités prises parmi nbOC capacités.

Nous introduisons tout d'abord la fonction Combinaison :

```
Combinaison : [Capacity], Vecteur de bits ⟶ [Option]
   { Combinaison (l, c) est la liste des options issues du produit cartésien des ensembles de services
   sélectionnés pour les capacités de la liste l de la combinaison décrite par c. }
Combinaison (LC, V) :
   listC : [Capacity]
   listC ⟵ [ ]
   pour i = 1..longueur(LC)
      si V[i] = 1 alors
         listC ⟵ listC + LC[i]
   return listC
End Combinaison
```

L'algorithme est finalement :

```
ListOfOptions(cp) :
{ ListOfOptions(cp) est la liste des options potentielles pour la composition cp. }
   Res : [Option]                                          { Le résultat : une liste d'options. }
   OptMin : [Option]
      { La liste des options construites à partir des services sélectionnés dans les capacités obliga-
      toires. }
   Res ⟵ [ ]; OptMin ⟵ [ ]
   MC, OC : [Capacity]
      { MC est la liste des capacités obligatoires, OC est celle des capacités facultatives.. }
   nbMC, nbOC : integer
   nbMC ⟵ ‖ MC ‖
   nbOC ⟵ ‖ OC ‖

   { Phase de construction des options du type minimal. }
   MC ⟵ MandatoryCapacities (cp)
   OptMin ⟵ ⟵ ∏_{i=1}^{nbMC} ServicesForCapacity (cp, MC[i])

   pour i = 1..nbOC-1
      C[i] ⟵ 0
   C[nbOC] ⟵ 1
   pour i = 1..2^nbOC - 1
      Res ⟵ Res ∪ Combinaison (OC, C)
      PlusUnBinaire (C)                          { Ajoute un à l'entier binaire représenté dans C. }
   Res ⟵ (OptMin X Res) ∪ OptMin
   return Res
End ListOfOptions
```

La figure B.3, dans l'annexe B, page 128, montre une partie de la liste initiale d'options générée par cet algorithme.

7.2.3 Module de réécriture

Ce module est chargé de produire pour chaque option un processus BPEL exécutable qui tient compte des propriétés transactionnelles des services qui participent à l'option.

Pour cela, il s'agit de réécrire le processus BPEL annoté par le concepteur de manière à remplacer les actions de communication (exprimées en terme de l'interface abstraite de la communauté) par des actions de communication en termes de l'interface du service participant. Cette réécriture est guidée par les annotations effectuées par le concepteur.

Dans un premier temps nous introduisons un ensemble prédéfini d'annotations. A terme, il est possible d'étudier un langage d'annotations permettant de générer de nouvelles annotations et par suite de rendre extensible le processus de réécriture [CFRC05].

Les annotations considérées actuellement par le processus de réécriture permettent de caractériser les activités du processus selon leur rôle :

- Identification de la zone transactionnelle,
- Demande d'information ou consultation,
- Réservation de ressources,
- Validation d'acquisition de ressources,
- Annulation,
- Point de synchronisation,
- Point de sélection,
- Opération interne.

Dans la section 6.2.3, page 94 nous avons donné les étapes principales de l'algorithme de réécriture que nous présentons ci-dessous de manière détaillée. Rappelons qu'il s'agit de mettre en œuvre la fonction FinalOrchestration(cp, o) (voir page 6.2.3).

```
FinalOrchestration(cp, o) :
{ FinalOrchestration(cp, o) est le processus exécutable réécrit à partir du processus annoté de la
composition cp, en fonction des propriétés des services constituant l'option o. }
   ANN : Annotation
   RS : Resource
   AC, ACfinal : Activity                                          { activité courante }
   CAC : Capacity                                   { capacité dans laquelle est définite AC }
   BPR : BPELProcess                                          { le processus BPEL réécrit }
   BPA : BPELProcess                                          { le processus BPEL annoté }
   BPA ⟵ FineOrchestration(cp)
   nbAC ⟵ nombre d'activités dans la ZoneTransactionnelle
   nbACav ⟵ nombre d'activités avant de rentrer à la ZoneTransactionnelle
   nbACap ⟵ nombre d'activités après de sortir de la ZoneTransactionnelle
   Flag ⟵ false
   Initialiser (BPR)
   AC ⟵ première (BPA)                              { AC est la première activité de BPA. }
   While nbACav ≠ 0 and AC ∉ ZoneTransactionnelle (BPA)
      Ajouter (AC, BPR) ;
      nbACav ⟵ nbACav - 1 ;
      AC ⟵ suivante (BPA)
      { recopie les activités non réécrites placées avant la zone transactionnelle }
   endWhile
   While nbAC ≠ 0 and AC ∈ ZoneTransactionnelle (BPA)
      ANN ⟵ GetAnnotation (AC)
      Switch ANN
         ANN = DemandeInformation then
            Copier (AC, BPR) ;
            Ajouter (PointSynchro, BPR) ;
```

```
            nbAC ⟵ nbAC - 1
        ANN = PointSynchro then
            Copier (PointSynchro, BPR) ;
            nbAC ⟵ nbAC - 1
        ANN = OperationInterne
            Copier (OperationInterne, BPR) ;
            { s'il y a un PointSyncho avant, la copier du coté "ok" }
            nbAC ⟵ nbAC - 1
        ANN = AcquisitionRessource then
            S ⟵ ServiceForOption (o, cp, CAC) ;
            TP ⟵ TransactionalProperty (S) ;
            nbAC ⟵ nbAC - 1
            Switch TP
                TP = atomique
                    Ajouter (Reserve (S, RS), BPR) ;
                    Ajouter (PointSynchro, BPR)
                TP = quasi-atomique
                    Ajouter (Validate (S, RS), BPR) ;
                    Ajouter (PointSynchro, BPR)
                TP = non-atomique
                    ACfinal ⟵ AC ;
                    { cas particulier lorsque AC est la dernière }
                    Flag ⟵ vrai
            endSwitch (TP)
    endSwitch (ANN)
    AC ⟵ ActSuivante(BPA)
endWhile
If Flag
    Du coté "non ok" du point de synchronisation :
        Ajouter (Finalisation, BPR)
    Du coté "ok" du point de synchronisation :
        Ajouter (Validate (S, RS), BPR) ;
        Ajouter (PointSynchro, BPR)
            Du coté "non ok" du point de synchronisation :
                Fusionner les points de synchronisation créés précédemment
                { pour tous les services quasi-atomiques dans la zone transactionnelles : }
                Ajouter (Compensate (S, RS), BPR) ;
                { pour tous les services atomiques dans la zone transactionnelles : }
                Ajouter (Annulation (S, RS), BPR)
            Du coté " ok"
                Fin de la zone transactionnelle
                { continuer à copier les activités placées après la zone transactionnelle }
endIf
While nbACap ≠ 0 and AC ∉ ZoneTransactionnelle (BPA)
    Ajouter (AC, BPR) ;
    nbACap ⟵ nbACap -1 ;
    AC ⟵ suivante (BPA) ;
    { recopie les activités non réécrites placées après la zone transactionnelle du coté "ok" du
    dernier PointSynchro }
endWhile
```

Une fois réécrit, le processus exécutable produit est soumis au moteur d'exécution BPEL. Si l'exécution s'achève avec succès la transaction a réussi. Dans le cas d'un échec, l'option

suivante dans la liste est considérée. La transaction a échoué si aucune option n'a pu s'exécuter avec succès, voir l'algorithme d'exécution des options de la page 94.

7.3 Mise en œuvre du modèle de communauté

Comme nous l'avons annoncé dans le chapitre 4, les communautés exposent deux facettes, la première à destination de son administrateur et des fournisseurs de services, la seconde à destination des applications clientes (voir la figure 7.3). Chacun des modules associés utilisent un ensemble d'opérations internes qui permettent en particulier l'accès et la mise à jour des informations de la communauté stockées dans une base de données.

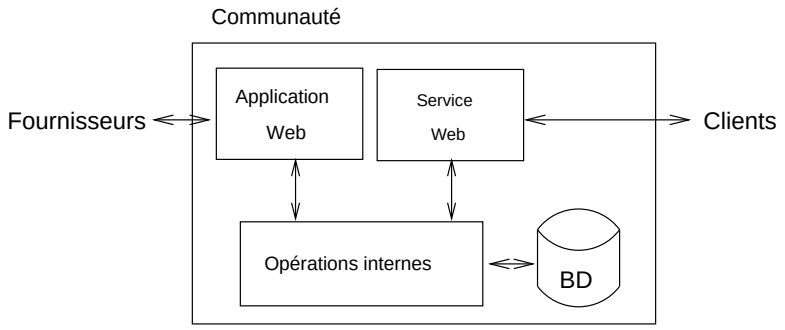


Fig. 7.3 – Vue globale d'une communauté

Ces deux facettes sont :

- Une application web offrant toutes les opérations destinées aux fournisseurs (voir le chapitre 6). L'application permet de télé-charger le modèle de qualité de la communauté et l'interface abstraite de la communauté. En ce qui concerne son administration, les fonctions offertes permettent de gérer d'une part le modèle de qualité et d'autre part l'interface abstraite.
- Un service web destiné aux clients de la communauté. Ce service rend disponible l'opération Select décrite dans le chapitre 4, permettant la sélection de services parmi ceux enregistrés dans la communauté.

7.3.1 Modèle de données

La figure 7.4 montre le diagramme UML qui modélise les données gérées par la communauté. Ce modèle est issu de la formalisation proposée dans le chapitre 6. Une communauté est décrite par un ensemble de services enregistrés chacun associé à un modèle de qualité défini par un ensemble de critères de qualité. Le modèle de qualité de la communauté est défini de même. Les données gérées par la communauté sont stockées dans une base de données relationnelles.

