

Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web

Sami Bhiri

▶ **To cite this version:**

Sami Bhiri. Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web. Génie logiciel [cs.SE]. Université Henri Poincaré - Nancy 1, 2005. Français. NNT : 2005NAN10116 . tel-01746679v2

HAL Id: tel-01746679

https://theses.hal.science/tel-01746679v2

Submitted on 16 Feb 2006

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.

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web

THÈSE

présentée et soutenue publiquement le 06/10/2005

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Sami Bhiri

Composition du jury

Rapporteurs :	M. Mohand-Saïd Hacid, Professeur à l'Université Claude Bernard, Lyon 1
	M. Jean-Bernard Stefani, Directeur de recherche INRIA, Grenoble
Examinateurs :	M. Vincent Chevrier, Maître de Conférence à l'Université Henri Poincaré, Nancy 1
	M. Claude Godart, Professeur à l'Université Henri Poincaré, Nancy 1
	M. Olivier Perrin, Maître de Conférence à l'Université Nancy 2

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

L'approche «services Web» étend le rôle du Web d'un simple support d'information vers un intergiciel d'applications B2B. Un des concepts intéressants qu'offre cette technologie est la possibilité de définir un nouveau service par composition de services Web existants. Dans cette thèse nous nous intéressons à assurer des compositions fiables de services Web.

Pour ce faire, nous proposons un modèle qui (i) enrichit la description des services Web pour mieux exprimer leurs propriétés transactionnelles et qui (ii) étend et fusionne les systèmes de workflow et les modèles transactionnels avancés (MTA).

Nous proposons trois approches, basées sur ce modèle, pour assurer des compositions fiables. Contrairement aux MTA, la première approche part des spécifications des concepteurs pour déterminer les mécanismes transactionnels permettant d'assurer des exécutions correctes.

La deuxième approche procède par ré-ingénierie du service composé (SC). Elle permet d'améliorer les mécanismes de recouvrement d'un SC après analyse de ses traces d'exécutions.

La troisième approche repose sur le concept de «patron transactionnel», un nouveau concept que nous introduisons. Un «patron transactionnel» est un patron qui combine la flexibilité des workflows et la fiabilité des MTA. Nous définissons un SC en connectant des patrons transactionnels. Nous utilisons un ensemble de règles de cohérence pour assurer que le SC défini est fiable.

La contribution majeure de notre proposition est qu'elle a pu fusionner la flexibilité des systèmes de workflow et la fiabilité des MTA. Ainsi elle permet d'assurer des exécutions correctes selon les critères spécifiés par les concepteurs.

Mots-clès : composition de services Web, modèles transactionnels avancés, systèmes de workflow, fiabilité.

Abstract

The Web services approach is extending the role of the Web from of a support of information interaction to a middleware for B2B interactions. One of the interesting concepts that this technology offers is the possibility to define a new service by combining existing Web services. In this thesis, we are interested in ensuring reliable Web services compositions.

For that purpose, we propose a model that (i) extends Web services description to express better their transactional properties and that (ii) extends and merge workflow systems and advanced transactional models (ATM).

We develop three approaches, based on this model, to ensure reliable Web services compositions. Contrary to the ATM, the first approach starts from the designers requirements to determine the transactional mechanisms allowing to ensure correct executions.

The second approach proceeds by re-engineering of the composite service (CS). This approach allows to improve a CS recovery mechanisms by analysing its executions traces.

The third approach is based on the concept of "Transactional Patterns", a new concept we introduce. A "Transactional Pattern" is a pattern that combines the workflow flexibility and the ATM reliability. We define a CS by connecting together a set of transactional patterns. We use a set of coherence rules to ensure that the defined CS is reliable.

The major contribution of our approach is her ability to merge the workflow systems flexibility and the ATM reliability. Thus, it allows to ensure correct executions according to the criterion specified by the designers.

Keywords : Web services composition, advanced transactional models, workflow systems, reliability.

Remerciements

Table des matières

1 Introduction générale **1**

2 Contexte et problématique **5**

2.1 Contexte de la thèse : Interactions B2B par composition de services Web 5

2.1.1 Intégration d'applications intra-entreprise : A2A 5

2.1.2 Intégration d'applications inter-entreprises : B2B 6

2.1.3 Différences entre A2A et B2B 8

2.1.4 Limite des technologies A2A pour les applications B2B 9

2.1.5 Interactions B2B par composition de services Web 9

2.2 Problématique de la thèse : Comment assurer des exécutions fiables de services Web composés ? 11

2.3 Objectifs, approche et contributions de la thèse 13

2.3.1 Principes directeurs 13

2.3.2 Approche 13

2.3.3 Contributions de la thèse 15

2.4 Conclusion 15

3 Etat de l'art **17**

3.1 Introduction 17

3.2 Modèles Transactionnels Avancés (MTA) 18

3.2.1 La genèse des MTA 18

3.2.2 Modèle des transactions emboîtées 19

3.2.3 Modèle des Sagas 20

3.2.4 Modèle des transactions flexibles 20

3.2.5 Synthèse 22

3.3 Systèmes de Workflow 22

3.3.1 Modèle de workflow 23

3.3.2 Patrons de Workflow 25

3.3.3 Workflows transactionnels 26

3.3.3.1 Introduction . . . 26
3.3.3.2 Les workflows transactionnels comme extensions des MTA . . . 27
3.3.3.3 Les workflows transactionnels comme extension des workflows . . 28
3.3.4 Synthèse . . . 30
3.4 Les services Web . . . 30
3.4.1 Architecture de référence . . . 31
3.4.2 Architecture avancée . . . 32
3.4.3 Coordination transactionnelle de services Web . . . 33
3.4.3.1 Coordination de services Web . . . 33
3.4.3.2 WS-Transaction . . . 34
3.4.3.3 WS-TXM . . . 35
3.4.4 Composition et chorégraphie de services Web . . . 36
3.4.4.1 BPEL4WS . . . 36
3.4.4.2 WS-CDL . . . 37
3.4.5 Autres technologies . . . 37
3.4.5.1 RosettaNet . . . 37
3.4.5.2 ebXML . . . 38
3.4.6 Synthèse . . . 38
3.5 Conclusion . . . 39

4 Modèle de services Web Transactionnels 41
4.1 Service Web transactionnel : SWT . . . 42
4.1.1 Propriétés transactionnelles d'un SWT . . . 42
4.1.2 Modélisation du comportement d'un SWT . . . 43
4.1.3 Diagramme à transition d'états d'un SWT . . . 44
4.1.4 Synthèse . . . 46
4.2 Service Web composé transactionnel : SCT . . . 46
4.2.1 Composition de services Web transactionnels . . . 47
4.2.2 Dépendances entre les services composants d'un SCT . . . 48
4.2.2.1 Dépendance et condition d'activation . . . 51
4.2.2.2 Dépendance et condition d'alternative . . . 52
4.2.2.3 Dépendance et condition d'abandon . . . 53
4.2.2.4 Dépendance et condition de compensation . . . 53
4.2.2.5 Dépendance et condition d'annulation . . . 54
4.2.2.6 Relations sémantiques entre les dépendances . . . 55
4.2.3 Flot de contrôle et flot transactionnel d'un SCT . . . 56
4.2.4 Synthèse . . . 59

4.3 Patrons de composition . . . 60
4.3.1 Patron AND-split . . . 60
4.3.2 Patron AND-join . . . 61
4.3.3 Patron XOR-split . . . 61
4.3.4 Spécification du flot de contrôle d'un SCT . . . 62
4.3.5 Synthèse . . . 63
4.4 Flot transactionnel potentiel . . . 64
4.4.1 Flot transactionnel potentiel d'un patron . . . 64
4.4.2 Flot transactionnel potentiel du patron AND-join . . . 64
4.4.3 Flot transactionnel potentiel du patron XOR-split . . . 65
4.4.4 Flot transactionnel potentiel du patron AND-split . . . 66
4.4.5 Flot transactionnel potentiel d'un flot de contrôle . . . 67
4.4.6 Synthèse . . . 67
4.5 Spécification d'un SCT . . . 67
4.6 Conclusion . . . 69

5 Assurer des compositions fiables de services Web 71
5.1 Approche par validation du SCT . . . 71
5.1.1 Critère de correction des exécutions d'un SCT : ETA . . . 73
5.1.2 Service composé valide . . . 74
5.1.3 Assurer des SCT valides . . . 77
5.1.3.1 Inclusion forte d'un flot transactionnel dans un autre . . . 78
5.1.3.2 Flot transactionnel induit par un ETA . . . 80
5.1.3.3 Méthodologie pour le calcul des `propriétés de validation` . . . 81
5.1.3.4 Calcul du flot transactionnel induit par un ETA . . . 81
5.1.3.5 Calcul des `propriétés de validation` . . . 84
5.1.4 Synthèse . . . 85
5.2 Approche par ré-ingénierie du SCT . . . 85
5.2.1 Extraction de l'ensemble des états de terminaison . . . 87
5.2.2 Découverte du flot transactionnel d'un SCT . . . 89
5.2.3 Amélioration du flot transactionnel d'un SCT . . . 89
5.2.4 Synthèse . . . 92
5.3 Approche par composition de patrons transactionnels . . . 93
5.3.1 Patrons transactionnels . . . 93
5.3.2 Composition de patrons transactionnsels . . . 94
5.3.3 Synthèse . . . 96
5.4 Conclusion . . . 97

6 Mise en oeuvre **99**
6.1 Environnement de conception . . . 99
6.2 Présentation de Bonita . . . 101
6.2.1 Vue générale de Bonita . . . 101
6.2.2 Architecture de Bonita . . . 102
6.3 Extension de Bonita via des plug-in . . . 103
6.3.1 Introduction . . . 103
6.3.2 Structure et rôle des plug-in . . . 104
6.3.2.1 Éléments à déléguer aux plug-in . . . 104
6.3.2.2 Architecture d'un plug-in . . . 105
6.3.3 Adaptation du moteur d'exécution . . . 106
6.4 Exemples de plug-in . . . 107
6.4.1 Plug-in rejouable . . . 107
6.4.2 Plug-in compensation . . . 108
6.5 Conclusion . . . 108

7 Bilan et perspectives **113**
7.1 Travail réalisé et contributions . . . 113
7.2 Perspectives . . . 114

A Patrons de composition **117**
A.1 Patron OR-split . . . 117
A.2 Patron OR-join . . . 117
A.3 Patron XOR-join . . . 118
A.4 Patron m-out-of-n . . . 118

B Fonctions «potentiel» des patrons de composition **119**
B.1 Flot transactionnel potentiel du patron OR-split . . . 119
B.2 Flot transactionnel potentiel du patron OR-join . . . 119
B.3 Flot transactionnel potentiel du patron XOR-join . . . 120
B.4 Flot transactionnel potentiel du patron m-out-of-n . . . 120

C Algorithmes de calcul du comportement transactionnel induit par un *ETA* **123**
C.1 Algorithme d'extraction de la condition d'annulation d'un service s induite par l'*ETA* . . . 123
C.2 Algorithme d'extraction de la condition d'alternative d'un service s induite par l'*ETA* . . . 123

D Demonstrations des lemmes **127**
D.1 Démonstration du lemme 5.1 127
D.2 Démonstration du lemme 5.2 127
D.3 Démonstration du lemme 5.3 128

E Acronymes **129**

Bibliographie **131**

Chapitre 1

Introduction générale

L'évolution récente des technologies de l'Internet, guidée par le langage XML et ses technologies sous-jacentes, étend le rôle du Web d'un simple support d'information vers un intergiciel d'applications B2B. Cette nouvelle vague de l'Internet est guidée par le concept de «services Web». Les services Web peuvent être définis comme des programmes modulaires, indépendants et auto-descriptifs, qui peuvent être découverts et invoqués via Internet ou un intranet. Grâce aux services Web, les entreprises peuvent encapsuler leurs procédés métiers et les publier comme des services, chercher et souscrire à d'autres services et échanger des informations au-delà des frontières des entreprises. L'approche «services Web» est la technologie clé pour automatiser les interactions B2B.

Un des concepts intéressants qu'offre cette technologie, et qui suscite beaucoup d'intérêt, est la possibilité de créer un nouveau service à valeur ajoutée par composition de services Web existants, éventuellement offerts par plusieurs entreprises. La composition de services Web partage plusieurs points communs avec la gestion de procédés d'entreprises. Toutes les deux nécessitent de coordonner un ensemble de tâches (gérer le flot de contrôle et le flot de données entre elles) pour atteindre un objectif commun. Cependant la composition de services Web est plus complexe à cause du degré d'autonomie, d'hétérogénéité et de dynamisme. Différents des composants de procédés métiers traditionnels, les services Web sont généralement fournis par des organisations différentes et indépendamment de tout contexte d'exécution. Puisque chaque organisation possède ses propres règles de travail, les services Web doivent être traités comme des unités strictement autonomes. À cause de cette autonomie et de cette hétérogénéité inhérente aux services Web, il est difficile de prévoir le comportement global d'un service composé.

Le problème auquel nous nous intéressons est comment assurer des compositions fiables de services Web. Par composition fiable, nous entendons une composition dont toutes ses exécutions sont correctes (de point de vue métier). Une exécution correcte est une exécution qui réagit au(x) échec(s) de certain(s) services composants conformément aux besoins des concepteurs du service composé. Les technologies actuelles des services Web sont incapables de résoudre ce problème d'une manière efficace. Ces technologies reposent principalement sur deux concepts forts pré-existants : les systèmes de workflow et les modèles transactionnels avancés (MTA). Les MTA visent à assurer un certain niveau de correction des exécutions d'un ensemble d'opérations encapsulées dans une même unité de traitement appelée transaction. Tandis que les systèmes de workflow s'intéressent plutôt à l'aspect coordination et organisationnel des procédés métiers. Ces deux approches présentent des limites qui ne sont plus acceptables dans le contexte de services Web du fait de l'autonomie des services et du manque de confiance entre les partenaires.

L'approche «workflow» est une technologie clé pour l'automatisation des procédés métiers en permettant leurs modélisations, leurs analyses et leurs exécutions. Émanant des procédés administratifs, les workflows considèrent principalement les problèmes liés à la coordination de tâches et à l'acheminement de documents, et ignorent les problèmes liés à la fiabilité notamment en présence d'échecs.

Les MTA étendent le modèle transactionnel classique en supportant des structures de contrôle plus complexes et en relâchant certaines des propriétés ACID (principalement l'atomicité et l'isolation). Cependant, même ces modèles montrent rapidement leurs limites lorsqu'ils sont appliqués au contexte de procédés métiers qui sortent rapidement du contexte des bases de données traditionnelles. Leurs limites découlent principalement de leur rigidité en terme de structure de contrôle et des contraintes qu'ils imposent aux concepteurs sans que ceux-ci peuvent les négocier.

Plusieurs propositions ont tenté de réconcilier ces deux approches en combinant la flexibilité des workflows et la fiabilité des MTA. Cependant, aucune de ces propositions n'a réussi a fusionner d'une manière efficace les avantages de ces deux technologies (le résultat était toujours au profit d'une approche sur l'autre).

L'objectif de notre travail est de proposer une approche pour assurer des compositions fiables de services Web. Pour ce faire, nous avons choisi d'enrichir la description des services Web avec des propriétés transactionnelles pour mieux exprimer leurs comportements. Ensuite, nous avons développé un modèle de composition de services Web. Ce modèle étend et fusionne les modèles à flots de tâches et les modèles transactionnels. Selon notre modèle, un service composé décrit à la fois un aspect «coordination» et un aspect «transactionnel». D'une part il peut être considéré comme un workflow de services. D'autre part, il peut être considéré comme une transaction structurée où les services composants sont des sous-transactions et les interactions sont des dépendances transactionnelles.

Nous avons développé une première approche, au dessus de ce modèle, qui permet d'assurer des compositions fiables de services Web. L'originalité de notre approche à ce niveau est la flexibilité que nous offrons aux concepteurs pour spécifier leurs besoins en terme de structure de contrôle et de correction. Contrairement aux MTA, nous partons des spécifications des concepteurs pour déterminer les mécanismes transactionnels permettant d'assurer des compositions fiables selon leurs besoins.

Nous avons mis en œuvre deux approches connexes. La première procède par ré-ingénierie des services composés après analyse de leurs traces d'exécutions effectives. Dans cette approche, nous améliorons le comportement transactionnel du service composé en assurant certaines règles de cohérence. Dans la deuxième approche, nous introduisons un nouveau concept appelé «patron transactionnel» que nous utilisons pour définir des services Web composés fiables. La notion de patron transactionnel est l'aboutissement de notre travail. Ce nouveau concept permet de combiner la flexibilité des workflows et la fiabilité des modèles transactionnels. Il étend et fusionne les patrons de workflows et les MTA. Nous définissons un service composé en connectant simplement un ensemble de patrons transactionnels. L'utilisation des patrons permet d'assurer un niveau d'abstraction élevé. La faisabilité de notre approche est démontrée par un environnement de conception de services composés et un prototype de système exécutif de tels services.

Plan de la thèse

Le document est organisé comme suit :

Dans le chapitre 2, nous présentons le contexte de notre travail : mener des applications B2B par composition de services Web. Nous montrons en particulier les différences entre l'intégration des applications intra-entreprise (A2A) et l'intégration des applications inter-entreprises (B2B),

les limites des technologies A2A pour satisfaire les besoins des applications B2B et pourquoi et comment l'approche des services Web émerge comme la technologie clé pour automatiser les interactions B2B. Nous présentons ensuite la problématique de notre thèse : comment assurer des compositions fiables de services Web ? Nous motivons notre problématique via un exemple illustratif et nous synthétisons les limites des technologies actuelles pour résoudre ce problème. Enfin, nous présentons les principes directeurs de notre travail, notre approche pour résoudre la problématique posée et les contributions de notre thèse.

Dans le chapitre 3, nous présentons les deux approches principales où les problèmes abordés présentent des similitudes avec la nôtre. Ces deux approches sont les modèles transactionnels avancés (MTA) et les systèmes de workflow. Nous montrons en particulier les avantages et les limites de chaque approche. Nous présentons ensuite les technologies de services Web liées à notre problématique, notamment celles visant à assurer l'interopérabilité au niveau procédé. Nous montrons comment ces technologies se sont inspirées des MTA et des systèmes de workflow et comment elles ont hérité de certaines de leurs limites.

Les chapitres 4 et 5 constituent le cœur de notre travail. Dans le chapitre 4, nous présentons notre modèle de composition de services Web qui sert de base aux diverses techniques développées au chapitre 5. Au départ, nous introduisons la notion de service Web transactionnel et nous précisons comment nous modélisons son comportement selon ses propriétés transactionnelles. Nous montrons ensuite comment nous combinons un ensemble de services Web transactionnels pour offrir un service plus complexe et à valeur ajoutée. Nous distinguons en particulier le flot de contrôle et le flot transactionnel d'un service Web composé transactionnel (SCT). Nous introduisons la notion de patron de composition et nous illustrons comment nous l'utilisons pour spécifier un SCT (spécifier son flot de contrôle et son flot transactionnel).

Dans le chapitre 5, nous présentons comment nous procédons pour assurer des compositions fiables de SCT. Nous distinguons trois approches principales. La première approche repose sur la validation du modèle de composition du SCT conformément aux besoins des concepteurs. La deuxième approche procède par ré-ingénierie du modèle de composition après analyse des traces d'exécutions. La troisième approche procède par composition de *patrons transactionnels* ; un nouveau concept que nous introduisons.

Dans le chapitre 6, nous illustrons l'implantation de notre approche dans un environnement de conception de services composés. Cet environnement est ensuite intégré dans BONITA un système de workflow développé par l'équipe ECOO. Nous montrons en particulier comment nous avons modifié BONITA, via l'utilisation des plug-in, pour qu'il puisse supporter des services Web transactionnels.

Finalement, le chapitre 7 dresse le bilan et les contributions de notre travail et présente les éventuelles perspectives.

Chapitre 2

Contexte et problématique

Table des matières

2.1 Contexte de la thèse : Interactions B2B par composition de services Web 5
2.1.1 Intégration d'applications intra-entreprise : A2A 5
2.1.2 Intégration d'applications inter-entreprises : B2B 6
2.1.3 Différences entre A2A et B2B 8
2.1.4 Limite des technologies A2A pour les applications B2B 9
2.1.5 Interactions B2B par composition de services Web 9
2.2 Problématique de la thèse : Comment assurer des exécutions fiables de services Web composés ? 11
2.3 Objectifs, approche et contributions de la thèse 13
2.3.1 Principes directeurs 13
2.3.2 Approche 13
2.3.3 Contributions de la thèse 15
2.4 Conclusion 15

2.1 Contexte de la thèse : Interactions B2B par composition de services Web

2.1.1 Intégration d'applications intra-entreprise : A2A

Une entreprise est souvent composée d'un ensemble de départements et d'unités organisationnelles. Le fonctionnement de la compagnie pour traiter un service donné est exprimé par un ensemble d'interactions et de coopérations entre ces diverses parties. Chacune de ces unités doit accomplir un sous traitement particulier.

Pour un meilleur rendement au niveau des délais et des coûts, chaque département commence à automatiser ses activités sans prendre en considération les diverses interactions et coopérations à mener avec les autres unités. Ceci a donné naissance à des applications commerciales isolées.

Des étapes manuelles sont donc nécessaires pour mener à bien les interactions et les coopérations entre les différentes unités impliquées dans un processus commercial particulier. Ces interventions manuelles augmentent les délais de traitement et les coûts des produits ou des services rendus.

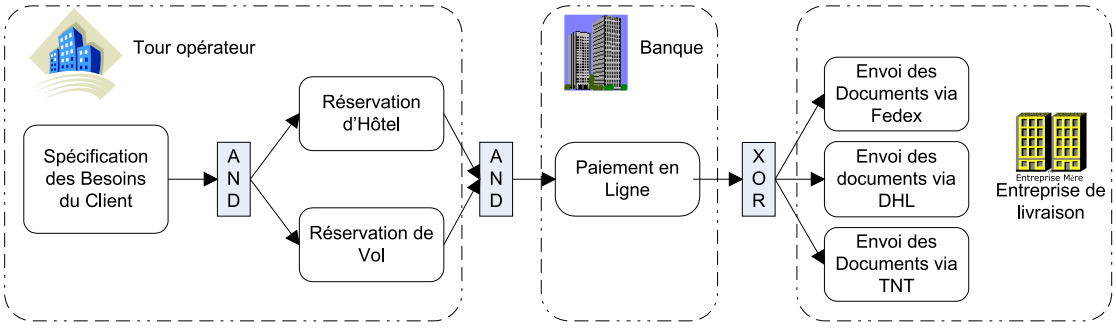
Ainsi, automatiser cette intégration est susceptible de réduire les délais et les charges et d'améliorer le rendement de l'entreprise. L'intégration d'applications intra-entreprise est souvent désignée sous l'anglicisme A2A (Application-to-Application) [Bus03]. Les technologies développées à ce propos concerne les intergiciels et les systèmes de workflows [WFM95; vdAvH02]. Les intergiciels, du RPC[1] [BN83] au courtiers messages [IBM02; Cor01; Web02; Gro01] et en passant par les courtiers objets [Gro02; Gro97; Pla99], veillent à masquer l'hétérogénéité technique des systèmes à intégrer et à fournir une abstraction indépendante de leurs environnements. Les systèmes de workflow se situent à un niveau plus haut. Ils permettent de décrire explicitement la logique d'exécution d'un procédé métier. Les systèmes de workflow reposent sur les intergiciels comme une plate-forme d'interactions qui masque l'hétérogénéité des différentes applications à intégrer.

2.1.2 Intégration d'applications inter-entreprises : B2B

Si on généralise cette idée au delà des frontières des entreprises, on s'aperçoit que les compagnies font elles même partie d'un processus commercial plus complet. En effet, peu d'entreprises transforment toutes seules des matières premières en produits finaux. Ces entreprises sont obligées d'interagir et de coopérer avec d'autres partenaires pour mener à bien leurs travaux.

De nos jours, les entreprises expriment de plus en plus leurs relations commerciales via le Web et tendent à automatiser leurs interactions et leurs coopérations. Cette coopération inter-entreprises et cette automatisation de procédés d'interaction manuels via le Web sont souvent désignés sous l'anglicisme «Business-to-Business» (B2B). Les applications B2B désignent l'utilisation de systèmes informatiques (e.g serveurs Web, services réseaux, bases de données, etc.) pour mener des interactions commerciales (e.g échanges de documents commerciaux, vente de produits, etc.) entre les différents partenaires [MBB+03; DRR03].

Figure 2.1 Application B2B : service d'organisation de voyage en ligne.



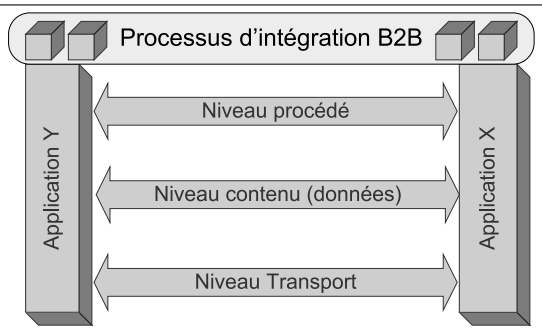
Par exemple, considérons une application d'organisation de voyage en ligne dont la logique d'exécution est illustrée par la figure 2.1. D'abord, le client spécifie ses besoins en terme de destinations et d'hôtels via la tâche «**S**pécification des **B**esoins du **C**lient (SBC)». Ensuite, l'application lance deux tâches en parallèles «**R**éservation du **V**ol (RV)» et «**R**éservation d'**H**ôtel (RH)» pour réserver un vol et un hôtel selon les choix du client. Une fois les réservations faites, la tâche «**P**aiement en **L**igne (PL)» permet au client d'effectuer le paiement en ligne. Enfin, les documents de voyage (billet d'avion et réservation d'hôtel) sont envoyés au client via l'une des tâches «**E**nvoi de **D**ocuments par **F**edex (EDF)», «**E**nvoi de **D**ocuments par **D**HL (EDD)» ou «**E**nvoi de **D**ocument par **T**NT (EDT)».

[1] RPC est l'acronyme de l'anglicisme «Remote Procedure Call» pour appel de procédures (méthodes) à distance

L'integration d'applications B2B se situe à différents niveaux : **transport**, **donnée** et **procédé métier** [MBB+03; DRR03] (voir figure 2.2).

- **Niveau transport** : Ce niveau s'intéresse aux échanges de messages entre les différentes parties. Il est fort probable que les différents partenaires utilisent des protocoles de communication et des environnements hétérogènes. Par exemple, l'agence de voyage et le tour opérateur peuvent utiliser respectivement JavaRMI (Remote Method Invocation) [Sun03] et IBM's MQSeries [IBM03] pour leurs communications internes. L'objectif de l'intégration à ce niveau est de masquer cette hétérogénéité en traduisant et en convertissant les messages entre les divers protocoles.
- **Niveau donnée** : Ce niveau s'intéresse aux échanges des documents (commerciaux) entre les différents partenaires. Il est indispensable que les différents acteurs parviennent à déterminer le type des documents, leurs contenus et la sémantique de chaque champ de donnée. L'hétérogénéité structurelle découle de l'utilisation de différents formats de données. L'hétérogénéité sémantique est le résultat des interprétations différentes d'un même concept. Par exemple, un champ de donnée «prix» peut être interprété différemment : toute taxe comprise ou hors taxe. L'objectif de l'intégration à ce niveau est de masquer l'hétérogénéité structurelle et sémantique des documents échangés pour assurer une compréhension commune des informations échangées. Les outils de conversion, de transformation et d'intégration d'informations sont nécessaires pour assurer la reconciliation entre les différents représentations, vocabulaires et sémantiques. Par exemple, si le tour opérateur utilise xCBL (XML Common Business Library) [eCO03] pour représenter les documents commerciaux et la banque attend les documents en cXML (Commerce XML) [cXM03], il y aura besoin de conversion entre ces deux formats.
- **Niveau procédé** : Ce niveau s'intéresse à l'aspect conversationnel entre les partenaires (c.à.d le procédé métier commun). Avant de s'engager dans une transaction, les partenaires doivent se mettre d'accord sur les procédures de leur procédé commercial commun. La sémantique des interactions doit être bien définie de façon à ce que l'interprétation de messages, les actions autorisées et les réponses attendues ne soient pas ambiguës. L'objectif des interactions à ce niveau est de permettre à des partenaires autonomes et hétérogènes d'interagir en ligne, de publier leurs fonctionnalités et leurs capacités, et de s'engager dans des interactions d'égal à égal avec d'autres partenaires. L'interopérabilité à ce haut niveau est un problème difficile parce qu'elle nécessite de comprendre la sémantique de procédés des partenaires [MBB+03].

Figure 2.2 Niveaux d'interopérabilité B2B.



2.1.3 Différences entre A2A et B2B

Les applications A2A et B2B partagent le même objectif global : l'intégration des applications. Cependant, l'intégration dans le cadre du B2B pose plus de problèmes à cause de la nature dynamique et inter-entreprises. En effet, les interactions intra-entreprise sont plutôt de nature statique contrairement aux coopérations inter-entreprises qui évoluent dans un environnement dynamique. En plus, dans le cadre du A2A, les applications à intégrer appartiennent à la même entreprise contrairement aux applications B2B où elles appartiennent à des organisations différentes ce qui pose d'autres problèmes d'interopérabilité, en particulier de confidentialité.

Ces deux facteurs sont les causes majeures des différences entre les applications A2A et les applications B2B (voir figure 2.3).

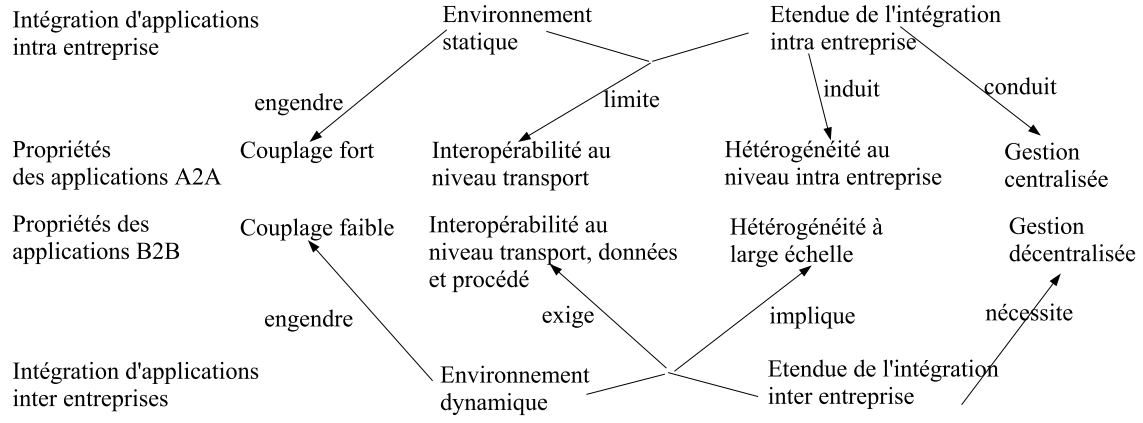
Le niveau de l'integration : l'intégration des applications intra-entreprise se situe généralement au niveau transport. En effet, vue la nature statique des applications A2A, la logique de l'intégration ainsi que l'échange des documents sont souvent «codés en dûr» (même dans le cas où ils sont explicitement définis indépendamment des détails d'implémentation). Cependant dans le cadre du B2B, et comme nous l'avons mentionné ci-dessus, l'intégration des applications inter-entreprises nécessite plus d'efforts au niveau des données échangées ainsi qu'au niveau du procédé commun mettant en oeuvre l'intégration.

La nature de l'intégration : dans le cadre du A2A, les applications sont fortement couplées vue la nature statique de l'intégration. En effet, les applications lors de leurs créations sont sensées travailler ensemble, même si la logique de l'intégration est indépendante de l'implémentation de ces applications. Cependant dans le cadre du B2B, les applications sont indépendantes de tout contexte d'exécution. Leur intégration se fait d'une façon plus dynamique et en conséquence elles sont faiblement couplées.

L'échelle de l'hétérogénéité : l'échelle de l'hétérogénéité des applications A2A est relativement limitée par rapport à celle des applications B2B. Ceci est dû en fait à l'étendue intra entreprise et à la nature statique des applications A2A par comparaison à l'étendue inter entreprises et à la nature dynamique des applications B2B.

La gestion de l'integration : vue que les applications A2A sont gérées dans le même domaine, leurs gestions sont en général centralisées. Cependant, cette gestion centralisée n'est plus possible dans le cadre du B2B à cause du manque de confiance entre les divers partenaires.

Figure 2.3 Différences entre A2A et B2B.



2.1.4 Limite des technologies A2A pour les applications B2B

Ces différences entre les applications A2A et B2B rendent les technologies A2A limitées pour répondre aux problèmes des applications B2B. En effet, ces technologies ont été développées pour travailler dans un environnement fortement couplé où la gestion est plutôt centralisée.

Les protocoles d'intergiciel conventionnels, comme le 2PC[2], sont conçus sur l'hypothèse qu'ils ne vont pas supporter des interactions inter-entreprises. Par exemple, au niveau transactionnel ces intergiciels supposent un coordinateur centralisé des transactions et la possibilité au coordinateur de bloquer des ressources. Cependant dans le cadre du B2B, les aspects de manque de confiance et de confidentialité s'opposent à une coordination centralisée. C'est pourquoi 2PC doit être redéfini pour travailler d'une façon totalement décentralisée et doit être étendu pour permettre plus de flexibilité. Les mêmes arguments sont valables pour tous les protocoles de coordination et d'interaction et en général pour plusieurs autres propriétés fournies par les intergiciels conventionnels, comme la fiabilité et la garantie de livraison.

En plus ces technologies s'intéressent plus à l'interopérabilité au niveau d'échange de messages, mais moins à l'interopérabilité au niveau des données et des procédés.

Cependant le plus grand problème des technologies A2A réside dans leur façon de procéder pour masquer l'hétérogénéité en se basant sur des convertisseurs. En effet, cette solution est plus adaptée au sein d'une seule entreprise où l'échelle d'hétérogénéité est limitée. Cependant, l'application de cette solution dans le cadre du B2B implique qu'une entreprise doive supporter autant de convertisseurs (aux différents niveaux) que de technologies hétérogènes existantes chez ses partenaires. Ce qui est inconcevable vu l'échelle de l'hétérogénéité et la nature dynamique des applications B2B.

2.1.5 Interactions B2B par composition de services Web

La technologie «services Web» est la technologie clé permettant l'intégration des applications via le Web. Les services Web peuvent être définis comme des programmes modulaires, généralement indépendants et auto descriptifs, qui peuvent être découverts et invoqués via Internet ou un intranet. Grâce aux services Web, les entreprises peuvent encapsuler leurs procédés métiers et les publier comme des services. Elles peuvent chercher et invoquer d'autres services et échanger des informations au delà des frontières des entreprises. La composition de services Web offre la possibilité de créer un service à valeur ajoutée en combinant de services Web existants, éventuellement offerts par plusieurs entreprises. Dans notre exemple de référence, l'application d'organisation de voyage en ligne peut être définie par composition de services de réservations, de paiement en ligne et de livraison externalisés par d'autres entreprises (voir figure 2.5).

La contribution de services Web pour dépasser les limites des technologies A2A implique trois aspects essentiels : la standardisation, l'architecture orientée-service et l'environnement d'égal à égal.

Standardisation : les services Web combinent des aspects de développement basés sur les composants et le Web [Jac99; MM99; Pre97]. Comme les composants, les services Web exposent une interface qui peut être ré-utilisée sans se soucier de l'implémentation des services. Différents des technologies composants actuelles, les services Web ne sont pas accédés via des protocoles spécifiques à un modèle objet comme DCOM [CC95], RMI[Sun03], ou IIOP [Gro95]. A la place, les services Web sont accédés via des protocoles Web et des formats de données omniprésents, comme HTTP et XML qui sont independents des fournisseurs. C'est une différence clé entre les services

[2]2PC est l'acronyme de l'anglicisme «two-Phase Commit» pour le protocole transactionnelle à deux phases mettant en oeuvre des transactions atomiques dans un environnement distribué.

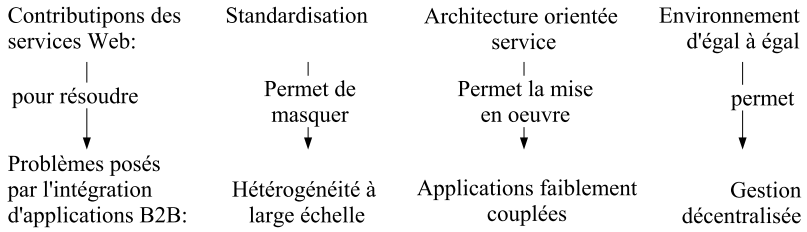
Web et les technologies actuelles basées sur les composants. Grâce à l'utilisation des protocoles et des formats de données omniprésents, les services Web assurent un niveau d'interopérabilité plus élevé que les technologies actuelles. Si **la standardisation** était importante pour les technologies A2A, elle **est indispensable** dans le cadre du B2B **afin de masquer l'hétérogénéité à large échelle**.

Architecture orientée services : les services Web évoluent selon une architecture orientée service. Une telle architecture suppose que chaque entreprise externalise son savoir faire comme un service. Les entreprises peuvent, par la suite, combiner leurs connaissances et offrir un nouveau service plus complexe à valeur ajoutée par composition de services Web existants. Les services Web sont dévéloppés et gérés par des entreprises différentes et sont définis indépendamment de tout contexte d'exécution (par opposition, par exemple, à deux objets CORBA qui sont développés par la même équipe pour travailler ensemble dans un contexte prédéfini). Par conséquence, les services Web sont des composants individuels **faiblement couplés** qui peuvent être ré-utilisés et agrégés plus facilement dans différents contextes d'exécution.

Environnement d'égal à égal : cet aspect de l'approche «services Web» concerne la re-définition des protocoles d'intergiciel pour travailler de façon d'égal à égal et inter-entreprises. Ce qui était accompli par une plate-forme centralisée doit être re-défini en des protocoles qui peuvent travailler de manière décentralisée et à travers des domaines de confiance. La figure 2.4 récapitule les contributions des services Web pour répondre aux problèmes posés par les applications B2B.

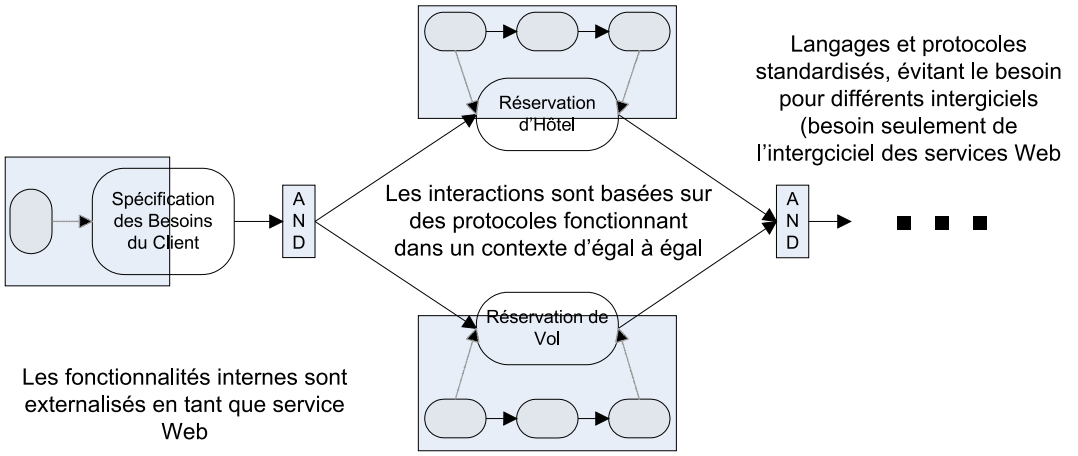
Le Web est lui même caractérisé par un haut niveau de standardisation qui lui a permis de fonctionner et de réussir sans coordination centralisée (à part le DNS) et a permis son expansion [ACKM04]. les technologies Web sont maintenant largement acceptées. Elles permettent les interactions entre les humains et les applications (via les navigateurs Web et les serveurs Web). Il est donc naturel que l'integration d'application B2B se fonde sur le Web et essaye de suivre le même chemin de succès en terme de standardisation et de décentralisation.

Figure 2.4 Contributions des services Web pour répondre aux problèmes d'intégration d'applications B2B.



La figure 2.5 résume cette discussion sur comment les services Web permettent l'intégration d'applications B2B. La figure montre que chaque partie expose ses opérations internes comme un service (Web), qui agit en conséquence comme un point d'entrée aux systèmes d'information locaux. Les interactions entre les entreprises se produisent d'égal à égal et prennent place via des protocoles standardisés, qui sont conçus pour supporter des interactions avec les mêmes propriétés que les intergiciels conventionnels, mais sans la présence d'une plate-forme centralisée [ACKM04]. Il incombe aux intergiciels de services Web de faciliter l'exécution de tels protocoles et de cacher au programmeur les complexités intrinsèques des problèmes d'intégration d'applications.

Figure 2.5 Intégration d'applications B2B par composition de services Web : service composé d'organisation de voyage en ligne



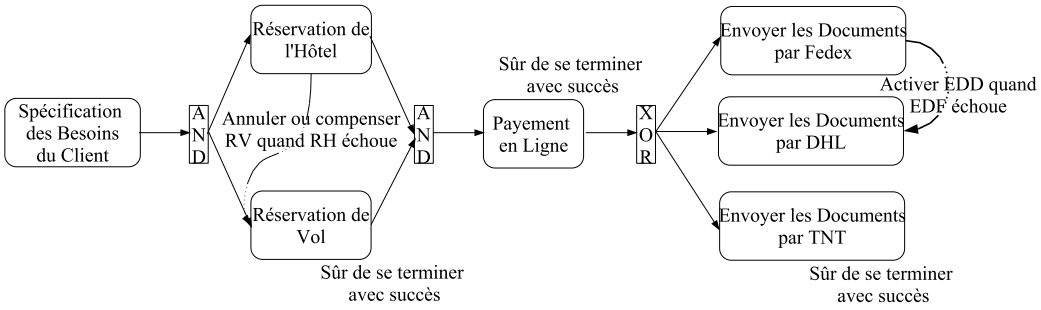
2.2 Problématique de la thèse : Comment assurer des exécutions fiables de services Web composés ?

La composition de services Web offre la possibilité de fournir un nouveau service à valeur ajoutée à partir de services Web existants, éventuellement offerts par d'autres entreprises. La composition de services Web partage plusieurs besoins avec la gestion de procédés métiers. Toutes les deux nécessitent de coordonner l'exécution d'un ensemble de tâches (gérer le flot de contrôle et le flot de données entre elles) comme une unité cohérente ayant un but global commun. En plus, ils ont besoin de fournir des garanties de fiabilité, de disponibilité, et de passage à l'échelle. Cependant la tâche de composer des services Web est plus complexe du fait de l'autonomie et de l'hétérogénéité de ces derniers, et de la nature dynamique de la composition [P. 02]. Différents des composants de procédés métiers traditionnels, les services Web sont généralement fournis par des organisations différentes et indépendamment de tout contexte d'exécution. Puisque, chaque organisation possède ses propres règles de travail, les services Web doivent être traités comme des unités strictement autonomes. A cause de cette hétérogénéité et de cette autonomie inhérente aux services Web, il est difficile de prévoir le comportement d'un service Web composé.

Un des problèmes majeurs, auquel nous nous intéressons, est comment garantir des compositions fiables de services Web. Par composition fiable, nous entendons toute composition dont toutes les exécutions sont correctes. Une exécution correcte est une exécution qui réagit au(x) échec(s) des services composants conformément aux besoins des concepteurs. Pour gérer les échecs des services composants, les concepteurs spécifient en plus du flot de contrôle des mécanismes de recouvrement. Par exemple, afin de garantir la fiabilité du service d'organisation du voyage en ligne, les concepteurs spécifient que (voir figure 2.6) (i) les services RV, PL et EDT sont sûr de se terminer avec succès, (ii) en cas d'échec du service RH, il faut annuler ou compenser le service RV (selon son état courant) et (iii) en cas d'échec du service EDF, il faut activer le service EDD comme une alternative. Le problème qui se pose à ce niveau est comment vérifier / assurer que la spécification d'un service composé garantit des exécutions fiables conformément aux exigences des concepteurs.

Les propositions actuelles ne permettent pas de résoudre ce problème d'une manière efficace. En effet, on peut dire que les technologies des services Web à l'état actuel sont des versions

Figure 2.6 Service Web composé d'organisation de voyage : flot de contrôle enrichi par des mécanismes de recouvrement.



standardisées (via l'utilisation de XML comme format de données et du Web comme infrastructure d'invocation) des technologies A2A adaptées au contexte des services Web (une architecture d'égal à égal et un environnement décentralisé). Si cette adaptation est nécessaire (pour masquer l'hétérogénéité technique à large échelle et offrir un environnement d'interactions faiblement couplé et décentralisé), elle n'est pas suffisante pour assurer l'interopérabilité au niveau donnée et procédé (indispensable pour mener des interactions B2B).

En effet, aux niveaux «transport» et «données», les services Web reposent principalement sur le protocole de transport SOAP [W3C03a] pour échanger les messages commerciaux, le langage WSDL [W3C03c] pour décrire les opérations offertes par un service Web et l'annuaire UDDI [W3C03b] pour publier et rechercher les services. Ces trois approches peuvent être vues comme la standardisation des technologies A2A adaptée à fonctionner dans un contexte faiblement couplé et un environnement décentralisé (par exemple, appel de procédure à distance pour SOAP, interface IDL de CORBA pour WSDL et annuaire de services CORBA pour UDDI).

Au niveau «procédé», les services Web reposent sur des langages tels que BPEL4WS [BM03] ou WS-CDL [KBRL04] pour spécifier respectivement l'orchestration ou la chorégraphie de services Web, et WS-AT [ea03], WS-BA [ea04] et WS-TXM [AFI+] pour mener des coordinations transactionnelles de services Web. BPEL4WS et WS-CDL adoptent une approche similaire à celle des workflows (basée sur le langage XML) pour spécifier la logique d'invocations des services Web au sein d'une composition ou d'une chorégraphie. WS-AT, WS-BA et WS-TXM reposent sur les modèles transactionnels avancés (MTA) déjà existants.

Malgré leurs intérêts, ces deux approches (workflows et MTA) présentent des limites qui ne sont plus acceptables dans le contexte de services Web du fait du l'autonomie des services et le manque de confiance entre les partenaires.

En effet, d'une part les workflows considèrent principalement les problèmes liés à la coordination de tâches et à l'acheminement de documents et ignorent les problèmes liés à la correction et à la fiabilité des exécutions. Si cette lacune a pu être gérée dans les applications A2A par des méthodes informelles, au cas par cas, et via des interventions humaines, elle n'est plus acceptable dans le cadre du B2B à cause de l'autonomie des services et du manque de confiance entre les partenaires.

D'autre part, les MTA se sont avérés limités pour l'automatisation des procédés métiers et par la suite pour la composition de services Web. Les limites des MTA découlent principalement de leur rigidité en terme de structure de contrôle et des contraintes qu'ils imposent aux concepteurs. En effet, les MTA ne peuvent pas supporter des applications ayant des structures de contrôle aussi complexes que les workflows et/ou ayant des composants avec des sémantiques transactionnelles différentes de celles exigées par le modèle adopté. Les MTA ne sont pas applicables dans

une approche bottom-up où le procédé métier et ses différents composants sont prédéfinis. Une application doit s'adapter aux contraintes (en terme de structure et de sémantiques transactionnelles) du modèle transactionnel adopté. De plus les MTA imposent leurs critères de correction et ne donnent pas la liberté aux concepteurs d'exprimer leurs besoins.

C'est pourquoi, l'adaptation des technologies A2A (telle quelles) pour fonctionner dans un environnement à large échelle d'hétérogénéité, décentralisé et faiblement couplé n'est pas suffisant pour que les services Web arrivent à satisfaire les besoins des applications B2B. Il faut en plus étendre ces technologies existantes afin d'assurer l'interopérabilité au niveau donnée et procédé.

2.3 Objectifs, approche et contributions de la thèse

2.3.1 Principes directeurs

Notre objectif est donc de proposer une approche pour assurer des exécutions correctes de services Web composés. Compte tenu du contexte de notre travail et des besoins décrits ci-dessus, nous avons identifié quatres dimensions à prendre en compte pour résoudre notre problématique. Ces dimensions représentent les principes directeurs de notre approche (voir figure 2.7).

- **Flexibilité** : contrairement aux MTA, il faut laisser la liberté aux concepteurs de spécifier leurs besoins tant au niveau de la structure de contrôle de l'application qu'au niveau du degré de correction exigé. Ceci sous entend pouvoir supporter des structures de contrôles assez complexes et de fournir un moyen aux concepteurs pour spécifier leurs critères de correction.
- **Correction** : contrairement aux workflows, il faut pouvoir assurer des exécutions correctes conformément aux besoins des concepteurs (structure de contrôle et degré de correction). Ceci implique développer un ensemble de techniques permettant d'assurer des compositions fiables compte tenu des spécifications des concepteurs.
- **Besoins des procédés métiers** : il faut que l'ensemble des techniques développées et des sémantiques utilisées soit approprié aux caractéristiques des procédés métiers, notamment la longue durée des exécutions et les tâches non annulables. Par exemple, assurer la fiabilité en vérifiant le critère de correction exigé à la fin de l'exécution (annuler le travail effectué si le critère de correction n'est pas vérifié) n'est pas adéquat au contexte des procédés métiers. En effet, d'une part il n'est pas pratique d'annuler un travail qui a duré des jours voir des mois. D'autre part, il n'est pas toujours évident d'annuler le travail effectué dans le contexte de procédés métiers.
- **Contexte des services Web** : il faut respecter le contexte des services Web en particulier le couplage faible et la gestion décentralisée.

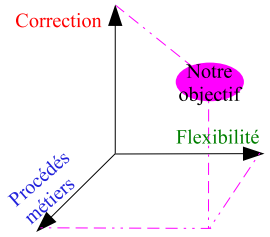
Un dernier principe directeur, qui est commun à toute approche, est de garder le maximum de simplicité possible tant au niveau des concepts utilisés qu'au niveau de leur utilisation.

La figure 2.7 récapitule cette discussion et illustre que pour résoudre notre problématique, il faut arriver à intégrer les trois premières dimensions. Dans cette figure (et les figures qui suivent au chapitre suivant), nous avons ignoré la quatrième dimension parce que l'intégration des trois premières dimensions est la plus difficile à réaliser. La quatrième dimension peut être ensuite intégrée.

2.3.2 Approche

Afin d'atteindre notre objectif tout en prenant en compte les principes directeurs que nous nous sommes fixés ci-dessus, nous avons adopté une démarche qui repose sur les points suivants :

Figure 2.7 Les trois principales dimensions à prendre en compte pour résoudre notre problématique.



- Enrichir la description des services Web pour mieux exprimer leurs propriétés transactionnelles. En effet, l'utilisation potentielle d'un service est étroitement liée à sa description. Cependant, WSDL décrit seulement la syntaxe des opérations offertes par un service et ne renseigne rien sur leurs sémantiques (transactionnelles). Connaître les propriétés transactionnelles (des propriétés) d'un service permettra d'assurer des compositions plus fiables. Ainsi dans notre exemple, il est évident que le service composé doit connaître les propriétés transactionnelles de ses services composants (comme par exemple que RV se termine toujours avec succès et peut être compensé, que les services EDD et EDT se terminent toujours avec succès, etc.) pour spécifier ses mécanismes de gestion d'échecs et de recouvrement. Dans notre approche, nous considérons les propriétés transactionnelles suivantes : *rejouable*, *compensable* et *pivot*. Nous utilisons les diagrammes à transition d'états pour modéliser le comportement transactionnel d'un service Web.
- Développer un modèle qui fusionne la fiabilité des modèles transactionnelles et la flexibilité des systèmes de workflow. Ce modèle permet la spécification des services Web composés (SWC) et sert de base à l'ensemble des techniques développées. Dans notre approche, nous distinguons le flot de contrôle et le flot transactionnel d'un SWC. Le flot de contrôle décrit l'ordre d'exécution des services composants. Le flot transactionnel définit les mécanismes de gestion d'échecs et de recouvrement. Dans notre exemple (voir figure 2.6), le flot de contrôle est représenté par des lignes continues et le flot transactionnel par des lignes pointillées. Nous utilisons des patrons de composition pour définir les flots de contrôle des services composés. Nous exploitons l'aspect transactionnel caché de ces patrons pour définir les flots transactionnels des Services Web composés.
- Fournir un moyen aux concepteurs pour exprimer leurs besoins de fiabilité en terme de semi atomicité[3]. Pour ce faire, nous utilisons dans notre approche, la notion «Ensemble d'États de Terminaison Acceptés» (ETA). Comme son nom l'indique, cet ensemble détient les états dans lesquels les concepteurs acceptent que leur service composé se termine. Une exécution du service composé est correcte *si et seulement si* elle se termine dans un état de terminaison accepté. ETA est ainsi le critère de correction spécifié par les concepteurs. Un service composé est valide *si et seulement si* toutes les exécutions de ses instances sont correctes.
- Développer les techniques nécessaires permettant d'assurer des exécutions correctes de services Web composés selon les besoins des concepteurs (en terme de structure de contrôle

[3]Le terme semi atomicité est notre traduction de l'anglicisme «failure atomicity». La semi atomicité est la relaxation de la propriété d'atomicité du modèle ACID. Elle fait référence à des exécutions correctes même en présence d'échecs de certains sous-transactions.

et de de semi atomicité exigée). Dans une première approche, nous avons développé un ensemble de techniques (algorithme(s) + règles) permettant de déterminer l'ensemble de mécanismes de gestion d'échecs et de recouvrement (flot transactionnel) assurant des exécutions correctes selon le critère de correction *ETA* et le flot de contrôle spécifiés par les concepteurs.
Nous avons mis en œuvre deux approches connexes. La première procède par ré-ingénierie des services composés après analyse de leurs exécutions effectives. Dans cette approche, nous améliorons le flot transactionnel du service composé en utilisant des règles de cohérence. Dans la deuxième approche, nous avons introduit un nouveau concept appelé «patron transactionnel» que nous utilisons pour définir des services Web composés. La notion de patron transactionnel est l'aboutissement de notre travail. Ce nouveau concept permet de combiner la flexibilité des workflows et la fiabilité des modèles transactionnels. Il peut être considéré comme une extension des patrons de workflows et des MTA. Un service composé peut être simplement défini en connectant un ensemble de patrons transactionnels.

2.3.3 Contributions de la thèse

Les contributions de notre thèse sont :

- Un modèle de composition de services Web qui fusionne les modèles à flot de tâches et les modèles transactionnels, deux technologies fortes, souvent considérées concurrentes. Selon notre modèle, un service composé révèle à la fois un aspect transactionnel et un aspect coordination. D'une part, il peut être considéré comme un modèle à flots de services. D'autre part, il peut être vu comme une transaction structurée, où les services composants sont les sous-transactions et les interactions sont des dépendances transactionnelles. Par rapport aux MTA, notre modèle permet des structures de contrôle plus complexes. Par comparaison aux systèmes de workflow, notre modèle intègre le flot transactionnel une dimension relativement ignorée. En plus l'utilisation des patrons comme concept de base de notre modèle assure un haut niveau d'abstraction et de simplicité. Ce modèle ne se limite pas à spécifier des compositions de services Web, mais il sert de base à l'ensemble des techniques développées pour assurer des compositions fiables.
- Une approche transactionnelle dont la correction est définie par les concepteurs. Notre approche offre la flexibilité aux concepteurs de spécifier leurs besoins en terme de structure de contrôle et de semi atomicité. Ainsi contrairement aux MTA, nous suivons le sens inverse en partant des spécifications des concepteurs pour définir les mécanismes transactionnels nécessaires pour assurer la fiabilité selon leurs besoins.
- L'introduction du concept de «patron transactionnel» qui peut être considéré comme l'aboutissement de notre approche. Un *patron transactionnel* est un patron de composition de services Web révélant un aspect coordination et un aspect transactionnel en même temps. Un patron transactionnel est un patron de composition (aspect coordination) enrichi par des mécanismes transactionnels (aspect transactionnel). Les patrons transactionnels permettent des compositions fiables de services Web d'une façon relativement simple et pratique.

2.4 Conclusion

Trois facteurs principaux ont permis l'expansion des services Web comme une technologie clé pour automatiser les interactions B2B : la standardisation, l'architecture orientée service et l'interaction d'égal à égal. En effet, la standardisation a permis de masquer l'hétérogénéité

à large échelle. L'architecture orientée service a rendu possible le développement d'applications faiblement couplées. Enfin, l'environnement d'égal à égal a engendré des exécutions décentralisées.

Grâce à ces facteurs, les services Web et ses technologies sous-jacentes sont en train de transformer le Web d'un support interactif à un intergiciel universel d'interactions B2B. L'un des concepts qui suscitent beaucoup d'intérêt pour mener des applications B2B est le pouvoir de composer de nouveaux services Web plus complexes et à valeurs ajoutées à partir de services offerts.

Cependant, l'hétérogénéité et l'autonomie inhérente aux services Web rendent difficile la prévision du comportement d'un service composé, notamment en présence d'échecs de services composants. L'un des problèmes majeurs qui se pose à ce niveau, et auquel nous nous intéressons dans cette thèse, est comment assurer des compositions fiables de services Web.

Plusieurs technologies ont été proposées. Ces propositions (proches pour un même niveau) reposent principalement sur deux concepts forts : les systèmes de workflow pour l'orchestration de services Web et les modèles transactionnels avancés pour assurer un certain niveau de correction des exécutions. Les technologies actuelles sont des versions standardisées de ces deux concepts adaptées pour fonctionner dans un environnement faiblement couplé et décentralisé. Cependant ces deux approches présentent des limites qui ne sont plus acceptables dans le contexte de services Web du fait de leurs autonomies et de manque de confiance entre les partenaires.

L'objectif de notre travail est de proposer une approche pour assurer des exécutions fiables de services Web composés. Pour ce faire, nous avons choisi d'enrichir la description des services Web avec des propriétés transactionnelles pour mieux exprimer leurs comportements. Ensuite, nous avons développé un modèle de composition de services Web. Ce modèle réconcilie et étend les modèles à flots de tâches et les modèles transactionnels vers un point de convergence. Ce modèle sert de base pour les diverses techniques développées pour assurer des exécutions correctes de services Web composés. L'originalité de notre approche à ce niveau est la flexibilité que nous offrons aux concepteurs pour spécifier leurs besoins en terme de structure de contrôle et de correction. Contrairement aux MTA, nous partons des spécifications des concepteurs pour définir les mécanismes transactionnels permettant d'assurer la fiabilité des exécutions selon les besoins de correction. Nous avons mis en œuvre deux approches connexes qui complémentent cette première approche. La première procède par ré-ingénierie des services composés après analyse de leurs exécutions effectives. Dans la deuxième approche, nous avons introduit un nouveau concept appelé «patron transactionnel» que nous utilisons pour définir des services Web composés ayant des exécutions fiables.

Chapitre 3

Etat de l'art

Table des matières

3.1 Introduction 17
3.2 Modèles Transactionnels Avancés (MTA) 18
3.2.1 La genèse des MTA 18
3.2.2 Modèle des transactions emboîtées 19
3.2.3 Modèle des Sagas 20
3.2.4 Modèle des transactions flexibles 20
3.2.5 Synthèse 22
3.3 Systèmes de Workflow 22
3.3.1 Modèle de workflow 23
3.3.2 Patrons de Workflow 25
3.3.3 Workflows transactionnels 26
3.3.4 Synthèse 30
3.4 Les services Web 30
3.4.1 Architecture de référence 31
3.4.2 Architecture avancée 32
3.4.3 Coordination transactionnelle de services Web 33
3.4.4 Composition et chorégraphie de services Web 36
3.4.5 Autres technologies 37
3.4.6 Synthèse 38
3.5 Conclusion 39

3.1 Introduction

Plusieurs thématiques de recherche ont abordé des problèmes similaires à ceux posés par notre problématique. Avant de présenter ces approches, il est important de rappeler les principales dimensions à intégrer pour résoudre notre problématique (que nous avons identifié au chapitre précédent) :

- flexibilité : laisser la liberté aux concepteurs de spécifier leurs besoins tant au niveau de la structure de contrôle de l'application qu'au niveau du degré de correction exigé,
- correction : pouvoir assurer des exécutions correctes conformément aux besoins des concepteurs (structure de contrôle et degré de correction),

- besoins des procédés métiers : il faut que l'ensemble des techniques développées et des sémantiques utilisées soient appropriées aux caractéristiques des procédés métiers, notamment la durée longue des exécutions et les tâches non annulables.

Nous distinguons dans la littérature deux approches principales où les problèmes abordés présentent des similitudes avec notre problématique et qui nous ont été sources d'inspiration. Ces deux approches sont les modèles transactionnels avancés (MTA) [ELLR90; (Ed92; GMS87; GMGK+91; Mos81; WS92] et les systèmes de workflow [JB96; Law97; Coa96; AM97; Ell99; vdAvH02; Fis00]. Ces deux approches ont énormément influencé les technologies actuelles des services Web. Elles abordent le problème d'exécution d'un ensemble de tâches encapsulées comme une seule unité de traitement. Pour des raisons historiques, elles abordent ce problème de deux points de vue différents. Basés sur le modèle transactionnel ACID, les MTA ont pour but d'assurer des exécutions correctes d'un ensemble de tâches encapsulées comme une seule unité de traitement appelée transaction. Issus des procédés administratifs, les workflows s'intéressent plutôt aux aspects coordination et organisationnel d'un ensemble d'activités encapsulées comme une seule unité de traitement appelée procédé d'exécution.

Dans ce chapitre, nous présentons ces deux approches ainsi que les technologies des services Web émergeantes en relation avec notre travail. Nous montrons en particulier les limites de chacune de ces approches en étudiant leurs capacités d'intégrer les trois dimensions de notre problématique.

3.2 Modèles Transactionnels Avancés (MTA)

Les MTA ont été proposés comme solution pour dépasser les limites du modèle transactionnel ACID afin de supporter des applications plus complexes que de simples interactions avec une base de données.

3.2.1 La genèse des MTA

L'approche transactionnelle a émergé initialement dans le contexte des bases de données. Le modèle des transactions ACID permet d'encapsuler une séquence d'opérations de lecture ou d'écriture sur une base de données comme une unité atomique de traitement, appelée transaction, et qui vérifie les propriétés suivantes :

- Atomicité : l'exécution d'une transaction est atomique ; soit toutes les opérations de la transaction sont exécutées soit aucune n'est exécutée,
- Cohérence : une transaction prise individuellement fait passer la base de données d'un état cohérent vers un état cohérent,
- Isolation : les effets d'une transaction en cours d'exécution sont invisibles aux transactions concurrentes,
- Durabilité : les effets d'une transaction validée sont permanents et ne peuvent plus être remis en cause, ni par une défaillance, ni par une autre transaction.

L'intérêt de l'approche transactionnelle est qu'elle assure (i) une exécution correcte (en terme de fiabilité) d'une transaction prise individuellement et (ii) des exécutions correctes (en terme de cohérence de données) de plusieurs transactions concurrentes[4]. L'approche transactionnelle a été étendue et adoptée par des applications plus avancées voulant bénéficier de cette fiabilité et de cette correction. Ces applications varient de simples interactions avec plusieurs bases de données à l'automatisation de procédés métiers. Cependant, le modèle ACID s'est avéré limité pour

[4]Notre problématique concerne plutôt le premier point

supporter des telles applications qui (i) nécessitent des structures de contrôle plus complexes qu'une simple séquence d'opérations, (ii) ont des exécutions de longues durées et (iii) où la sémantique de l'annulation n'est pas aussi évidente que pour les opérations de bases de données classiques.

Les modèles transactionnels avancés (MTA) ont été alors proposés pour répondre à ces nouveaux besoins. Dés lors, une transaction désigne plus généralement toute unité de travail encapsulant un ensemble d'opérations. Les MTA permettent de grouper des opérations dans une structure hiérarchique et de relâcher, dans la plupart des cas, certaines des propriétés ACID (généralement l'isolation et l'atomicité). Ces modèles peuvent être classés selon plusieurs caractéristiques, à savoir la structure de la transaction, la concurrence intra transaction, les dépendances d'exécution, les besoins d'isolation et de semi atomicité [WS97].

Dans la suite, nous présentons quelques modèles transactionnels qui nous semblent être les plus appropriés pour supporter des applications à flot de tâches.

3.2.2 Modèle des transactions emboîtées

Une transaction emboîtée [Mos81] est un ensemble de sous-transactions qui peuvent récursivement contenir d'autres sous-transactions, formant ainsi un arbre de transactions emboîtées. Une transaction fille ne peut démarrer qu'après que sa transaction mère ait démarré. Une transaction mère ne peut se terminer que lorsque toutes ses transactions filles sont terminées. La validation d'une transaction est conditionnée par la validation de sa transaction mère. L'abandon d'une transaction entraîne l'abandon de toutes ses transactions filles. Cependant, une transaction mère peut survivre à l'échec d'une de ses transactions filles en exécutant, par exemple une sous-transaction alternative. Une transaction est isolée par rapport à toutes transactions n'appartenant pas à sa descendance.

Les transactions emboîtées ouvertes [WS92] relâchent l'isolation en rendant visibles les résultats des sous-transactions validées aux autres transactions emboîtées s'exécutant en parallèle. Ainsi une sous-transaction peut valider et libérer ses ressources avant que sa transaction mère termine correctement et valide. En cas d'abandon d'une transaction, le travail effectué par ses sous-transactions validées peut être sémantiquement compensé en exécutant des sous-transactions de compensation. Une transaction de compensation t^{-} annule sémantiquement le travail d'une transaction t. Ainsi l'état d'une base de données avant et après l'exécution de la séquence tt^{-} est un état éventuellemnt différent de l'état initial avant l'exécution, mais considéré comme cohérent.

L'accès aux résultats de transactions qui seront ultérieurement compensées peut entraîner des incohérences. Pour éviter ce problème, seules les transactions qui commutent avec celles qui valident peuvent accéder à leurs résultats. Deux transactions s et t commutent si la séquence d'exécution $s\ t$ est équivalente (a le même effet au niveau des résultats et de l'état de la base de données) à la séquence d'exécution $t\ s$. Le problème est ainsi résolu puisque la séquence d'exécution $t\ s\ t^{-}$ sera équivalente à $t\ t^{-}\ s$ et par suite équivalente à s.

◇ ◇ ◇ ◇

La contribution majeure des transactions emboîtées est l'extension de la structure plate des transactions ACID vers une structure hiérarchique. Les principaux avantages des transactions emboîtées par rapport aux transactions ACID sont :

- Un niveau de modularité plus élevé : Chaque transaction peut être décomposée en une structure hiérarchique de sous-transactions.
- La gestion d'erreur à un niveau de granularité plus fin : Les actions de recouvrement sont prises au niveau de la sous-transaction échouée.

– Le parallélisme intra transaction : En effet, les sous-transactions non-conflictuelles peuvent s'exécuter en concurrence.

Bien que le modèle des transactions emboîtées assure plus de flexibilité par rapport aux transactions ACID, il reste limité à des applications base de données et loin de supporter des applications telle que les procédés métiers. Ceci est dû à sa structure de contrôle primitive par comparaison à celle des modèles à flot de tâches.

3.2.3 Modèle des Sagas

Le modèle des Sagas [GMS87] a été proposé comme une solution pour les transactions de longue durée. L'idée de base est de permettre à une transaction de libérer certaines ressources qu'elle a acquises avant de valider. Une transaction de longue durée ou Saga est une séquence de sous-transactions, T_1, ..., T_n, qui peuvent s'intercaler avec d'autres transactions. Chaque sous-transaction T_i est une transaction ACID qui préserve la cohérence de la base de données. Les exécutions partielles d'une saga sont inacceptables. Si une saga abandonne (suite à l'échec de l'une de ses sous-transactions) alors le travail réalisé jusqu'à lors (par les précédentes sous-transactions) doit être compensé. Ainsi, chaque sous-transaction T_i possède une transaction de compensation C_i qui annule sémantiquement son travail. Plus formellement, le système assure que : soit la séquence T_1, ..., T_n est exécutée en cas de non-problème, soit T_1, T_2, ..., T_i, C_i, ..., C_2, C_1 en cas d'échec de la sous-transaction T_i. Dans sa première version, un seul niveau d'emboîtement est autorisé. Ce modèle initial a été ensuite étendu [GMGK+91].

◇ ◇ ◇ ◇

Le modèle des Sagas, comme les transactions emboîtées, adopte une structure hiérarchique pour définir une transaction structurée. Il repose sur la compensation pour relâcher la propriété d'isolation. Ce qui permet de réduire l'isolation au niveau des sous-transactions, d'éviter le blocage de ressources inutilement pour de longues durées et d'augmenter ainsi la concurrence inter transactions.

Cependant, le modèle des sagas présente certaines lacunes qui limitent l'étendue de son application. En effet, il présente comme les transactions emboîtées, une structure de contrôle limitée par rapport aux besoins des procédés métiers. En plus il suppose certaines contraintes qui ne sont pas forcement respectées dans beaucoup d'applications. Par exemple, il faut pouvoir découper à l'avance l'application à supporter en sous-transactions, où chacune possède une transaction de compensation. En plus la propriété d'atomicité qu'il préserve n'est pas approprié au contexte des procédés métiers notamment de longue durée.

3.2.4 Modèle des transactions flexibles

Le modèle des transaction flexibles [ELLR90; ZNBB94; ARS92; SABS02] a subi plusieurs évolutions depuis sa proposition. Initialement, les transactions flexibles [ELLR90] ont été proposées comme un modèle transactionnel convenable dans un environnement multi bases de données. Une transaction flexible est un ensemble de tâches, chacune ayant un ensemble de sous-transactions fonctionnellement équivalentes. Un ensemble de dépendances d'exécution sur les sous-transactions, incluant des dépendances d'échec, de succès, ou des dépendances liées à des événements externes, peut être spécifié. Pour relâcher l'isolation, les transactions flexibles utilisent la compensation. L'atomicité globale est relâchée en permettant la spécification d'états de terminaison acceptés dans lesquels certaines transactions peuvent avoir avorté.

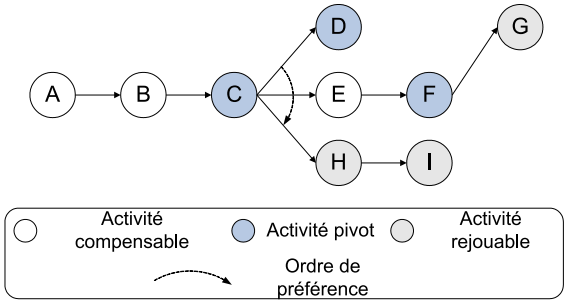
Le modèle des transactions flexibles a été ensuite étendu et adopté pour supporter des applications plus avancées [ZNBB94; ARS92]. [SABS02] adopte les sémantiques transactionnelles de tâche **rejouable**, **compensable** et **pivot** [MRKS92]. Une tâche t est dite **rejouable** si elle se termine toujours avec succès après un nombre fini d'activation. t est dite **compensable** si son travail peut être sémantiquement annulé. Une tâche est dite **pivot** si une fois qu'elle se termine avec succès, ses effets ne peuvent pas être compensés [SABS02].

[ZNBB94] et [SABS02] exploitent ces propriétés transactionnelles pour définir des transactions flexibles et correctes. Une transaction flexible doit respecter certaines règles :

- toute tâche entre deux pivots ou avant le premier pivot doit être compensable,
- après un pivot, plusieurs branches alternatives peuvent exister avec un ordre de préférence entre elles,
- la dernière branche alternative après un pivot doit être sûre de se terminer. Ceci implique que toutes les tâches de la dernière branche alternative doivent être rejouables.

Ces propriétés assurent qu'une transaction flexible (i) est recouvrable en arrière tant que son premier pivot n'est pas exécuté et (ii) termine toujours une fois ce dernier exécuté. La figure 3.1 illustre un exemple de transaction flexible $\mathcal{T}$. Les tâches A, B et E sont compensables. Les tâches C, D et F sont pivots. Les tâches H, I et G sont rejouables. $\mathcal{T}$ est recouvrable en arrière tant que son premier pivot C n'est pas exécuté puisque A et B sont compensables. Après l'exécution de C s'offre trois alternatives dont la dernière est sûre de se terminer puisque H et I sont rejouables. De même $\mathcal{T}$ est recouvrable jusqu'à C si les des deux autres alternatives échouent. Enfin, après l'exécution de son deuxième pivot F, $\mathcal{T}$ est sûre de se terminer puisque G est sûre de se terminer.

Figure 3.1 Un exemple de transaction flexible.



◇ ◇ ◇ ◇

Le modèle des transactions flexibles est un modèle hybride à mi-chemin entre l'approche transactionnelle et les systèmes de gestion de procédés. L'originalité de ce modèle est l'utilisation de propriétés transactionnelles en plus de la compensation permettant d'atteindre plus de flexibilité tout en préservant un certain degré de correction. Ainsi, l'utilisation des alternatives comme mécanismes de recouvrement en avant permet d'éviter d'annuler la totalité du travail effectué en cas d'échec et permet de continuer l'exécution et atteindre l'objectif désiré en suivant un autre chemin alternatif.