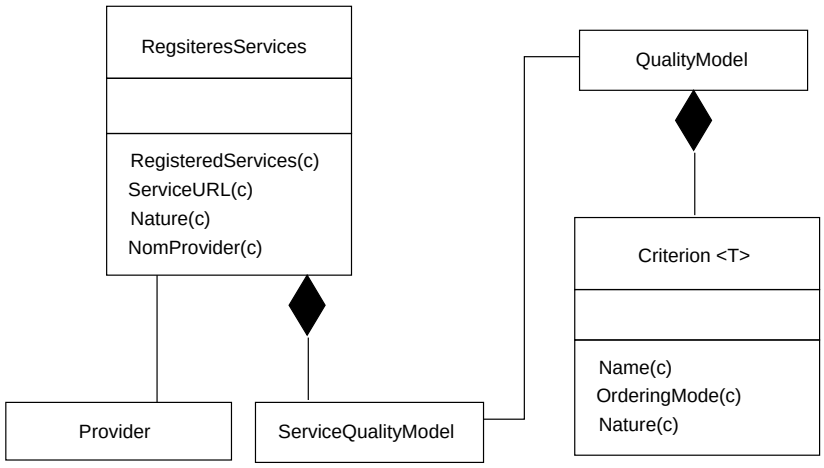


Fig. 7.4 – Modélisation des informations d'une communauté

7.3.2 L'application web

Les tâches décrites ci-dessous sont accessibles via l'interface donnée dans la figure A.1 de l'annexe A.

Les tâches liées aux fournisseurs Les fournisseurs peuvent inscrire (opération Register) et dés-inscrire (opération Quit) leurs services auprès de la communauté, aussi bien que consulter l'interface abstraite de la communauté et son modèle de qualité.

Le modèle de qualité et l'interface abstraite peuvent être télé-chargés sous la forme de documents XML et WSDL respectivement.

Les tâches liées à l'administrateur Pour permettre l'administration d'une communauté de services, il est nécessaire d'offrir les opérations pour :

- Référencer des fournisseurs qui alors pourront enregistrer les services web qu'ils proposent.
- Changer/mettre à jour le modèle de qualité de la communauté.
- Changer/mettre à jour l'interface abstraite de la communauté.
- Surveiller les niveaux de qualité des services participants (pour vérifier par exemple l'adéquation de ce qui était annoncé avec la réalité).

Ces opérations ne sont pas aujourd'hui développées au sein du prototype.

7.3.3 Le service web

Ce service web rend disponible l'opération de sélection de services selon un ensemble de critères de qualité (voir la section 4.2.3, page 60) :

$$\mathsf{Select} : \mathsf{Community}, \mathsf{integer}, \{\mathsf{RequiredQuality}\} \longrightarrow [\mathsf{RegisteredService}]$$

Nous rappelons ci-après les types utilisés par cette fonction :

type Restriction : (type $\longrightarrow$ boolean)
type Weight : real in [0..1]
RequiredQuality : $\langle$ c : Criterion$\langle$T$\rangle$, p : Weight, r : Restriction $\rangle$
{ *Soit nq donné dans RequiredQuality. nq décrit un niveau de qualité relativement au critère nq.c. La restriction nq.r exprime une contrainte requise sur ce critère et nq.p est le poids donné à ce critère.* }

La mise en œuvre de la fonction Select d'ordre supérieur pose problème : en Java par exemple il n'y a pas de solution directe (il est nécessaire de lister *a priori* les fonctions qui seront passées en paramètre et d'utiliser la notion d'interface).

Pour proposer une solution générale respectant le principe d'indépendance des langages des services web, il est donc nécessaire d'encoder en XML l'expression de la fonction de restriction.

La version actuelle du prototype est limitée à des restrictions de valeurs à des intervalles donnés chacun par leurs bornes inférieure et supérieure.

7.4 Conclusion

Le prototype que nous décrit dans ce chapitre a été développé en utilisant les technologies web standards. La facette application web des communautés s'appuie sur le canevas ZK[3].

La mise en œuvre du prototype a soulevé les questions ci-dessous :

- Quelle stratégie définir afin de déterminer combien d'options il est nécessaire de calculer ? Dans la version actuelle du prototype, les options sont construites de manière exhaustive ce qui, dans le cas d'une application réelle, conduirait à une explosion combinatoire. Il est donc nécessaire d'effectuer des expérimentations afin de mesurer le coût véritable induit, puis si nécessaire d'étudier des stratégies afin de diminuer ce coût.
- Les fournisseurs accepteront-ils d'exposer le comportement transactionnel de leurs services web ?
- Des mesures permettant d'évaluer le sur-coût occasionné par l'exécution de la fonction de sélection s'avèrent nécessaires. Il est important de revoir le modèle de qualité de la communauté dans le but d'introduire le traitement multicritère.

Le prototype doit être complété par le développement du module de composition dédié à la phase de conception proposée dans le canevas `Tcows`. Nous pensons que ce module doit s'appuyer sur un outil existant pour le développement d'applications.

Dans le chapitre 8 nous discutons de conclusions et de perspectives générales liées à l'approche elle-même.

[3]http ://www.zkoss.org/

Troisième partie

Bilan et perspectives

CHAPITRE 8

CONCLUSION ET PERSPECTIVES

Tout au long de ce document nous avons présenté le travail dont est issu le canevas `Tcows`. Nous présentons maintenant un bilan et nous dressons plusieurs perspectives, envisageables tant sur le plan théorique que sur le plan pratique.

8.1 Conclusion

Cette thèse a présenté un canevas appelé `Tcows`[1] qui a pour but la conception et l'exécution de compositions de services web qui tiennent compte des propriétés transactionnelles hétérogènes des composants.

Notre approche couvre plusieurs domaines liés à la composition de services. Tout d'abord, nous avons défini le concept de communauté de service web, un référentiel pour l'enregistrement de services web, et la sélection sur des propriétés extra fonctionnelles (par exemple un niveau de qualité). S'appuyant sur ce concept, le modèle de composition de `Tcows` s'exprime en termes de capacités de services et non pas en termes de services web spécifiques. Cette caractéristique rend les compositions plus adaptables, dans le sens qu'elles ne sont pas limitées être composées toujours des mêmes services.

Le modèle que nous proposons permet au concepteur de spécifier les capacités obligatoires et celles facultatives. Cette caractéristique permet de limiter la portée de la transaction au minimum requis, ce qui assouplit la propriété d'atomicité, et ce qui a pour conséquence d'augmenter la probabilité de réussite de la composition.

Le canevas `Tcows` permet aussi, la paramétrisation de la composition à travers des contraintes et des préférences fixées par le concepteur et qui guident la sélection des services au moment de l'exécution de la composition. Ensuite, la liste des options de composition peut être construite de telle sorte que si la validation d'une de ces options échoue, il est possible de tenter la validation d'une autre option, ainsi de suite jusqu'à épuisement de la liste. A nouveau, ceci augmente la probabilité de réussite de la composition.

[1] Transactional Composition Of Web Services

D'autre part, comme chaque service expose sa propriété d'atomicité, au moment de l'exécution d'une option, l'ensemble des propriétés hétérogènes des composants seront prises en compte de manière à ne tenter cette option que si son exécution peut être conduite avec la sémantique du tout ou rien.

Donc, si nous revenons sur les questions soulevées par la discussion présentée dans la section 1.2, nous constatons le canevas `Tcows`, développé dans cette thèse, y répond. Cependant, nous avons laissé de côté des pistes qui ouvrent des perspectives au travail présenté ici. Nous les détaillons dans la section suivante.

8.2 Perspectives

Dans le chapitre 6, la figure 6.1 donnée dans la page 88 montre le processus global allant de la conception d'une composition transactionnelle à base de services web jusqu'à son exécution tel que nous l'avons abordé dans la thèse présentée ici. La figure 8.1, reprenant le même processus, met en avant les points qui ouvrent des perspectives.

La présentation de ces perspectives est organisée selon l'étape du processus à laquelle elle se rapporte : conception de la composition, sélection des participants potentiels puis exécution des différentes options obtenues pour la composition.

8.2.1 Phase de conception

Pour la phase de conception il est nécessaire disposer d'un outil de modélisation qui permette l'expression des besoins en termes des capacités de services obligatoires et celles facultatives, en termes des données d'entrée et de sortie, et en termes des fonctions de restriction et de préférence et enfin qui permette de décrire un flot de contrôle dans un langage quelconque. Dans l'état actuel les informations issues de cette phase sont élaborées de manière *ad-hoc* :

- Le modèle d'orchestration à gros grains est exprimé sous forme d'un processus BPEL. Les capacités sont représentées par les partenaires du processus et les données en entrée et en sortie sont les données du processus.
- Les fonctions de restriction et de préférence sont décrites en XQuery [W3C], à part du processus.
- Pour chaque capacité (c'est-à-dire pour chaque partenaire du processus), une description XML indique son caractère facultatif ou obligatoire.

Des outils existent aujourd'hui pour le développement, la validation et l'exécution de processus BPEL. Il existe aussi des moteurs plus ou moins sophistiqués pour l'évaluation de requêtes XQuery. La perspective envisagée ici fait référence à un travail d'intégration de ces outils.