Le degré de flexibilité élevé, par comparaison aux modèles des sagas et des transactions emboîtées, a élargi l'étendue des transactions flexibles au-delà des applications centrées bases de données. Cependant, malgré ce succès relatif ce modèle reste limité à quelques applications spécifiques et inadéquat pour les modèles à flot de tâches à large échelle. Comme les deux modèles précédents, ceci est dû à sa structure de contrôle qui reste primitive par rapport à celle

des services composés et à l'ensemble des contraintes qu'il définit et qui doivent être respectées par les concepteurs d'applications.

3.2.5 Synthèse

Le modèle transactionnel traditionnel a évolué au cours du temps pour incorporer des structures de transaction plus complexes et relâcher les propriétés d'atomicité et d'isolation. Les modèles transactionnels avancés, résultats de cette évolution, ont permis de dépasser les limites du modèle ACID pour satisfaire les besoins de nouvelles applications avancées. Bien que ces modèles aient été proposés dans des contextes plus ou moins similaires, ils partagent certaines caractéristiques :

- une structure plus complexe qu'une simple séquence d'opérations,
- un degré de concurrence et de parallélisme intra transaction plus élevé,
- le relâchement des propriétés d'atomicité et d'isolation,
- l'adoption de nouvelles sémantiques transactionnelles comme la compensation (qui correspond au recouvrement en arrière) et l'alternative (qui correspond au recouvrement en avant).

Cependant, malgré le degré de flexibilité qu'ils apportent, ces modèles restent limités dans le cadre des modèles à flot de tâches [RS95; WS97; GC02; SABS02] à large échelle. Leurs limites découlent principalement :

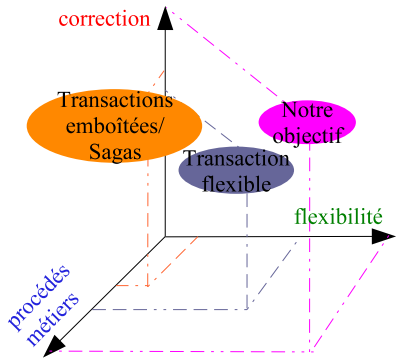
- de leurs structures de contrôle limitées : bien que les MTA aient étendu la structure plate du modèle ACID vers des structures plus complexes, elles restent primitives par comparaison à celles des services Web composés qui nécessitent des constructeurs de branchement et de synchronisation plus évolués. Par exemple, aucun des modèles présentés ci-dessus n'est capable de modéliser le service d'organisation de voyage en ligne, aussi simple soit-il.
- des contraintes qu'ils exigent : les MTA imposent des contraintes, au niveau de la structure et des sémantiques transactionnelles adoptées, que les concepteurs doivent respecter lors de la spécification de leurs applications. De plus, en adoptant un modèle donné, les concepteurs doivent adhérer au critère de correction implicitement défini par l'ensemble des règles prédéfinies. Ceci rend les MTA plus appropriés pour des approches top-down que pour des approches bottom-up. Par approche top-down nous entendons une approche qui part d'un MTA et essaye de modéliser l'application selon ce modèle en respectant ses règles et ses contraintes. Par approche bottom-up, nous entendons une approche qui part d'une application existante (avec un flot de contrôle et un ensemble de services existants) et essaye d'assurer sa fiabilité.

  En conclusion, nous pouvons dire que les MTA ont étendu le modèle ACID pour assurer plus de flexibilité. Cependant, ils restent limités et n'arrivent pas à intégrer d'une manière efficace les trois dimensions de notre problématique (voir figure 3.2).

3.3 Systèmes de Workflow

Les systèmes de workflow facilitent la description, la modélisation, l'analyse et l'exécution des procédés métiers. Une *définition de procédé workflow* (ou plus simplement un *modèle de procédé*) est la représentation formelle d'un procédé. Un modèle de procédé spécifie principalement les activités et leurs interdépendances (le flot de contrôle et le flot de données), les ressources et les programmes informatiques associés à chaque activité. Un système de gestion de workflow (WfMS) est un système informatique qui transforme la représentation explicite d'un modèle de procédé

Figure 3.2 Limite des MTA à intégrer les principales dimensions de notre problématique.



dans un format interne et exécutable et fournit un environnement opérationnel pour l'exécution, l'administration et la surveillance des procédés.

Dans ce qui suit, nous présentons le modèle de référence de workflow. Nous présentons ensuite les «patrons de workflow» un concept simple et fort pour définir des modèles de workflow. Nous montrons en particulier, à ce niveau, comment cette approche a pu intégrer les dimensions «flexibilité» et «procédés métiers» et a (relativement) ignoré la dimension «correction». Nous présentons ensuite les «workflows transactionnels» qui ont tenté de remédier à cette lacune en intégrant des sémantiques transactionnelles au workflow afin d'assurer un certain degré de fiabilité.

3.3.1 Modèle de workflow

Tout les systèmes de workflows sont centrés au tour du concept de procédé métier. Le modèle de référence définit un procédé comme «une procédure où des documents, des informations, ou des tâches sont transmis entre les participants selon un ensemble de règles définies, pour atteindre ou contribuer à un but commercial / métier global» [WFM95]. Un (modèle de) workflow est une représentation d'un procédé métier dans un format interpretable par la machine. Un système de gestion de workflow est «un système qui définit, gère et exécute les workflows via l'exécution d'un logiciel dont l'ordre d'exécution est guidé par la représentation en machine de la logique du workflow» [WFM95].

Un modèle de workflow est un graphe acyclique dans lequel les nœuds représentent les étapes d'exécution et les arcs représentent le flot de control et le flot de données entre ces étapes. Le meta model proposé par WfMC [5] [WFM95] distingue les composants décrits ci-dessous.

- Modèle de procédé : un modèle de procédé est la représentation d'un procédé dans une forme qui permet sa manipulation automatique (modélisation, exécution) par un système de gestion de workflow. Un modèle de procédé est constitué d'un réseau d'activités et des dépendances entre elles, des critères pour spécifier le démarrage et la terminaison d'un procédé et des informations sur les activités individuelles (participants, applications, données informatiques associées, etc.) [WFM95]. Les activités et les procédés ont des données en entrée et en sortie. Les données d'entrée et de sortie sont représentées comme des ensembles d'éléments de données, appelés conteneurs.
- Activité : une activité est une étape dans un procédé. Chaque activité a un nom, un type,

[5] Workflow Management Coalition

des pre et des post conditions et des contraintes d'ordonnancement. Elles peuvent être des activités programme ou des activités procédé. A chaque activité programme, un programme est associé qui sera exécuté quand l'activité sera exécutée. Une activité est exécutée en l'attribuant à des utilisateurs qui sont capables de l'exécuter. Chaque utilisateur a une liste des activités qui nécessitent d'être exécutées. A chaque activité procédé est attribué un procédé. Ainsi tout un procédé est exécuté quand l'activité est exécutée. Les activités procédés sont utilisés pour une conception modulaire. Chaque activité a un conteneur des données en entrée et un conteneur des données résultats.

- Flot de contrôle : un modèle de procédé décrit chaque unité de travail, mais aussi la séquence adéquate d'unité de travail pour atteindre un certain objectif. Un flot de contrôle définit l'ordre dans lequel les activités sont exécutées en spécifiant des connecteurs de contrôle entre les activités.
- Données de procédé : A chaque modèle correspond un ensemble de données qui décrivent toutes les informations nécessaires pour son exécution. Ces données incluent (i) des informations requises en entrée des activités, (ii) des informations requises pour l'évaluation des conditions et (iii) des données qui doivent être échangées entre les activités. Ainsi, nous distinguons :
  - le conteneur d'entrée : ensemble de variables et structures typées qui sont utilisées comme entrée à l'application invoquée.
  - le conteneur de sortie : ensemble de variables et structures typées dans lesquelles les résultats de l'application invoquée sont stockés.
- Flot de données : un flot de données est spécifié via des connecteurs de données entre les activités mettant en œuvre une série de correspondance entre des conteneurs de données en sortie et des conteneurs des données en entrée permettant l'échange d'information entre les activités. La définition des données du procédé comprend d'une part la spécification des conteneurs d'entrée et de sortie, des activités et des conditions, et d'autre part la spécification du flot de donnée entre les activités. Soit A et B deux activités, si B utilise en entrée des données qui sont supposées être produites par A, cette dépendance de données entre A et B est exprimée par un connecteur de données entre A et B.
- Conditions : les conditions spécifient quand certains événements se produisent. Il y a trois types de conditions. `Les conditions de transitions` sont associées aux connecteurs de contrôle et spécifient quand un connecteur est évalué à vrai ou à faux. `Les conditions de démarrage` spécifient quand une activité va être démarrée : par exemple, quand tous ses connecteurs de contrôle entrants seront évalués à vrai, ou quand un d'eux sera évalué à vrai. `Les conditions de sortie` spécifient quand une activité est considérée terminée. Après l'exécution d'une activité, `sa condition de sortie` est vérifiée. Si elle est vraie alors l'activité s'est proprement terminée, sinon l'activité est re-ordonnancé pour être exécutée.

◇ ◇ ◇ ◇

L'approche «workflow» intègre parfaitement les dimensions «flexibilité»[6] et «procédés métiers». En effet un modèle de workflow est **la représentation d'un procédé métier** dans un format interprétable par la machine. Et un système de gestion de workflow permet l'exécution de ce modèle de workflow.

Cependant, le modèle de workflow (et par conséquence l'approche «workflow») ignore la dimension «correction». En effet, il se contente seulement de modéliser le flot de contrôle et le

[6]Rappelons que par cette dimension «flexibilité» nous entendons : donner la liberté aux concepteurs de définir le flot de contrôle (par comparaison aux MTA qui imposent un modèle à suivre). Il est important de ne pas la confondre avec les workflows flexibles qui permettent une interprétation flexible du flot de contrôle.

flot de données et ne considère pas les mécanismes de recouvrement en cas d'échec (voir figure 3.5).

3.3.2 Patrons de Workflow

Séparer la logique d'exécution des détails techniques d'implémentation est intéressant dans le sens où il permet d'analyser le modèle de procédé et de raisonner dessus. Plusieurs langages ont été proposés pour la spécification des modèles de workflow. Certains langages se basent sur les techniques de modélisations existantes comme les réseaux de Petri [vdAvH02], les diagrammes d'activités [Gro99; CLSMS+00; WWWD96], l'algèbre des procédés [MPW92], etc. D'autres langages sont spécifiques.

Cette variété de langages et des concepts rend difficile une compréhension commune des systèmes existants et par la suite leur comparaison. En plus elle pénalise l'interopérabilité entre les systèmes de workflows.

Dans le but de remédier à ces problèmes et afin de donner une vue commune, abstraite et assez complète des interactions dans les systèmes de workflow, Van der Aalst et al [vdABtHK00] ont défini les patrons de workflows dans lesquels ils décrivent d'une façon abstraite les différentes formes d'interactions recensées dans les systèmes de workflow existants.

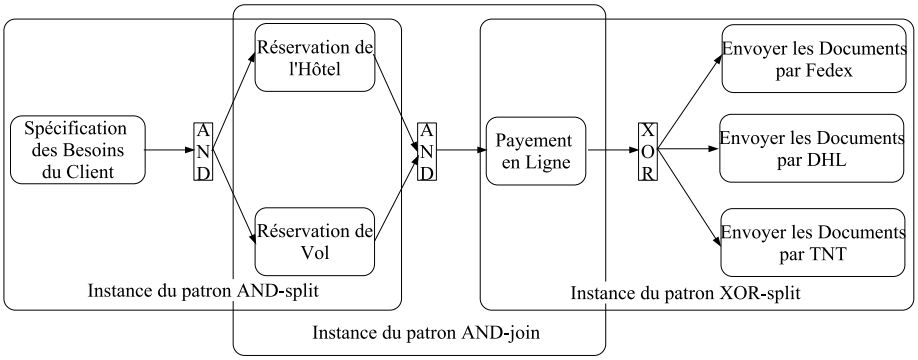
D'une façon générale, un patron est une description abstraite d'une forme récurrente dans un contexte spécifique [GHJV95]. Gamma et al [GHJV95] spécifie quelques patrons de conception qui décrivent les plus petites interactions récurrentes dans les systèmes orientés objet. Les patrons de conception fournissent une indépendance par rapport aux technologies d'implémentation.

Un patron de workflow est une description abstraire d'une classe d'interactions. Les patrons de workflows varient de constructeurs de flot de contrôle simples comme le routage séquentiel à des patrons complexes nécessitant des mécanismes de routage plus avancés comme le patron «discriminator». Van der Aalst et al [vdABtHK00] distingue 20 patrons pertinents classifiés en 6 catégories :

- Patrons basiques : cette première classe regroupe des patrons basiques qui spécifient des concepts élémentaires de flot de contrôle définis par WFMC [WFM95]. Cette classe regroupe les patrons *sequence*, *parallel split*, *synchronisation*, *exclusive choice* et *simple merge*.
- Patrons de branchement et de synchronisation avancés : cette classe s'intéresse à des patrons de branchement et de synchronisation avancés. Cette classe regroupe les patrons *multi-choice*, *synchronizing merge*, *multi-merge* et *discriminator*.
- Patrons structurels : cette classe regroupe des patrons de structuration comme les boucles. Cette classe regroupe les patrons *arbitrary cycles* et *implicit termination*.
- Patrons d'instances multiples : cette classe regroupe des patrons où une ou plusieurs activités peuvent être instanciées plusieurs fois en même temps. Cette classe regroupe les patrons d'instances multiples *without synchronisation*, *with a prior design time knowledge*, *with a prior runtime knowledge*, et *without a prior runtime knowledge*.
- patrons basés états : comme son nom l'indique, cette classe décrit un ensemble de patrons où les interactions sont basées sur le changement d'états des activités. Cette classe regroupe les patrons *deferred choice*, *interleaved parallel routing* et *milestone*.
- Patrons d'annulation : cette classe regroupe des patrons permettant d'annuler une activité en attente d'exécution ou tout un workflow. Cette catégorie regroupe les patrons *cancel activity* et *cancel case*.

La figure 3.3 illustre comment nous pouvons décrire d'une façon abstraite et relativement simple le flot de contrôle du service d'organisation de voyage en ligne en utilisant les patrons *parallel split* (AND-split), *synchronisation* (AND-join) et *exclusive choice* (XOR-split).

Figure 3.3 Spécification du flot de contrôle du service d'organisation de voyage en ligne en utilisant les patrons AND-split, AND-join et XOR-split.



◇ ◇ ◇ ◇

L'utilisation des «patrons de workflow» pour définir des modèles de workflow est une idée intéressante pour les raisons suivantes :

- leur simplicité relative : grâce au niveau d'abstraction qu'ils présentent et leurs représentation intuitive (graphique), les «patrons de workflow» permettent de masquer la complexité des langages de spécification des modèles de workflow.
- leurs cotés pratiques : les «patrons de workflow» comme tout patron de conception, représente une bonne pratique déduite à partir des systèmes existants et résultat d'une bonne expérience. Les patrons augmentent la réutilisabilité et le niveau de compréhension entre les concepteurs.

En plus, l'abstraction des «patrons de workflow» n'exclut pas des raisonnements formels. Au contraire, un patron de workflow peut être formalisé en utilisant les langages de modélisation de workflow existants, tirant profit des différentes méthodes développées pour chaque langage.

Cependant, les «patrons de workflows» permettent seulement de modéliser le flot de contrôle et sont insuffisants pour modéliser les mécanismes de recouvrement en cas d'échecs. Les «patrons de workflows» nécessitent une extension si l'on désire qu'ils puissent traiter le comportement transactionnel de services Web composés.

3.3.3 Workflows transactionnels

3.3.3.1 Introduction

Les systèmes de workflow ont toujours été sujets à de nombreuses critiques à cause de l'absence de mécanismes de correction et de fiabilité [AAAM97]. Le terme de workflow transactionnel [SR93] a été introduit pour reconnaître la pertinence des transactions dans le contexte des workflows pour assurer un certain degré de correction et de fiabilité. Les workflows transactionnels concernent l'exécution coordonnée de plusieurs tâches et suggèrent l'utilisation sélective de propriétés transactionnelles pour des tâches individuelles ou pour des workflows tout entiers. Ils fournissent les mêmes fonctionnalités que les workflows en terme d'automatisation des exécutions des procédés métiers. En plus, ils visent à assurer des exécutions fiables et correctes en présence de la concurrence et des échecs.

Ceci n'implique pas que les workflows sont similaires ou équivalents aux transactions des bases de données, ou qu'ils supportent toutes les propriétés ACID. Néanmoins, des tels workflows

partagent les objectifs de quelques MTA dans le sens où ils veillent à imposer des contraintes transactionnelles à un ensemble d'activités. Par comparaison aux modèles transactionnels avancés, les workflows transactionnels s'intéressent aux problèmes de cohérence d'un point de vue métier plutôt que d'un point de vue base de données. La portée des workflows transactionnels s'étend au-delà de celle des transactions bases de données et MTA.

Deux approches principales ont été adoptées pour étudier et définir des workflows transactionnels. La première approche part des modèles transactionnels avancés en essayant de les étendre. La deuxième approche considère les workflows transactionnels comme des workflows enrichis par un comportement et une sémantique transactionnelle.

3.3.3.2 Les workflows transactionnels comme extensions des MTA

Dans cette approche, les MTA sont enrichis pour incorporer des concepts de workflow et étendre leurs fonctionnalités et leurs applicabilités. Le degré auquel chacun de ces modèles incorpore les caractéristiques transactionnelles varie et dépend largement des besoins (comme la flexibilité, l'atomicité et l'isolation des exécutions de tâches individuelles et des multiples instances de workflows, etc.) du procédé métier qu'il essaye de modéliser.

Les ATMs (Activities/Transaction Model) [DHL90; DHL91] permettent d'intégrer des transactions ACID et des activités qui ne doivent pas être ACID. Une activité de longue durée est modélisée comme un ensemble d'unités d'exécution qui peuvent consister en d'autres activités ou transactions, de manière récursive. Le flot de contrôle et le flot de données peuvent être spécifiés statiquement dans un script ou dynamiquement par des règles événement-condition-action. Le recouvrement est supporté par la spécification de règles de compensation ou de transaction de contingence qui représentent des alternatives.

Le TSME (Transaction Specification and Execution Environement) [GHKM94] est un système programmable qui supporte la spécification des workflows transactionnels et la configuration des gestionnaires de transactions pour implanter les modèles désirés. Il fournit un langage de spécification pour exprimer les différents types de dépendances intra et inter-transactions et les critères de correction que la transaction doit satisfaire.

Similaire à TSME, *transaction toolkits* permet l'implantation par programmation des modèles de transactions avancées en fournissant un ensemble de primitives de transactions. En utilisant une interface de programmation C++, ASSET [BDG+94] permet à un concepteur de transaction de compiler des spécifications de transactions avancées dans un code exécutable. ASSET se base sur des primitives de transactions dérivées du formalisme ACTA [7] [CR94].

Pour démontrer que les modèles de workflow sont un sur-ensemble des modèles transactionnels avancés, le projet Exotica [MAGK95; AAA+96] a décrit des méthodes et des outils pour implanter des modèles de transactions avancées au dessus de Flowmark (le système de workflow qui est le prédécesseur de IBM MQ Series). L'idée de base a été de fournir un modèle de workflow étendu qui intégre des concepts de transactions avancées. En particulier, les auteurs ont montré comment les SAGAs et les transactions flexibles pourraient être implantées au dessus de Flowmark. L'utilisateur peut définir une activité de compensation pour chaque activité de procédé. Un preprocesseur transforme ces spécifications en FDL (Flowmark Definition Language) en insérant des chemins de compensation après une activité ou groupe d'activités, qui sont exécutées de manière conditionnelle après l'échec d'une activité.

[7] ACTA est un métamodèle de transactions qui permet la spécification des modèles de transactions avancées

3.3.3.3 Les workflows transactionnels comme extension des workflows

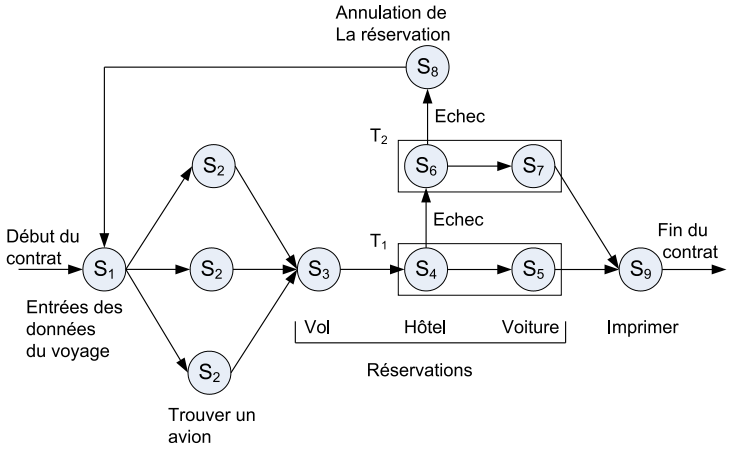
Cette approche utilise un modèle de workflow qui est senser supporté des procédés métiers et l'étend en utilisant des sémantiques transactionnelles pour assurer la fiabilité, la cohérence, et d'autres caractéristiques transactionnelles. Nous distinguons en particulier le modèle des contracts [WR92] et le système METEOR [RS95; KS95; SKM+96] que nous décrivons ci-dessous.

Le modèle des contrats

Un contrat est une exécution cohérente et tolérante aux pannes d'une séquence d'actions prédéfinies (étapes ou steps) selon un scénario prédéfini (script) qui décrit l'enchaînement des étapes. Les étapes constituent les éléments unitaires d'un contrat. Une étape est une transaction ACID qui implémente un traitement particulier (réservation de vol, réservation d'hôtel). Un script décrit la structure (flot de contrôle) du contrat. Un des points clefs du modèle réside dans la notion de recouvrement en avant. En cas d'incident, le gestionnaire de contrat réinstancie les contrats interrompus dans l'état cohérent le plus récent et continue l'exécution du script.

La figure 3.4 présente les étapes d'une réservation pour un voyage d'affaires. Intuitivement, à partir des données du voyage, il faut consulter les horaires d'avions, faire la réservation du vol. de l'hôtel et d'une voiture pour finir le contrat. S1. S2. ..., S9 représentent les étapes du contrat. Le concepteur de ces étapes n'a pas à prendre en compte des aspects tels que la concurrence d'accès, la communication, la synchronisation. les pannes du système, ou encore le recouvrement. C'est au gestionnaire de contrats de prendre en compte ces aspects. Le concepteur de contrats peut définir quelles étapes doivent être regroupées au sein d'une transaction ACID. Dans la figure 3.4, les actions «réserver l'hôtel et voiture» doivent être groupées au sein d'une seule transaction ACID. Il peut décrire des relations entre transactions comme, par exemple, si Tl échoue alors exécuter T2. Un contrat, lui-même, n'est pas une transaction ACID et les mises à jour peuvent être rendues publiques avant la fin du contrat. Un contrat est en mesure de lire des résultats partiels en provenance d'un autre contrat. Par exemple, si le vol est réservé, la réservation est effective sitôt l'action correspondante terminée.

Figure 3.4 Exemple de contrat : réservation pour un voyage d'affaire.



◇ ◇ ◇ ◇

Le modèle des contrats intègre assez bien les dimensions «flexibilité» et «procédés métiers» de notre problématique (voir figure 3.5). En effet, comme le modèle de workflow, il permet de spécifier la logique d'exécution d'une application.

De plus, le modèle des contrats intègre des sémantiques transactionnelles. Ainsi, il considère chaque étape du contrat comme une transaction ACID. Il permet de regrouper certaines étapes dans une même transaction ACID et/ou de définir des relations entre les étapes. Il permet de définir ainsi des mécanismes de recouvrement en avant.

Par comparaison aux modèles transactionnels, le degré de fiabilité du modèle des contrats est limité (voir figure 3.5). En effet, les mécanismes de recouvrement sont définis d'une façon ad-hoc par les concepteurs. Le modèle des contrats ne définit pas un critère de correction ni des mécanismes pour vérifier la cohérence et la correction des spécifications des concepteurs.

Le système METEOR

L'exécution des dépendances inter-activités dans les workflows transactionnels a été discutée dans [RS95; KS95] et a été repris dans le système METEOR [SKM+96]. Un workflow dans METEOR est une collection de plusieurs tâches. Les activités sont modélisées en fournissant leurs états visibles et les événements correspondant aux transitions d'états. Les dépendances inter-activités peuvent être classées comme des dépendances qui peuvent être activées, rejetées ou retardées. Les dépendances entre les activités, comme la dépendance d'ordre ou d'existence entre des événements significatifs, sont spécifiées formellement en utilisant la logique computationnelle d'arbre. Ce modèle supporte des tâches ayant des sémantiques transactionnelles et non transactionnelles.

Pour relâcher l'atomicité, les auteurs dans [RS95] utilisent la notion d'états de terminaison acceptés (ETA) comme critère de correction. Tout état n'appartenant pas à ETA est un état de terminaison non accepté, dans lequel la semi atomicité est non respectée. On distingue les états de terminaison acceptés `de validation` et les états de terminaison acceptés `d'abandon`. Un état de terminaison `de validation` est un état dans lequel le workflow a atteint ses objectifs. Au contraire, `un état de terminaison accepté d'abandon` est une état de terminaison valide dans lequel le workflow a échoué et n'a pas atteint ses objectifs. Si un état de terminaison accepté `d'abandon` est atteint, tous les effets indésirables de l'exécution partielle du workflow doivent être annulés conformément aux besoins définis.

◇ ◇ ◇ ◇

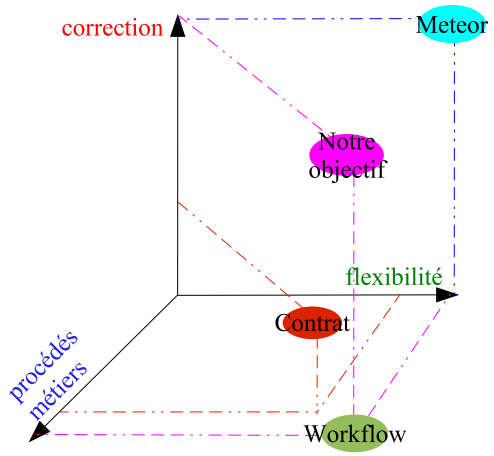
Le système METEOR présente deux points intéressants par rapport aux systèmes de workflow et au modèle des contrats.

Tout d'abord il définit la logique d'exécution en spécifiant des dépendances entre les activités (basées sur leurs états). Ce qui permet de définir aussi bien du flot de contrôle que de mécanismes de recouvrement. De plus le système METEOR définit un critère de correction des exécutions. La combinaison de la flexibilité de l'approche «workflow» et des mécanismes transactionnels permet d'intégrer parfaitement les dimensions «flexibilité» et «correction» de notre problématique (voir figure 3.5).

Cependant, le grand problème du système METEOR est que le mécanisme développé pour assurer le critère de correction n'est pas approprié au contexte des procédés métiers. En effet, pour assurer la fiabilité, le système METEOR vérifie à la fin de l'exécution si l'application s'est terminée dans un état acceptable ou non. Si oui ses effets sont validés, sinon ils sont annulés. Cette façon de procéder n'est pas appropriée au contexte des procédés métiers à cause de leurs

exécutions de longue durées. En plus, il n'est pas toujours possible d'annuler le travail effectué. C'est pourquoi le système METEOR reste limité aux applications centrées bases de données.

Figure 3.5 Limites de l'approche «workflow» pour résoudre notre problématique.



3.3.4 Synthèse

Les systèmes de workflow présentent une approche intéressante permettant de modéliser des procédés métiers complexes et de supporter leurs exécutions. Par comparaison aux MTA, les modèles de workflow ont une sémantique plus riche et abordent des problèmes qui ne sont pas pris en compte par les modèles transactionnels avancés.

Émanant des procédés administratifs, les workflows ont considéré les problèmes liés à la coordination de tâches et à l'acheminement de documents. Et ils ont ignoré les problèmes liés à la fiabilité des exécutions en cas d'échecs. Dans le contexte des procédés métiers, les échecs des activités ont été résolus de façon ad-hoc, au cas par cas et éventuellement via des interventions humaines. C'est pourquoi, les systèmes de workflows ont été toujours critiqués à cause de ce manque de fiabilité.

Les workflows transactionnels incorporent des sémantiques transactionnelles pour intégrer la dimension «correction» afin d'assurer un certain degré de fiabilité. Cependant aucun modèle n'a pu intégrer efficacement les trois dimensions de notre problématique. Généralement chaque modèle réussit à intégrer, au meilleur des cas, deux dimensions au détriment du troisième (voir figure 3.5).

3.4 Les services Web

La définition précise d'un «service Web» est en cours d'évolution. La notion de «service Web» désigne essentiellement une application (un programme) mise à disposition sur Internet par un fournisseur de service, et accessible par les clients à travers des protocoles Internet standards [FBM02; CS01]. Le consortium W3C[8] définit un service Web comme étant une application ou un composant logiciel (i) identifié par un URI, (ii) dont ses interfaces et ses liens (binding) peuvent être décrits en XML, (iii) sa définition peut être découverte par d'autres services Web et (iv) il

[8]World Wide Web Consortium : http ://www.w3.org/2002/ws/

peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet [KT03].

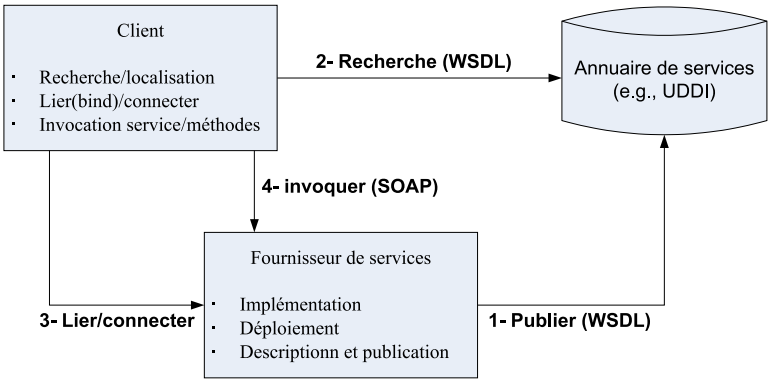
3.4.1 Architecture de référence

Les interactions entre les services Web impliquent trois participants [KT03] : le fournisseur de services, l'annuaire de services et le client (utilisateur du service) (voir figure 3.6).

- Le fournisseur de services correspond au propriétaire du service. D'un point de vue technique, il est constitué par la plate-forme d'accueil du service.
- Le client correspond au demandeur du service. D'un point de vue technique, il est constitué par l'application de recherche et d'invocation d'un service. L'application client peut être elle même un service Web.
- L'annuaire des services correspond à un registre de descriptions de services offrant des facilités de publication de services pour les fournisseurs ainsi que des facilités de recherche pour les clients.

Les interactions de base entre ces trois rôles incluent les opérations de publication, de recherche et de liaison. Le fournisseur de services définit la description de son service et la publie dans un annuaire de service. Le client utilise les facilités de recherche disponibles au niveau de l'annuaire pour retrouver et selectionner un service donné. Il examine ensuite la description du service selectionné pour récupérer les informations nécessaires lui permettant de se connecter au fournisseur du service et d'interagir avec l'implémentation du service considéré [KT03].

Figure 3.6 Les interactions entre les services Web impliquent trois intervenants : le fournisseur de services, l'annuaire de services et le client.



Trois initiatives de standardisation majeures ont été proposées au consortium W3C pour supporter les interactions entre les service Web : SOAP [W3C03a], WSDL [W3C03c] et UDDI [W3C03b].

SOAP est un protocole de transport permettant l'échange de documents XML dans un environnement distribué et décentralisé, initialement proposé par Microsoft, IBM et d'autres intervenants. L'avantage majeur de SOAP est qu'il est basé sur le standard XML et pas sur des formats binaires comme les autres modèles d'exécution distribués.

WSDL est un langage de description des capacités de services Web basé sur XML. Un document WSDL décrit essentiellement ce qu'un service Web offre, où il réside et comment on peut l'invoquer. WSDL sépare les définitions abstraites des implantations concrètes et des formats de données spécifiques.

UDDI est une spécification qui définit les mécanismes qui permettent aux entreprises de publier leurs services et de découvrir et interagir avec d'autres services via le Web. UDDI se base sur SOAP et suppose que les requêtes et les réponses sont des objets UDDI envoyés comme des messages SOAP.

3.4.2 Architecture avancée

SOAP, WSDL et UDDI constituent les technologies de bases pour les interactions entre les services Web. Cependant, elles sont insuffisantes à elles seules pour répondre aux besoins des applications B2B. En effet, ces trois propositions permettent la description, la publication et l'invocation d'un service Web. Cependant, une application B2B nécessite d'invoquer un ensemble de services dans un ordre précis et selon une logique bien définie. Or SOAP, WSDL et UDDI ne s'intéressent pas à ce problème et se situe plutôt au niveau transport et données. A ce propos, plusieurs technologies ont été proposées. Ces technologies s'intéressent à différents problèmes et à différents niveaux, des propriétés non fonctionnelles (comme la sécurité et la fiabilité) au niveau transport à la qualité de services au niveau procédé. Dans ce qui suit, nous nous intéressons aux technologies proposées pour assurer l'interopérabilité au niveau procédé. Avant de présenter ces technologies, nous introduisons quelques concepts et définitions nécessaires pour situer ces technologies.

Chorégraphie et coordination de services Web

Dans le contexte du B2B, un service Web encapsule une application interne d'une entreprise. Les opérations offertes par un service représentent donc les points d'entrés de cette application via lesquels elle interagit avec le monde extérieur. L'interaction avec un service Web implique donc un échange de messages dans un ordre bien défini. Nous distinguons alors :

- **Une chorégraphie de services Web** : désigne la coordination de services Web selon un protocole de coordination.
- **Un protocole de coordination** : est un ensemble de règles spécifiant l'ensemble de conversations possibles/acceptées.
- **Une conversation de services Web** : désigne la séquence de messages échangés entre un ou plusieurs service(s) Web. Une conversation est une instance d'un protocole de coordination.

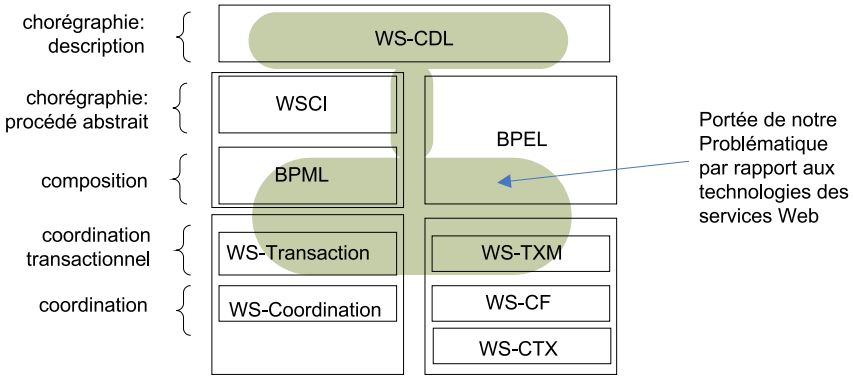
Il est important de noter que dans une chorégraphie un ensemble de services Web collaborent pour atteindre un objectif commun **sans pour autant créer un nouveau service**. La gestion d'une chorégraphie est gérée d'une façon décentralisée par les différents partenaires. La chorégraphie s'intéresse plutôt au coté externe des services Web. Ceci implique la description (i) des procédés abstraits des services et (ii) la collaboration entre les services. Un procédé abstrait désigne l'interface publique d'un service Web (les messages échangés et dans quel ordre) spécifiant les conversations que le service peut supporter.

Composition et orchestration de services Web

La composition de services Web désigne la création d'un nouveau service en réutilisant et en combinant des services Web existants. La composition définit un procédé exécutable, externalisé comme un nouveau service, dont les activités composantes sont d'autres services Web. L'exécution d'un service composé est géré par un seul organisme, celui qui l'a créé. La composition implique, entre autres, l'orchestration des services composants. L'orchestration décrit l'interac-

tion des services composants au niveau message mettant en oeuvre la logique d'exécution. La composition s'intéresse ainsi au coté interne (implémentation) d'un service Web.

Figure 3.7 Pile des technologies avancées des services Web.



La figure 3.7 donne une vue globale des technologies des services Web et les situe par rapport à la terminologie introduite ci-dessus. Elle illustre en plus la portée de notre problématique par rapport à ces propositions.

La composition et la chorégraphie s'intéressent à la problématique de définir la logique d'invocation d'un ensemble de services Web. Ce point est partiellement abordé lors de la définition des procédés abstraits. La coordination transactionnelles vise à coordonner un ensemble de services pour atteindre un objectif commun et à assurer un certain niveau de correction.

Il est important de mentionner que ces technologies se caractérisent en l'état actuel par une forte évolution et une grande dynamicité. Beaucoup de ces propositions sont en cours de développement et rares sont celles qui ont atteint le stade de stabilité. Certaines propositions disparaissent et généralement se fusionnent pour faire émerger des nouvelles. En plus, plusieurs technologies similaires sont proposées au même niveau. Néanmoins, cela n'empêche que certaines propositions imposent leurs philosophies plus que d'autres.

3.4.3 Coordination transactionnelle de services Web

3.4.3.1 Coordination de services Web

WS-Coordination [ea02] et WS-CAF[9] [AFI+] (plus précisement l'environnement de coordination WS-CF de WS-CAF) sont les deux approches principales proposées pour supporter la coordination de services Web. Leur but est de fournir un cadre pour supporter des protocoles de coordination. Dans cette perspective, chacune est sensée être une meta spécification qui gouvernera les spécifications qui implémentent les formes concrètes de coordination (e.g coordination transactionnelle).

WS-Coordination et WS-CF présentent globalement les mêmes concepts. Les entités de bases sont les coordinateurs et les participants qui seront coordonnés lors de l'exécution d'une conversation.

WS-Coordination comme WS-CF se base sur le concept de «contexte» pour la correlation des messages. Un contexte de coordination est une structure de données utilisée pour marquer

[9]WS-CAF offre un environnement pour assurer des coordinations fiables de services Web. Cette proposition est composée de trois Parties : WS-CTX pour la gestion des contextes, WS-CF pour la coordination de services Web et WS-TXM pour la coordination transactionnelle.

les messages appartenant à une même conversation. Tous les messages SOAP échangés entre les participants dans le cadre d'une conversation donnée incluent dans leurs entêtes les contextes de coordination appropriés.

WS-Coordination et WS-CF distinguent trois formes d'interactions entre un coordinateur et et ses participants :

- **Activation**, un participant demande à un coordinateur de créer un nouveau «*contexte de coordination*». Un nouveau contexte est créé quand un participant initie une instance d'un type de coordination (une conversation). Ceci se produit par exemple quand un service Web initie une transaction atomique.
- **Registration**, un participant s'enregistre à un protocole de coordination. En s'enregistrant, un service Web déclare qu'il va participer à l'exécution du protocole et qu'il devra être notifié quand les étapes du protocole de coordination seront exécutées. Par exemple, un service Web peut s'enregistrer comme un participant dans une transaction atomique, en fournissant son rôle et son port.
- **Interactions spécifiques** le coordinateur et les participants échangent les messages qui sont spécifiques à un protocole de coordination. Par exemple, le coordinateur peut envoyer des messages de validation ou d'annulation aux participants, et spécifiquement aux ports qui les ont spécifiés lors de la phase de l'enregistrement.

Les interactions d'activation et d'enregistrement sont indépendantes du type de la coordination. En d'autres termes, elles ne changent pas d'un type de coordination à un autre. Ainsi, les interfaces qui doivent être imlplémentées par les coordinateurs et les participants pour ces deux types d'interactions font partie de ces deux spécifications (WS-Coordination et WS-CF). D'autre part, les interfaces nécessaires pour les interactions des protocoles spécifiques varient d'un protocole à un autre. C'est pourquoi WS-Coordination et WS-CF ne spécifient pas ces interfaces.

Comme pour les intergiciels traditionnels, l'un des principaux protocoles requis par les applications est celui permettant un support transactionnel. Dans le contexte de services Web, de tels protocoles sont définis principalement par WS-Transaction [ea03; ea04] et WS-TXM [AFI+], deux propositions bâties respectivement sur WS-Coordination et WS-CF pour définir des protocoles de coordination transactionnelle.

3.4.3.2 WS-Transaction

WS-Transaction [ea03; ea04] hérite de WS-Coordination la distinction entre *protocole de coordination* et *type de coordination*. Un *protocole de coordination*, est un ensemble de règles contrôlant les conversations entre un coordinateur et ses participants. 2PC[10] est un exemple de protocole de coordination. Un *type de coordination*, comprend un ensemble de *protocoles de coordination*, logiquement liés les un aux autres. Par exemple, nous pouvons considérer une transaction atomique comme un *type de coordination*, regroupant logiquement le protocole 2PC et le protocole de notification du résultat (ce dernier est exécuté par les participants qui veulent être informés de résultat de 2PC). Une instance d'un *type de coordination* peut impliquer l'exécution de plusieurs instances d'un même ou de plusieurs protocoles. Par exemple, l'exécution d'une instance de type de coordination transaction atomique peut engendrer l'exécution d'une conversation 2PC et une ou plusieurs conversations de notification du résultat, selon le nombre des parties intéressées par le résultat.

WS-Transaction distingue deux types de protocoles : *les transactions atomiques* (WS-AT : WS-Atomic Transaction) et *les procédés métiers* (WS-BA : WS-Business Activity).

[10] 2PC est l'acronyme de l'anglicisme «Two Phase Commit», un protocol permettant la mise en oeuvre de transaction atomique.

WS-AT comprend plusieurs *protocoles de coordination*, exécutés en séquence ou en alternative par les services Web participants ou par le coordinateur, selon ce que doit être fait durant les différentes phases d'une transaction distribuée. Cinq protocoles constitue ce type de coordination : `Completion`, `2PC`, `CompletionWithAck`, `PhaseZero`, et `OutcomeNotification`.

Pour gérer les transactions commerciales de longue durées entre les services Web sans bloquer les ressources en utilisant 2PC, WS-Transaction définit un autre *type de coordination*, appelé *Business Activity*. Ce *type de coordination* comprend deux protocoles : `Business Agreement` et `Business Agreement With Complete`. Le protocole `Business Agreement` est initié par une service Web participant pour informer le coordinateur de son état d'exécution («Exited», «Completed», «Faulted»). Après avoir obtenu un consensus sur le fait que la transaction continue ou abandonne, le coordinateur répond à tous les participants avec l'un des messages «Close», «Complete», «Compensate», ou «Forget». Le protocole `Business Agrement With Complete` est similaire au protocole `BusinessAgreement` mais en plus le coordinateur doit informer un participant quand toutes les tâches qu'il attend, comme une partie de la transaction, ont été invoquées. Ceci permet au participant de se préparer pour la terminaison ou la compensation qui va suivre.

3.4.3.3 WS-TXM

WS-TXM fournit des modèles conçus pour s'adapter à différents cas d'utilisation, TX-ACID pour des transactions fortement couplées, TX-LRA pour des transactions de longues durées et TX-BP pour des transactions orientées procédés métiers.

Cet ensemble de protocole n'est pas censé être complet. L'intention est que si d'autres modèles sont développés pour mieux convenir à d'autres cas d'utilisation, alors ces modèles peuvent être ajoutés à WS-TXM.

TX-ACID : ce modèle est conçu pour supporter l'interopérabilité des systèmes transactionnels existants via les services Web. Bien que les transactions ACID ne soient pas convenables pour tous les services Web, elles le sont pour certains comme par exemple les interactions financières. Dans le modèle ACID, chaque activité est encapsulée dans une transaction de telle manière que la fin d'une activité déclenche la terminaison (validation ou annulation) de la transaction associée.

TX-LRA : le modèle des actions de longue durée (Long Running Activities) est conçu spécifiquement pour les interactions commerciales de longue durée. Avec ce modèle, une activité reflète des interactions commerciales : tout le travail effectué dans l'étendue d'une application doit être compensable. Par conséquent, le travail d'une activité est soit effectué avec succès soit annulé.

TX-BP : selon ce modèle, toutes les parties impliquées dans un procédé métier réside dans des domaines commerciaux, qui peuvent eux même utiliser des procédés métier pour effectuer le travail. Les transactions «procédés métiers» sont responsables de la gestion des interactions entre ces domaines. Un procédé métier est divisé en tâches commerciales et chaque tâche s'exécute dans un domaine commercial spécifique. Un domaine commercial peut être lui même divisé en plusieurs autres domaines commerciaux (procédés métiers) de façon recursive. Chaque domaine peut présenter un modèle transactionnel différent si une telle fédération de modèles est plus appropriée à l'activité.

◇ ◇ ◇ ◇

WS-Coordination et WS-CF sont deux approches similaires qui spécifient les mécanismes de base permettant de supporter la coordination d'un ensemble de services selon un protocole donné. Le problème traité par ces deux approches se situe à un niveau plus bas que celui de notre problématique. En effet, elles ne traitent ni le problème d'orchestration de services Web ni celui des exécutions fiables de services composés.

WS-Transaction et WS-TXM définissent un ensemble de protocoles de coordination basés respectivement sur WS-Coordination et WS-CF. Ces deux approches définissent des variantes des modèles transactionnels (avancés) existants. Par conséquence, elles héritent de certaines limites de ces modèles, surtout au niveau de la rigidité des structures de contrôle.

3.4.4 Composition et chorégraphie de services Web

Même si la composition et la chorégraphie ont des buts différents, elles partagent une même problématique, celle de spécifier la logique d'invocation de services au sein d'une composition ou d'une chorégraphie. Dans ce paragraphe nous présentons BPEL4WS [BM03] pour la composition et WS-CDL [KBRL04] pour la chorégraphie.

3.4.4.1 BPEL4WS

Le standard BPEL4WS représente une convergence des idées initialement proposées par les langages XLANG [Cor02] et WSFL [Ley01]. La version 1.1 délivrée en Mai 2003 fournit une grammaire pour décrire l'orchestration de services Web. Les services Web sont définis en WSDL. BPEL4WS définit comment les opérations WSDL doivent être ordonnancées.

La spécification BPEL4WS supporte des activités élémentaires et des activités structurées. Une activité élémentaire peut être considérée comme un composant qui interagit avec une entité externe au procédé lui-même. Par exemple, les activités élémentaires vont s'occuper de la réception et de la réponse aux requêtes de messages ainsi que de l'invocation des services externes.

Au contraire, les activités structurées gèrent le flot global du procédé. Elles indiquent quelle activité doit être exécutée et dans quel ordre. Ces activités permettent aussi des boucles avec des conditions et de branchements dynamiques.

Les «conteneurs» et les «partenaires» [11] sont deux autres éléments importants dans BPEL4WS. Un «conteneur» identifie l'échange de données dans un flot de message, qui correspond en particulier à un type de message WSDL. Quand un procédé BPEL reçoit un message, le «conteneur» approprié est rempli de telle façon que les requêtes ultérieures peuvent accéder aux données dont elles ont besoins. Un «partenaire» peut être un service que le procédé invoque ou n'importe quel «service» qui invoque le procédé. A chaque «partenaire» est associé à un rôle spécifique. Un «partenaire» particulier peut jouer un seul rôle dans un procédé mais peut jouer un rôle totalement différent dans un autre procédé. Les «conteneurs» sont alors utilisés pour gérer la persistance des données à travers les requêtes des services Web.

BPEL4WS fournit des mécanismes pour gérer les transactions et les exceptions, au dessus de WS-Coordination et WS-Transaction. Dans BPEL4WS, un ensemble d'activités peut être groupé dans une seule transaction en utilisant l'étiquette «contexte» (scope est le mot utilisé par BPEL). Cette étiquette signifie que cet ensemble est atomique. Ainsi, les concepteurs peuvent spécifier des gestionnaires de compensation qui seront invoqués en cas d'erreur. Par exemple, si une partie d'un procédé de réservation de voyage échoue, le gestionnaire doit identifier comment annuler les autres parties de la transaction. BPEL fournit un mécanisme de gestion d'exception via l'utilisation des clauses «throw» et «catch», similaires à celles utilisées par le langage de programmation Java.

[11] Ces deux termes sont notre traduction des anglicismes de BPEL containors et partners

3.4.4.2 WS-CDL

WS-CDL est un langage de description de chorégraphie. Il définit une chorégraphie comme un contrat multi-partenaires qui décrit, d'un point de vue global, le comportement commun observable des services participants à une collaboration.

Une description de chorégraphie en WS-CDL est un conteneur d'un ensemble d'activités qui peuvent être effectuées par un ou plusieurs participants. WS-CDL distnigue trois classes d'activités : «unitaire», «flot de contrôle», et «basiques».

Une activité «unitaire» décrit une exécution conditionnelle et, éventuellement, répétitive. La classe «flot de contrôle» inclut trois (types) d'activités, «séquence», «parallèle» et «choix», qui permet de spécifier les constructeurs basiques de flot de contrôle.

La troisième classe «basique» inclut les activités «NoAction», «SilentAction», «Assign», «Perform» et «Interaction». Les activités «NoAction» et «SilentAction» décrivent des points dans une chorégraphie où un rôle n'effectue aucune action ou exécute une action cachée qui n'influe pas sur le reste de la chorégraphie. L'activité «Assign» est utilisée pour affecter la valeur d'une variable à une autre. L'activité «Perform» est utilisée pour réaliser une autre chorégraphie dans le contexte de la chorégraphie courante.

Un élément important de WS-CDL est l'activité «Interaction». Cette activité décrit un échange d'information entre les parties. Trois types d'interactions sont définis : envoyer une requête, envoyer une réponse, ou envoyer une requête qui nécessite une réponse. Une description d'une activité «Interaction» comprend trois parties correspondantes (i) aux participants impliqués, (ii) aux informations échangées, et (iii) au canal (channel) pour d'échange des informations.

◇ ◇ ◇ ◇

Bien que BPEL et WS-CDL aient été proposés dans des contextes différents, ils partagent plusieurs points communs. En effet, ces deux propositions adoptent plusieurs concepts des systèmes de workflow pour spécifier la logique d'exécution des invocations de services. De même que les workflows, ces deux propositions permettent d'intégrer les dimensions «flexibilité» et «procédés métiers» de notre problématique, mais elles restent très limitées concernant la dimension «correction» (voir figure 3.8). En plus, puisque BPEL est destiné à la composition il ne supporte que des exécutions centralisées.

3.4.5 Autres technologies

Dans ce qui suit, nous présentons RosettaNet et ebXML deux propositions basées sur XML qui présentent des approches plus complètes pour mener des interactions B2B.

3.4.5.1 RosettaNet

RosettaNet [ros03] vise à standardiser la description des produits et des procédés métiers des applications de chaîne logistique du secteur informatique. Elle s'intéresse aux trois niveaux d'interopérabilité B2B.

Au niveau transport, RosettaNet repose sur les protocoles de transports Internet habituels. L'environnement d'implémentation de RosettaNet (RosettaNet Implementation Framework) spécifie le contenu des messages, les protocoles de transport (HTTP, CGI, email, SSL) pour la communication et les mécanismes de sécurité (certificat électronique, signature électronique).

Au niveau donnée, RosettaNet utilise des schémas XML. Pour résoudre les problèmes d'hétérogénéité sémantique, cette proposition utilise un dictionnaire qui contient les vocabulaires qui

peuvent être utilisées pour décrire les propriétés commerciales (e.g. nom de l'entreprise, l'adresse, l'identificateur de taxe) et un dictionnaire technique qui contient les propriétés qui peuvent être utilisées pour décrire les caractéristiques des produits (e.g un ordre d'achat).

Au niveau procédé, RosettaNet se contente de donner une base commune pour les interactions B2B via la définition des PIP[12] [Ros00]. Un PIP définit un procédé abstrait. L'integration des PIPs avec les procédés métiers internes est effectuée par les partenaires.

3.4.5.2 ebXML

L'initiative ebXML est similaire à celle de RosettaNet mais elle est plus générique et n'est pas restreinte à un domaine précis. L'objectif de la spécification ebXML est de remplacer les formats d'échanges ad-hoc par une infrastructure neutre qui vise à gérer de manière globale les échanges de messages électroniques entre plusieurs entreprises. ebXML prévoit de spécifier des schémas XML de représentation des procédés de collaboration et des éléments de base associés. Pour cela, ebXML propose un environnement pour la définition de vocabulaires communs (business objects), de modèles (business models) et de procédés (business processes). L'interopérabilité doit être assurée par l'utilisation des objets communs à tous les modèles. ebXML définit également les mécanismes d'identification des entreprises et des services qu'elles proposent. Toutes les informations sont stockées dans des registres accessibles par tous les partenaires. De cette façon, un annuaire global des services est constitué permettant aux entreprises d'exécuter des transactions commerciales sur Internet via le Web.

◇ ◇ ◇ ◇

ebXML et RosettaNet assurent l'interopérabilité en définissant des concepts communs (aux trois niveaux, transport, données et procédé) à partager et à respecter par les partenaires dans une collaboration donnée. Par exemple au niveau procédé, une entreprise qui veut publier son service doit respecter les modèles de procédés pré-définis («business processes» pour ebXML et PIP pour RosettaNet). En fait, ces deux propositions peuvent être vues comme des versions plus souples de EDI [EDI] adaptées au contexte des services Web.

L'avantage de ces propositions est qu'elles intègrent bien les dimensions «correction» et «procédés métiers» de notre problématique. Leur inconvénient est que cette intégration est réalisé au détriment de la flexibilité (voir figure 3.8). Cette rigidité est héritée en fait de EDI. En effet, les partenaires sont contraints à respecter et se conformer aux spécifications communes définies par ebXML ou RosettaNet.

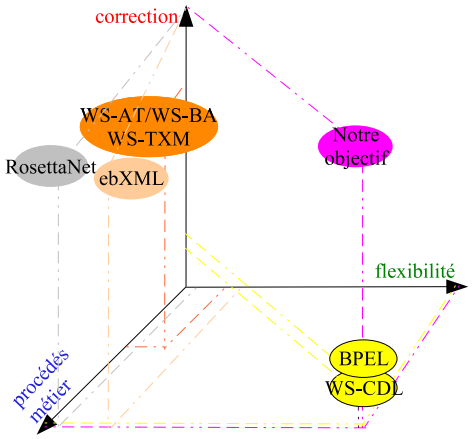
3.4.6 Synthèse

Plusieurs propositions (proches) constituent la pile des technologies des services Web. Au niveau transport et données, les problèmes traités concernent principalement l'échange de documents commerciaux (SOAP), la description des capacités des services Web (WSDL), la publication et la recherche de services Web (UDDI). Au niveau procédé, les problèmes traités concernent la coordination (transactionnelle) (WS-Coordination au dessus de WS-Transaction et WS-CAF), la composition (BPEL4WS) et la chorégraphie (WS-CDL) de services Web. Ces technologies au niveau procédé se sont inspirées :

- des systèmes de workflows pour la modélisation de l'orchestration et la chorégraphie de services Web. En effet, BPEL et WS-CDL définissent des constructeurs de flot de contrôle (semblables à ceux utlisés dans les workflows) en utilisant le langage XML.

[12] PIP est l'acronyme de Partner Interface Processes

Figure 3.8 Limite des technologies des services Web à intégrer les principales dimensions de notre problématique.



– des modèles transactionnels (avancés) pour définir des protocoles de coordination transactionnels. WS-Transaction et WS-TXM définissent un ensemble de spécifications au dessus de WS-Coordination et WS-CF permettant la mise en oeuvre des (variantes) de modèles/protocoles transactionnels pré-existants de façon décentralisée.

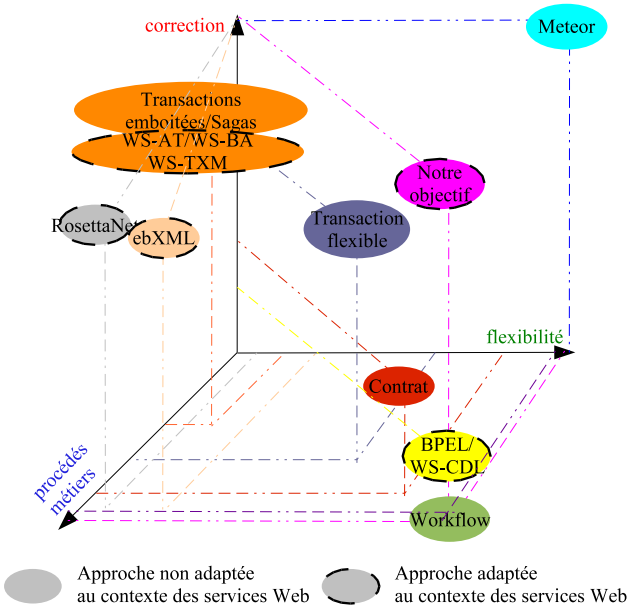
On peut dire que les technologies des services Web actuelles sont des versions standardisées (via l'utilisation de XML comme format de données et du Web comme infrastructure d'invocation) des systèmes de workflows et des MTA adaptées à un contexte orienté service et à un environnement décentralisé. Par conséquence, les propositions actuelles présentent les mêmes limites que les MTA et que les systèmes de workflow (voir figure 3.8).

3.5 Conclusion

Dans ce chapitre, nous avons présenté les approches principales qui abordent des problèmes similaires à ceux posés par notre problématique : les modèles transactionnels avancés (MTA), les systèmes de «workflow» et les technologies des services Web. Nous avons montré en particulier leurs limites pour résoudre notre problématique en étudiant leur degré d'intégration des trois principales dimensions que nous avons mises en avant : «flexibilité», «correction» et «procédés métiers».

Généralement, les MTA assurent un bon niveau de correction au détriment des dimensions «flexibilité» et «procédés métiers». Les systèmes de workflow assurent un bon niveau de flexibilité et sont appropriés aux contextes des procédés métiers au détriment de la correction. Seul le système METEOR a pu intégrer les dimensions «correction» et «flexibilité». Cependant les mécanismes offerts par ce système pour assurer des exécutions correctes ne sont pas applicables dans le contexte des procédés métiers. Les technologies actuelles des services Web adoptent des concepts des MTA (pour la coordination transactionnelle) et des systèmes de workflow (pour l'orchestration et la chorégraphie de services Web). Par la suite, elles présentent les limites de ces deux approches. La figure 3.9 illustre les limites des différentes approches présentées en les situant par rapport aux trois dimensions de notre problématique.

Figure 3.9 Les différentes technologies présentées situées par rapport aux dimensions de notre problématique.



Chapitre 4

Modèle de services Web Transactionnels

Table des matières

4.1 Service Web transactionnel : SWT 42
4.1.1 Propriétés transactionnelles d'un SWT 42
4.1.2 Modélisation du comportement d'un SWT 43
4.1.3 Diagramme à transition d'états d'un SWT 44
4.1.4 Synthèse 46
4.2 Service Web composé transactionnel : SCT 46
4.2.1 Composition de services Web transactionnels 47
4.2.2 Dépendances entre les services composants d'un SCT 48
4.2.3 Flot de contrôle et flot transactionnel d'un SCT 56
4.2.4 Synthèse 59
4.3 Patrons de composition 60
4.3.1 Patron AND-split 60
4.3.2 Patron AND-join 61
4.3.3 Patron XOR-split 61
4.3.4 Spécification du flot de contrôle d'un SCT 62
4.3.5 Synthèse 63
4.4 Flot transactionnel potentiel 64
4.4.1 Flot transactionnel potentiel d'un patron 64
4.4.2 Flot transactionnel potentiel du patron AND-join 64
4.4.3 Flot transactionnel potentiel du patron XOR-split 65
4.4.4 Flot transactionnel potentiel du patron AND-split 66
4.4.5 Flot transactionnel potentiel d'un flot de contrôle 67
4.4.6 Synthèse 67
4.5 Spécification d'un SCT 67
4.6 Conclusion 69

Dans ce chapitre, nous présentons notre modèle de composition de services Web [BPG05]. Nous distinguons en particulier l'aspect coordination et l'aspect transactionnel d'un service Web composé (SWC). D'une part, un SWC peut être considéré comme un flot de services autonomes

et hétérogènes. D'autre part, il peut être considéré aussi comme une transaction structurée où les services composants sont les sous-transactions et les interactions sont les dépendances.

La section 4.1 introduit la notion de service Web transactionnel. Nous présentons les propriétés transactionnelles que nous considérons et nous montrons comment nous modélisons le comportement d'un service Web selon ses propriétés transactionnelles. La section 4.2, illustre comment nous combinons un ensemble de services Web transactionnels pour offrir un service plus complexe à valeur ajoutée. Nous montrons comment nous modélisons l'orchestration des service composants à différents niveaux d'abstraction. Nous distinguons en particulier le `flot de contrôle` (aspect coordination) et le `flot transactionnel` (aspect transactionnel) d'un SWC. Dans la section 4.3, nous présentons la notion de patron de composition. Nous montrons comment nous utilisons ce concept pour définir les flots de contrôle des SWCs. Dans la section 4.4, nous dévoilons le coté transactionnel caché des patrons de composition que nous exploitons, dans la section 4.5, pour définir les flots transactionnels des SWCs.

Il est important de noter que nous utilisons le terme «composition de services» cohérent avec la pratique pour désigner la composition des opérations offertes par ces services [ACKM04].

4.1 Service Web transactionnel : SWT

4.1.1 Propriétés transactionnelles d'un SWT

Un service Web est un programme modulaire auto descriptif qui peut être publié, découvert et invoqué via le Web. Typiquement les services Web reposent sur XML, SOAP et UDDI [UDD00]. Un service Web est défini par son profil (fichier WSDL) qui spécifie ses entrées, ses sorties et les détails techniques pour l'invoquer. Nous désignons par $\mathcal{P}rofil$ l'ensemble de tous les profils de tous les services Web.

Un service Web transactionnel (SWT) est un service Web dont le comportement manifeste des propriétés transactionnelles. Les propriétés transactionnelles que nous considérons dans notre approche sont **rejouable**, **compensable** et **pivot** [MRKS92]. Un service s est dit rejouable (s^r) s'il se termine toujours avec succès après un nombre fini d'activation. s est dit compensable (s^{cp}) s'il offre des mécanismes de compensation pour annuler sémantiquement son travail. Un service s est dit pivot (s^p) si une fois qu'il se termine avec succès, ses effets ne peuvent pas être compensés. L'ensemble des combinaisons de propriétés possibles est $\{\emptyset; r; cp; p; (r, cp); (r, p)\}$.

Rappelons que dans notre travail, nous nous intéressons à assurer des exécutions correctes de services Web composés en présence des échecs de services composants. Ce problème a été abordé par la plupart des modèles transactionnels (par exemple le modèle des transactions ACID assure des exécutions correctes en forçant la propriété d'atomicité). Généralement, d'un point de vue transactionnel, ce problème mène à traiter deux points essentiels : *les échecs de sous transactions* et *les mécanismes de recouvrement*. Les propriétés que nous utilisons ont été introduites dans certains modèles transactionnels avancés (principalement le modèle des transactions flexibles). L'utilisation et l'appellation (transactionnelle) de ces propriétés sont justifiées par le fait qu'elles permettent de caractériser le comportement d'un service. Ainsi, la propriété **rejouable** permet d'informer si un service est succeptible d'échouer ou non (le premier point du problème à traiter). De même les propriétés **compensable** et **pivot** informe si un service est recouvrable ou non (deuxième point du problème à traiter).

4.1.2 Modélisation du comportement d'un SWT

A chaque service transactionnel est associé un diagramme à transition d'états qui modélise son comportement (transactionnel). Ce diagramme décrit les états possibles par lesquels le service peut passer durant son cycle de vie et les transitions possibles entre ses états. Nous détaillons l'ensemble des états et des transitions, que nous considérons dans notre approche, ci-dessous. Nous notons $\mathcal{E}tats$ et $\mathcal{T}ransitions$ respectivement l'ensemble de tous les états et l'ensemble de toutes les transitions de tous les SWT transactionnels. Un SWT est défini, en plus de son profil, par son diagramme à transition d'état qui permet d'exprimer son comportement transactionnel.

DÉFINITION 4.1 (SERVICE WEB TRANSACTIONNEL)
Un service Web transactionnel, *st*, est un triplet $st = (profil \in \mathcal{P}rofil, E \in \mathcal{E}tats, T \in \mathcal{T}ransistions)$ où $profil$ est son profil, E et l'ensemble de ses états et T l'ensemble des transitions entre ses états.

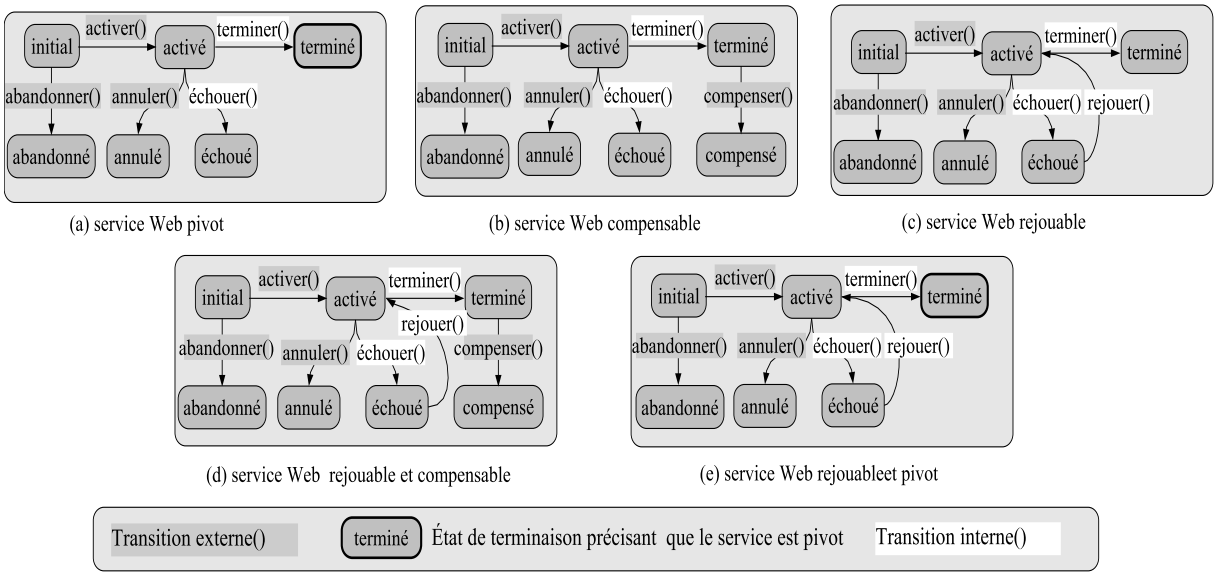
Nous Notons $\mathcal{SWT}$ l'ensemble de tous les services Web transactionnels. Nous définissons la fonction *états* : $\mathcal{SWT} \longrightarrow \mathcal{P}(\mathcal{E}tats)$[13] qui permet de retourner l'ensemble des états d'un SWT donné.

L'ensemble des états que nous considérons est {**initial, abandonné, activé, annulé, échoué, terminé, compensé**} (voir le diagramme UML de la figure 4.2). Au départ, quand un service est instancié, il est dans l'état **initial**. Ensuite, il peut être soit **abandonné** (avant d'être activé pour l'exécuter) ou **activé** (pour être exécuté). Une fois activé, il peut continuer normalement son exécution ou peut être **annulé** (durant son exécution). Si un service n'est pas annulé durant son exécution, il peut soit atteindre ses objectifs et se **terminer** avec succès soit **échouer**. Un service compensable peut être compensé après sa terminaison et passer ainsi à l'état **compensé**.

Une transition désigne le changement d'état d'un service. L'activation d'une transition fait transiter le service en question d'un état initial vers un nouveau état. Une transition $t()$ d'un service s est un couple $t()$=($e_1 \in$ *états(s)*, $e_2 \in$ *états*(s)) avec $e_1 \neq e_2$ et où e_1 est l'état de s avant l'activation de $t()$ et e_2 est l'état de s après l'activation de $t()$. L'ensemble des transitions que nous considérons est {`abandonner()`, `activer()`, `annuler()`, `échouer()`, `terminer()`, `compenser()`, `rejouer()`}. La transition `abandonner()` permet d'abandonner l'exécution du service en question (avant d'être activé) et le fait passer ainsi de l'état `initial` à l'état `abandonné` (`s.abandonner() = (s.initial,s.abandonné)`[14]). La transition `activer()` permet d'activer un service et de le faire ainsi passer de l'état `initial` à l'état `activé` (`s.activer() = (s.initial,s.activé)`). La transition `annuler()` permet d'annuler un service (en cours de son exécution) et le fait ainsi transiter de l'état `activé` vers l'état `annulé` (`s.annuler() = (s.activé,s.annulé)`). La transition `échouer()` permet de marquer l'échec d'un service et le fait ainsi transiter de l'état `activé` vers l'état `échoué` (`s.échouer() = (s.activé,s.échoué)`). La transition `terminer()` permet de marquer la terminaison d'un service avec succès et de le faire ainsi transiter de l'état `activé` vers l'état `terminé` (`s.terminer() = (s.activé,s.terminé)`). La transition `compenser()` permet de compenser sémantiquement le travail effectué par le service en question et le fait ainsi transiter de l'état `terminé` vers l'état `compensé` (`s.compenser() = (s.terminé,s.compensé)`). La transition `rejouer()` permet d'activer un service après son échec et le fait ainsi transiter de l'état `échoué` vers l'état `activé` (voir figure 4.1).

[13] Étant donné un ensemble E, nous notons $\mathcal{P}(E)$ pour désigner l'ensemble des sous ensemble de E.
[14] $s.t()$ désigne la transition $t()$ du service s et $s.e$ désigne l'état e du service s

Figure 4.1 Diagrammes à transitions d'états des services Web transactionnels selon leurs propriétés transactionnelles.



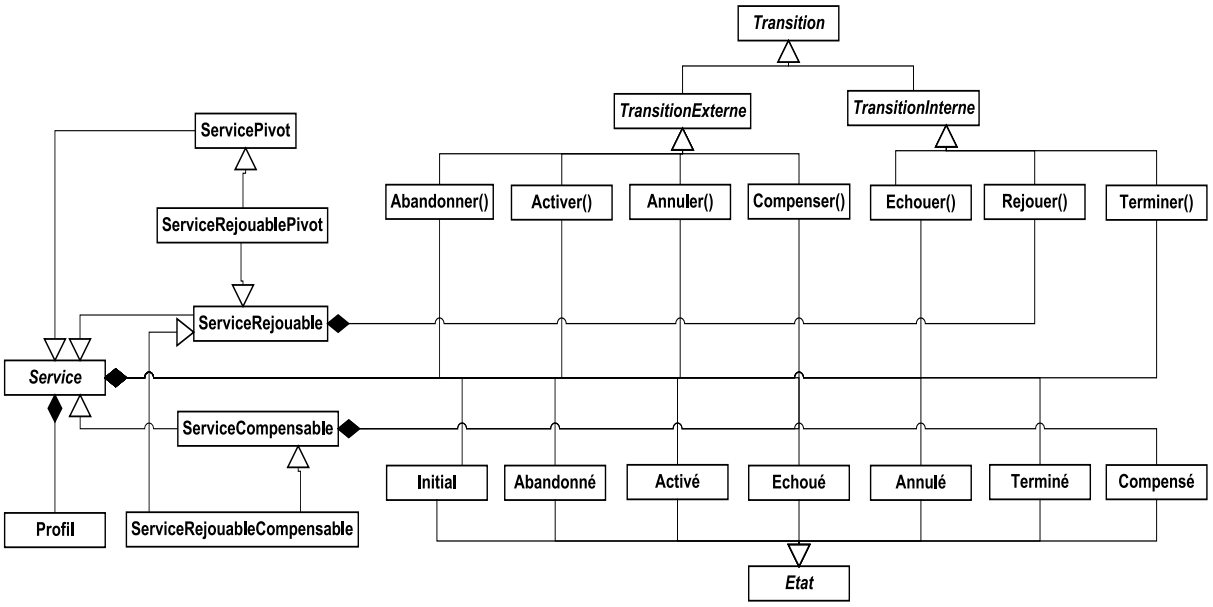
Nous distinguons les transitions externes et les transitions internes d'un service Web transactionnel (voir le digramme UML de la figure 4.2). Une transition externe est activée par une entité externe. Les transitions externes permettent à un service d'interagir avec l'extérieur. En particulier, elles permettent de mettre en oeuvre l'orchestration des services au sein d'un service Web composé (voir la section 4.2). Les transitions externes que nous considérons dans notre approche sont : `abandonner()`, `activer()`, `annuler()` et `compenser()`. Une transition interne est activée par le service lui-même (l'agent responsable de l'exécuter). Les transitions internes que nous considérons dans notre approche sont `échouer()`, `terminer()` et `rejouer()`. Nous notons $\mathcal{T}ransEx t$ et $\mathcal{T}ransInt$ respectivement l'ensemble de toutes les transitions externes et l'ensemble de toutes les transitions internes de tous les SWT. Nous définissons l'ensemble des fonctions suivantes nécessaires pour la formalisation des divers concepts introduits par la suite.