8.2.2 Phase de sélection

Nous avons considéré une zone d'atomicité unique et connexe. Lever cette restriction implique d'étudier l'emboîtement des zones transactionnelles (comme dans le cas des Sagas

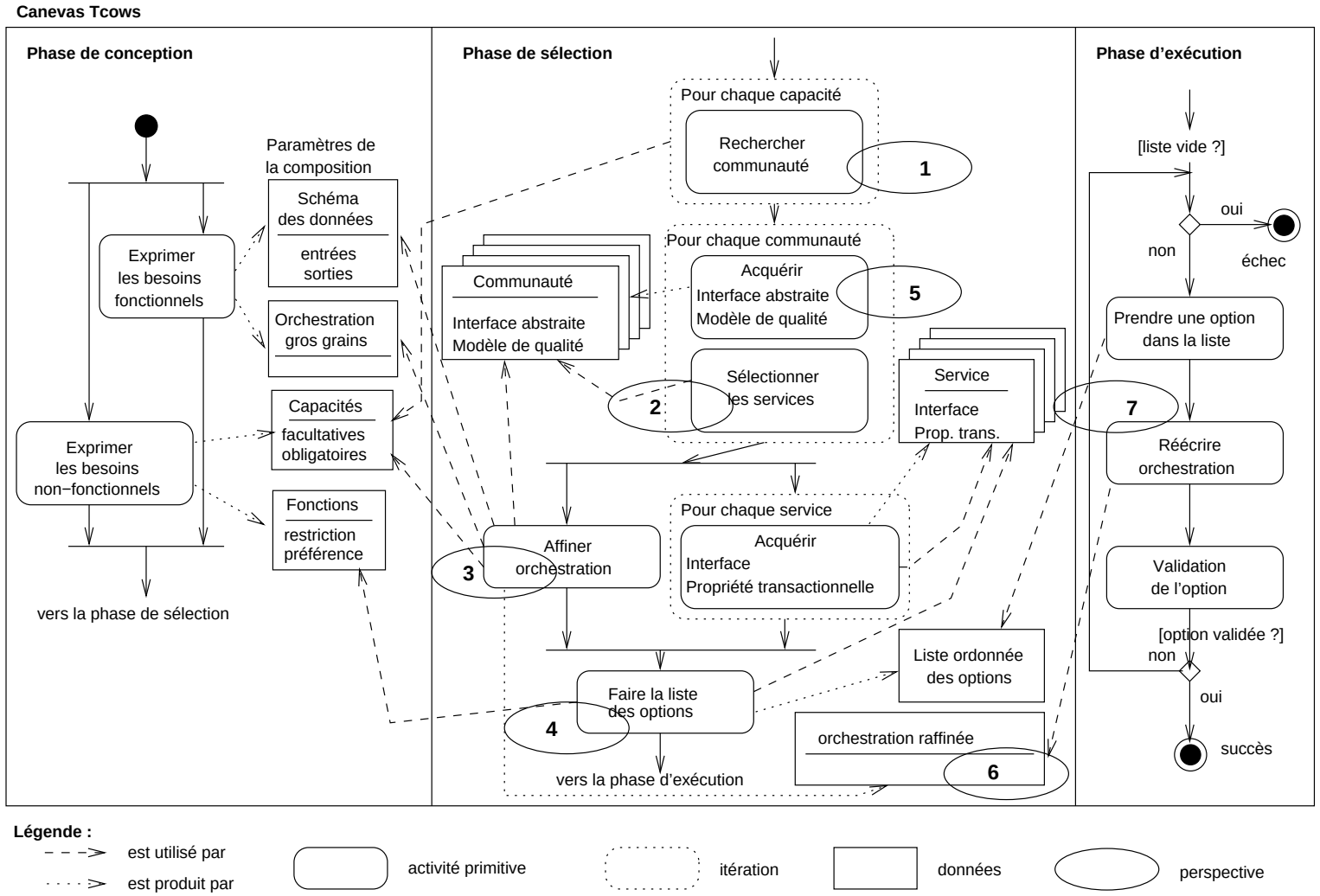


FIG. 8.1 – Schéma général du Canevas de composition et les différentes perspectives

par exemple, voir [GMS87]) puis les relations des zones transactionnelles les unes par rapport aux autres (marque numéro 3 dans la figure 8.1).

L'évaluation des valeurs pour les critères du modèle de qualité est réalisée par la communauté correspondante, au moment de la sélection des services. Entre le moment de cette évaluation et celui du tri de la liste des options les valeurs peuvent être modifiées. Une optimisation possible consiste à retarder le plus possible l'évaluation des valeurs des critères afin de construire une liste dont l'ordre est le plus pertinent (marque numéro 4 dans la figure 8.1).

Le nombre de communautés à contacter et le nombre de services à sélectionner dans chaque communauté sont potentiellement importants, il s'en suit une possible explosion combinatoire pour la construction des options. Il est nécessaire d'étudier un mécanisme pour fixer de manière optimale le nombre de services à sélectionner dans chaque communauté (marque numéro 2 dans la figure 8.1). Une direction à étudier est sans doute celle d'une étude expérimentale destinée à produire des résultats statistiques sur lesquels pourrait s'appuyer le choix de ces nombres.

Dans notre approche la composition de services s'appuie sur la notion de capacités, chacune se traduisant, dans la phase de sélection, par une communauté. Mais il peut arriver, qu'au moment de la recherche de communauté, aucune ne soit trouvée. Il faut considérer dans ce cas un service web à la place de la communauté concernée (marque numéro 5 dans la figure 8.1).

Cette phase construit un processus BPEL annoté, basé sur les interfaces abstraites des communautés. Pour se faire, nous avons défini un ensemble d'annotations qui est pour l'instant, très limité. La complétude de ce ensemble doit être étudiée (marque numéro 6 dans la figure 8.1).

8.2.3 Phase d'exécution

La réécriture effectuée dans cette phase est réalisée par le biais d'un algorithme dont il faut étudier la complexité, ainsi que les possibilités d'optimisation du processus BPEL généré (marque numéro 7 dans la figure 8.1).

8.2.4 Communautés de services web

Il faut disposer d'un outil pour aider à la recherche des communautés spécifiques à une composition donnée. Ainsi la description des communautés via des ontologies, par exemple, pourra faciliter cette recherche (marque 1 dans la figure 8.1).

D'autre part, le modèle de qualité tel que nous le proposons doit être enrichi pour couvrir un éventail plus large de critères de qualité ou bien pour permettre de réaliser des conversions entre des valeurs d'un même critère exprimées dans différentes unités.

Dans l'approche que nous avons proposée, si certains services offrent un modèle de qualité qui n'inclut pas tous les critères utilisés dans une sélection, ils ne sont pas considérés dans cette sélection. Cette contrainte doit être relâchée, surtout lorsqu'aucun service enregistré dans la communauté n'est muni d'un modèle de qualité correspondant à tous les critères utilisés dans la sélection. De façon plus générale, nous avons adopté une vision simplifiée

du choix multicritère en nous ramenant, par l'introduction de pondération, à un choix monocritère. Une perspective consiste à intégrer des résultats obtenus dans ce domaine (voir par exemple [BC06]). Enfin, un outil doit être mis à la disposition des administrateurs de communautés afin qu'ils puissent modifier le modèle de qualité de la communauté, répondant ainsi aux évolutions des besoins des clients et des fournisseurs.

Par ailleurs, nous avons pris pour hypothèse que l'adaptation entre l'interface abstraite de la communauté et celles fournies par les services de la communauté était une tâche à la charge des fournisseurs de services. Une interface peut être décrite selon la dimension structurelle, comportementale ou encore extra fonctionnelle des services. Ainsi l'adaptation doit être étudiée selon chacune de ces dimensions (voir par exemple [ABF07] pour l'adaptation comportementale). Un travail en cours vise à proposer un outil dans la perspective d'automatiser ou, au moins, d'assister le fournisseur lors de la définition des règles de mise en correspondance entre deux interfaces selon leur dimension structurelle. A terme, ce jeu de règles sera utilisé par le composant OSC pour la réécriture, en termes de l'interface fournie du service sélectionné, des appels exprimés dans le code du client en terme de l'interface abstraite de la communauté.

8.2.5 Validation expérimentale

Nous avons développé un prototype qui valide notre proposition de manière expérimentale. Mais il manque sa validation par sa confrontation à une véritable application industrielle.

D'autre part, notre modèle prône pour la non-négociation au préalable entre les partenaires d'une composition. Il reste à prouver, ici aussi dans le cadre d'une vraie application que les services "acceptent" d'exposer leur propriété transactionnelle.

Quatrième partie

Annexes

ANNEXE A

DESCRIPTION WSDL DE L'INTERFACE ABSTRAITE DE LA COMMUNAUTÉ VOL

La figure A.1 montre, d'un manière graphique, l'interface abstraite du fichier WSDL de la communauté Vol

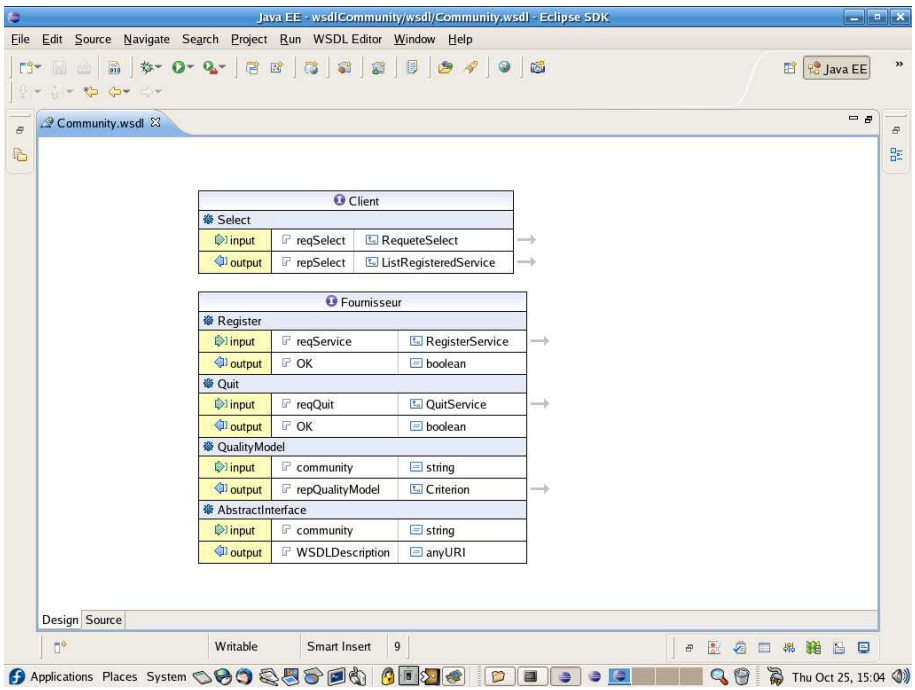


Fig. A.1 – L'interface abstraite WSDL de la communauté Vol

Sur cette figure nous pouvons voir que le PortType Client expose l'opération Select, destinée aux clients qu'utilisent la communuaté du coté service web. Le PortType Fournisseur expose les opérations destinées particuliérement aux fournisseurs et qu'ont été implémenté du coté Application web. Cette disposition facilite la consultation de la part des utilisateurs et montre, dans un format convenable, les type d'opérations que la communauté expose.

L'outil (plugin) proposé par Eclipse permet de traduire cette interface pour produire un fichier WSDL contenant la description des opérations montrées dans la figure A.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://lig.mrim.fr/Vol/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Vol"
  targetNamespace="http://lig.mrim.fr/Vol/">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://lig.mrim.fr/Vol/">

      <xsd:complexType name="RegisterService">
        <xsd:sequence>
          <xsd:element name="com" type="xsd:string"></xsd:element>
          <xsd:element name="nomService" type="xsd:string"></xsd:element>
          <xsd:element name="urlService" type="xsd:string"></xsd:element>
          <xsd:element name="urlAdpServ" type="xsd:string"></xsd:element>
          <xsd:element name="nomFournisseur" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="QuitService">
        <xsd:sequence>
          <xsd:element name="comm" type="xsd:string"></xsd:element>
          <xsd:element name="urlServReg" type="xsd:string"></xsd:element>
          <xsd:element name="nomFournisseur" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Criterion">
        <xsd:sequence>
          <xsd:element name="nonCriterion" type="xsd:string"></xsd:element>
          <xsd:element name="orderMode" type="xsd:string"></xsd:element>
          <xsd:element name="nature" type="tns:NatureCriterion"></xsd:element>
          <xsd:element name="typeDonneCriterion" type="xsd:string"></xsd:element>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="NatureCriterion">
        <xsd:sequence>
          <xsd:element name="semantique" type="xsd:string"></xsd:element>
          <xsd:element name="unite" type="xsd:string"></xsd:element>
        </xsd:sequence>
```

```
    </xsd:complexType>


    <xsd:complexType name="Restriction">
      <xsd:sequence>
        <xsd:element name="term1" type="xsd:string"></xsd:element>
        <xsd:element name="operator" type="xsd:string"></xsd:element>
        <xsd:element name="term2" type="xsd:string"></xsd:element>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="RequiredQuality">
      <xsd:sequence>
        <xsd:element name="criterion" type="tns:Criterion"></xsd:element>
        <xsd:element name="poid" type="xsd:float"></xsd:element>
        <xsd:element name="restriction" type="tns:Restriction"></xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="RequeteSelect">
      <xsd:sequence>
        <xsd:element name="nomComm" type="xsd:string"></xsd:element>
        <xsd:element name="nbServ" type="xsd:int"></xsd:element>
        <xsd:element name="requiredQ" type="tns:RequiredQuality"
                     maxOccurs="unbounded">
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ListRegisteredService">
      <xsd:sequence>
        <xsd:element name="urlRegServ" type="xsd:anyURI"
                      maxOccurs="unbounded"></xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</wsdl:types>

<wsdl:message name="demandeItinerairesRequest">
  <wsdl:part name="reqItineraire" type="tns:RequeteItineraire"></wsdl:part>
</wsdl:message>
<wsdl:message name="demandeItinerairesResponse">
  <wsdl:part name="repItineraire" type="tns:Itineraire"></wsdl:part>
</wsdl:message>
<wsdl:message name="reservationItineraireRequest">
  <wsdl:part name="reqReservation" type="tns:Itineraire"></wsdl:part>
  <wsdl:part name="nom" element="tns:nomClient"></wsdl:part>
</wsdl:message>
<wsdl:message name="reservationItineraireResponse">
  <wsdl:part name="repBillet" type="tns:Billet"></wsdl:part>
</wsdl:message>
<wsdl:message name="RegisterRequest">
  <wsdl:part name="reqService" type="tns:RegisterService"></wsdl:part>
</wsdl:message>
```

```
<wsdl:message name="RegisterResponse">
  <wsdl:part name="OK" type="xsd:boolean"></wsdl:part>
</wsdl:message>
<wsdl:message name="QuitRequest">
  <wsdl:part name="reqQuit" type="tns:QuitService"></wsdl:part>
</wsdl:message>
<wsdl:message name="QuitResponse">
  <wsdl:part name="OK" type="xsd:boolean"></wsdl:part>
</wsdl:message>
<wsdl:message name="QualityModelRequest">
  <wsdl:part name="community" type="xsd:string"></wsdl:part>
</wsdl:message>
<wsdl:message name="QualityModelResponse">
  <wsdl:part name="repQualityModel" type="tns:Criterion"></wsdl:part>
</wsdl:message>
<wsdl:message name="AbstractInterfaceRequest">
  <wsdl:part name="community" type="xsd:string"></wsdl:part>
</wsdl:message>
<wsdl:message name="AbstractInterfaceResponse">
  <wsdl:part name="WSDLDescription" type="xsd:anyURI"></wsdl:part>
</wsdl:message>
<wsdl:message name="SelectRequest">
  <wsdl:part name="reqSelect" type="tns:RequeteSelect"></wsdl:part>
</wsdl:message>
<wsdl:message name="SelectResponse">
  <wsdl:part name="repSelect" type="tns:ListRegisteredService"></wsdl:part>
</wsdl:message>

<wsdl:portType name="Fournisseur">
  <wsdl:operation name="Register">
    <wsdl:input message="tns:RegisterRequest"></wsdl:input>
    <wsdl:output message="tns:RegisterResponse"></wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Quit">
    <wsdl:input message="tns:QuitRequest"></wsdl:input>
    <wsdl:output message="tns:QuitResponse"></wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="QualityModel">
    <wsdl:input message="tns:QualityModelRequest"></wsdl:input>
    <wsdl:output message="tns:QualityModelResponse"></wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="AbstractInterface">
    <wsdl:input message="tns:AbstractInterfaceRequest"></wsdl:input>
    <wsdl:output message="tns:AbstractInterfaceResponse"></wsdl:output>
  </wsdl:operation>
</wsdl:portType>

<wsdl:portType name="Client">
  <wsdl:operation name="Select">
    <wsdl:input message="tns:SelectRequest"></wsdl:input>
    <wsdl:output message="tns:SelectResponse"></wsdl:output>
  </wsdl:operation>
```

```
  </wsdl:portType>
</wsdl:definitions>
```

ANNEXE B

DÉTAILS SPÉCIFIQUES DE LA MISE EN ŒUVRE

Cet annexe montre quelques détails d'implantation à un niveau plus technique, par rapport à ceux décrits dans le chapitre 7.

Nous présentans cet annexe en deux parties. La première partie est dediée à l'implantation du canevas `Tcows` et la deuxième partie à l'implantation de la communauté.

B.1 Le modèle de composition `Tcows`

La figure B.1 décrit le fichier des adresses des communautés trouvées par le Module de communication à partir des capacités de services envoyées par le Module de composition que le Module de communication envoie au module Construction des options. Ce fichier contient les adresses des communautés trouvées pour la composition en cours.

La figure B.2 montre un fichier au format XML avec les adresses de services web recupérés auprès de chaque communauté participant à la composition. Ce fichier est construit par le Module de communication en exploitant le fichier contenant les adresses des communautés (voir figure B.1).

Ainsi, la figure B.1 décrit le fichier UrlsCommunities.xml qui contient les adresses des cinq communautés nécessaires à la composition du voyage de notre exemple.

Nous avons codé une fonctionnalité qui reçoit le fichier de la figure B.2 comme paramètres d'entrée et produit la liste d'options montrée dans la figure B.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<Communities>
  <Community name=''Vol'' mandatory=''true''>
    <url>http://the.address.of.communityVol</url>
  </Community>
  <Community name=''Hotel'' mandatory=''true''>
    <url>http://the.address.of.communityHotel</url>
  </Community>
  <Community name=''Voiture'' mandatory=''false''>
    <url>http://the.address.of.communityVoiture</url>
  </Community>
  <Community name=''Vélo''  mandatory=''false''>
    <url>http://the.address.of.communityVélo</url>
  </Community>
  <Community name=''Météo''  mandatory=''false''>
    <url>http://the.address.of.communityMétéo</url>
  </Community>
</Communities>
```

Fig. B.1 – Document XML décrivant les adresses urls des communautés

B.2 Les communautés de services web

Dans le chapitre 7, la figure 7.4 modélise le modèle de qualité de la communauté via un diagrammme UML. Puis, nous passons de cette diagramme UML à la spécification d'une relation. Etant donné que Criterion est un type paramétré et que le modèle de qualité dépends du nombre de critères, la figure B.4 traduit la partie correpondant au modèle de qualité décrit dans la figure 7.4, page 106, et décrit la relation que nous avons implémentée.

QualityModel				
name	natureSem	natureUnit	orderingMode	type
Reputation	nil	nil	descendant	integer
TempRéponse	durée	millisecondes	ascendant	float
Fiabilité	nil	nil	descendant	integer

Tab. B.1 – Le modèle de qualité de la communauté Vol

Ainsi, les méthodes de la classe Criterio deviennent des attributs de la relation QualityModel. Conformément à l'exemple que nous sommes en train de décrire tout à long de cette thèse, le modèle de qualité de la communauté Vol est décrit via un ensemble de n-tuples, comme le montre la table B.1.

Quand les fournisseurs et les clients de la communauté demandent la consultation du modèle de qualité, une procédure JDOM va parcourir cette table pour générer un fichier en format XML, en réponse à la requête. La figure B.5 montre le modèle de qualité de la communauté Vol exposé dans le chapitre 4.