- Nous définissons la fonction *source* qui permet de retourner l'état source d'une transition ($\bowtie_i (t)$ dénote la projection de t selon le i^{me} attribut)
$$source : \mathcal{T}ransitions \longrightarrow \mathcal{E}tats$$
$$t \longmapsto source(t) = \bowtie_1 (t)$$
- Nous définissons la fonction *cible* qui permet de retourner l'état cible d'une transition
$$cible : \mathcal{T}ransitions \longrightarrow \mathcal{E}tats$$
$$t \longmapsto cible(t) = \bowtie_2 (t)$$
- Nous définissons la fonction $tex : \mathcal{SWT} \longrightarrow \mathcal{P}(\mathcal{T}rans\mathcal{E}x)$ qui permet de renvoyer l'ensemble des transitions externes d'un SWT.
- Nous définissons la fonction $tin : \mathcal{SWT} \longrightarrow \mathcal{P}(\mathcal{T}rans\mathcal{I}nt)$ qui permet de renvoyer l'ensemble des transitions internes d'un SWT.

4.1.3 Diagramme à transition d'états d'un SWT

L'ensemble des états et des transitions dépendent des propriétés transactionnelles du service.

Figure 4.2 Diagramme UML des services Web transactionnels.



Service Web pivot : Un service pivot possède un ensemble minimal d'états (`initial`, `abandonné`, `activé`, `annulé`, `échoué` et `terminé`) et un ensemble minimal de transitions (`abandonner()`, `activer()`, `annuler()`, `échouer()` et `terminer()`). Le diagramme à transition d'états d'un service pivot est illustré par la figure 4.1.a. Formellement, un service Web pivot, *sp*, est un service Web tq : *états(sp)* = {`initial, abandonné, activé, échoué, annulé, terminé`}, *tex(sp)* = {`abandonner() ; activer() ; annuler()`} et *tin(sp)* = {`échouer() ; terminer()`}.

Ces deux ensembles minimaux (d'états et de transitions) peuvent être étendus pour exprimer les propriétés transactionnelles des services Web.

Service Web compensable : Ainsi, un service compensable possède en plus un état `compensé` et une transition `compenser()`. La figure 4.1.b montre le diagramme à transition d'états d'un service compensable (non rejouable). Formellement, un un service Web compensable, *sc*, est un service Web transactionnel tq : *états(sc)* = {`initial, abandonné, activé, échoué, annulé, terminé, compensé`}, *tex(sc)* = {`abandonner() ; activer() ; annuler() ; compenser()`} et *tin(sc)* = {`échouer() ; terminer()`}.

Service Web rejouable :[15] Un service rejouable possède une transition `rejouer()` qui ramène le service à son état activé à chaque fois qu'il échoue. La figure 4.1.c illustre le diagramme à transition d'états d'un service rejouable. Formellement, un service Web rejouable, *sr*, est un service Web transactionnel tq : *états(sr)* = {`initial, abandonné, activé, échoué, annulé, terminé`}, *tex(sr)* = {`abandonner() ; activer() ; annuler()`} et *tin(sr)* = {`échouer() ; terminer() ; rejouer()`}.

Service Web compensable et rejouable : Un service peut être à la fois compensable et rejouable. La figure 4.1.d illustre le diagramme à transition d'états d'un service rejouable

[15] Un service Web rejouable et pivot possède les mêmes transitions et états qu'un service rejouable. Cependant dans un service rejouable et pivot, il est précisé que l'état **terminé** est un état final (c.à.d le service est pivot).

et compensable. Formellement, un service Web rejouable et compensable, *scr*, est un service Web transactionnel tq : *états(scr)* = {`initial, abandonné, activé, échoué, annulé, terminé, compensé`}, *tex(scr)* = {`abandonner() ; activer() ; annuler() ; compenser()`} et *tin(scr)* = { `échouer() ; terminer() ; rejouer()`}.

4.1.4 Synthèse

Dans cette section, nous avons présenté le concept de service Web transactionnel. Un service Web transactionnel est un service Web dont le comportement manifeste des propriétés transactionnelles. Les propriétés transactionnelles que nous considérons dans notre approche sont *rejouable*, *compensable* et *pivot* [MRKS92].

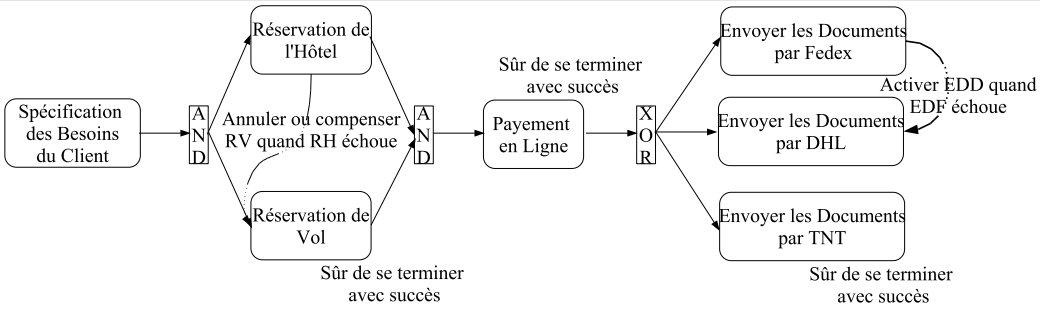
A chaque service transactionnel est associé un diagramme à transition d'états qui modélise son comportement (transactionnel). Ce diagramme décrit les états possibles par lesquels une instance du service peut passer durant son cycle de vie et les transitions possibles entre ses états. L'ensemble d'états et de transitions dépendent des propriétés transactionnelles du service.

Nous distinguons entre les transitions externes et les transitions internes d'un service Web transactionnel. Une transition externe est activée par une entité externe. Les transitions externes permettent à un service d'interagir avec l'extérieur. En particulier, elles permettent de mettre en oeuvre l'orchestration des services au sein d'un service Web composé. Une transition interne est activée par le service lui-même (l'agent responsable de l'exécuter).

4.2 Service Web composé transactionnel : SCT

Dans cette section, nous montrons comment nous combinons un ensemble de services Web transactionnels pour offrir un nouveau service plus complexe à valeur ajoutée. Nous illustrons en particulier comment nous modélisons l'orchestration des services composants à différents niveaux d'abstraction. Au départ, nous montrons comment un service composé définit des préconditions sur les transitions externes de ses services composants (paragraphe 4.2.1). Ensuite, nous montrons comment ces préconditions permettent d'exprimer à un niveau d'abstraction plus haut des dépendances entre les services (paragraphe 4.2.2). Ces dépendances définissent à leur tour (à un niveau d'abstraction plus haut) le `flot de contrôle` et le `flot transactionnel` du sercice composé (paragraphe 4.2.3). Nous reprenons tout au long de cette section notre exemple de service d'organisation de voyage en ligne, décrit à la figure 4.3, pour illustrer les divers concepts que nous présentons.

Figure 4.3 Service Web composé d'organisation de voyage en ligne.



4.2.1 Composition de services Web transactionnels

Un service Web composé orchestre un ensemble de services pour atteindre un objectif commun. Un service Web composé transactionnel (SCT) est un service Web composé dont les services composants sont des services transactionnels. Un tel service exploite les propriétés transactionnelles de ses services composants pour spécifier des mécanismes de recouvrement.

Pour définir l'orchestration de ses services composants, un service composé spécifie des préconditions sur leurs transitions externes (voir diagramme UML de la figure 4.4). Ces préconditions spécifient pour chaque service quand est-ce qu'il sera abandonné, activé, annulé ou compensé. Par exemple, le service composé défini dans la figure 4.3 spécifie que le service «**P**ayement en **L**igne» (PL) sera activé après la terminaison des services «**R**éservation d'**H**ôtel» (RH) et «**R**éservation de **V**ol» (RV). Ceci signifie que la précondition de l'activation de la transition $activer()$ du service PL est la terminaison du service RH et la terminaison du service RV.

Ainsi, un service Web composé transactionnel peut être défini comme le couple de l'ensemble de ses services composants et l'ensemble des préconditions définies sur leurs transitions externes.

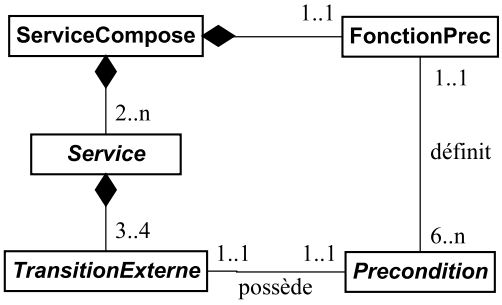
Définition 4.2 (Service Web Composé Transactionnel : SCT)
Un service Web composé transactionnel sct est un couple $sct = (ES \subset \mathcal{SWT}, \mathcal{P}rec)$ où ES est l'ensemble de ses services Web composants et $\mathcal{P}rec$ une fonction qui définit pour chaque transition externe d'un service composant une precondition pour son activation (voir définition 4.3) :

$$\begin{array}{rccc} \mathcal{P}rec : & \mathcal{T} & \longrightarrow & \mathcal{PR}ec_{sc} \\ & t() & \longmapsto & \mathcal{P}rec(t()) \end{array}$$

où $\mathcal{T}$ est l'ensemble de toutes les transitions externes de tous les services composants de sct et $\mathcal{PR}ec(sc)$ est l'ensemble de toutes les préconditions tenant compte seulement des événements générés par les services composants de sc.

Nous notons $\mathcal{SCT}$ l'ensemble de tous les services Web composés transactionnels. Nous définissons la fonction $services : \mathcal{SCT} \longrightarrow \mathcal{P}(\mathcal{SWT})$ qui permet de retourner l'ensemble des services composants d'un SCT.

Figure 4.4 Un service Web composé transactionnel est défini par l'ensemble de ses services composants et par la fonction qui définit une précondition pour chaque transition externe d'un service composant.



Les préconditions des transitions externes des services composants spécifient pour chacun d'eux comment il réagit aux changements d'états des autres services composants et comment il agit sur leurs comportements. En fait la fonction $\mathcal{P}rec$ définit pour chaque transition externe $t()$

d'un service composant l'ensemble de conditions susceptibles de l'activer. Puisque ces conditions sont exclusives, nous les écrivons comme une seule condition sous forme normale disjonctive exclusive.

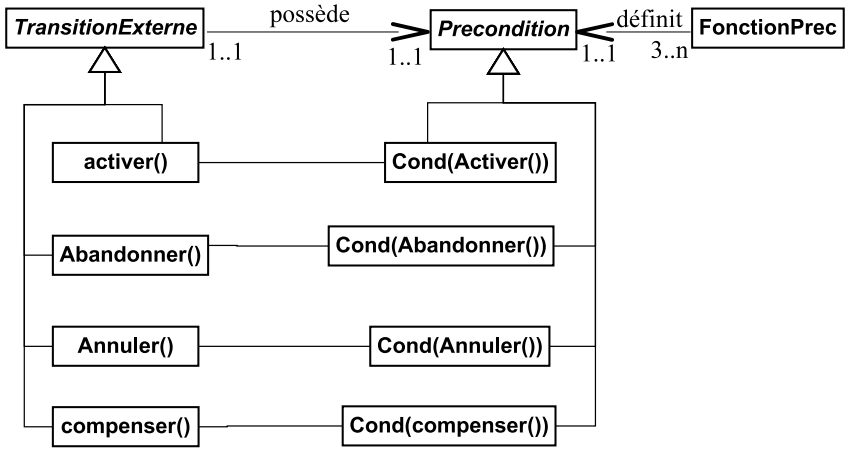
DÉFINITION 4.3 (PRÉCONDITION D'UNE TRANSITION EXTERNE D'UN SERVICE COMPOSANT) Soit sc un service composé, s un service composant de sc ($s \in services(sc)$) et $s.t()$ une transition externe de s ($s.t() \in tex(s)$), la précondition de $s.t()$ défini par la fonction $\mathcal{P}rec$ de sc est une formule logique de premier ordre dont les prédicats sont des états de certain(s) service(s) composant de sc.

Nous notons $\mathcal{P}recond$ l'ensemble de toutes les préconditions. Nous définissons la fonction *prédicats* : $\mathcal{P}recond \longrightarrow \mathcal{P}(\mathcal{E}tats)$ qui permet de retourner les prédicats d'une précondition.

Ainsi, nous distinguons pour chaque service composant, s, une pré condition pour chacune de ses transitions externes, *activer()*, *abandonner()*, *annuler()* et *compenser()*, que nous notons respectivement *Cond(s.activer())*, *Cond(s.abandonner())*, *Cond(s.annuler())* et *Cond(s.compenser())* (voir diagramme UML de la figure 4.5).

Par rapport à notre exemple, le service composé spécifie que le service RV sera compensé quand le service RH échoue. Ceci signifie que $Cond(RV.compenser()) = RH.échoué$. De même, ce service spécifie que le service «**E**nvoyer les **D**ocuments par **D**HL» (EDD) sera activé soit après la terminaison de PL soit après l'échec du service «**E**nvoyer les **D**ocuments par **F**edex» (EDF). Ceci signifie que $Cond(EDD.activer()) = (PL.\text{terminé} \bigwedge \text{service } EDD \text{ est sélectionné pour envoyer les documents}) \bigoplus EDF.échoué$.

Figure 4.5 Chaque service composant possède une précondition pour chacune de ses transitions externes.



4.2.2 Dépendances entre les services composants d'un SCT

Les préconditions expriment à un niveau d'abstraction plus haut des relations (successions, alternative, ...) entre les services composants sous forme de dépendances. Ces dépendances expriment comment les services sont couplés et comment le comportement de certains service(s) influence le comportement d'autre(s) service(s). Par rapport à notre exemple, la précondition de la transition *activer*() du service EDD ($Cond(EDD.activer())$= $PL.terminé$ (et EDD est choisi

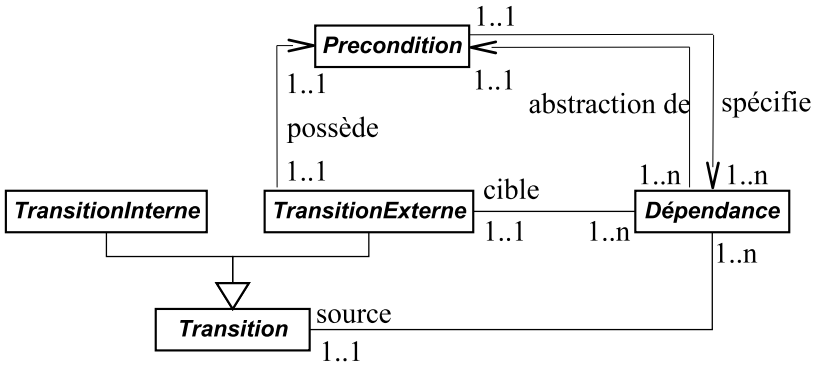
pour l'envoi des documents) $\bigoplus$ *EDF.échoué*) manifeste une dépendance d'activation de PL vers EDD (voir définition 4.5) et une dépendance d'alternative de EDF vers EDD (voir définition 4.7). Ces dépendances expriment respectivement une relation de succession entre PL et EDD et une relation d'alternative entre EDF et EDD. D'une façon générale une dépendance d'un service s_1 vers un service s_2 existe s'il existe une transition t_1 de s_1 et une transition externe t_2 de s_2 telles que l'activation de t_1 peut entraîner l'activation de t_2.

DÉFINITION 4.4 (DÉPENDANCE D'UNE TRANSITION $s_1.t_1()$ VERS UNE TRANSITION $s_2.t_2()$)
Soit *sc* un service composé, s_1 et s_2 deux services composants de *sc*, $s_1.t_1()$ une transition de s_1 et $s_2.t_2()$ une transition externe de s_2, une dépendance de $s_1.t_1()$ vers $s_2.t_2()$, notée $dep(s_1.t_1(), s_2.t_2())$, existe *si et seulement si* l'activation de la transition $s_1.t_1()$ peut entraîner l'activation de la transition $s_2.t_2()$. En termes d'états et de préconditions, une telle dépendance existe *si et seulement si* l'état cible de $s_1.t_1()$ est un prédicat de la pré condition de $s_2.t_2()$. Plus formellement :

$$\exists\ dep(s_1.t_1(), s_2.t_2()) \Leftrightarrow cible(s_1.t_1()) \in prédicats(\mathcal{P}rec(s_2.t_2())).$$

Nous appelons $s_1.t_1()$ (respectivement $s_2.t_2()$) la transition source (respectivement cible) d'une dépendance $dep(s_1.t_1(), s_2.t_2())$.

Figure 4.6 Les préconditions expriment à un niveau d'abstraction plus haut des dépendances entre les services.

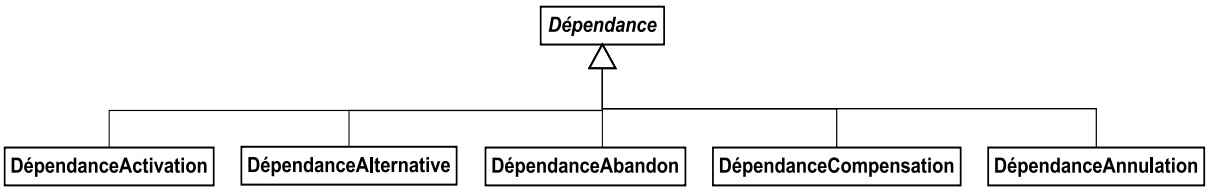


Les dépendances permettent d'exprimer différents types de relations (*succession, alternative, compensation, etc.*) qui peuvent exister entre les services. La définition 4.4 donne une définition générale d'une dépendance qui permet d'exprimer tout type de relation. Dans notre approche, nous considérons les dépendances suivantes (voir diagramme UML de la figure 4.7) : dépendance d'activation (définition 4.5), d'alternative (définition 4.7), d'abandon (définition 4.9), de compensation (définition 4.11) et d'annulation (définition 4.13). Les dépendances d'activation permettent d'exprimer des relations de succession entre les services. Les dépendances d'abandon permettent de propager l'abandon du service composé en question. Les dépendances d'alternative permettent d'exprimer des alternatives d'exécution comme mécanismes de recouvrement en avant. Les dépendances de compensation permettent de mettre en oeuvre des mécanismes de recouvrement en arrière par compensation. Enfin, les dépendances d'annulation permettent de signaler les échecs de certains services à d'autres services en cours d'exécution en les annulant si nécessaire. Par exemple, le service composé décrit à la figure 4.3 spécifie, parmi d'autres :

- une dépendance d'activation du service «**S**pécification des **B**esoins du **C**lient» (SBC) vers le service RH indiquant que le service RH est activé après la terminaison du service SBC. Cette dépendance exprime ainsi une relation de succession entre SBC et RH.
- une dépendance d'abandon du service SBC vers le service RH indiquant que le service RH est abandonné quand le service SBC échoue ou abandonne.
- une dépendance d'alternative de EDF vers EDD indiquant que le service EDD est activé quand le service EDF échoue. Cette dépendance permet de spécifier EDD comme une alternative de livraison pour EDF permettant ainsi de définir un mécanisme de recouvrement en avant en cas d'échec de EDF.
- une dépendance de compensation de RV vers RH indiquant que RV est compensé (compenser la réservation d'avion si elle est déjà faite) quand le service RH échoue. Cette dépendance permet de définir un mécanisme de recouvrement en arrière pour faire face à l'échec de RH.
- une dépendance d'annulation de RH vers RV indiquant que RV est annulé en cours de son exécution quand RH échoue. Cette dépendance permet de signaler l'échec de RH vers RV en annulant ce dernier.

Chacune des définitions 4.5, 4.7, 4.9, 4.11 et 4.13 spécifie la définition 4.4, en restreignant les transitions cibles et sources, pour exprimer exclusivement un type de relation.

Figure 4.7 Les différents types de dépendances que nous considérons dans notre approche.



Les dépendances expriment des relations entre les services, cependant elles ne décrivent pas d'une façon explicite et précise les interactions entre ces services. Une dépendance $dep(s_1.t_1(), s_2.t_2())$ ne précise pas quand est-ce que $s_2.t_2()$ sera activée suite à l'activation de $s_1.t_1()$. Le même type de dépendance (activation, abandon, alternative, compensation ou annulation) peut exister dans des contextes d'interaction différents. Dans notre exemple d'organisation de voyage en ligne (figure 4.3), il existe une dépendance (d'activation) du service SBC vers le service RH : $dep(SBC.terminer(), RH.activer())$. Le même type de dépendance existe du service RH vers le service PL : $dep(RH.terminer(), PL.activer())$. Ces deux dépendances expriment des relations de succession entre les services concernés. Cependant, elles ne spécifient pas le même type d'interaction. En effet, dans le premier cas la terminaison de SBC est une condition nécessaire et suffisante pour l'activation de RH. Alors que dans le deuxième cas, la terminaison de RH est une condition nécessaire mais pas suffisante pour l'activation de PL. Pour une dépendance $dep(s_1.t_1(), s_2.t_2())$, seule la précondition de $s_2.t_2()$ précise si l'activation de $s_1.t_1()$ est une condition nécessaire ou/et suffisante pour l'activation de $s_2.t_2()$. Nous disons que $dep(s_1.t_1(), s_2.t_2())$ est définie conformément à $\mathcal{P}rec(s_2.t_2())$.

Dans ce qui suit, nous illustrons plus en détail ces différents types de dépendances que nous considérons dans notre approche.

4.2.2.1 Dépendance et condition d'activation

Une dépendance d'activation est une dépendance qui exprime une relation de succession entre deux services. Elle définit un ordre total entre leurs activations. Une dépendance d'activation de s_1 vers s_2 existe *si et seulement si* la terminaison de s_1 peut déclencher l'activation de s_2.

DÉFINITION 4.5 (DÉPENDANCE D'ACTIVATION DE s_1 VERS s_2)
Une dépendance d'activation de s_1 vers s_2, notée $depAct(s_1, s_2)$, existe *si et seulement si* l'activation de la transition $terminer()$ de s_1 ($s_1.terminer()$) peut déclencher l'activation de la transition $activer()$ de s_2 ($s_2.activer()$). En terme d'états et de précondition, $depAct(s_1, s_2)$ existe *si et seulement si* l'état *terminé* de s_1 est un prédicat de la précondition de la transition $s_2.activer()$. Plus formellement et par référence à la définition 4.4 :

$$depAct(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.terminer(), s_2.activer()).$$

$$\exists\ depAct(s_1, s_2) \Longleftrightarrow \exists\ dep(s_1.terminer(), s_2.activer()) \Longleftrightarrow s_1.terminé \in pr\acute{e}dicats(\mathcal{P}rec(s_2.activer()))$$

Une dépendance d'activation de s_1 vers s_2 exprime seulement une relation de succession entre eux. Mais, elle ne précise pas quand est-ce que s_2 sera activé suite à la terminaison de s_1. Vu sa définition, une dépendance d'activation $depAct(s_1, s_2)$ est définie conformément à $\mathcal{P}rec(s_2.activer())$ et plus précisément conformément à la condition d'activation de s_2.

La condition d'activation (par succession) d'un service s, notée $CondAct(s)$, détermine quand est-ce qu'il sera activé en tant que successeur d'autre(s) service(s) (c.à.d seulement après la terminaison de ces services). $CondAct(s)$ est donc la sous-condition de $\mathcal{P}rec(s.activer())$ dont les prédicats sont des états de terminaison.

DÉFINITION 4.6 (CONDITION D'ACTIVATION D'UN SERVICE)
Soit sc un service composé, la condition d'activation d'un service composant s est une formule logique de premier ordre dont les prédicats sont des états de terminaison d'autres services composants. Plus formellement :

$$CondAct(s) \text{ est la sous condition de } \mathcal{P}rec(s.activer()) \mid [\forall\ e \in pr\acute{e}dicats(CondAct(s)) \\ (e = \text{événement externe à } sc) \bigvee (\exists\ s' \in services(sc) \mid e = s'.terminé)].$$

Nous écrivons $CondAct(s)$ sous forme normale disjonctive exclusive ; $CondAct(s) = \bigoplus_i CondAct_i(s)$. Ainsi $CondAct_i(s)$ est une condition d'activation (parmi d'autres) de s.

Exemple Le service d'organisation de voyage en ligne (décrit à la figure 4.3) spécifie une dépendance d'activation du service SBC vers le service RH, $depAct(SBC, RH)$, conformément à la condition d'activation de RH, $CondAct(RH) = SBC.terminé$. Ainsi, le service RH sera activé après la terminaison du service SBC. Ce service spécifie aussi une dépendance d'activation du service RH vers le service PL, $depAct(RH, PL)$ conformément à la condition d'activation de PL, $CondAct(PL) = RH.terminé \bigwedge RV.terminé$. Ainsi, le service PL sera activé après la terminaison de RH et la terminaison de RV.

4.2.2.2 Dépendance et condition d'alternative

Les dépendances d'alternative permettent de définir des alternatives d'exécution comme mécanismes de recouvrement en avant. Une dépendance d'alternative de s_1 vers s_2 existe *si et seulement si* l'échec de s_1 peut déclencher l'activation de s_2.

DÉFINITION 4.7 (DÉPENDANCE D'ALTERNATIVE DE s_1 VERS s_2)
Une dépendance d'alternative de s_1 vers s_2, notée $depAlt(s_1, s_2)$, existe *si et seulement si* l'activation de la transition *échouer*() de s_1 ($s_1.échouer()$) peut déclencher l'activation de la transition *activer*() de s_2 ($s_2.activer()$). En terme d'états et de précondition, $depAlt(s_1, s_2)$ existe *si et seulement si* l'état *échoué* de s_1 est un prédicat de la précondition de la transition $s_2.activer()$. Plus formellement et par référence à la définition 4.4 :

$$depAlt(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.échouer(), s_2.activer()).$$

$$\exists\ depAlt(s_1, s_2) \Longleftrightarrow \exists\ dep(s_1.échouer(), s_2.activer()) \Longleftrightarrow s_1.échoué \in prédicats(\mathcal{P}rec(s_2.activer()))$$

Vu sa définition, une dépendance d'alternative $depAlt(s_1, s_2)$ est définie conformément à $\mathcal{P}rec(s_2.activer())$ et plus précisément conformément à la condition d'alternative de s_2. La condition d'alternative d'un service s, notée *CondAlt(s)*, détermine quand est-ce qu'il sera activé en tant qu'alternative à d'autre(s) service(s) (c.à.d seulement après l'échec d'autre(s) service(s)). *CondAlt(s)* est donc la sous condition de $\mathcal{P}rec(s.activer())$ dont les prédicats sont des états d'échec.

DÉFINITION 4.8 (CONDITION D'ALTERNATIVE D'UN SERVICE)
Soit sc un service composé, la condition d'alternative d'un service composant s est une formule logique de premier ordre dont les prédicats sont des états d'échec d'autre(s) service(s) composant(s). Plus formellement :

$CondAlt(s)$ est la sous condition de $\mathcal{P}rec(s.activer) \mid [\forall\ e \in prédicats(CondAlt(s))\ (e = faux) \bigvee (\exists\ s' \in services(sc) \mid e = s'.échoué)]$.

Nous écrivons $CondAlt(s)$ sous forme normale disjonctive exclusive ; $CondAlt(s) = \bigoplus_i CondAlt_i(s)$. Ainsi $CondAlt_i(s)$ est une condition d'alternative (parmi d'autres) de s.

Exemple Le service d'organisation de voyage en ligne (décrit à la figure 4.3) spécifie une dépendance d'alternative du service EDF vers le service EDD, $depAlt(EDF, EDD)$, conformément la condition d'alternative de EDD, $CondAlt(EDD) = EDF.échoué$. Ainsi, le service EDD sera activé quand EDF échoue.

Notez bien que la condition d'activation de la transition *activer*() d'un service s, $s.activer()$, est défini par la condition d'activation de s (en tant que successeur), $CondAct(s)$, et la condition d'alternative de s $CondAlt(s)$: $\mathcal{P}rec(s.activer()) = CondAct(s) \bigoplus CondAlt(s)$.

4.2.2.3 Dépendance et condition d'abandon

Une dépendance d'abandon permet de propager les échecs (causant l'abandon du SCT) d'un service vers son ou ses successeur(s) en le(s) abandonnant. Une dépendance d'abandon de s_1 vers s_2 existe *si et seulement si* l'échec, l'annulation ou l'abandon de s_1 peut déclencher l'abandon de s_2.

DÉFINITION 4.9 (DÉPENDANCE D'ABANDON DE s_1 VERS s_2)
Une dépendance d'abandon de s_1 vers s_2, notée $depAbn(s_1, s_2)$, existe *si et seulement si* l'activation d'une des transitions abandonner(), annuler() ou *échouer*() de s_1 ($s_1.abandonner()$, $s_1.annuler()$ ou $s_1.échouer()$) peut déclencher l'activation de la transition *abandonner*() de s_2 ($s_2.abandonner()$). En terme d'états et de précondition, $depAbn(s_1, s_2)$ existe *si et seulement si* l'état *abandonné*, l'état *échoué* ou l'état *annulé* de s_1 est un prédicat de la précondition de la transition $s_2.abandonner()$. Plus formellement et par référence à la définition 4.4 :

$$depAbn(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.abandonner(), s_2.abandonner()) \bigvee dep(s_1.échouer(), s_2.abandonner()) \bigvee dep(s_1.annuler(), s_2.abandonner()).$$

$$\exists\, depAnb(s_1, s_2) \Longleftrightarrow \exists\, dep(s_1.abandonner(), s_2.abandonner()) \bigvee \exists\, dep(s_1.échouer(), s_2.abandonner()) \bigvee \exists\, dep(s_1.annuler(), s_2.abandonner()) \Longleftrightarrow s_1.abandonné \in prédicats(\mathcal{P}rec(s_2.abandonner())) \bigvee s_1.échoué \in prédicats(\mathcal{P}rec(s_2.abandonner())) \bigvee s_1.annulé \in prédicats(\mathcal{P}rec(s_2.abandonner())).$$

Une dépendance d'abandon, $depAbn(s_1, s_2)$, est définie conformément à $\mathcal{P}rec(s_2.abandonner())$. $\mathcal{P}rec(s.abandonner())$ définit la condition d'abandon du service s que nous notons *CondAbn(s)* : $CondAbn(s) = \mathcal{P}rec(s.abandonner())$.

DÉFINITION 4.10 (CONDITION D'ABANDON D'UN SERVICE)
Soit *sc* un service composé, la condition d'abandon d'un service composant s est une formule logique de premier ordre dont les prédicats sont des états d'abandon, d'échec ou d'annulation d'autre(s) service(s) composant(s). Plus formellement :

$$\mathcal{P}rec(s.abandonner()) = CondAbn(s) \mid [\forall\, e \in prédicats(CondAbn(s))\ (e = \text{événement externe à } sc) \bigvee (\exists\, s' \in services(sc) \mid e = s'.abandonné \bigvee e = s'.échoué \bigvee e = s'.annulé)].$$

Nous écrivons $CondAbn(s)$ sous forme normale disjonctive exclusive ; $CondAbn(s) = \bigoplus_i CondAbn_i(s)$. Ainsi $CondAbn_i(s)$ est une condition d'abandon (parmi d'autres) de s.

4.2.2.4 Dépendance et condition de compensation

Une dépendance de compensation permet de définir un mécanisme de recouvrement en arrière par compensation. Une dépendance de compensation de s_1 vers s_2 existe *si et seulement si* l'échec ou la compensation de s_1 peut déclencher la compensation de s_2.

DÉFINITION 4.11 (DÉPENDANCE DE COMPENSATION DE s_1 VERS s_2)
Une dépendance de compensation de s_1 vers s_2, notée $depCps(s_1, s_2)$, existe *si et seulement si* l'activation d'une des transitions *échouer*() ou *compenser*() de s_1 ($s_1.échouer()$ ou $s_1.compenser()$) peut déclencher l'activation de la transition *compenser*() de s_2 ($s_2.compenser()$). En terme d'états et de précondition, $depCps(s_1, s_2)$ existe *si et seulement si* l'état *échoué* ou l'état *compensé* de s_1 est un prédicat de la précondition de la transition $s_2.compenser()$. Plus formellement et par référence à la définition 4.4 :

$$depCps(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.échouer(), s_2.compenser()) \bigvee dep(s_1.compenser(), s_2.compenser()).$$

$$\exists\ depCps(s_1, s_2) \Longleftrightarrow \exists\ dep(s_1.échouer(), s_2.compenser()) \bigvee \exists\ dep(s_1.compenser(), s_2.compenser()) \Longleftrightarrow s_1.échoué \in prédicats(\mathcal{P}rec(s_2.compenser())) \bigvee s_1.compensé \in prédicats(\mathcal{P}rec(s_2.compenser()))$$

Une dépendance de compensation $depCps(s_1, s_2)$ est définie conformément à $\mathcal{P}rec(s_2.compenser())$. $\mathcal{P}rec(s.compenser())$ définit la condition de compensation de s que nous notons *CondCps(s)*. *CondCps(s)* spécifie quand s sera compensé.

DÉFINITION 4.12 (CONDITION DE COMPENSATION D'UN SERVICE)
Soit *sc* un service composé, la condition de compensation d'un service composant s est une formule logique de premier ordre dont les prédicats sont des états d'échec ou de compensation d'autre(s) service(s) composant(s). Plus formellement :

$$\mathcal{P}rec(s.compenser()) = CondCps(s) \mid [\forall\ e \in prédicats(CondCps(s))\ (e = faux) \bigvee (\exists\ s' \in services(sc)\ e = s'.échoué \bigvee e = s'.compensé)].$$

Nous écrivons $CondCps(s)$ sous forme normale disjonctive exclusive ; $CondCps(s) = \bigoplus_i CondCps_i(s)$. Ainsi $CondCps_i(s)$ est une condition de compensation (parmi d'autres) de s.

Exemple Le service d'organisation de voyage en ligne (décrit à la figure 4.3) spécifie une dépendance de compensation du service RH vers le service RV, $depCps(RH, RV)$, conformément la condition de compensation du service RV, $CondCps(RV) = RH.échoué$. Ainsi, le service RV sera compensé (s'il s'est déjà terminé avec succès) suite à l'échec de RH.

4.2.2.5 Dépendance et condition d'annulation

Une dépendance d'annulation permet de signaler les échecs d'exécution d'un service à d'autre(s) service(s) s'exécutant en parallèle en provoquant si nécessaire leur annulation. Une dépendance d'annulation de s_1 vers s_2 existe *si et seulement si* l'échec de s_1 peut déclencher l'annulation de s_2.

DÉFINITION 4.13 (DÉPENDANCE D'ANNULATION DE s_1 VERS s_2)
Une dépendance d'annulation de s_1 vers s_2, notée $depAnl(s_1, s_2)$, existe *si et seulement si* l'activation de la transition *échouer*() de s_1 ($s_1.échouer()$) peut déclencher l'activation de la transition *annuler*() de s_2 ($s_2.annuler()$). En terme d'états et de précondition, $depAnl(s_1, s_2)$ existe *si et seulement si* l'état *échoué* de s_1 est un prédicat de la précondition de la transition $s_2.annuler()$. Plus formellement et par référence à la définition 4.4 :

$$depAnl(s_1, s_2) \stackrel{\text{def}}{=} dep(s_1.échouer(), s_2.annuler())$$

$$\exists\, depAnl(s_1, s_2) \Longleftrightarrow \exists\, dep(s_1.échouer(), s_2.annuler()) \Longleftrightarrow s_1.échoué \in prédicats(\mathcal{P}rec(s_2.annuler()))$$

Une dépendance d'annulation, $depAnl(s_1, s_2)$, est définie conformément à $\mathcal{P}rec(s_2.annuler())$. $\mathcal{P}rec(s.annuler())$ définit la condition d'annulation du service s, *CondAnl(s)*, qui spécifie quand il sera annulé.

DÉFINITION 4.14 (CONDITION D'ANNULATION D'UN SERVICE)
Soit *sc* un service composé, la condition d'annulation d'un service composant s est une formule logique de premier ordre dont les prédicats sont des états d'échec d'autres services composants. Plus formellement :

$$\mathcal{P}rec(s.compenser()) = CondAnl(s) \mid [\forall\, e \in prédicats(CondAnl(s))\ (e = faux) \bigvee (\exists\, s' \in services(sc)\ e = s'.échoué.$$

Nous écrivons $CondAnl(s)$ sous forme normale disjonctive exclusive ; $CondAnl(s) = \bigoplus_i CondAnl_i(s)$. Ainsi $CondAnl_i(s)$ est une condition d'annulation (parmi d'autres) de s.

Exemple Le service d'organisation de voyage en ligne (illustré par la figure 4.3) spécifie une dépendance d'annulation du service RH vers le service RV, $depAnl(RH, RV)$, conformément à la condition d'annulation de RV, $CondAnl(RV) = RH.échoué$. Ainsi, le service RV sera annulé (s'il est encore en cours d'exécution) quand RH échoue.

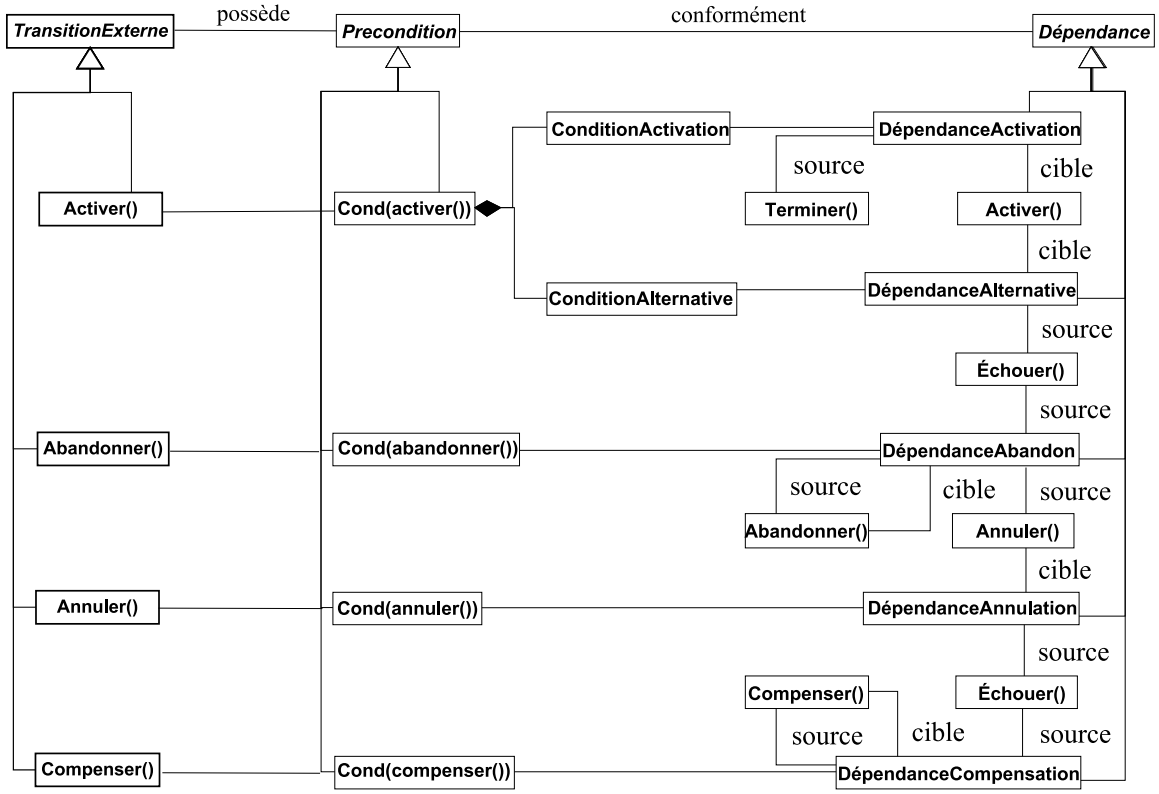
La figure 4.8 présente une vue d'ensemble des transitions externes, des préconditions et des différents types de dépendances que nous considérons.

4.2.2.6 Relations sémantiques entre les dépendances

Nous appelons les dépendances de compensation, d'annulation et d'alternative des `dépendances transactionnelles`. Les dépendances d'abandon et les `dépendances transactionnelles` dépendent des dépendances d'activation. Par exemple, une dépendance d'abandon de s_1 vers s_2 ne peut exister que si une dépendance d'activation de s_2 vers s_1 existe. Les dépendances d'abandon peuvent être déduites à partir des dépendances d'activation. Les dépendances transactionnelles dépendent des dépendances d'activation selon les relation suivantes :

- **R1 :** Une dépendance de compensation de s_1 vers s_2 peut exister seulement si une dépendance d'activation de s_2 vers s_1 existe ou si s_1 et s_2 s'exécutent en parallèle et se synchrosnisent.

Figure 4.8 Vue d'ensemble : Transitions externes, préconditions et dépendances.



- **R2 :** Une dépendance d'annulation de s_1 vers s_2 peut exister seulement si s_1 et s_2 s'exécutent en parallèle et se synchronisent.
- **R3 :** Une dépendance d'alternative de s_1 vers s_2 existe seulement si les exécutions de s_1 et s_2 sont exclusives.

4.2.3 Flot de contrôle et flot transactionnel d'un SCT

Les dépendances d'activation et les `dépendances transactionnelles` expriment à un niveau d'abstraction plus haut respectivement le `flot de contrôle` et le `flot transactionnel` d'un SCT (voir diagramme UML à la figure 4.9).

Flot de contrôle

Le `flot de contrôle` d'un SCT définit un ordre partiel entre les activations de ses services composants. Intuitivement, le flot de contrôle d'un SCT est défini par l'ensemble de ses dépendances d'activation. Formellement, nous définissons un `flot de contrôle` comme un SCT dont les seules dépendances sont des dépendances d'activation.

Définition 4.15 (Flot de contrôle)
Un `flot de contrôle` est un SCT, $fc = (ES, \mathcal{P}rec)$ tq $\forall s \in ES$ $\mathcal{P}rec(s.activer()) = CondAct(s)$ (autrement dit $CondAlt(s) = faux$) ; $\mathcal{P}rec(s.annuler()) = faux$; $\mathcal{P}rec(s.compenser()) = faux$.

Nous notons $\mathcal{FC}ontrôle$ l'ensemble des `flots de contrôle` de tous les SCT. Nous définissons la fonction *getFContrôle* qui permet de retourner le `flot de contrôle` d'un SCT donné.

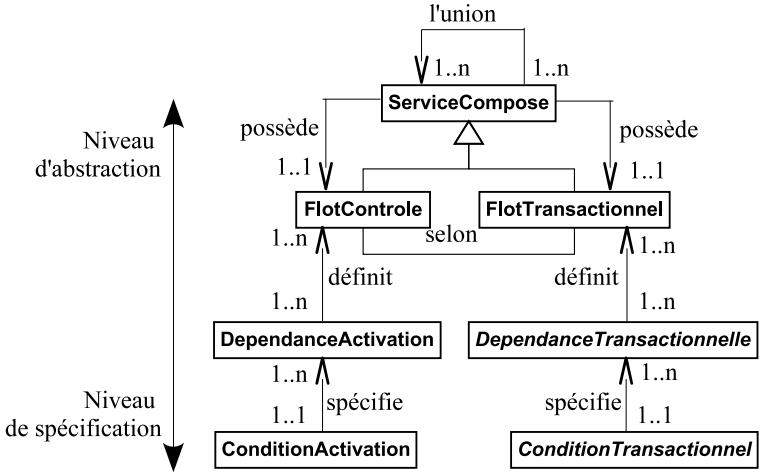
Définition 4.16 (Fonction getFContrôle)
Nous définissons la fonction *getFContrôle* qui permet de retourner le `flot de contrôle` d'un service Web composé transactionnel.

$$\begin{array}{rccc} getFContrôle : & \mathcal{SCT} & \longrightarrow & \mathcal{FC}ontrôle \\ & sc = (ES, \mathcal{P}rec) & \longmapsto & fc = (ES', \mathcal{P}'rec) \end{array}$$

tq $ES' = ES$ et $\forall s \in ES\ \mathcal{P}'rec(s.activer()) = CondAct(s)$; $\mathcal{P}'rec(s.annuler()) = faux$; $\mathcal{P}'rec(s.compenser()) = faux$.

Exemple La figure 4.10.b illustre le `flot de contrôle` du service d'organisation de voyage en ligne défini à la figure 4.10.a.

Figure 4.9 Nous distinguons entre le `flot de contrôle` et le `flot transactionnel` d'un SCT.



Flot transactionnel

Le `flot transactionnel` d'un SCT définit les mécanismes de recouvrement en cas d'échecs. Intuitivement, le flot transactionnel d'un SCT est défini par les propriétés transactionnelles de ses services composants et l'ensemble de ses dépendances transactionnelles. Formellement, nous définissons un `flot transactionnel` comme un SCT dont les dépendances sont seulement des dépendances transactionnelles.

Définition 4.17 (Flot transactionnel)
Un `flot transactionnel` est un SCT, $ft = (ES, \mathcal{P}rec)$ tq $\forall s \in ES\ \mathcal{P}rec(s.activer()) = CondAlt(s)$ (autrement dit $CondAct(s) = faux$).

Les dépendances transactionnelles dépendent (de la sémantique) des dépendances d'activation. Ceci implique qu'un `flot transactionnel` est toujours défini selon un `flot de contrôle`. Nous notons $\mathcal{FT}ransactionnel$ l'ensemble des `flots transactionnels` de tous les SCT. Nous

définissons la fonction *getFTransactionnel* qui permet de retourner le `flot transactionnel` d'un SCT donné.

DÉFINITION 4.18 (FONCTION GETFTRANSACTIONNEL)
Nous définissons la fonction *getFTransactionnel* qui permet de retourner le flot transactionnel d'un service Web composé.
$getFTransactionnel : \mathcal{SCT} \longrightarrow \mathcal{FT}ransactionnel$
$s = (ES, \mathcal{P}rec) \longmapsto s' = (ES', \mathcal{P}'rec)$
tq $ES' = ES$ et $\forall s \in ES'$ $\mathcal{P}'rec(s.activer()) = condAlt(s)$; $\mathcal{P}'rec(s.annuler()) = \mathcal{P}rec(s.annuler())$; $\mathcal{P}'rec(s.compenser()) = \mathcal{P}rec(s.compenser())$.

Exemple La figure 4.10.c illustre le `flot transactionnel` du service d'organisation de voyage en ligne défini à la figure 4.10.a. Ce `flot transactionnel` est défini selon le `flot de contrôle` illustré par la figure 4.10.b.

Il est important de préciser que (voir diagramme UML de la figure 4.9) :

- Définir le `flot de contrôle` d'un SCT revient à définir pour chaque service composant, s, sa condition d'activation $CondAct(s)$.
- Définir le `flot transactionnel` d'un SCT revient à définir pour chaque service composant, s, ses propriétés transactionnelles, sa condition de compensation $CondCps(s)$, sa condition d'annulation $CondAnl(s)$ et sa condition d'alternative $CondAlt(s)$.

Union de services composés

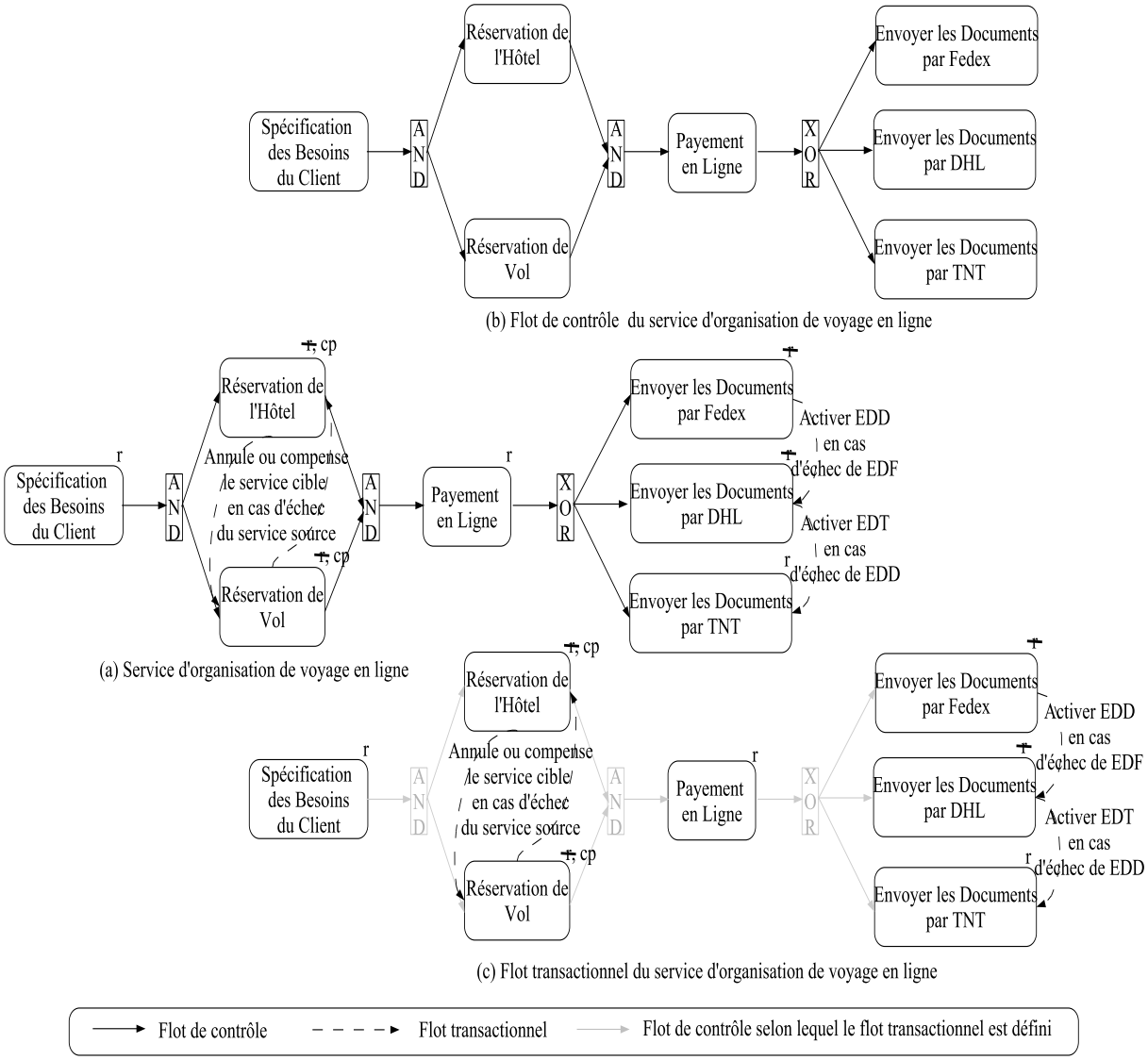
L'union de deux services composés sc_1 et sc_2 est un service composé sc dont (i) l'ensemble de ses services composants est l'union des deux ensembles de services composants des services sc_1 et sc_2 et (ii) la précondition d'une transition externe d'un service composant s est : celle définie par cs_1 si s appartient seulement à sc_1, celle définie par cs_2 si s appartient seulement à sc_2, ou la disjunction exclusive de ces deux préconditions si s appartient à cs_1 et cs_2.

DÉFINITION 4.19 (UNION DE DEUX SERVICES COMPOSÉ)
Soit deux SCT cs_1 et cs_2 : $cs_1 = (ES_1, \mathcal{P}rec_1)$ et $cs_2 = (ES_2, \mathcal{P}rec_2)$. L'union de cs_1 et cs_2 est le SCT défini comme suit : $cs = cs_1 \bigcup cs_2 = (ES, \mathcal{P}rec)$ où
– $ES = ES_1 \bigcup ES_2$.
– $\forall s \in ES \quad \mathcal{P}rec(s) = \begin{cases} \mathcal{P}rec_1(s) \text{ si } s \in ES_1 \bigwedge s \notin ES_2 \\ \mathcal{P}rec_2(s) \text{ si } s \in ES_2 \bigwedge s \notin ES_1 \\ \mathcal{P}rec_1(s) \bigoplus \mathcal{P}rec_1(s) \text{ si } s \in ES_1 \bigwedge s \in ES_2 \end{cases}$

Ainsi pour tout SCT sc, sc est le résultat de l'union de son `flot de contrôle` et de son `flot transactionnel` : $sc = getFContrôle(sc) \bigcup getFTransactionnel(sc)$. Par exemple, le SCT d'organisation de voyage en ligne décrit à la figure 4.10.a est l'union de son `flot de contrôle` (figure 4.10.b) et de son `flot transactionnel` (figure 4.10.c).

De même un `flot de contrôle` (respectivement transactionnel) peut aussi être l'union de plusieurs flots de contrôle (respectivement transactionnels).

Figure 4.10 `Flot de contrôle` et `flot transactionnel` du service d'organisation de voyage en ligne.



4.2.4 Synthèse

Dans cette section, nous avons présenté le concept de service Web composé transactionnel (SCT). Un SCT est un service Web composé dont les services composants sont des services Web transactionnels. Un SCT définit l'orchestration de ses services composants en spécifiant des préconditions sur leurs transitions externes. Ces préconditions spécifient pour chaque service composant quand il sera activé en tant que successeur, quand il sera activé en tant qu'alternative, quand il sera compensé, quand il sera annulé et quand il sera abandonné. Ces préconditions définissent à un niveau d'abstraction plus haut des dépendances entre les services composants. Nous distinguons entre les dépendances d'activation et les `dépendances transactionnelles`. Ces dépendances expriment à leur tour à un niveau d'abstraction plus haut le `flot de contrôle`

et le `flot transactionnel` d'un SCT. Le `flot de contrôle` définit un ordre partiel entre les activations des services composants. Le `flot transactionnel` spécifie les mécanismes de recouvrement en cas d'échecs. Un SCT est ainsi bien défini par son `flot de contrôle` et son `flot transactionnel`.

Dans la suite, nous montrons comment nous procédons pour spécifier le `flot de contrôle` et le `flot transactionnel` d'un SCT.

4.3 Patrons de composition

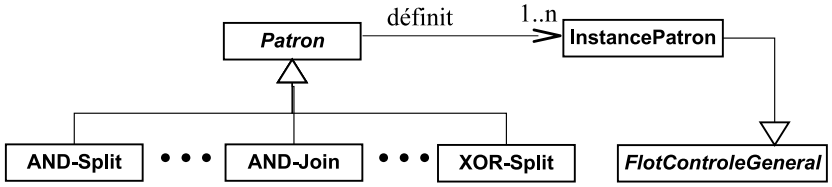
Dans cette section, nous introduisons le concept de patron de composition. Nous montrons en particulier comment nous l'utilisons pour définir le flot de contrôle d'un SCT.

Un patron permet d'exprimer d'une façon abstraite une forme récurrente dans un contexte spécifique [GHJV95]. Un patron de composition peut être vu comme une description abstraite d'une classe d'interactions. Par exemple, le patron *AND-join* [vdABtHK00] (voir figure 4.13.b) décrit une classe d'interactions où un service sera activé après la terminaison d'autres services. Formellement, nous définissons un patron de composition comme une fonction qui retourne un `flot de contrôle` (`InstancePatron` dans le diagramme UML de la figure 4.11) à partir d'un ensemble de services[16].

Définition 4.20 (Patron de composition)
Un patron de composition, $pat : \mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FC}ontrôle$, est une fonction qui définit un `flot de contrôle`, $pat(S)$, à partir d'un ensemble de services S.

Nous notons $\mathcal{P}atron$ l'ensemble de tous les patrons de composition. Dans notre modèle, nous considérons les patrons suivants : *sequence*, *AND-split*, *OR-split*, *XOR-split*, *AND-join*, *OR-join*, *XOR-join* et *m-Out-Of-n* [vdABtHK00]. Dans ce qui suit, nous détaillons les patrons que nous utilisons dans notre exemple et nous donnons leurs définitions formelles. Les explications et les définitions des autres patrons sont données dans l'annexe A.

Figure 4.11 Un patron de composition est une fonction qui définit un `flot de contrôle` (InstancePatron) à partir d'un ensemble de services donné.



4.3.1 Patron AND-split

[vdABtHK00] définit un patron *AND-split* comme un point dans le procédé d'un workflow où un processus de contrôle se divise en plusieurs processus de contrôle en parallèle, permettant ainsi une exécution simultanée d'un ensemble de processus d'exécution.

[16] Étant donné un ensemble E, nous notons $\mathcal{P}(E)$ pour désigner l'ensemble des sous ensemble de E. Nous reprenons cette notation dans la suite.

Selon notre approche, un patron *AND-split* spécifie qu'un ensemble de services seront activés après la terminaison d'un autre service. La figure 4.13.a illustre le `flot de contrôle` résultat de l'application du patron *AND-split* à l'ensemble de services $\{SBC, RH, RV\}$.

DÉFINITION 4.21 (PATRON *AND-split*)
Nous définissons le patron de composition *AND-split* comme la fonction suivante :

$$\begin{array}{rccc} \text{AND-split :} & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FC}ontrôle \\ & \{s_0, s_1, \ldots, s_n\} & \longmapsto & fc = (ES, \mathcal{P}rec) \text{ tq} \end{array}$$

- $ES = \{s_0, s_1, \ldots, s_n\}$,
- $\mathcal{P}rec(s_0.activer()) =$ *événement externe à* fc et
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = s_0.terminé$.

4.3.2 Patron AND-join

[vdABtHK00] définit un patron *AND-join* comme un point dans le procédé d'un workflow où plusieurs processus de contrôle en parallèle convergent et se synchronisent en un seul processus d'exécution.

Selon notre approche, un patron *AND-join* spécifie qu'un service sera activé après la terminaison d'un ensemble d'autres services. La figure 4.13.b illustre le `flot de contrôle` résultat de l'application du patron *AND-join* à l'ensemble de services $\{RH, RV, PL\}$.

DÉFINITION 4.22 (PATRON *AND-join*)
Nous définissons le patron de composition *AND-join* comme la fonction suivante :

$$\begin{array}{rccc} \text{AND-join :} & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FC}ontrôle \\ & \{s_1, \ldots, s_n, s_0\} & \longmapsto & fc = (ES, \mathcal{P}rec) \text{ tq} \end{array}$$

- $ES = \{s_1, \ldots, s_n, s_0\}$,
- $\mathcal{P}rec(s_0.activer()) = \bigwedge_{i=1\ldots n} s_i.terminé$ et
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) =$ *événement externe à* fc.

4.3.3 Patron XOR-split

[vdABtHK00] définit un patron *XOR-split* comme un point dans le procédé d'un workflow où, selon une condition, un seul processus d'exécution parmi d'autres sera choisi.

Selon notre approche, un patron *XOR-split* spécifie qu'un service, parmi plusieurs, sera activé suite à la terminaison d'un autre service. La figure 4.13.c illustre le `flot de contrôle` résultat de l'application du patron *XOR-split* à l'ensemble de services $\{PL, EDF, EDD, EDT\}$.

DÉFINITION 4.23 (PATRON *XOR-split*)
Nous définissons le patron de composition *XOR-split* comme la fonction suivante :

$$\begin{array}{rccc} \text{XOR-split :} & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FC}ontrôle \\ & \{s_0, s_1, \ldots, s_n\} & \longmapsto & fc = (ES, \mathcal{P}rec) \end{array}$$

- $ES = \{s_0, s_1, \ldots, s_n\}$,
- $\mathcal{P}rec(s_0.activer()) =$ *événement externe à* fc et
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = s_0.terminé \bigwedge c_i$ | il y a toujours un seul c_j $(1 \leq j \leq n)$ évalué à vrai après la terminaison de s_0.

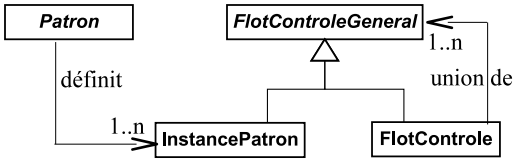
4.3.4 Spécification du flot de contrôle d'un SCT

Nous appelons `instance de patron`, le `flot de contrôle` résultat de l'application d'un patron à un ensemble de services donné. Soit un patron *pat* et soit S un ensemble de services, $pat(S)$ est une `instance du patron` *pat*.

Dans notre approche nous définissons le `flot de contrôle` d'un SCT comme l'union d'`instances de patron` (voir le diagramme UML de la figure 4.12). Plus formellement :

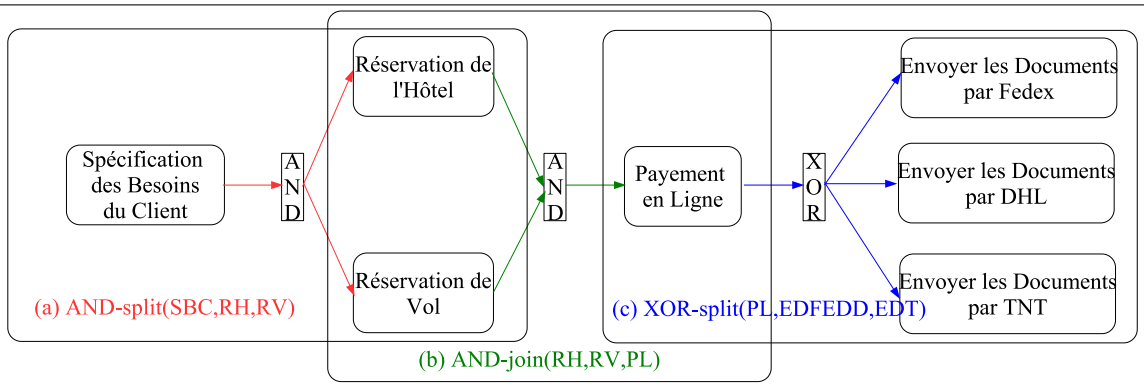
$$\forall \text{ SCT } sc = (ES, \mathcal{P}rec) \; \exists \text{ un ensemble de patrons } \{P_1, \ldots, P_n\} \text{ et une partition } S \text{ de } ES : S = \{S_1, \ldots S_n\} \text{ (avec } ES = \bigcup_{1 \leq i \leq n}(S_i) \mid getFContrôle(sc) = \bigcup_{1 \leq i \leq n} P_i(S_i)$$

Figure 4.12 Le `flot de contrôle` d'un SCT est l'union d'un ensemble d'instances de patron.



Exemple Le `flot de contrôle` du service d'organisation de voyage en ligne illustré par la figure 4.13 est défini comme l'union d'instances de patron suivante : $AND\text{-}split(SBC,RH,RV) \bigcup AND\text{-}join(RH,RV,PL) \bigcup XOR\text{-}split(PL,EDF,EDD,EDT)$.

Figure 4.13 Le `flot de contrôle` du service d'organisation de voyage en ligne peut être défini comme une union cohérente d'`instances de patron`.



Le flot de contrôle du service d'organisation de voyage en ligne est défini comme l'union suivante d'instances de patron

AND-split(SBC,RH,RV) ∪ AND-join(RH,RV,PL) ∪ XOR-split(PL,EDFEDD,EDT)

Afin de ne pas avoir des unions non sensées (instances disjointes) ou incohérentes (avec des sémantiques incompatibles), nous utilisons la grammaire contextuelle à gauche illustrée au tableau 4.1. Nous considérons une union d'instances de patron comme un mot dont les terminaux sont les «`instances de patron`» et le symbol «$\bigcup$». Cette grammaire définit le langage des unions cohérentes. Un `flot de contrôle` est cohérent *si et seulement si* l'union des instances correspondante ou une union équivalente est un mot généré par cette grammaire. Deux unions sont équivalentes si elles sont constituées des mêmes instances de patron.

$$
\begin{array}{rcl}
\mathcal{G} & \longrightarrow & Sequence(a,b) \bigcup \mathcal{A} \mid \\
 & & AND-split(s_0,s_1,\ldots,s_n) \bigcup \mathcal{B} \mid \\
 & & OR-split(s_0,s_1,\ldots,s_n) \bigcup \mathcal{C} \mid \\
 & & XOR-split(s_0,s_1,\ldots,s_n) \bigcup \mathcal{D} \mid \\
Sequence(a,b) \bigcup \mathcal{A} & \longrightarrow & Sequence(a,b) \bigcup Sequence(b,c) \bigcup \mathcal{A} \mid \\
 & & Sequence(a,b) \bigcup AND-split(b,b_1,\ldots,b_n) \bigcup \mathcal{B} \mid \\
 & & Sequence(a,b) \bigcup OR-split(b,b_1,\ldots,b_n) \bigcup \mathcal{C} \mid \\
 & & Sequence(a,b) \bigcup XOR-split(b,b_1,\ldots,b_n) \bigcup \mathcal{D} \mid \\
 & & Sequence(a,b) \bigcup \varepsilon \\
split(s_0,s_1,\ldots,s_n) \bigcup \mathcal{B} & \longrightarrow & split(s_0,s_1,\ldots,s_n) \bigcup AND-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
 & & split(s_0,s_1,\ldots,s_n) \bigcup OR-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
 & & split(s_0,s_1,\ldots,s_n) \bigcup XOR-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
 & & split(s_0,s_1,\ldots,s_n) \bigcup m-out-of-n(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
split(s_0,s_1,\ldots,s_n) \bigcup \mathcal{C} & \longrightarrow & split(s_0,s_1,\ldots,s_n) \bigcup OR-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
 & & split(s_0,s_1,\ldots,s_n) \bigcup XOR-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
split(s_0,\ldots,s_n) \bigcup \mathcal{D} & \longrightarrow & split(s_0,\ldots,s_n) \bigcup XOR-join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} \mid \\
 & & split(s_0,\ldots,s_n) \bigcup \varepsilon \\
join(s_1,\ldots,s_n,s_{n+1}) \bigcup \mathcal{E} & \longrightarrow & join(s_1,\ldots,s_n,s_{n+1}) \bigcup Sequence(s_{n+1},s') \bigcup \mathcal{A} \mid \\
 & & join(s_1,\ldots,s_n,s_{n+1}) \bigcup AND-split(s_{n+1},s'_1,\ldots,s'_n) \bigcup \mathcal{B} \mid \\
 & & join(s_1,\ldots,s_n,s_{n+1}) \bigcup OR-split(s_{n+1},s'_1,\ldots,s'_n) \bigcup \mathcal{C} \mid \\
 & & join(s_1,\ldots,s_n,s_{n+1}) \bigcup XOR-split(s_{n+1},s'_1,\ldots,s'_n) \bigcup \mathcal{D} \mid \\
 & & join(s_1,\ldots,s_n,s_{n+1}) \bigcup \varepsilon
\end{array}
$$

TAB. 4.1 – Grammaire définissant toutes les unions cohérentes `d'instances de patron`.

Cette grammaire assure la connexion cohérente des instances de patron d'un SCT. Ainsi, elle postule que :

- un flot de contrôle doit commencer par une instance d'un des patrons *sequence*, *AND-split*, *OR-split* ou *XOR-split*.
- une instance du patron *sequence* ou d'un patron de synchronisation est suivie par une instance d'un des patrons *sequence*, *AND-split*, *OR-split* ou *XOR-split*.
- une instance du patron *AND-split* doit être suivie par une instance d'un des patrons *AND-join*, *OR-join* ou *XOR-join*.
- une instance du patron *OR-split* doit être suivie par une instance d'un des patrons *OR-join* ou *XOR-join*.
- une instance du patron *XOR-split* doit être suivie par une instance du patron *XOR-join*.

4.3.5 Synthèse

Dans cette section, nous avons introduit le concept de patron de composition et nous avons illustré comment nous l'utilisons pour spécifier le flot de contrôle d'un SCT. Un patron de composition *pat* est une fonction qui retourne à partir d'un ensemble de service S un `flot de contrôle` $pat(S)$. Nous appelons $pat(S)$ une instance du patron *pat*. Nous définissons le `flot de contrôle` d'un SCT comme une union cohérente d'`instances de patron` (mot généré par la grammaire illustré au tableau 4.1).

4.4 Flot transactionnel potentiel

Les dépendances transactionnelles d'un SCT dépendent de la sémantique de ses dépendances d'activation. Ceci sous-entend que pour un `flot de contrôle` donné fc, il existe un ensemble de dépendances transactionnelles potentielles qui peuvent être définies conformément à ses dépendances d'activation. Une dépendance transactionnelle potentielle est une dépendance transactionnelle qui peut être éventuellement définie par un SCT ayant fc comme `flot de contrôle`. Par exemple, étant donné le `flot de contrôle` défini à la figure 4.13, une dépendance de compensation du service PL vers le service RH peut être éventuellement défini par un SCT défini selon ce `flot de contrôle`. Ainsi $depCps(PL, RH)$ est une dépendance de compensation potentielle induite par ce `flot de contrôle`. Cependant définir une dépendance d'alternative du service RV vers le service RH est incohérent avec la sémantique de ce flot de contrôle. Nous appelons `flot transactionnel potentiel` d'un `flot de contrôle` le `flot transactionnel` incluant toutes les dépendances transactionnelles potentielles qui peuvent être éventuellement définies conformément à ce `flot de contrôle`.

4.4.1 Flot transactionnel potentiel d'un patron

A chaque patron, pat, nous associons une fonction $potentiel_{pat}$ qui permet de retourner, à partir d'un ensemble de services, S, le `flot transactionnel potentiel` du `flot de contrôle` $pat(S)$ (voir diagramme UML de la figure 4.14).

Définition 4.24 (fonction «potentiel» d'un patron)
Soit un patron, pat, nous définissons la fonction $potentiel_{pat}$ comme suit :

$$\begin{array}{rccc} potentiel_{pat} : & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FT}ransactionnel \\ & S & \longmapsto & ft = (ES, \mathcal{P}rec) \end{array}$$

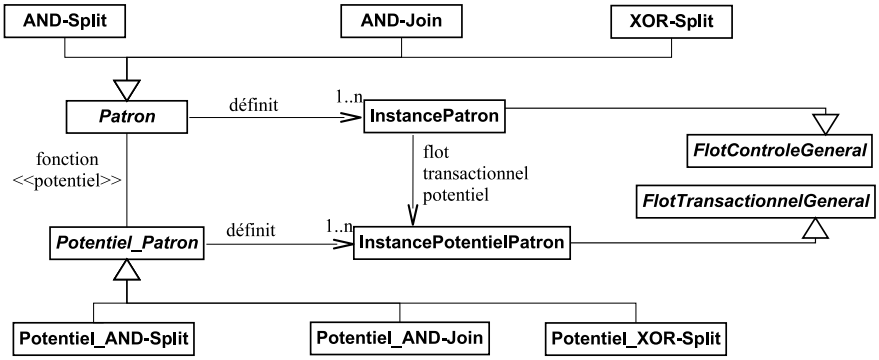
Avec ft est le `flot transactionnel potentiel` du `flot de contrôle` $pat(S)$, $ES = S$ et la fonction $\mathcal{P}rec$ définit pour chaque service $s \in S$:
- $CondPtCps(s)$: la **Cond**ition **P**otentielle de **Comp**ensation de s. $CondPtCps(s)$ spécifie quand s sera éventuellement compensé.
- $CondPtAnl(s)$: la **Cond**ition **P**otentielle d'**An**nul**a**tion de s. $CondPtAnl(s)$ spécifie quand s sera éventuellement annulé.
- $CondPtAlt(s)$: la **Cond**ition **P**otentielle d'**Alt**ernative de s. $CondPtAlt(s)$ spécifie quand s sera éventuellement activé en tant qu'alternative.

$CondPtCps(s)$, $CondPtAnl(s)$ et $CondPtAlt(s)$ et les propriétés transactionnelles correspondantes définissent le flot transactionnel potentiel.

4.4.2 Flot transactionnel potentiel du patron AND-join

La fonction «potentiel» du patron *AND-join*, $potentiel_{AND-join}$ définit à partir d'un ensemble de services ($\{s_1, \ldots, s_n, s_0\}$) les dépendances transactionnelles suivantes : tout service s_i en échec ($i \neq 0$) va compenser ou annuler tout service s_j (selon son état courant) ($j \neq i$ et $j \neq 0$). Le service s_0 va compenser tout les autres services en cas d'échec ou de compensation.

Figure 4.14 Chaque patron de composition *pat* possède une fonction «potentiel» qui définit, à partir d'un ensemble de services S, le `flot transactionnel potentiel` du `flot de contrôle` $pat(S)$.



DÉFINITION 4.25 (FONCTION «POTENTIEL» D'UN *AND-join*)
La fonction «potentiel» du patron *AND-join* est définie comme suit :

$$potentiel_{AND-join} : \mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FT}ransactionnel$$
$$S = \{s_1, \ldots, s_n, s_0\} \longmapsto ft = (ES, \mathcal{P}rec)$$

avec ft le flot `transactionnel potentiel` du `flot de contrôle` *AND-join(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_1, \ldots, s_n, s_0\}$:
- $CondPtAlt(s_i) = faux \ \forall 0 \leq i \leq n$,
- $CondPtCps(s_0) =$ *événement externe à ft*,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) = s_0.échoué \bigoplus s_0.compensé \bigoplus_{1 \leq j \leq n, i \neq j} s_j.échoué$,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) = \bigoplus_{1 \leq j \leq n, i \neq j} s_j.échoué$.