Pour mettre en œuvre les services qui vont être enregistrés dans une communauté, nous avons traduit la figure 7.4, page 7.4, en un schéma relationnel pour obtenir la base de

```
<?xml version="1.0" encoding="UTF-8"?>
<UrlsServicesWeb>
  <Community name=''Vol'' mandatory=''true''>
    <url>http://the.address.of.AirFrance</url>
    <url>http://the.address.of.Avianca</url>
    <url>http://the.address.of.Qantas</url><Community>
  </Community>
  <Community name=''Hotel'' mandatory=''true'' >
    <url>http://the.address.of.Vauban</url>
    <url>http://the.address.of.WalkAbout</url>
  </Community>
  <Community name=''Voiture'' mandatory=''false''>
    <url>http://the.address.of.Avis</url>
    <url>http://the.address.of.Rentacar</url>
  </Community>
  <Community name=''Vélo'' mandatory=''false''>
    <url>http://the.address.of.BikeSolutions</url>
    <url>http://the.address.of.Birds</url>
  </Community>
  <Community name=''Météo'' mandatory=''false''>
    <url>http://the.address.of.MétéoFrance</url>
    <url>http://the.address.of.BOM</url>
  </Community>
</UrlsServicesWeb>
```

Fig. B.2 – Les adresses de services web par communauté

données de la communauté Vol (voir la figure B.6). Dans la table RegisteredServices se trouvent toutes les références des services enregistrés auprès de cette communauté. La table ServiceQualityModel contient le modèle de qualité associé à chaque service enregistré dans la table RegisteredServices. L'association entre la table ServiceQualityModel et la table QualityModel permet d'assurer que les critères du modèle de qualité des services enregistrés se trouvent parmi ceux du modèle de qualité de la communauté.

Nous avons implémenté ce schéma en utilisant la base de données relationnelle MySQL. Cette structure de données permet de répondre à toutes les tâches liées au fonctionnement de la communauté, tant selon le point de vue des fournisseurs et des clients, que selon le point de vue de l'administrateur de la communauté.

La figure B.7 montre la base de données utilisée pour le développement de toutes les communautés.

La figure B.8 montre la page d'accueil de l'application que nous avons développée pour la communauté Vol. L'aplication montre tois onglets, le premier est destiné aux faournisseurs de services. Dans cette page, ils enregistrent les services web et son modèle de qualité associé. Ici, il est possible aussi de télécharger l'interface abstraite de la communauté via un fichier WSDL, ainsi comme le modèle de quelité. Le deuxième onglet est destiné aux clients et donne seulement l'URL du service web qui fournit l'opération Select(). Le troisième onglet est destiné à l'administrateur de la communauté, mais nous n'avons pas encore développé des fonctionnalités pour cette tâche.

```
<?xml version="1.0" encoding="UTF-8"?>
<AnswerOptions>
  <Option>
    <url>http://the.address.of.AirFrance</url>
    <url>http://the.address.of.Vauban</url>
  </Combination>
  <Option>
    <url>http://the.address.of.AirFrance</url>
    <url>http://the.address.of.WalkAbout</url>
  </Option>
  <Option>
    <url>http://the.address.of.Avianca</url>
    <url>http://the.address.of.Vauban</url>
  </Option>
  <Option>
    <url>http://the.address.of.Avianca</url>
    <url>http://the.address.of.WalkAbout</url>
  </Option>
  <Option>
    <url>http://the.address.of.Qantas</url>
    <url>http://the.address.of.Vauban</url>
  </Option>
    ...
  <Option
    <url>http://the.address.of.Qantas</url>
    <url>http://the.address.of.WalkAbout</url>
    <url>http://the.address.of.Avis</url>
  </Option>
   ...
  <Option>
    <url>http://the.address.of.Qantas</url>
    <url>http://the.address.of.WalkAbout</url>
    <url>http://the.address.of.Avis</url>
    <url>http://the.address.of.MétéoFrance</url>
  </Option>
  <Option>
    <url>http://the.address.of.Qantas</url>
    <url>http://the.address.of.WalkAbout</url>
    <url>http://the.address.of.Avis</url>
    <url>http://the.address.of.BOM</url>
  </OptionCombination>
  <Option>
    <url>http://the.address.of.Qantas</url>
    <url>http://the.address.of.WalkAbout</url>
    <url>http://the.address.of.Rentacar</url>
    <url>http://the.address.of.MétéoFrance</url>
  </Option>
    ...
</AnswerOptions>
```

Fig. B.3 – Fichier en format XML de la liste initiale d'options

QualityModel				
name	natureSem	natureUnit	orderingMode	type

FIG. B.4 – Relation qui implémente le modèle de qualité de la communauté

```
<?xml version="1.0" encoding="UTF-8"?>
<Community name = ‘‘Vol’’>
 <Criteria>
  <Criterion name="Fiabilite">
    <Nature />
    <Ordering mode="descendant" />
    <Type name="integer" />
  </Criterion>
  <Criterion name="Reputation">
    <Nature />
    <Ordering mode="descendant" />
    <Type name="integer" />
  </Criterion>
  <Criterion name="TempReponse">
    <Nature semantic="duree" unit="millisecondes" />
    <Ordering mode="ascendant" />
    <Type name="float" />
  </Criterion>
 </Criteria>
</Community>
```

FIG. B.5 – Fichier en format XML du modèle de qualité de la communauté Vol

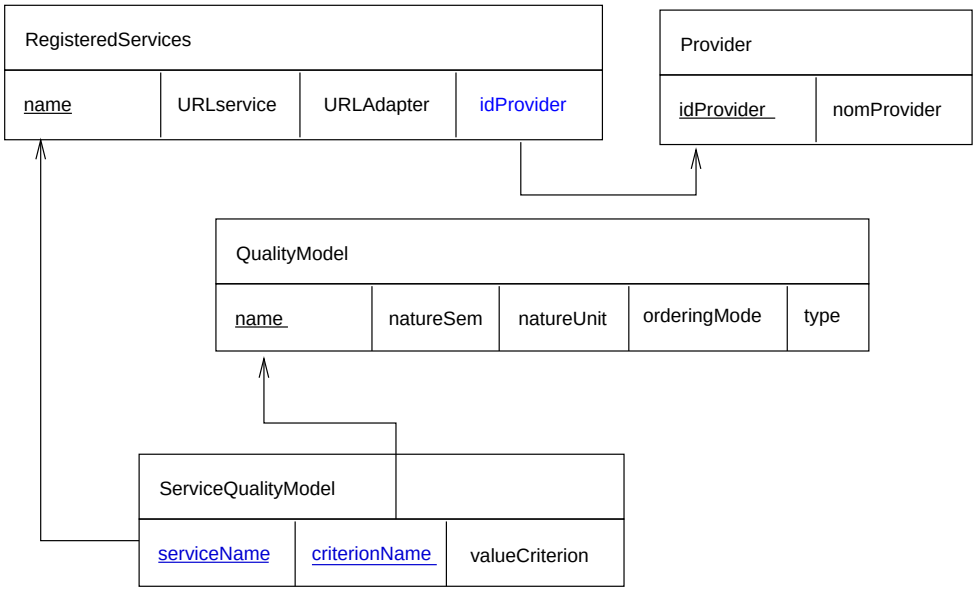


FIG. B.6 – Modèle relationnel de la base de données de la communauté Vol

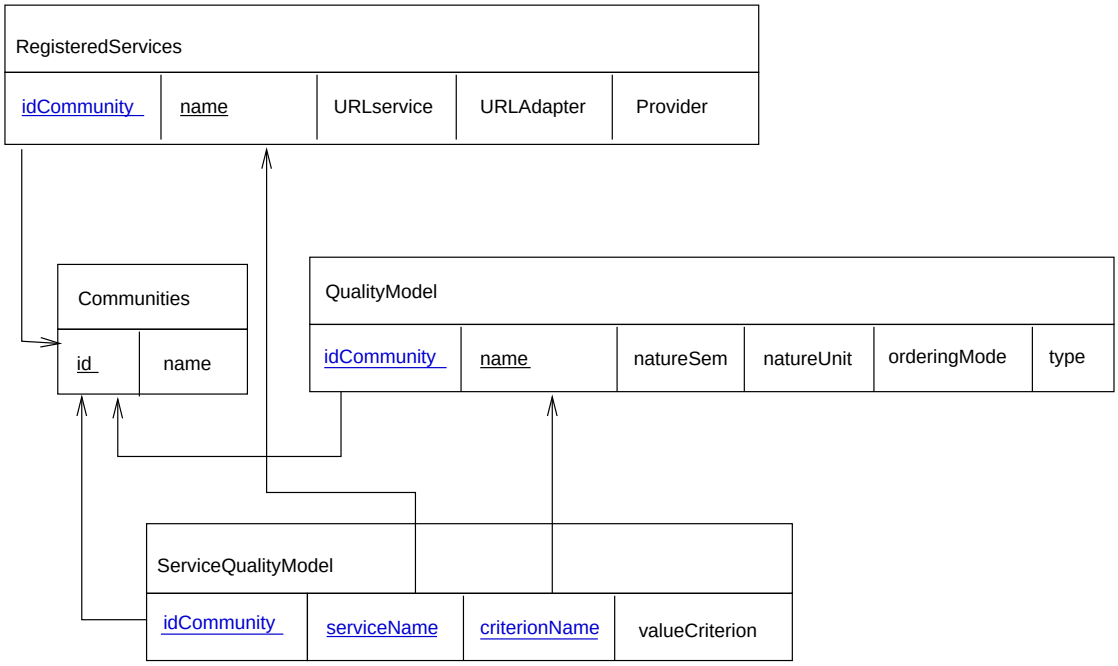


FIG. B.7 – Modèle de la base de données pour les communautés

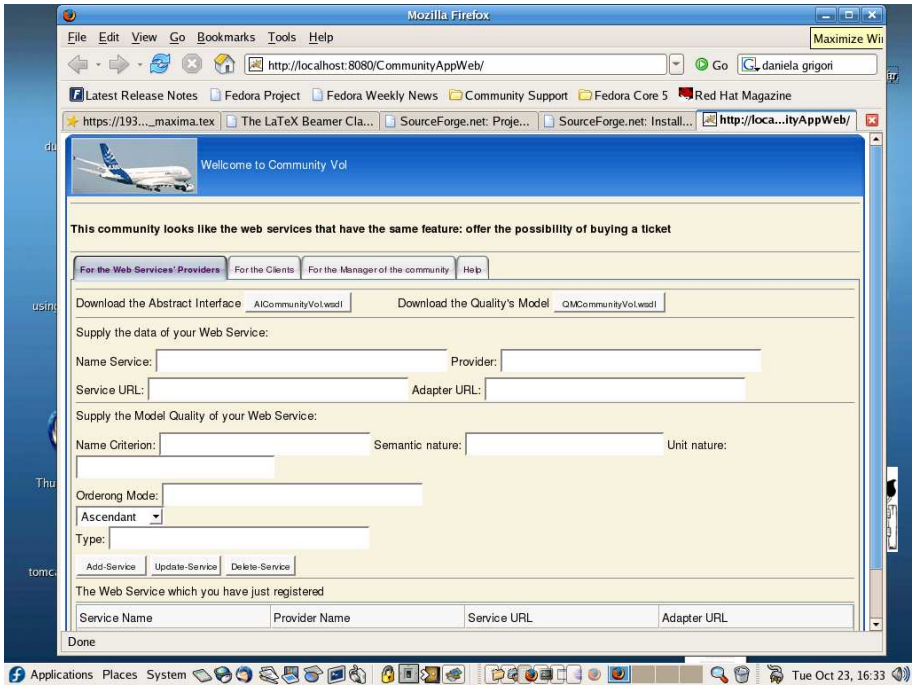


FIG. B.8 – L'"Application web" de la communuauté Vol

ANNEXE C

PUBLICATIONS

Les publications issues de la thèse.

1. H. Duarte-Amaya, *Vers un modèle transactionnel pour des services Web*. Dans le 22ème Congrès INFORSID'04 (Informatique des Organisations et Systèmes d'Information et de Décision), Biarritz - France, 25-28 Mai 2004.
2. H. Duarte and M.-C. Fauvet and M. Dumas and B. Benatallah, *Vers un modèle de composition de services Web avec propriétés transactionnelles*, Revue Ingénierie des Systèmes d'Information - Numéro spécial Services Web, vol10, no3/2005, pp. 9-28, 2005.
3. M.-C. Fauvet and H. Duarte and M. Dumas and B. Benatallah, *Handling Transactional Properties in Web Service Composition*, in WISE2005 International Conference, New York City (NY), pp. 273-289, November 2005.
4. M. -C. Fauvet, H. Duarte, Y. Taher, D. Benslimane, *Sélection dynamique de services Web – Une approche à base de communautés*, Dans le XXVème congrès INFORSID'07 (Informatique des Organisations et Systèmes d'Information et de Décision), Perros-Guirec - France, 22-25 Mai 2007

BIBLIOGRAPHIE

[AAF+02] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, I. Takacsi-Nagy, P.and Trickovic, and S. Zimek. Web Service Choreography Interface - WSCI. *http ://www.w3.org/TR/2002/NOTE-wsci-20020808/*, W3C note 8 August 2002. BEA, Intalio, SAP, Sun.

[AAP+99] J.-M. Andreoli, D. Arregui, F. Pacull, M. Riviere, and J.-Y. Vion-Dury. CLF/Mekano : a Framework for Building Virtual-enterprise Applications. In *EDOC'99*. In Proc. of Enterprise Distributed Object Computing EDOC'99, 1999. Mannheim, Germany.

[ABF07] A. Ait-Bachir and M.-C. Fauvet. Un Médiateur pour la Réconciliation de Conversations entre Services Web. In *Actes du Congrès INFORSID*, Perros-Guirec, 2007. A paraître.

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, K. Liu, D. Roller, D. Smith, S. Thatte, and I. Trickovic. Business Process Executions Langage for Web Services, BPEL4WS. Version 1.1. http ://www.ibm.com/developerworks/library/ws-bpel, 5 May 2003. BEA, IBM, Microsoft, SAP AG, Siebel Systems.

[ACF96] J. Andrade, M. Carges, and S. Felts. *The Tuxedo System : Software for Constructing and Managing Distributed Business Applications*. Addison-Wesley Professional, 1996.

[ACKM03] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer Verlag, 2003.

[AFI+03] Arjuna, Fujitsu, Iona, Oracle, and Sun-Microsystems. http ://www.oasis-open.org/committees/ws-caf/charter.php, 28 jullet 2003.

[Alo05] G. Alonso. *Service Oriented Software System Engineering*, chapter Transactional Business Processes. IDEA Group Publishing, 2005. to appear.

[alp02] IBM alphaWorks. BPWS4J A Platform for Creating and Executing BPEL4WS Processes. http ://www.alphaworks.ibm.com/tech/bpws4j, 9 Aout 2002.

[APR00] D. Arregui, F. Pacull, and M. Riviere. Heterogeneous Component Coordination : The CLF Approach. In *EDOC*, pages 194–2003. 4th International Enterprise Distributed Object Computing Conference, Makuhari, Japan, IEEE Computer Society, 25-28 September 2000.

[BBC+02] C. Banerji, A.and Bartolini, D. Beringer, V. Chopella, V. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, and S. Sharma, S.and Williams. Web Service Conversation Langage - WSCL, v. 1.0. *www.w3.org/TR/2002/NOTE-wscl10-20020314/*, W3C note 14 March 2002. Hewlett-Packard Company.

[BBCT04] K. Baïna, B. Benatallah, F. Casati, and F. Toumani. Model-Driven Web Services Development. *16th International Conference on Advanced Information Systems Engineering (CAISE'04), Riga, Latvia. Springer-Verlag*, 3084, 7-11 June 2004.

[BC06] D. Barreiro Claro. SPOC - Un canevas pour la composition automatique de services web dédiés à la réalisation de devis. Thèse de Doctorat, Ecole Doctorale d'Angers, 2006.

[BCC+02] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescou. Model-driven Specification of Web Services Composition and Integration with Data-intensive Web Applications. *IEEE Computer Society Technical Committee on Data Engineering Bulletin*, November 2002.

[BCTH03] B. Benatallah, F. Casati, F. Toumani, and R. Hamadi. Conceptual Modeling of Web Services Conversations. In *CAiSE*. Proc. of the 15th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'03), no. 2681 in Lecture Notes in Computer Science, Klagenfurt/Velden, Austria., Springer, 16-18 June 2003.

[BDDM05] B. Benatallah, R. Dijkman, M. Dumas, and Z. Maamar. *Service Oriented Software System Engineering*, chapter Service Composition : Concepts, Techniques, Tools and Trends. IDEA Group Publishing, 2005. to appear.

[BDM02] B. Benatallah, M. Dumas, and Z. Maamar. Definition and Execution of Composite Web Services the SELF-SERV Projet. *IEEE Computer Society Technical Committee on Data Engineering Bulletin*, 25(4), 2002.

[BDS05] B. Benatallah, M. Dumas, and Q. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Database*, 17(1) :5–37, 2005. Springer Science, Business Media.

[BDSN02] B. Benatallah, M. Dumas, Q.-Z. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proceedings of ICDE'02 Conference*, San Jose, California, 2002. IEEE Computer Society Press.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelshon, H-F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocole - SOAP - Version 1.1. http ://www.w3.org/TR/SOAP, May 2000. W3C note.

[Ber96] P. Bernstein. Middleware : A Model for Distributed Systems Services. *Communications of the ACM*, 2(39) :8698, 1996.

[BGP05] S. Bhiri, C. Godart, and O. Perrin. Reliable Web Services Composition using a Transactional Approach. In *SCC'04*. IEEE International Conference on e-Tecnology, e-Commerce and e-Service, Honk Kong, 29 March - 1 April 2005.

[Bhi04] S. Bhiri. A Transactional Framework for Reliable Web Services Compositions to Support Interentreprises Cooperations. In *CAiSE'04*, pages 83–94. 11th Doctoral Consortium on Advanced Information Systems Engineering, Riga, Latvia, June 2004.

[Bhi05] S. Bhiri. *Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web*. PhD thesis, Université Henri Poincaré, 2005.

[BHRT03] B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. *The SemanticWeb - ISWC 2003*, chapter Request Rewriting-Based Web Service Discovery, pages 242–257. LNCS. Springer Berlin-Heidelberg, 2003.

[Blo92] J. Bloomer. *Power Programming with RPC*. O'Reilly, First edition, 1992.

[BMT+07] D. Benslimane, Z. Maamar, Y. Taher, M. Lahkim, M.-C. Fauvet, and M. Mrissa. Multi-Layer and Multi-Perspective Approach for Web Services Composition. In *AINA'07*, Niagara Falls, Canada, 2007. Proc. of the 21st International Conference on Advanced Information Networking and Applications, IEEE.

[BN97a] P. Bernstein and E. Newcomer. *Principles of Transaction Processing for the Systems Professional*. Morgan Kaufmann Publishers, 1997.

[BN97b] Ph. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, 1997.

[BPM04] BPMI. Business Process Modeling Notation, Version 1.0. Specification, May 2004.

[BSD03] B. Benatallah, Q. Z. Sheng, and M. Dumas. The SELF-SERV Environment for Web Service Composition. *Internet Computing, IEEE*, 7(1) :40–48, 2003.

[Bus03] C. Bussler. *B2B Integration. Concepts and Architectures*. Springer Verlag, 2003.

[CCC+02a] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, D. Langworthy, and D. Orchard. Web Service Coordination, WS-Coordination. http ://www.ibm.com/developerworks/library/ws-coor/, W3C note 9 August 2002. IBM, Microsoft, BEA.

[CCC+02b] F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey, and S. Thatte. Web Service Transaction, WS-Transaction. http ://www.ibm.com/developerworks/library/ws-transpec/, W3C note 9 August 2002. IBM, Microsoft, BEA.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Langage - WSDL. Version 1.1. http ://www.w3.org/TR/wsdl.html, March 2001. W3C note.

[CFK05] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based Resource Management. In *Proceedings of the IEEE*, volume 93, pages 631– 643, March 2005.

[CFRC05] C. Consel, L. Fabien, L. Réveillère, and P. Cointe. Une Approche de Programmation Générative au Développement de Compilateurs de DSL - Domain-Specific Languages. Technical report, INRIA, http ://www.inria.fr/rrrt/rr-5550.html, Avril 2005. Equipe PHOENIX.