Exemple Selon le `flot de contrôle` défini à la figure 4.13, le service RV peut être éventuellement compensé (respectivement annulé) suite à l'échec de RH, l'échec de PL ou la compensation de PL (respectivement l'échec de RH). Ceci signifie que $CondPtCps(RV) = RH.échoué \bigoplus PL.échoué \bigoplus PL.copmensé$ et que $CondPtAnl(RV) = RH.échoué$. Réciproquement, le service RH peut être éventuellement compensé (respectivement annulé) suite à l'échec de RV, l'échec de PL ou la compensation de PL (respectivement l'échec de RV). Ceci veut dire que $CondPtCps(RH) = RV.échoué \bigoplus PL.échoué \bigoplus PL.copmensé$ et que $CondPtAnl(RH) = RV.échoué$. La figure 4.16.b illustre le `flot transactionnel potentiel` du `flot de contrôle` *AND-join*(RH, RV, PL).

4.4.3 Flot transactionnel potentiel du patron XOR-split

La fonction «potentiel» du patron *XOR-split*, $potentiel_{XOR-split}$ définit à partir d'un ensemble de services ($\{s_0, s_1, \ldots, s_n\}$) les dépendances transactionnelles suivantes : tout service s_i est une alternative d'un service s_j avec 1≤i,j≤n et i≠j. Tout service s_i (1≤i≤n) va compenser le service s_0 en cas d'échec ou de compensation.

DÉFINITION 4.26 (FONCTION «POTENTIEL» D'UN *XOR-split*)
La fonction «potentiel» du patron *XOR-split* est définie comme suit :

$$potentiel_{XOR-split} : \quad \mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FT}ransactionnel$$
$$S = \{s_0, s_1, \ldots, s_n\} \longmapsto ft = (ES, \mathcal{P}rec)$$

avec ft le `flot transactionnel potentiel` du `flot de contrôle` *XOR-split(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_0, s_1, \ldots, s_n\}$:
- $CondPtAlt(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAlt(s_i) = \bigoplus_{1 \leq j \leq n, i \neq j} s_j.échoué$,
- $CondPtCps(s_0) = \bigoplus_{1 \leq j \leq n} s_j.échoué \bigoplus_{1 \leq j \leq n} s_j.compensé$,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) =$ *événement externe à ft*,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) = faux$.

Exemple Selon le `flot de contrôle` défini à la figure 4.13 le service *EDD* peut éventuellement être activé comme alternative à *EDF* ou *EDT*. Ceci signifie que $CondPtAlt(EDD) = (EDF.échoué \bigwedge EDF.échoué) \bigoplus (EDF.échoué \bigwedge EDF.initial) \bigoplus (EDF.initial \bigwedge EDF.échoué)$. La figure 4.16.c illustre le `flot transactionnel potentiel` du `flot de contrôle` *XOR-split*(*PL*,*EDF*,*EDD*,*EDT*).

4.4.4 Flot transactionnel potentiel du patron AND-split

La fonction «potentiel» du patron *AND-split*, $potentiel_{AND-split}$ définit à partir d'un ensemble de services (s_0, s_1, ..., s_n) les dépendances transactionnelles suivantes : il y a une dépendance de compensation de s_i ($1 \leq i \leq n$) vers s_0 conformément à la politique de synchronisation des services s_1, ..., s_n.

DÉFINITION 4.27 (FONCTION «POTENTIEL» D'UN *AND-split*)
La fonction «potentiel» du patron *AND-split* est définie comme suit :

$$potentiel_{AND-split} : \quad \mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FT}ransactionnel$$
$$S = \{s_0, s_1, \ldots, s_n\} \longmapsto ft = (ES, \mathcal{P}rec)$$

avec ft le `flot transactionnel potentiel` du `flot de contrôle` *AND-split(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_0, s_1, \ldots, s_n\}$:
- $CondPtAlt(s_i) = faux\ \forall 0 \leq i \leq n$,
- $CondPtCps(s_0) = \bigoplus_{1 \leq i \leq n} CondPtCps(s_i)$,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) =$ *défini par le patron de synchronisation*,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) =$ *défini par le patron de synchronisation*.

Exemple Selon le `flot de contrôle` défini à la figure 4.13, le service *SBC* peut éventuellement être compensé en cas d'échec de *RH* ou de *RV*. Ceci signifie que $CondCpsPt(SBC) = RH.échoué \bigvee RV.échoué$. La figure 4.16.a illustre le `flot transactionnel potentiel` du `flot de contrôle` *AND-split*(*SBC*,*RH*,*RV*).

4.4.5 Flot transactionnel potentiel d'un flot de contrôle

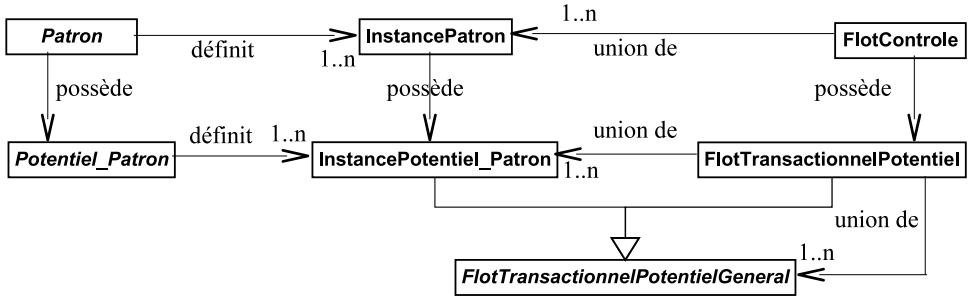
Le `flot transactionnel potentiel` d'un `flot de contrôle` est l'union des `flots transactionnels potentiels` de ses `instances de patron` (voir diagramme UML de la figure 4.15).

DÉFINITION 4.28 (FONCTION «POTENTIEL» D'UN `FLOT DE CONTRÔLE`)
Nous définissons la fonction *potentielle* qui permet de retourner le `flot transactionnel potentiel` d'un `flot de contrôle` donné.

$$\begin{array}{lll} potentielle : & \mathcal{FC}ontrôle \longrightarrow & \mathcal{FT}ransactionnel \\ & fc = \bigcup_i pat_i(S_i) \longmapsto & ftp = \bigcup_i potentiel_{pat_i}\ (S_i) \end{array}$$

Exemple La figure 4.16 illustre le flot transactionnel potentiel du flot de contrôle du service d'organisation de voyage en ligne, fc, illustré par la figure 4.13. Ce flot transactionnel est l'union des flots transactionnels potentiels des instances des patrons du flot de contrôle :
$potentielle(fc)$ = potentielle[*AND-split* $(\{SBC, RH, RV\})$ $\bigcup$ *AND-join*($\{RH, RV, PL\}$) $\bigcup$ *XOR-split* $(\{PL, EDD, EDF, EDT\})$] = $potentiel_{AND-split}$ $(\{SBC, RH, RV\})$ $\bigcup$ $potentiel_{AND-join}$ $(\{RH, RV, PL\})$ $\bigcup$ $potentiel_{XOR-split}$ $(\{PL, EDD, EDF, EDT\})$.

Figure 4.15 Le `flot transactionnel potentiel` d'un `flot de contrôle` est l'union des `flots transactionnels potentiels` des ses `instances de patron`.



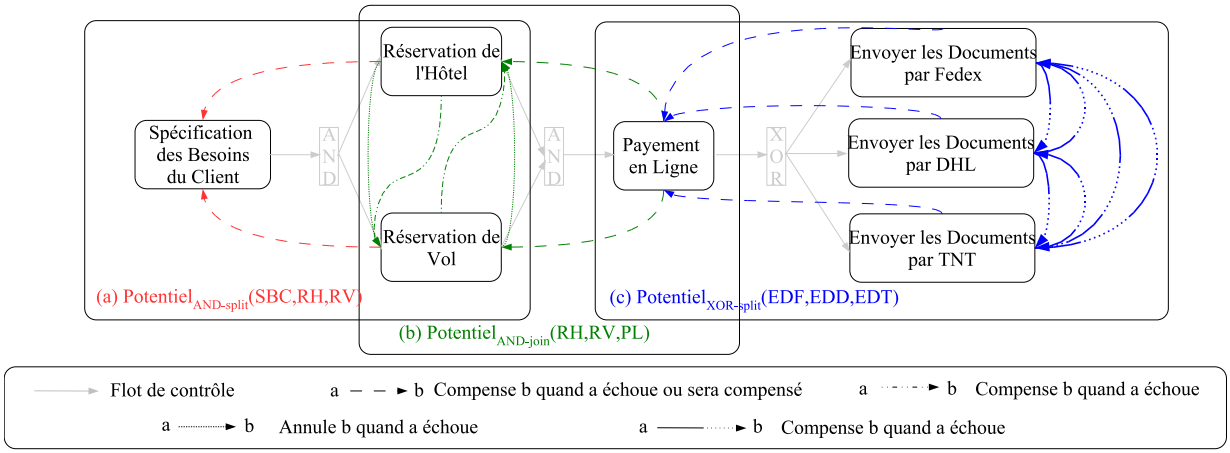
4.4.6 Synthèse

Dans cette section, nous avons montré qu'un `flot de contrôle` fixe implicitement un ensemble de dépendances transactionnelles potentielles qui peuvent éventuellement être définies conformément aux sémantiques de ses dépendances d'activation. Ainsi, chaque `flot de contrôle` possède un `flot transactionnel potentiel` incluant toutes ses dépendances transactionnelles potentielles. Nous avons défini pour chaque patron de composition une fonction «potentiel». Une fonction «potentiel» d'un patron détermine le `flot transactionnel potentiel` de tout flot de contrôle défini par ce patron. Le `flot transactionnel potentiel` d'un flot de contrôle est l'union des `flots transactionnels potentiels` de ses instances de patron.

4.5 Spécification d'un SCT

Dans la section 4.2, nous avons montré qu'un SCT est bien défini par son `flot de contrôle` et son `flot transactionnel`. Ensuite, nous avons montré dans la section 4.3 comment nous

Figure 4.16 `Flot transactionnel potentiel` du `flot de contrôle` du service d'organisation de voyage en ligne.



Le flot transactionnel potentiel du flot de contrôle du service d'organisation de voyage en ligne est défini comme l'union des fonctions <<potentiel>> de ses instances de patrons

$potentiel_{AND\text{-}split}(SBC,RH,RV) \cup potentiel_{AND\text{-}join}(RH,RV,PL) \cup potentiel_{XOR\text{-}split}(EDF,EDD,EDT))$

définissons le flot de contrôle d'un SCT en utilisant des patron de composition. Dans cette section, nous montrons comment nous définissons un SCT et en particulier son `flot transactionnel`.

Étant donné deux flots transactionnels $ft1$ et $ft2$ définis selon le même `flot de contrôle`, nous disons que $ft1$ est *inclus simplement* dans $ft2$ ($ft1 \subseteq ft2$) *si et seulement si* l'ensemble des dépendances transactionnels de $ft1$ est inclus dans celle de $ft2$.

DÉFINITION 4.29 (INCLUSION SIMPLE D'UN `FLOT TRANSACTIONNEL` DANS UN AUTRE)
Soit fc un `flot de contrôle`, $ft1$ et $ft2$ deux flots transactionnels définis selon fc. $ft2$ est *inclus simplement* dans $ft1$, $ft2 \subseteq ft1$, *si et seulement si* il existe $ft3$ un `flot transactionnel` défini selon fc tq $ft1 = ft2 \bigcup ft3$.

Exemple Les flots transactionnels des services cs_1 et cs_2 (figures 4.18.a et 4.18.b) sont *inclus simplement* dans le flot transactionnel potentiel illustré par la figure 4.16.

Plusieurs SCT peuvent être définis selon le même `flot de contrôle` fc. Chacun de ces SCT étend fc par un `flot transactionnel` *inclus simplement* dans le `flot transactionnel potentiel` de fc. Plus formellement :

$$\forall \text{ SCT } sc \text{ défini selon un flot de contrôle } fc$$
$$sc = getFContrôle(sc) \bigcup getFTransactionnel(sc) \text{ tel que}$$
$$getFContrôle(sc) = fc \text{ et } getFTransactionnel(sc) \subseteq potentielle(fc).$$

Exemple La figure 4.18 illustre deux SCT définis selon le même flot de contrôle décrit à la figure 4.13. Chacun de ces services étend ce flot de contrôle par un flot transactionnel inclus dans son flot transactionnel potentiel illustré par la figure 4.16.

Figure 4.17 Le `flot transactionnel` d'un SCT est *inclus simplement* dans le `flot transactionnel potentiel` de son `flot de contrôle`.

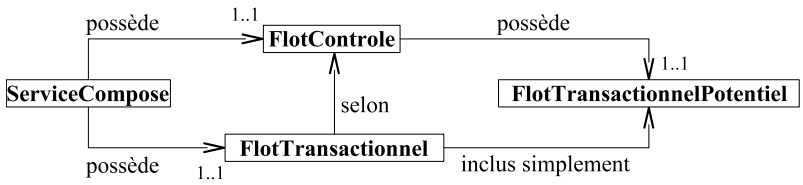
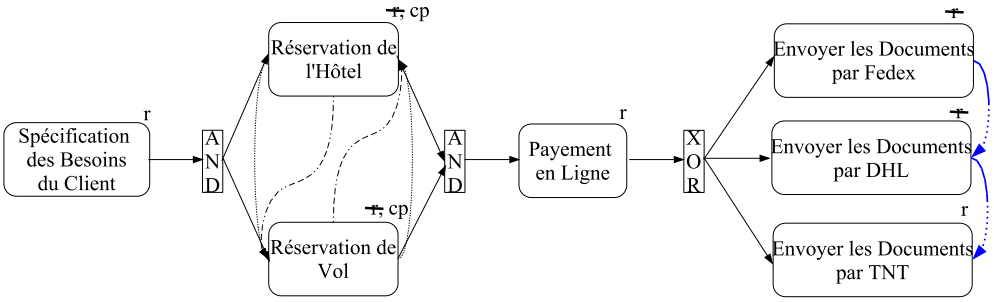
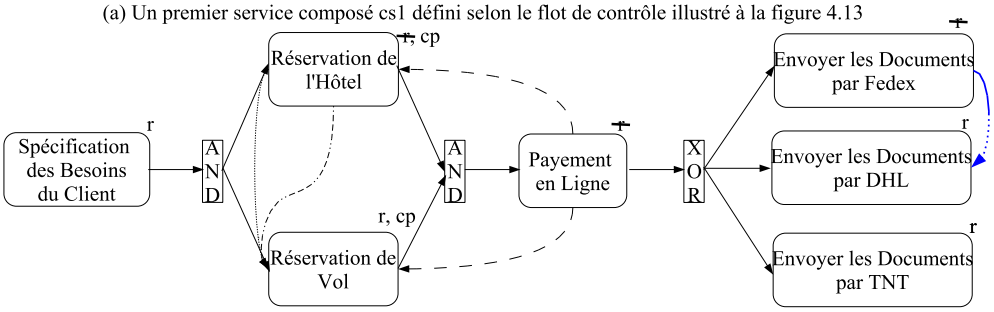


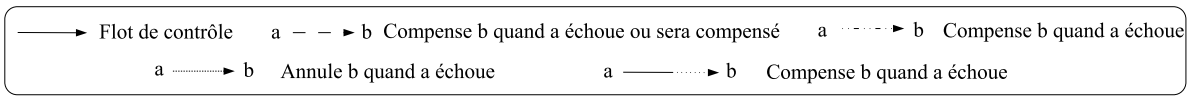
Figure 4.18 Deux services d'organisation de voyage en ligne définis selon le même flot de contrôle.



(a) Un premier service composé cs1 défini selon le flot de contrôle illustré à la figure 4.13



(b) Un deuxième service composé cs2 défini selon le même flot de contrôle illustré à la figure 4.13



4.6 Conclusion

Dans ce chapitre, nous avons présenté notre modèle de services Web transactionnels. Le but de ce modèle est :

- d'enrichir la description des services Web pour mieux exprimer leurs propriétés transactionnelles. Ce qui permet aux concepteurs de services Web composés d'exploiter ces propriétés pour assurer des compositions plus correctes et fiables.
- de combiner l'aspect coordination et l'aspect transactionnel. Ce qui permet d'assurer un niveau élevé de flexibilité et de développer un ensemble de techniques pour préserver un certain degré de correction.

Pour ce faire, nous avons introduit au départ la notion de service Web transactionnel. Un service Web transactionnel (SWT) est un service Web qui manifeste des propriétés transactionnelles. Nous modélisons le comportement transactionnel d'un SWT via un diagramme à

transitions d'états.

Ensuite, nous avons décrit comment nous combinons un ensemble de SWT pour définir des services composés transactionnels (SCT). Nous avons distingué en particulier entre le flot de contrôle et le flot transactionnel d'un SCT. Le flot de contrôle définit l'ordre d'exécution des services composants. Le flot transactionnel définit les mécanismes de recouvrement en cas d'échecs.

Enfin, nous avons présenté la notion de patron de composition et nous avons montré comment nous l'utilisons pour spécifier un SCT : définir son flot de contrôle et son flot transactionnel.

L'originalité de notre modèle est qu'il a pu réconcilier les systèmes de workflow et les MTA, deux technologies fortes, souvent considérées concurrentes. En effet, d'une part un SCT peut être considéré comme un flot de services autonomes et hétérogènes. D'autre part, il peut être considéré aussi comme une transaction structurée où les services composants sont les sous-transactions et les interactions sont les dépendances transactionnelles. Par rapport au système de workflow, notre modèle permet d'intégrer le flot transactionnel un concept qui a été toujours (relativement) ignoré. Par comparaison au MTA, notre modèle permet des structures de contrôles plus complexes. En effet, ce modèle sert de base pour l'ensemble des techniques développées (que nous présentons dans le chapitre suivant) pour assurer des exécutions correctes et des compositions fiables.

Chapitre 5

Assurer des compositions fiables de services Web

Table des matières

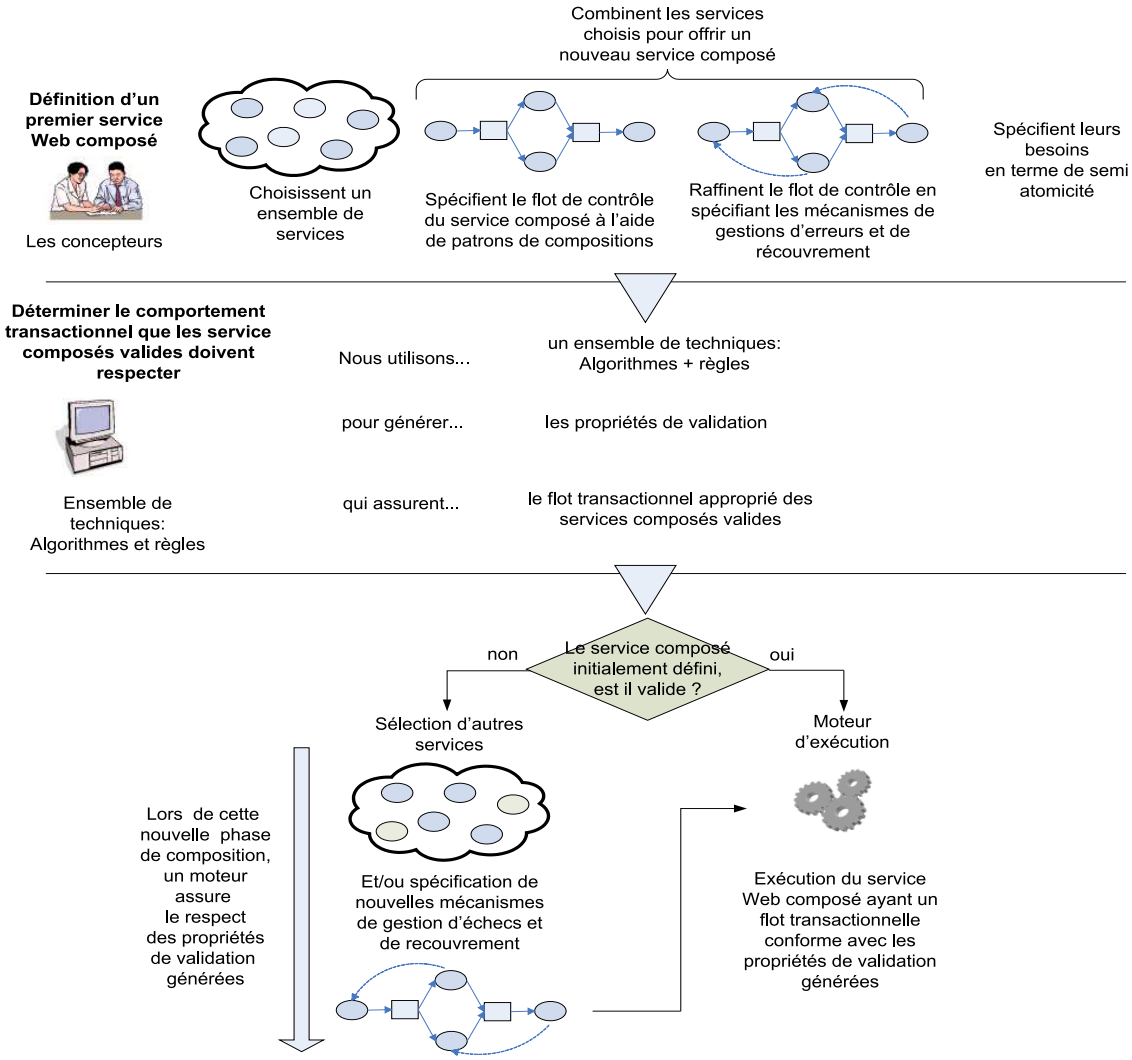
5.1 Approche par validation du SCT 71
5.1.1 Critère de correction des exécutions d'un SCT : *ETA* 73
5.1.2 Service composé valide 74
5.1.3 Assurer des SCT valides 77
5.1.4 Synthèse 85
5.2 Approche par ré-ingénierie du SCT 85
5.2.1 Extraction de l'ensemble des états de terminaison 87
5.2.2 Découverte du flot transactionnel d'un SCT 89
5.2.3 Amélioration du flot transactionnel d'un SCT 89
5.2.4 Synthèse 92
5.3 Approche par composition de patrons transactionnels 93
5.3.1 Patrons transactionnels 93
5.3.2 Composition de patrons transactionnsels 94
5.3.3 Synthèse 96
5.4 Conclusion 97

Nous distinguons trois approches principales pour assurer des compositions fiables de services. Ces trois approches peuvent être appliquées séparément comme elle peuvent être combinées. La première approche repose sur la validation du modèle de composition du SCT conformément aux besoins des concepteurs [BPG05; BGP05; BGP04]. Toutes les exécutions des instances d'un modèle valide sont correctes conformément aux besoins transactionnels exigés. La deuxième approche procède par ré-ingénierie du modèle de composition après analyse de ses traces d'exécution [GBG04b; GBG04a]. La troisième approche procède par composition de `patrons transactionnels` [BGPG05]. Un `patron transactionnel` est un nouveau concept que nous introduisons. Il combine la flexibilité des workflows et la fiabilité des modèles transactionnels.

5.1 Approche par validation du SCT

Dans cette approche, nous procédons par validation du modèle de composition pour assurer des exécutions correctes de SCT. Toutes les exécutions des instances d'un modèle valide (voir

Figure 5.1 Vue globale de notre approche par validation du modèle de composition.



définition 5.4) sont correctes (voir paragraphe 5.1.1) conformément aux besoins des concepteurs en terme de fiabilité. Avant de présenter en détail les différents concepts et techniques utilisés, nous présentons une vue globale de cette approche (voir la figure 5.1).

- **Définition d'un SCT initial :** Au départ, les concepteurs choisissent à la volée un ensemble de services et les combinent pour définir un service Web composé. Ils utilisent un ensemble de patrons de composition (*Sequence*, *AND-join*, *AND-split*, etc.) pour spécifier le flot de contrôle du SCT. Ensuite, ils peuvent enrichir ce flot de contrôle par un flot transactionnel inclus dans le flot transactionnel potentiel défini par les patrons choisis. Enfin, ils spécifient leurs besoins transactionnels en terme de semi atomicité[17] (voir paragraphe 5.1.1).
- **Calcule des propriétés de validation :** Nous utilisons un ensemble de techniques pour

[17] Rappelons que le terme semi atomicité est notre traduction de l'anglicisme «failure atomicity». La semi atomicité est la relaxation de la propriété d'atomicité du modèle ACID. Elle fait référence à des exécutions correctes (en terme de fiabilité) même en présence d'échecs de certaines sous-transactions (les services composants dans notre cas) d'une transaction globale.

calculer à partir du flot de contrôle spécifié et des besoins transactionnels demandés un ensemble de `propriétés de validation`. Ces propriétés définissent les flots transactionnels appropriés des SCT valides (voir définition 5.4). Tout SCT valide doit respecter ces `propriétés de validation`.

- **Définition d'un SCT valide :** Si le SCT initial n'est pas valide (ne respecte pas les `propriétés de validation`), les concepteurs peuvent (i) choisir de nouveaux services (avec des nouvelles propriétés transactionnelles) et/ou (ii) enrichir le flot de contrôle spécifié avec de nouvelles dépendances. Durant cette phase, un moteur assiste les concepteurs pour composer des SCT valides en respectant les `propriétés de validation` générées. Une fois qu'un SCT valide est composé, il peut être déployé pour être exécuté.

Le reste de cette section est organisé comme suit. Dans le paragraphe 5.1.1, nous introduisons notre critère de correction des exécutions (qui prend en compte les besoins des concepteurs en terme de fiabilité). Dans le paragraphe 5.1.2, nous définissons qu'est ce qu'un service composé valide selon ce critère de correction. Nous détaillons ensuite, dans le paragraphe 5.1.3, l'ensemble des techniques que nous utilisons pour générer les `propriétés de validation` que les services composés valides doivent vérifier.

Nous reprenons notre exemple de service d'organisation de voyage en ligne pour illustrer les divers concepts et techniques utilisés. Nous supposons que les concepteurs spécifient un service initial dont le flot de contrôle est donné à la figure 5.2.

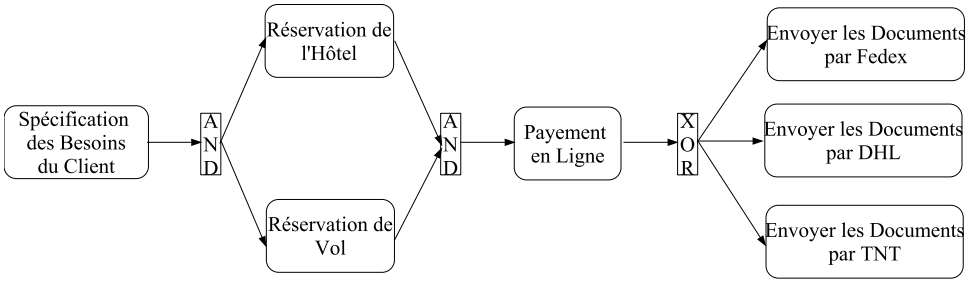
5.1.1 Critère de correction des exécutions d'un SCT : *ETA*

Plusieurs exécutions peuvent être instanciées selon le même modèle de composition. L'état d'une instance d'un SCT composé de n services à un instant donné est le tuple $(x_1, x_2, \ldots x_n)$, avec x_i l'état du service s_i à cet instant. L'ensemble des états de terminaison d'un SCT est l'ensemble de tous les états de terminaison possibles de ses instances. Nous notons $\mathcal{ET}$ le domaine des ensembles des états de terminaison de tous les services composés. La figure 5.6 illustre l'ensemble des états de terminaison du service composé illustre par la figure 5.8.

Définition 5.1 (Ensemble d'états de terminaison)
Nous définissons la fonction $calculET : \mathcal{SCT} \longrightarrow \mathcal{P}(\mathcal{ET})$ qui permet de retourner l'ensemble des états de terminaison d'un service composé sc : $calculET(sc)$.

Les concepteurs précisent leurs besoins en terme de semi atomicité en spécifiant l'`ensemble d'états de terminaison acceptés` que nous notons ETA. Un `état de terminaison accepté` est un état dans lequel les concepteurs acceptent qu'une instance se termine. Une instance atteint un tel état si (i) elle réussit à atteindre ses objectifs et se termine avec succès ou (ii) si elle échoue et compense tous les effets indésirables, dus à l'execution partielle, conformément aux exigences des concepteurs. Une exécution d'un SCT est correcte *si et seulement si* elle se termine dans un état de terminaison accepté par les concepteurs. ETA est ainsi notre critère de correction. Il est important de préciser qu'un ETA est défini par rapport à un flot de contrôle pré-défini.

Figure 5.2 Flot de contrôle du service composé d'organisation de voyage en ligne.



5.1.2 Service composé valide

État de terminaison sans échec

Nous distinguons entre deux types d'état de terminaison d'un SCT. Le premier type correspond aux états atteints lors des exécutions normales (sans échec[18] et tel qu'il est décrit par le flot de contrôle du service). Nous notons $\mathcal{ET}_{sansEchec}$ le domaine des ensembles des `états de terminaison sans échec`. L'ensemble des `états de terminaison sans échec` d'un SCT est défini par son flot de contrôle.

DÉFINITION 5.2 (ENSEMBLE D'ÉTATS DE TERMINAISON SANS ÉCHEC)
Nous définissons la fonction $calculET_{sansEchec} : \mathcal{FC}ontrôle \longrightarrow \mathcal{P}(\mathcal{ET}_{sansEchec})$ qui permet de retourner l'ensemble des états de terminaison (sans échec) défini par un flot de contrôle donné.

Ainsi, l'ensemble des `états de terminaison sans échec` d'un service composé sc est $calculET_{sansEchec}(getFContrôle(sc))$. La figure 5.3 illustre l'ensemble des états de terminaison sans échec du flot du contrôle illustré par la figure 5.2.

Figure 5.3 L'ensemble des états de terminaison (sans échec) défini par le flot de contrôle du service d'organisation de voyage en ligne.

services / et	SBC	RV	RH	PL	EDF	EDD	EDT
et_1	(terminé,	terminé,	terminé,	terminé,	terminé,	initial,	initial)
et_2	(terminé,	terminé,	terminé,	terminé,	initial,	terminé,	initial)
et_3	(terminé,	terminé,	terminé,	terminé,	initial,	initial,	terminé)

État de terminaison avec échecs

Le deuxième type d'état de terminaison correspond aux états atteints en cas d'échecs de certains services (conformément au flot transactionnel spécifié). Nous notons $\mathcal{ET}_{avecEchecs}$ le domaine

[18] Les échecs que nous désignons ici sont les échecs inattendus qui ne sont pas prises en compte dans le flot de contrôle

des ensembles des `états de terminaison avec échecs`. L'ensemble des `états de terminaison avec échecs` d'un SCT est défini par son flot transactionnel.

DÉFINITION 5.3 (ENSEMBLE D'ÉTATS DE TERMINAISON AVEC ÉCHECS)
Nous définissons la fonction $calculET_{avecEchecs} : \mathcal{FT}ransactionnel \longrightarrow \mathcal{P}(\mathcal{ET}_{avecEchecs})$ qui permet de retourner l'ensemble des états de terminaison (avec échecs) défini par un flot transactionnel donné.

Ainsi, l'ensemble des `états de terminaison avec échecs` d'un service composé sc est $calculET_{avecEchecs}(getFTransactionnel(sc))$. La figure 5.4.a illustre l'ensemble des `états de terminaison avec échecs` définis par le flot transactionnel du service cs_1 illustré par la figure 5.7.a. La figure 5.4.b illustre l'ensemble des `états de terminaison avec échecs` définis par le flot transactionnel du service cs_2 illustré par la figure 5.7.b.

Figure 5.4 L'ensembles des états de terminaison (avec échecs) des services cs_1 et cs_2 illustrés à la figure 5.7.

services / et	SBC	RV	RH	PL	EDF	EDD	EDT
$etCS1_4$	(terminé,	compensé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
$etCS1_5$	(terminé,	annulé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
$etCS1_6$	(terminé,	terminé,	terminé,	terminé,	échoué,	terminé,	initial)

(a) Ensemble d'états de terminaison avec échecs du service cs_1

services / et	SBC	RV	RH	PL	EDF	EDD	EDT
$etCS2_4$	(terminé,	compensé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
$etCS2_5$	(terminé,	annulé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
$etCS2_6$	(terminé,	terminé,	terminé,	terminé,	échoué,	terminé,	initial)
$etCS2_7$	(terminé,	échoué	terminé,	abandonné,	abandonné,	abandonné,	abandonné)

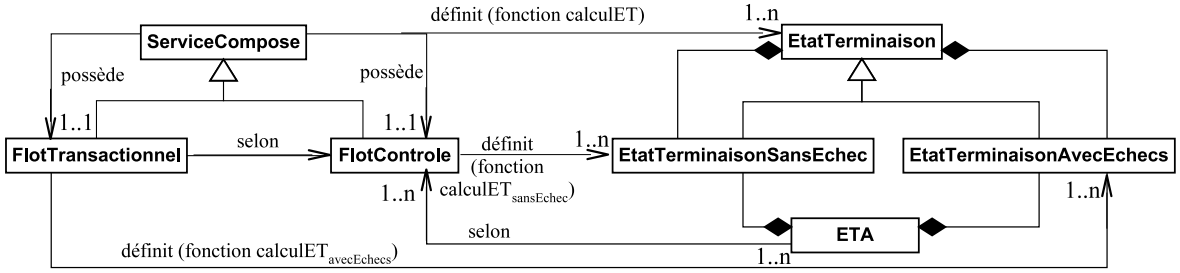
(b) Ensemble d'états de terminaison avec échecs du service cs_2

L'ensemble des états de terminaison d'un SCT, sc, est l'union des états de terminaison définis par son flot de contrôle et des états de terminaison définis par son flot transactionnel : $calculET(sc) = calculET_{sansEchec}(getFContrôle(sc)) \bigcup calculET_{avecEchecs}(getFTransactionnel(sc))$. Par exemple, les états de terminaison du service cs_1 (figure 5.7.a) est l'union des états de terminaison définis par son flot de contrôle (figure 5.3) et ceux définis par son flot transactionnel (figure 5.4.a). Les états de terminaison du service cs_2 (figure 5.7.b) est l'union des états de terminaison définis par son flot de contrôle (figure 5.3) et ceux définis par son flot transactionnel (figure 5.4.b).

De même, nous distinguons pour l'ensemble des états de terminaison acceptés, ETA, ces deux types d'état de terminaison. Nous désignons par $ETA_{sansEchec}$ et $ETA_{avecEchecs}$ l'ensemble des états de terminaison respectivement sans échec et avec échecs appartenant à ETA.

Dans ce qui suit, nous supposons que les concepteurs spécifient l'ensemble des états de terminaison illustré par la figure 5.6 comme ETA (exigé selon le flot de contrôle illustré par la figure

Figure 5.5 Nous distinguons entre deux types d'état de terminaison : *sans échec* et *avec échecs*.



5.2). L'ensemble $\{eta_1, eta_2, eta_3\}$ correspond à l'ensemble des `états de terminaison accepté sans échec` $ETA_{sansEchec}$. Il est important de noter que $ETA_{sansEchec}$ est égal à l'ensemble des `états de terminaison sans échec` défini par le flot de contrôle correspondant (illustré par la figure 5.3). L'ensemble $\{eta_4, eta_5, eta_6, eta_7, eta_8, eta_9\}$ correspond à l'ensemble des `états de terminaison accepté avec échecs` $ETA_{avecEchecs}$.

Service composé valide

Par définition, un service composé est dit valide *si et seulement si* toutes les exécutions de ses instances sont correctes. Ce qui signifie que l'état de terminaison de toute instance est un état de terminaison accepté.

Définition 5.4 (Service composé valide)
Un service composé cs est valide *si et seulement si* $calculET(cs) \subseteq ETA$.

Cette définition implique qu'un service sc est valide *si et seulement si* l'ensemble de ses `états de terminaison avec échecs` ($calculET_{avecEchecs}(getFTransactionnel(sc))$) est inclus dans l'ensemble des `états de terminaison acceptés avec échecs` ($ETA_{avecEchecs}$).

Lemme 5.1 (Service composé valide)
Un service composé cs est valide *si et seulement si* $calculET_{avecEchecs}(getFTransactionnel(cs)) \subseteq ETA_{avecEchecs}$.