[Cha04] D. Chappell. *Enterprise Service Bus*. O'Reilly Media, 2004.

[CIJ+00] F. Casati, S. Ilnicki, L.J. Jin, V. Krishnamoorthy, and M.C. Shan. eflow : A Platform for Developing and Managing Composite e-Services. In *2000 Academia-Industry Working Conference on Research Challenges (AI-WoRC 2000)*. IEEE Computer Society, 27-29 April, Buffalo, NY, USA. 2000.

[Coa96] WorkFlow Management Coalition. Workflow Management Coalition : Terminology and Glossary. Technical report, WorkFlow Management Coalition Brussels, Belgium 1996.

[Col00] C. Collet. The NODS project : Networked Open Database Services. In *In Proc. of the Int. Symposium on Objects and Databases, Sophia Antipolis, France*. Springer Verlag, 2000.

[Com05] D. Comer. *Internetworking with TCP/IP*. Prentice Hall, 2005.

[Cor] Microsoft Corporation. .NET. http ://www.microsoft.com/net.

[CT06] L. Chappell and E. Tittel. *Guide to TCP/IP*. Course Technology, 2006.

[Dav78] C.T Davies. Data Processing Spheres of Control. *IBM Systems Journal*, 17(2) :179–198, 1978.

[DD04] R. Dijkman and M. Dumas. Service-oriented Design : A Multi-viewpoint Approach. Technical Report CTIT Technical Report Series No. 04-09, Centre for Telematics and Information Technology, University of Twente, The Netherlands, February 2004.

[DDR05] C. Dominic, M. Dumas, and P. Roe. A Porgramming Langage for Web Service Development. Technical report, Centre for Information Technology Innovation, 2005.

[DHL01] U. Dayal, M. Hsu, and R. Ladin. Business Process Coordination : Sate of the Art, Trends, and Open Issues. In *VLDB'01*. Proceeding of the 27th VLDB Conference, 2001. Roma, Italy.

[Dic98] Dickman. *Designing Applications With Msmq : Message Queuing for Developers*. Addison-Wesley Professional, 1998.

[DKZD06] G. Decker, M. Kirov, J.M. Zaha, and M. Dumas. Maestro for Let's Dance : An Environment for Modeling Service Interactions. Technical Report Preprint no. 4708, Faculty of Information Technology, Queensland University of Technology, July 2006. https ://eprints.qut.edu.au/secure/00004708/01/bpmdemo-final.pdf.

[DOHE01] M. Dumas, j. O'Sullivan, M. Hervizadeh, and A. Edmond, D. ter Hofstede. Towards a Semantic Framework for Service Description. In *IFIP TC2/WG2.6*, pages 277 – 291. In Proc. of the IFIP TC2/WG2.6 Ninth Working Conference on Database Semantics : Semantic Issues in E-Commerce Systems, Hong Kong, April 25-28 2001.

[DTL+03] S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber. Coordinating Business Transactions on the Web. *IEEE Internet Computer Society Journal*, 7(1) :30–39, January-February 2003.

[Dum05] M. Dumas. Tutorial on Service Design, Implementation and Description. The 6th International Conference on Web Information Systems Engineering. New York City, New York, 20-22 Nov 2005.

[Elm90] Ahmed K. Elmagarmid, editor. *Database Transactions Models for Advanced Applications*. Morgan Kaufmann Publishers, 1990.

[Eme00] W. Emerich. Sofware Engineering and Middleware : A roadmap. *Proceedings of the conference on The future of Sofware Engineering, Limerick, Ireland*, pages 117–129, 2000.

[eXO98] Prentice-Hall end X Open. *X/Open CAE Specification : Distributed Transaction Processing : Cpi-C Specification, Version 2*. Prentice-Hall PTR, 1998.

[FAB06] M.-C. Fauvet and A. Ait-Bachir. An Automaton-Based Approach for Web Service Mediation. In *Proceedings of the International Conference on Concurrent Engineering*, pages 47–54, Antibes, France, September 2006.

[FDBP01] M.C. Fauvet, M. Dumas, B. Benatallah, and H.-H. Paik. Peer-To-Peer Traced Execution of Composite Services. In *Proceedings of the International Workshop on Technologies for E-Services (TES 2001).In cooperation with VLDB.*, pages 103–117, Roma, Italy, 2001. Springer.

[FDDB05] M.-C. Fauvet, H. Duarte, M. Dumas, and B. Benatallah. Handling Transactional Properties in Web Service Composition. In Springer-Verlag LNCS, editor, *WISE'05*, volume 3806, pages 273–289. The 6th International Conference on Web Information Systems Engineering. New York City, New York, 20-22 Nov 2005.

[FDTB07] M.-C. Fauvet, H. Duarte, Y. Taher, and D. Benslimane. Sélection Dynamique de Services Web – Une Approche à Base de Communautés. In *INFORSID'07*. XXVème congrès INFORSID (Informatique des Organisations et Systèmes d'Information et de Décision), Perros-Guirec, France, 22-25 Mai 2007.

[FFK+06] I. Foster, T. Freeman, K. Keahey, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual Clusters for Grid Communities. In *CCGRID*, pages 513–520. Sixth IEEE International Symposium on Cluster Computing and the Grid, Singapore, IEEE Computer Society, 16-19 May 2006.

[Gav03] L. Gavin, editor. *An EAI Solution Using Websphere Business Integration V4.1*. IBM Redbooks, 2003.

[GdT06] D.-Z.-G. Garcia and M.-B.-F. de Toledo. Semantics-enriched QoS Policies for Web Service Interactions. In *Proc. of the 12th Brazilian symposium on Multimedia and the web*, pages 35–44, New York, NY, USA, 2006. ACM Press.

[GMS87] H. Garcìa-Molina and K. Salem. Sagas. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pages 249–259, San Francisco, CA, USA, December 1987.

[GR93] J. Gray and A. Reuter. *Transaction Processing : Concepts and Tecniques*. Morgan Kaufmann Publishers, 1993.

[Gra81] J. Gray. The Transaction Concept : Virtues and Limitations (invited paper). In *VLDB*, pages 144–154. Very Large Data Bases, 7th International Conference, Cannes, France, IEEE Computer Society, 9–11 September 1981. Invited Paper.

[Gro03] Object Management Group. Object Constraint Langage. http ://www.omg.org/docs/formal/03-03-13.pdf, 2003.

[HA00] C. Hagen and G Alonso. Exception Handling in Workflow Management Systems. *Software Engineering, IEEE Transactions on*, 26(10) :943–958, October 2000.

[Har87] D. Harel. Statecharts : A visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3) :231–274, June 1987.

[HKY95] T. Howes, S. Kille, and W. Yeong. Lightweight Directory Access Protocol. www.ietf.org/rfc/rfc17777.txt, 1995.

[IBM] IBM. Websphere MQ. http ://www-306.ibm.com/software/integration/wmq.

[Icu96] Icube. OpenFlow. http ://www.openflow.it/EN/index-html, 1996.

[Jam05] S. Jamal. *Environnement de Procédé Extensible pour l'Orchestration - Application aux Services Web*. PhD thesis, Université Joseph Fourier, 13 décembre 2005.

[JMS06] M.B. Juric, B. Mathew, and P. Sarang. *Business Process Execution Language for Web Services*. Packt Publishing, 2006.

[Kal02] M. F. Kaleem. Transactions over Web Services - An Introduction to the Business Transaction Protocol. *http ://www.webservices.org/index.php/article/381/-1/1*, May 2002.

[KBR+04] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Service Choreography Description Langage - WSCDL, v. 1.0. *http ://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/*, W3C note 17 December 2004. Hewlett-Packard Company.

[KCH+04] M. Keen, J. Cavell, S. Hill, C.K. Kee, W. Neave, B. Rumph, and H Tran. BPEL4WS Business Processes with WebSphere, Business Integration : Understanding, Modeling, Migrating. IBM Redbook, December 2004.

[KGM98] S. P. Ketchpel and H. Garcia-Molina. A Sound and Complete Algorithm for Distributed Transactions. *Distributed Computing Journal*, 12(8) :13–29, 1998.

[Kra06] S. Krakowiak. Intergiciel et Construction d'Applications Réparties. Ecole d'été, 13 juillet 2006. Autrans, France.

[Lay01] F. Laymann. Web Service Flow Langage - WSFL. http ://www.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf, May 2001. IBM.

[LEK98] j. Lyon, K. Evans, and J. Klein. Transaction Internet Protocol - TIP. Version 3.0, July 1998.

[LL05] Z. Luo and J-S. Li. A Web Services Provisioning Optimization Model in a Web Services Community. In *ICEBE*, pages 689–696. IEEE International Conference on e-Business Engineering. Beijing, China, IEEE Computer Society, 18-20 October 2005.

[Loe03] S. Loecher. Model-Based Transaction Service Configuration for Components-Based Development. http ://www.sei.cmu.edu/pacc/CBSE7, November 2003.

[LZ04] B. Limthanmaphon and Y. Zhang. Web Service Composition Transaction Management. In Klaus-Dieter Schewe and Hugh Williams, editors, *ADC*, volume 27 of *CRPIT*. Database Technologies 2004, Proceedings of the Fifteenth Australian Database Conference, ADC 2004, Dunedin, New Zealand, Australian Computer Society, 18-22 January 2004.

[Mic01] Microsoft. Microsoft Biztalk Server Product. htpp ://www.microsoft.com/BizTalk, 2001.

[Mic04a] Biztalk Server 2004 Conceptual Overview, 2004.

[Mic04b] Microsoft. Biztalk Server 2004 Conceptual Overview. msdn.microsoft.com/library/en-us/introduction/htm, 2004.

[MM03] D-J. Mandell, , and S-A. McIlraith. *The SemanticWeb - ISWC 2003*, chapter Adapting BPEL4WS for the Semantic Web : The Bottom-Up Approach to Web Service Interoperation, pages 224–241. LNCS. Springer Berlin-Heidelberg, 2003.

[MPP02] M. Mecella, F.P. Presicce, and B. Pernici. Modeling E-Services Orchestration through Petri Nets. In *TES*, volume 2444, pages 38–47. Proc. 3rd Int. Workshop on Technologies for E-Services, Springer LNCS, 2002.

[MTR02] T. Mikalsen, S. Tai, and I. Rouvellou. Transactional Attitudes : Reliable Composition of Autonomous Web Services. In Workshop on Dependable Middleware-based systems (WDMS 2002), June 2002. Washington, D.C., USA.

[MTSM02] J. McGovern, S. Tyagi, M. Stevens, and S. Mathew. *Java Web Services Architecture*, chapter 14 :Transaction Management. Morgan Kaufmann, 2002.

[Mul97] P.-A. Muller. *Modélisation Objet avec UML*. Eyrolles, 1997. ISBN 2-212-08966-X.

[New02] E. Newcomer. *Understanding Web Services : XML, WSDL, SOAP and UDDI*. David Chapell. Addison Wesley, 2002.

[New04] E. Newcomer. Contex, Coordinators, and Transations - The Importance of WS-CAF. *http ://www.webservices.org/index.php/article/view/1297*, January 2004.

[OAS02] OASIS. Business Transaction Protocol - BTP. http ://www.oasis-open.org/, Juin 2002.

[OAS05] OASIS UDDI. Universal Description, Discovery and Integration version 3.0. http ://www.uddi.org, 2005.

[Obj92] http ://www.omg.org/, 1992.

[Ope93] Distributed Computing Environment. http ://www.sei.cmu.edu/str/descriptions/dce.html, 1993.

[Ora04] Oracle. Oracle BPEL Process Manager. http ://www.oracle.com/global/fr/corporate/press/collaxa.html, 29 juin 2004.

[OYP03] B. Orriens, J. Yang, and M. Papazoglou. Model Driven Service Composition. In *ICSOC*, number 2910 in Lecture Notes in Computer Science, pages 75–90. Proceedings of the First International Conference on Service Oriented Computing, Springer-Verlag, December 2003. Trento, Italy.

[Pap03] Mike P. Papazoglou. Web Services and Business Transactions. *World Wide Web*, 6(1) :49–91, 2003.

[PBT04] H-y. Paik, B. Benatallah, and F. Toumani. WS-Catalognet : Building Peer-to-Peer e-Catalogs. In *FQAS 2004*, pages 256–269. 6th International Conference on Flexible Query Answering Systems. Lyon, France, June 24-26 2004.

[Pel03] C. Peltz. Web Services Orchestration. *Hewlet-Packard Company Journal*, january 2003.

[PKPS02] M. Paolucci, T. Kawamura, T-R. Payne, and K-P. Sycara. Semantic Matching of Web Services Capabilities. In *ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 333–347. First International Semantic Web Conference, Sardinia, Italy, Springer, June 9-12 2002.

[Pla99] D.S. Platt. *Undertanding COM+*. Microsoft Press, 1999. 256 pp.

[pro] Intalion3 product. http ://www.intalio.com/products/index.xpg.

[QDvS04] D. Quartel, r. Dijkman, and M. van Sinderen. Methodological Support for Service-oriented Design with ISDL. In *ACM*. In Proceedings of the Second International Conference on Service Oriented Computing, November 2004.

[QRTY04] Z. Qi, R. Rao, F. Tang, and J. You. WebTP as a Case Study of Web Transaction Processing. In *PDPTA'04*, pages 881–886. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas, Nevada, USA, CSREA Press, 21-24 June 2004.

[RCMS01] J. Roberts, T. Collier, P. Malu, and K. Srinivasan. Tentative Hold Protocol - Part 2. http ://www.w3.org/TR/tenthold-2, November 2001.

[Red05] IBM Redbooks. *Websphere Message Broker Basics*. Vervante, 2005.

[Row00] M. Rowell. *Understanding EAI : Enterprise Application Integration*. Sams, 2000.

[RS01] J. Roberts and K. Srinivasan. Tentative Hold Protocol - Part 1. http ://www.w3.org/TR/tenthold-1, November 2001.

[SBS02] G. Schuldt, H.and Alonso, C. Beeri, and H.J. Schek. Atomicity and Isolation for Transactional Processes. *ACM Transactions on Database Systems*, 27 :63–116, 2002.

[SD04] B. A. Schmit and S. Dustar. Model-driven Development of Web Service Transactions. Technical Report TUV-1841-2004-27, Technical University of Vienna, 10 December 2004.

[Sky02] Java Skyline. Collaxa 2.0 BPEL4WS Server. http ://www.javaskyline.com/20030311-collaxa.html, 11 Mars 2002.

[TBFM06] Y. Taher, D. Benslimane, M-C. Fauvet, and Z. Maamar. Towards an Approach for Web Services Substitution. In *IEEE IDEAS 2006*, pages 166–173. Proc. of the 10th IEEE International Database Engineering and Applications Symposium, India 2006.

[Tha01] S. Thatte. Web Service for Business Process Design - XLANG. http ://www.gotdotnet.com/team/xmlwsspecs/xlang-c/default.htm, 2001. Microsoft.

[UC01] UN-CEFACT. UN-CEFACT's Modeling Methodology, Version 10. Specification, November 2001.

[vdHA02] W. van den Heuvel and S. Artyshchev. Developing a Three-Dimensional Transaction Model for Supporting Atomicity Spheres. In *WS-RSD'02*, volume 2593. International Workshop - Research, Standardization, and Deployment, Springer's LNCS Series, 2002.

[VV03] K. Vidyasankar and G. Vossen. A Multi-level Model for Web Service Composition. Technical report, Dept. of Information Systems, University of Muenster, 2003. Tech. Report No. 100.

[W3C] W3C. XML Query. http ://www.w3.org/XML/Query.

[WV02] G. Weikum and G. Vossen. *Transactional Informations Systems : Theory, Algorithms and the Practice of Concurrency Control and Recovery.* Kaufmann Publishers, 2002.

[WvdADtH03] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of Web Services Composition Languages : The case of BPEL4WS. *Proc. of the 22nd Intl. Conf. on Conceptuel Modelling (ER). Chicago IL, USA*, Octobre 2003.

[YZL07] T. Yu, Y. Zhang, and Kwei-Jay L. Efficient Algoritms for Web services Selection with End-to-End QoS Constraints. *ACM Transactions on the Web*, 1(1), May 2007.

[ZBtHA06] J. M. Zaha, M. Barros, A.and Dumas, and A. ter Hofstede A. Lets Dance : A Language for Service Behavior Modeling. Technical Report Preprint no. 4468, Faculty of Information Technology, Queensland University of Technology, February 2006. http ://eprints.qut.edu.au/archive/00004468.

[ZM05] W. Zahreddine and Q-H. Mahmoud. A Framework for Automatic and Dynamic Composition of Personalized Web Services. In *AINA*, pages 513–518. 19th International Conference on Advanced Information Networking and Applications, Taipei, Taiwan, IEEE Computer Society, 28-30 March 2005.

[ZNBB94] A. Zang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proceedings of the ACM SIGMOD Journal*, volume 23, pages 67–78, June 1994.

Résumé

Le développement et l'adoption des technologies associées aux services web permettent aux entreprises d'implanter de nouvelles applications en composant des services existants. Cependant, la mise en œuvre de processus métiers interagissant sur le web reste une tâche complexe. Le concept de service web, basé sur les standards de l'internet, vise à faciliter le développement de ce type de processus et les interactions entre plusieurs partenaires dans le but de produire un service à valeur ajoutée. Mais, paradoxalement, le développement de services créés par chaque entreprise de manière autonome a donné lieu à une hétérogénéité qui pose divers problèmes au moment de l'exécution de la composition obtenue, surtout lorsque celle-ci est munie de propriétés transactionnelles.

L'étude présentée dans ce document nous a permis d'identifier les problèmes liés d'une part, à la composition de services web, et d'autre part à l'association de propriétés transactionnelles à cette composition. Nous nous sommes intéressés à ces deux problématiques qui nous ont conduits à la conception du canevas `Tcows`[1] pour la composition de services web en tenant compte des propriétés transactionnelles des services composants. Le modèle permet aux concepteurs de composer des services et de prendre en compte des contraintes liées à la portée de la transaction et, par le biais de restrictions et de préférences, de fixer la qualité de service requise par les composants. Au moment de l'exécution, s'appuyant sur le concept de communautés de services, le canevas choisit, parmi les services accessibles via une communauté, ceux qui répondent le mieux aux besoins de la composition, tout en respectant ces caractéristiques transactionnelles.

Mots clés : Composition de services web, propriétés transactionnelles de services web, sélection dynamique, communautés de services web.

Abstract

With the development and adoption of web-based technologies, existing companies can establish new partnerships by composing existing services, thus offering new value-added services. However, the implementation of process interacting on the Web still remains a complex and tedious task. The concept of service, based on web technologies aims at facilitating the development of this type of process. Services that enter into compositions with other services may have transactional properties, especially those in the broad area of resource management (e.g. booking services). These transactional properties may be exploited in order to derive composite services which themselves exhibit certain transactional properties.

The work presented in this document led us to identify two main issues : (i) web service composition and (ii) transactional properties associated with this composition. As we are interested in both issues we have designed a framework called `Tcows`[1] which covers the design of compositions of Web services as well as their executions. The framework relies on a model for the composition of Web services associated with transactional properties that takes into account the expression of the transactional properties that composite services are required to fulfil. In addition, compositions can be parameterised so as to allow the end user to impose that certain quality of services constraints and preferences are satisfied by the executions of the composite service. At execution time and relying on web service communities, `Tcows` selects among services accessible through communities, those which best meet the composition's needs. The framework automatically ensures that transactional properties associated with the composition are fulfilled by exploiting the transactional properties of the underlying selected services.

Keywords : Web service composition, transactional properties of Web services, dynamic service selection, Web services communities.

[1]Transactional Composition Of Web Services