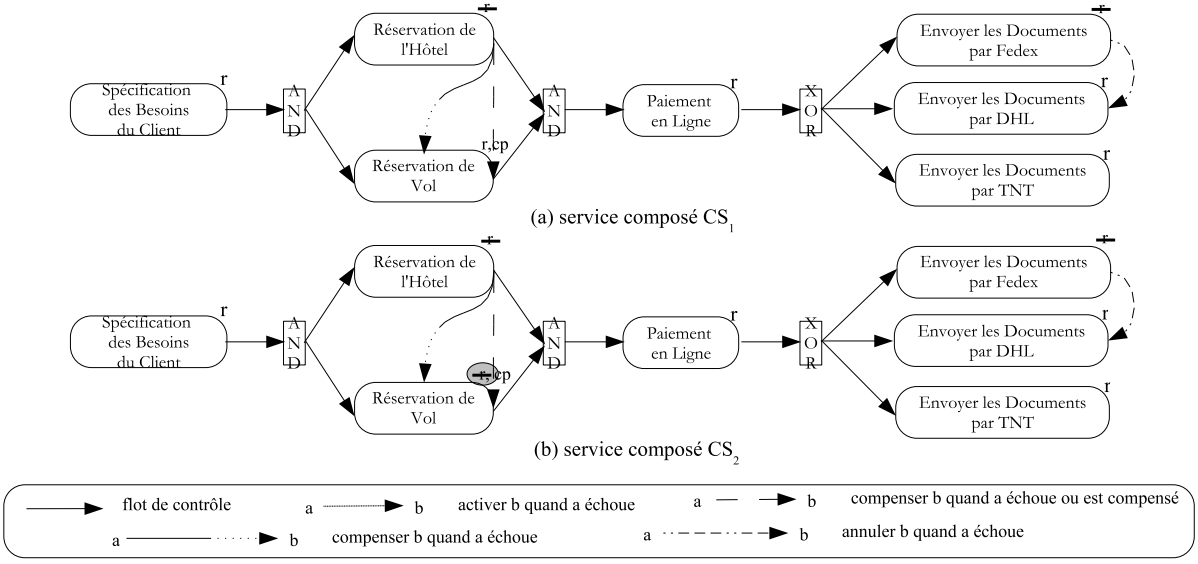
En effet, tous les services et ETA définis selon le même flot de contrôle partagent le même ensemble d'`états de terminaison sans échecs`. Tandis que l'ensemble des `états de terminaison avec échecs` varie d'un service à un autre selon son flot transactionnel et d'un ETA à un autre selon les besoins des concepteurs. Ainsi, pour un flot de contrôle donné et un ETA défini selon ce flot, tout `état de terminaison sans échec` de tout service défini selon ce flot est un état accepté. Par la suite tout service dont l'ensemble des `états de terminaison avec échecs` inclus dans $ETA_{avecEchecs}$ est valide puisque chacun de ses état de terminaison (des deux types) est un état accepté.

Exemple Selon la définition 5.4, le service cs_1 (figure 5.7.a) est valide puisque $calculET(cs_1) \subseteq ETA$. Tandis que le service cs_2 (figure 5.7.b) n'est pas valide puisque $calculET(cs_2) \not\subseteq ETA$. En effet, tous les états de terminaison du service cs_1, en particulier ceux avec échecs, sont des états acceptables. Par contre, l'état de terminaison $etCS2_7$ du service cs_2 (voir figure 5.4) n'est pas un état acceptable.

Figure 5.6 Ensemble d'états de terminaison acceptés spécifié par les concepteurs.

services / eta	SBC	RV	RH	PL	EDF	EDD	EDT
eta_1	(terminé,	terminé,	terminé,	terminé,	terminé,	initial,	initial)
eta_2	(terminé,	terminé,	terminé,	terminé,	initial,	terminé,	initial)
eta_3	(terminé,	terminé,	terminé,	terminé,	initial,	initial,	terminé)
eta_4	(terminé,	compensé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
eta_5	(terminé,	annulé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
eta_6	(terminé,	échoué,	compensé,	abandonné,	abandonné,	abandonné,	abandonné)
eta_7	(terminé,	échoué,	annulé,	abandonné,	abandonné,	abandonné,	abandonné)
eta_8	(terminé,	terminé,	terminé,	terminé,	échoué,	terminé,	initial)
eta_9	(terminé,	terminé,	terminé,	terminé,	échoué,	échoué,	terminé)

Figure 5.7 Deux services composés définis selon le même flot de contrôle.



5.1.3 Assurer des SCT valides

Dans cette section, nous présentons comment nous procédons pour assurer des SCT valides conformément aux spécifications des concepteurs (flot de contrôle et *ETA*).

Afin d'assurer des compositions valides, nous générons à partir du flot de contrôle et *ETA* spécifiés, un ensemble de `propriétés de validation`. Un SCT est valide *si et seulement si* il respecte ces propriétés. Avant de détailler pourquoi et comment nous générons ces propriétés, nous présentons un exemple illustratif qui montre leur applicabilité pour valider les SCT.

Dans ce qui suit, nous considérons un exemple où les concepteurs spécifient le flot de contrôle illustré par la figure 5.2 et l'*ETA* présenté à la figure 5.6. Nous donnons directement les `propriétés de validation` correspondantes et nous montrons comment elles permettent de valider les SCT conformément aux besoins des concepteurs. Par la suite, nous montrons pour-

quoi (paragraphe 5.1.3.1, 5.1.3.2 et 5.1.3.3) et comment (paragraphe 5.1.3.4 et 5.1.3.5) nous générons de telles propriétés.

Les `propriétés de validation` que nous obtenons compte tenu des spécifications des concepteurs sont { TP^{r}_{SBC}, TP^{r}_{EDT}, TP^{r}_{EDT}, $TP^{cp_1}_{RV}$, $TP^{cp_1}_{RH}$, $TP^{an_1}_{RV}$, $TP^{an_1}_{RH}$, $TP^{at_1}_{EDD}$, $TP^{at_1}_{EDT}$ } :

- TP^{r}_{SBC}, TP^{r}_{PL} et TP^{r}_{EDT} postulent respectivement que SBC, PL et EDT doivent être rejouables.
- $TP^{cp_1}_{RV}$ postule que si RH peut éventuellement échouer (n'est pas rejouable) alors RV doit être compensable et il faut spécifier que RV doit être compensé (s'il s'est déjà terminé) quand RH échoue.
- $TP^{cp_1}_{RH}$: postule que si RV peut éventuellement échouer (n'est pas rejouable) alors RH doit être compensable et il faut spécifier que RH doit être compensé (s'il s'est déjà terminé) quand RV échoue.
- $TP^{an_1}_{RV}$ postule que si RH peut éventuellement échouer (n'est pas rejouable) alors il faut spécifier que RV doit être annulé (s'il est en cours d'exécution) quand RH échoue.
- $TP^{an_1}_{RH}$ postule que si RV peut éventuellement échouer (n'est pas rejouable) alors il faut spécifier que RH doit être annulé (s'il est en cours d'exécution) quand RV échoue.
- $TP^{at_1}_{EDD}$ postule que si EDF peut éventuellement échouer (n'est pas rejouable) alors il faut spécifier que EDD doit être activé (en tant qu'alternative) quand EDF échoue.
- $TP^{at_1}_{EDT}$ postule que que si EDD peut éventuellement échouer (n'est pas rejouable) alors il faut spécifier que EDT doit être activé (en tant qu'alternative) quand EDD échoue.

Exemple Le service cs_1 (illustré par la figure 5.7.a), qui est valide, respecte toutes ces propriétés en particulier $TP^{cp_1}_{RH}$ (puisque RV est rejouable son échec n'est plus éventuel et par la suite il n'est plus nécessaire que RH soit compensable). Par contre le service cs_2 (illustré par la figure 5.7.b), qui n'est pas valide, ne respecte cette propriété $TP^{cp_1}_{RH}$. Ceci signifie que le service cs_2 peut se terminer dans l'état $etCS2_7$ qui n'est pas un état acceptable.

Dans la suite, nous montrons pourquoi et comment nous générons ces `propriétés de validation`. Pour cela, nous avons besoin d'introduire la notion d'`inclusion forte` d'un flot transactionnel dans un autre et la notion de `flot transactionnel induit` par l'ETA.

5.1.3.1 Inclusion forte d'un flot transactionnel dans un autre

Un flot transactionnel d'un SCT spécifie les échecs acceptés et les mécanismes de recouvrement. Par exemple, le flot transactionnel du service cs_{induit} décrit à la figure 5.8 accepte les échecs suivants et spécifie les mécanismes de recouvrement comme suit :

E1 : RH peut échouer. Dans ce cas, il faut annuler ou compenser RV et abandonner l'exécution.

E2 : RV peut échouer. Dans ce cas, il faut annuler ou compenser RH et abandonner l'exécution.

E3 : EDF peut échouer. Dans ce cas, il faut activer EDD comme une alternative.

E4 : EDD peut échouer. Dans ce cas, il faut activer EDT comme une alternative.

Étant donné deux flots transactionnels $ft1$ et $ft2$ définis selon le même flot de contrôle, nous disons que $ft2$ est *inclus fortement* dans $ft1$ (ft2 $\sqsubseteq$ $ft1$) *si et seulement si* $ft2$ n'accepte que les échecs acceptés par $ft1$ et pour chaque échec accepté, il le gère exactement comme il est géré dans $ft1$. $ft2$ peut ne pas accepter des échecs acceptés par $ft1$.

DÉFINITION 5.5 (INCLUSION FORTE D'UN FLOT TRANSACTIONNEL DANS UN AUTRE)
Soit fc un flot de contrôle, $ft1$ et $ft2$ deux flots transactionnels définis selon fc. $ft2$ est inclus fortement dans $ft1$ (ft2 $\sqsubseteq$ $ft1$) *si et seulement si* les trois conditions suivantes sont vérifiées (s est un service composant de fc. $\Diamond_{ft2}$ F signifie que F est éventuellement vraie dans $ft2$.)

C1 : $\forall$ s rejouable dans $ft1$, s est rejouable dans $ft2$

C2 : $\forall$ $\Diamond_{ft2}$ $CondCpsft1_i(s)$

(a) s est compensable dans $ft2$ et

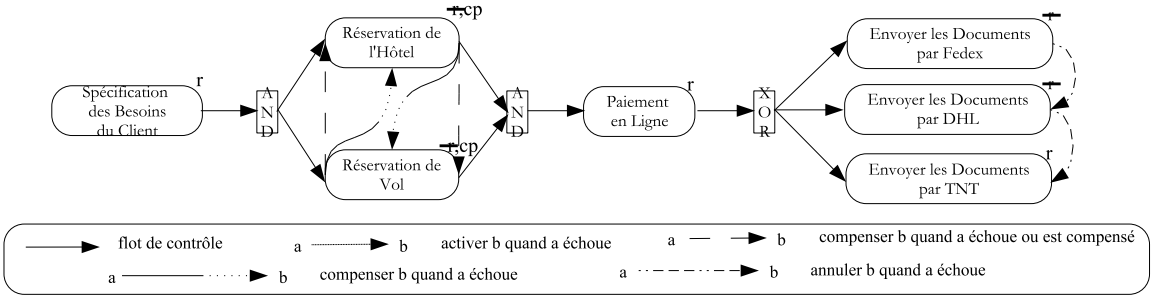
(b) $CondCpsft1_i(s) \in CondCpsft2(s)$

C3 : $\forall \Diamond_{ft2} CondAnlft1_i(s), CondAnlft1_i(s) \in CondAnlft2(s)$

C4 : $\forall \Diamond_{ft2} CondAltft1_i(s), CondAltft1_i(s) \in CondAltft2(s)$

avec $CondCpsfti(s)$, $CondAnlfti(s)$ et $CondAltfti(s)$ désigne respectivement la condition de compensation, d'annulation et d'alternative de s dans fti (i est égal à 1 ou 2).

Figure 5.8 Flot transactionnel induit par l'ETA.



La condition C1 assure que $ft2$ n'accepte que les échecs acceptées par $ft1$. Les conditions C2, C3 et C4 assurent que les échecs acceptés par $ft2$ seront gérés exactement comme ils le sont dans $ft1$. La condition C2 assure que toute condition de compensation de s dans $ft1$ éventuellement vraie dans $ft2$ est une condition de compensation de s dans $ft2$ (évidemment s sera par conséquence compensable dans $ft2$). La condition C3 assure que toute condition d'annulation de s dans $ft1$ éventuellement vraie dans $ft2$ est une condition d'annulation de s dans $ft2$. La condition C4 assure que toute condition d'alternative de s dans $ft1$ éventuellement vraie dans $ft2$ est une condition d'alternative de s dans $ft2$. L'éventualité d'une condition, spécifiée dans $ft1$, dans $ft2$ dépend des échecs acceptés par $ft2$.

Exemple Le flot transactionnel du service cs_1 (figure 5.7.a) est `inclus fortement` dans le flot transactionnel du service cs_{induit} (figure 5.8). En effet, le flot transactionnel du service cs_1 accepte que les échecs, E1, E3, et E4, acceptés par le flot transactionnel du service cs_{induit}. En plus il les gère comme ils sont gérés dans cs_{induit}. Notez bien que la condition de compensation de RH spécifié dans cs_{induit}, $RV.échoué$, n'est pas possible dans cs_1 puisque RV est rejouable (cs_1 ne tolère pas l'échec E2).

Par comparaison à l'inclusion simple (que nous avons introduit au chapitre précédent à la définition 4.29), l'`inclusion forte` est plus restrictive dans le sens ou elle exige que $ft2$ accepte que les échecs acceptés par $ft1$ et les gère comme $ft1$. Alors que l'inclusions simple exige seulement que l'ensemble des dépendances transactionnelles spécifiées par $ft2$ soient incluses dans celles du

flot transactionnel $ft1$. Ceci signifie, entre autres, que l'inclusion simple ($ft2 \subseteq ft1$) accepte que $ft2$ gère un échec partiellement (voir pas du tout) par rapport à ce qui est spécifié dans $ft1$. Par exemple, le flot transactionnel du service cs_2 (figure 5.7.b) est *inclus simplement* dans celui de cs_{induit} (figure 5.8) (puisque toutes les dépendances qu'il spécifient sont spécifiées dans cs_{induit}). Cependant, il n'est pas inclus fortement parce que il ne gère pas l'échec de RV comme il est spécifié dans cs_{induit}.

LEMME 5.2
Soit fc un flot de contrôle, $ft2$ et $ft1$ deux flots transactionnels définis selon fc. $ft2$ est inclus fortement dans $ft1 \iff calculET_{avecEchecs}(getFTransactionnel(ft2)) \subseteq calculET_{avecEchecs}(getFTransactionnel(ft1))$.

Ce lemme postule que $ft2$ est inclus fortement dans $ft1$ *si et seulement si* l'ensemble des `états de terminaison` avec échecs de $ft2$ est inclus dans celui de $ft1$.

En effet, soit $ft1$ et $ft2$ deux flots transactionnels tq $ft2 \sqsubseteq ft1$. La seule différence entre $ft2$ et $ft1$ est que $ft2$ peut ne pas accepter des échecs acceptés par $ft1$. Intuitivement, ceci signifie que $ft2$ est plus fiable que $ft1$. En terme d'états de terminaison, ceci se traduit par le fait que les états de terminaison de $ft1$ résultats des échecs acceptés par lui et non acceptés par $ft2$ ne seront plus dés états de terminaison de $ft2$. En plus les états de terminaison de $ft1$ suite à des échecs acceptés par $ft2$ sont des états de terminaison de $ft2$ puisque $ft2$ les gère exactement comme ils le sont dans $ft1$. Par la suite, l'ensemble des états de terminaison de $ft2$ est inclus dans celeui de $ft1$. Une démonstration plus formelle de ce lemme est donnée en annexe D.

Exemple Le flot transactionnel du service cs_1, $ft1$, (voir figure 5.7.a) est *inclus fortement* dans celui du service cs_{induit}, ft_{induit} (voir figure 5.8). L'ensemble des états de terminaison de $ft1$ (voir figure 5.4.a) est inclus dans celui de ft_{induit} (l'ensemble $\{eta_4, eta_5, eta_6, eta_7, eta_8, eta_9\}$ de la figure 5.6). Remarquez que les états de terminaison de ft_{induit} eta_6 et eta_7 ne sont pas des états de terminaison de $ft1$. En effet, ces deux états sont atteints dans ft_{induit} suite à l'échec de RV. Cependant RV est rejouable dans $ft1$. De même est pour l'état eta_9 qui est atteint dans ft_{induit} suite à l'échec de EDD qui n'est plus possible dans $ft1$ puisque EDD est rejouable.

5.1.3.2 Flot transactionnel induit par un ETA

Un `état de terminaison avec échecs` est atteint en cas d'échec de certains services. Ceci sous-entend qu'un tel état garde la trace des échecs reproduits et les mécanismes de recouvrement qui ont été appliqués. Prenons par exemple l'état de terminaison eta_6 (figure 5.6). Cet état est atteint suite à l'échec de RV. En plus, les mécanismes de recouvrement appliqués consistaient à compenser RH et à abandonner l'exécution globale.

Par conséquent, étant donné l'ensemble des `états de terminaison avec échecs` d'un service composé, nous pouvons déduire son flot transactionnel. Soit $ET_{avecEchecs}$ l'ensemble des `états de terminaison avec échecs` d'un service composé sc (dont on connaît son flot de contrôle), nous appelons `flot transactionnel induit` par $ET_{avecEchecs}$ le flot transactionnel (défini selon le flot de contrôle de sc) qui a pour ensemble d'états de terminaison $ET_{avecEchecs}$.

La fonction $calculET^{-1}_{avecEchecs}$ définit pour chaque service composant s :

- ses propriétés transactionnelles induites par $ET_{avecEchecs}$,
- $CondCpsET_{avecEchecs}(s)$: sa condition de compensation induite par $ET_{avecEchecs}$ qui spécifie quand s sera compensé selon $ET_{avecEchecs}$,
- $CondAnlET_{avecEchecs}(s)$: sa condition d'annulation induite par $ET_{avecEchecs}$ qui spécifie quand s sera annulé selon $ET_{avecEchecs}$,

DÉFINITION 5.6 (FLOT TRANSACTIONNEL INDUIT)
La fonction inverse de $calculET_{avecEchecs}$,
$calculET^{-1}_{avecEchecs} : \mathcal{P}(\mathcal{ET}_{avecEchecs}) \longrightarrow \mathcal{FT}ransactionnel$ définit le flot transactionnel (selon un flot de contrôle donné) induit par un ensemble d'états de terminaison avec échecs $ET_{avecEchecs}$, $calculET^{-1}_{avecEchecs}(ET_{avecEchecs})$.

– $CondAltET_{avecEchecs}(s)$: sa condition d'alternative induite par $ET_{avecEchecs}$ qui spécifie quand s sera activé en tant qu'alternative selon $ET_{avecEchecs}$.

Nous distinguons dans cette approche, en particulier, le flot transactionnel induit par l'`ensemble des états de terminaison acceptés avec échecs` $ETA_{avecEchecs}$: $calculET^{-1}_{avecEchecs}(ETA_{avecEchecs})$ (que nous appelons désormais flot transactionnel induit par l'ETA pour des raisons de lisibilité). Le flot transactionnel du service cs_{induit} (figure 5.8) est le flot transactionnel induit par l'ETA spécifié à la figure 5.6.

5.1.3.3 Méthodologie pour le calcul des `propriétés de validation`

Le flot transactionnel induit par ETA, $calculET^{-1}_{avecEchecs}(ETA)$, spécifie les échecs acceptés et leur gestion conformément aux besoins des concepteurs.

LEMME 5.3
sc est valide $\Longleftrightarrow getFTransactionnel(sc) \sqsubseteq calculET^{-1}_{avecEchecs}(ETA_{avecEchecs})$.

Ce lemme postule qu'un SCT sc est valide *si et seulement si* son flot transactionnel est inclus fortement dans le flot transactionnel induit par l'ETA spécifié. La démonstration de ce lemme est donnée à l'annexe D.

Ainsi pour garantir qu'un service est valide, il suffit d'assurer que son flot transactionnel est *inclus fortement* dans le flot transactionnel induit par ETA. Et c'est justement ce que les `propriétés de validation` permettent d'assurer. Tout flot transactionnel qui respecte ces propriétés est *inclus fortement* dans le flot transactionnel induit par ETA et par la suite le service correspondant est valide.

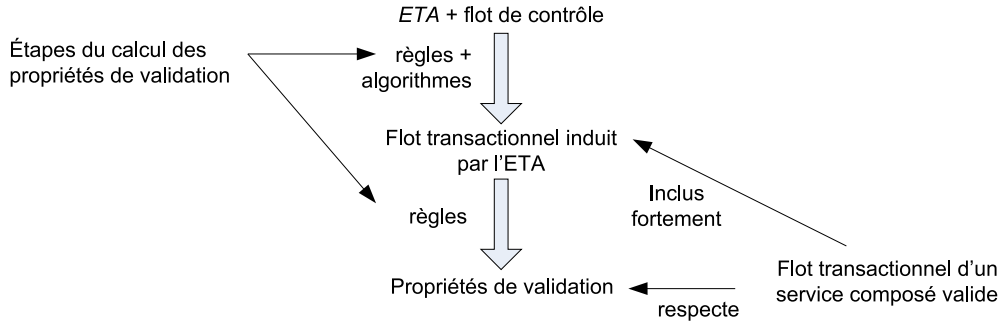
Pour calculer ces `propriétés de validation`, nous procédons en deux étapes (voir figure 5.9). La première consiste à calculer le flot transactionnel induit par l'ETA à partir du flot de contrôle et de l'ETA spécifié par les concepteurs. La deuxième étape consiste à générer les `propriétés de validation` à partir de ce flot transactionnel induit.

Dans la suite, nous détaillons chacune de ces deux étapes. Nous présentons les règles et l'algorithme que nous utilisons pour déduire le flot transactionnel induit par l'ETA. Ensuite, nous illustrons les règles que nous utilisons pour générer les propriétés transactionnels de validation à partir du flot transactionnel induit par l'ETA.

5.1.3.4 Calcul du flot transactionnel induit par un ETA

Dans ce paragraphe, nous montrons comment nous procédons pour calculer le `flot transactionnel induit` par ETA étant donné ETA et le flot de contrôle correspondant. Définir le flot transactionnel induit par l'ETA revient à définir pour chaque service composant s, ses propriétés transactionnelles induites, ses conditions de compensation, d'annulation et d'alternative induites par l'ETA : $CondCpsETA(s)$, $CondAnlETA(s)$ et $CondAltETA(s)$.

Figure 5.9 Méthodologie pour le calcul des propriétés de validation.



Calcul des propriétés transactionnelles induites par un ETA

Étant donné un ETA, nous pouvons déduire l'ensemble des états de terminaison acceptés de chaque service composant s que nous notons ETA(s). Par exemple l'ETA spécifié dans la figure 5.6 définit, parmi d'autres, $ETA(EDT) = \{initial, terminé, abandonné\}$ et $ETA(RH) = \{terminé, échoué, compensé, annulé\}$. Nous utilisons les règles ci-dessous pour définir les propriétés transactionnelles des services composants induites par ETA.

RÈGLE 5.1
$\forall$ service composant s,

1. si $s.échoué \notin ETA(s)$ alors s est rejouable.
2. si $s.échoué \in ETA(s)$ alors s n'est pas rejouable.
3. si $s.compensé \in ETA(s)$ alors s est compensable.
4. si $s.compensé \notin ETA(s)$ alors s n'est pas compensable.

Les règles 1 et 2 permettent respectivement de spécifier si un service s est rejouable ou non. Les règles 3 et 4 permettent de spécifier respectivement si un service s est compensable ou non. Pour notre exemple, l'application de la première règle permet de déduire que les services SBC, PL et EDT sont rejouables. L'application de la deuxième règle permet de déduire que les services RV, RH, EDF et EDD ne sont pas rejouables. L'application de la troisième règle permet de déduire que les services RV et RH sont compensables. L'application de la quatrième règle permet de déduire que les services SBC, PL, EDF, EDD et EDT ne sont pas compendables. Les propriétés transactionnelles des services composants induites par ETA sont illustrées par la figure 5.8.

Calcul des conditions transactionnelles induites par un ETA

Dans ce paragraphe, nous montrons comment nous procédons pour calculer la condition de compensation d'un service s induite par l'ETA. Nous procédons similairement pour calculer les conditions d'annulation et d'alternative unduites par ETA d'un service s (voir annexe C).

L'algorithme 1 permet d'extraire pour chaque service s sa condition de compensation $CondCpsETA(s)$ induite par l'ETA. Le principe est le suivant : une condition de compensation potentielle d'un service s devient une condition de compensation induite par l'ETA si elle est vérifiée dans un état de terminaison accepté dans lequel l'état du service s est compensé. Ainsi l'algorithme va parcourir ETA état par état (ligne 4 à la ligne 14). Pour chaque état de

terminaison accepté dans lequel l'état du service s est compensé (ligne 5), l'algorithme va chercher la condition de compensation potentielle de s qui est vérifiée dans cet état (ligne 6 à la ligne 13). Les lignes 7 et 13 permettent de parcourir les conditions de compensation potentielles de S. La variable booléenne «satisfait» (lignes 6 et 11) permet de marquer si la condition potentielle de compensation de s courante est vérifiée ou non dans l'état de terminaison courant (variable «eta»). Une condition de compensation potentielle qui se trouve vérifiée dans «eta» sera considérée comme condition de compensation induite par ETA de s (ligne 10). Cette condition sera retirée de la liste des conditions potentielle de s pour ne pas être re vérifiée (ligne 12) dans les autres états de terminaison.

Nous procédons de façon similaire pour extraire les conditions d'annulation, $CondAnlETA(s)$, et d'alternatives, $CondAltETA(s)$, induites par ETA des services composants.

Entrées : ETA : l'ETA exigé par les concepteurs
$CondPtCps(s)$: La condition de compensation potentielle de s
Sorties : $CondCpsETA(s)$: La condition de compensation de s induite par l'ETA
Données : *eta* :l'état de terminaison accepté courant (dans ETA)
$CondPtCps_i(s)$: la condition de compensation potentielle courante dans $CondPtCps(s)$
$satsif$: une variable booléenne mis à vrai si $CondPtCps_i(s)$ est satisfaite dans *eta*

```
1  début
2      CondCpsETA(s) ⟵ ∅
3      eta ⟵ eta suivant dans ETA
4      tant que eta ≠ null faire
5          si l'état de s dans eta est compensé alors
6              satisf ⟵ faux
7              CondPtCps_i(s) ⟵ CondPtCps_i(s) suivant dans CondPtCps(s)
8              tant que non satisfé et CondPtCps_i(s) ≠ null faire
9                  si CondPtCps_i(s) est satisfait dans eta alors
10                     CondCpsETA(s) ⟵ CondCpsETA(s) ⊕ CondPtCps_i(s)
11                     satisf ⟵ vrai
12                     CondPtCps(s) ⟵ CondPtCps(s) - CondPtCps_i(s)
13                 CondPtCps_i(s) ⟵ CondPtCps_i(s) suivant dans CondPtCps(s)
14         eta ⟵ eta suivant dans l'ETA
15 fin
```

Algorithme 1 : Extraction de la condition de compensation d'un service s induite par ETA

Exemple Dans notre exemple, la condition potentielle de compensation de RV, *RH.échoué*, devient une condition de compensation induite par l'ETA puisqu'il est satisfaite dans eta_4 (dans lequel l'état de RV est compensé). Puisque eta_4 est le seul état dans lequel RV est compensé nous pouvons déduire alors que la condition de compensation de RV induite par ETA est l'échec de RH : $CondCpsETA(RV) = RH.échoué$. De même nous pouvons déduire la condition d'annulation de RV induite par l'ETA. eta_5 est le seul état dans lequel RV est annulé. En plus, la condition potentielle d'annulation de RV, *RH.échoué*, est satisfaite dans eta_5. Nous déduisons alors que $CondAnlETA(RV) = RH.échoué$. Réciproquement, nous pouvons déduire à partir de eta_6 (respectivement eta_7) la condition de compensation de RH induite par ETA :

$CondCpsETA(RH) = RV.échoué$ (respectivement la condition d'annulation de RH induite par ETA : $CondAnlETA(RH) = RV.échoué$). Enfin, nous pouvons déduire de la même manière à partir de eta_8 et $CondPtAlt(EDD)$ (respectivement à partir de eta_9 et $CondPtAlt(EDT)$) que $CondAltETA(EDD) = EDF.échoué$ (respectivement $CondAltETA(EDT) = EDD.échoué$).

5.1.3.5 Calcul des `propriétés de validation`

Une fois que nous avons calculé le flot transactionnel induit par l'ETA, nous utilisons un enesmble de règles pour générer les propriétés de validition. Tout flot transactionnel inclus fortement dans le flot transactionnel induit par l'ETA doit respecter ces propriétés de validition.

En fait, ces règles assure via les propriétés générées le respect des conditions C1, C2, C3 et C4 de la définition 5.5 pour que le flot transactionnel en question, ft, soit inclus fortement dans le flot transactionnel induit par ETA.

RÈGLE 5.2
$\forall$ service composé s ($\Diamond_{ft}$F désigne que F est éventuellement vrai dans ft) :

1. s est rejouable dans le flot transactionnel induit par l'$ETA \Longrightarrow$ *générer la propriété transactionnelle* TP_s^r : *s doit être rejouable.*
2. $\forall CondCpsETA_i(s) \in CondCpsETA(s)$, *générer la propriété transactionnelle* $TP_s^{cp_i} : (\Diamond_{ft}(CondCpsETA_i(s))) \Longrightarrow$
   (a) s doit être compensable et
   (b) $CondCpsETA_i(s) \in CondCps(s)$
3. $\forall CondAnlETA_i(s) \in CondAnlETA(s)$, *générer la propriété transactionnelle* $TP_s^{an_i} : (\Diamond_{ft}(CondAnlETA_i(s))) \Longrightarrow CondAnlETA_i(s) \in CondAnl(s)$
4. $\forall CondAltETA_i(s) \in CondAltETA(s)$, *générer la propriété transactionnelle* $TP_s^{at_i}$:
   $(\Diamond_{ft}(CondAltETA_i(s))) \Longrightarrow CondAltETA_i(s) \in CondAlt(s)$

La propriété générée par la règle 1 assure le respect de la condition C1 pour le service s. Cette règle postule que si s est rejouable dans le flot transactionnel induit par l'ETA alors il doit aussi l'être dans ft. Cette règle assure que seuls les échecs acceptés par le flot transactionnel induit par ETA sont acceptés.

Les propriétés générées par la règle 2 assurent le respect de la condition C2 pour chaque service s. Cette règle postule que chaque condition de compensation $CondCpsETA_i(s)$ d'un service s induite par l'ETA éventuellement vraie dans ft doit être une condition de compensation de s (s sera par conséquence compensable).

Les propriétés générées par la règle 3 assurent le respect de la condition C3 pour chaque service s. Cette règle postule que chaque condition d'annulation $CondAnlETA_i(s)$ d'un service s induite par l'ETA éventuellement vraie dans ft doit être une condition d'annulation de s.

Les propriétés générées par la règle 4 assurent le respect de la condition C4 pour chaque service s. Cette règle postule que chaque condition d'alternative $CondAltETA_i(s)$ d'un service s induite par l'ETA éventuellement vraie dans ft doit être une condition d'alternative de s.

Ces trois dernières règles (2, 3 et 4) assurent que les mécanismes de gestion des échecs acceptés se déroulent conformément à ce qui est spécifié dans le flot transactionnel induit par l'ETA.

Exemple L'application de ces règles à notre exemple permet de générer les `propriétés de validation` présentées précédemment { $TP^r_{SBC}, TP^r_{EDT}, TP^{cp_1}_{RV}, TP^{cp_1}_{RH}, TP^{an_1}_{RV}, TP^{an_1}_{RH}, TP^{at_1}_{EDD}$,

$TP^{at_1}_{EDT}$ } :

- En appliquant la règle 1, et puisque les services SBC, PL et EDT sont rejouables dans le flot transactionnel induit par l'ETA alors nous obtenons les `propriétés de validation` TP^{r}_{SBC}, TP^{r}_{PL} et TP^{r}_{EDT}.
- En appliquant la règle 2, et puisque $CondCpsETA(RV) = RH.échoué$ (respectivement $CondCpsETA(RH) = RV.échoué$) nous obtenons la `propriété de validation` $TP^{cp_1}_{RV}$ (respectivement $TP^{cp_1}_{RH}$).
- En appliquant la règle 3, et puisque $CondAnlETA(RV) = RH.échoué$ (respectivement $CondAnlETA(RH) = RV.échoué$) nous obtenons la `propriété de validation` $TP^{an_1}_{RV}$ (respectivement $TP^{an_1}_{RH}$).
- En appliquant la règle 4, et puisque $CondAltETA(EDD) = EDF.échoué$ (respectivement $CondAnlETA(EDT) = EDD.échoué$) nous obtenons la `propriété de validation` $TP^{at_1}_{EDD}$ (respectivement $TP^{at_1}_{EDT}$).

Le flot transactionnel du service cs_1 (figure 5.7.a) est inclus fortement dans le flot transactionnel induit par ETA (figure 5.8) et par la suite cs_1 est valide. cs_1 respecte toutes ces propriétés en particulier $TP^{cp_1}_{RH}$. En effet, puisque RV est rejouable son échec n'est plus à considérer. Par contre, cette propriété n'est pas respectée dans le service cs_2 (figure 5.7.b). En effet, l'échec de RV est possible dans cs_2 puisque il n'est pas rejouable. Or il n'est pas spécifié que le service RH sera compensé quand RV échoue. Ce qui signifie que le service cs_2 peut se terminer dans l'état $etCS2_7$ qui n'est pas acceptable. Par la suite le service cs_2 n'est pas valide.

5.1.4 Synthèse

Dans cette section, nous avons présenté notre approche par validation du modèle de composition pour assurer des compositions fiables de services Web.

Selon, cette approche les concepteurs précisent leurs besoins en terme de structure de contrôle et de semi atomicité. Ensuite, nous utilisons un ensemble de techniques pour générer des propriétés de validition. Ces propriétés caractérisent les flots transactionnels des SCT valides. Un SCT est valide conformément aux spécifications des concepteurs *si et seulement si* il respecte ces propriétés de validité.

L'originalité de notre approche est la flexibilité que nous offrons aux concepteurs pour spécifier leurs besoins en terme de structure de contrôle et de correction. Contrairement aux MTA, nous partons des spécifications des concepteurs pour déterminer les mécanismes transactionnels permettant d'assurer des compositions fiables selon leurs besoins. Notre approche permet alors de prendre en compte des contextes spécifiques (où la correction est définie conformément aux besoins des concepteurs et non conformément à des règles implicites de bonne conduite comme les MTA).

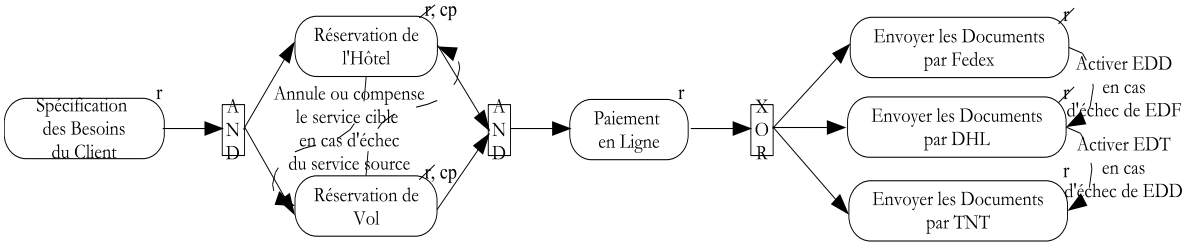
Il est important de souligner que cette approche peut être appliquée d'une façon locale pour garantir certaines propriétés à des parties du service global.

5.2 Approche par ré-ingénierie du SCT

Dans cette approche, nous procédons par ré-ingénierie du modèle de composition pour assurer des exécutions correctes. Nous corrigeons et améliorons les mécanismes de recouvrement après analyse des traces d'exécution d'un nombre suffisant grand d'instances. Cette approche est motivée par le fait que garantir une composition valide n'est pas suffisant, dans certains cas, pour assurer des exécutions correctes. Ceci est du au fait que le flot transactionnel spécifié au niveau du modèle de composition repose sur des hypothèses pouvant ne pas coïncider avec la réalité.

Supposons, par exemple, que le service d'organisation de voyage en ligne défini par les concepteurs est celui illustré par la figure 5.10. Selon ce modèle, le service PL est rejouable contrairement aux services RV et EDD. Supposons maintenant qu'en réalité (après observation d'un nombre suffisant grand d'exécutions) les services RV et EDD n'échouent jamais et le service PL peut échouer. Ceci sous-entend (i) qu'il n'y a pas besoin que le service RH supporte des mécanismes de compensation (ce qui peut être coûteux) et (ii) que le paiement peut échouer alors que les réservations de vol et d'hôtel sont toujours maintenues.

Figure 5.10 Service d'organisation de voyage en ligne initialement défini.

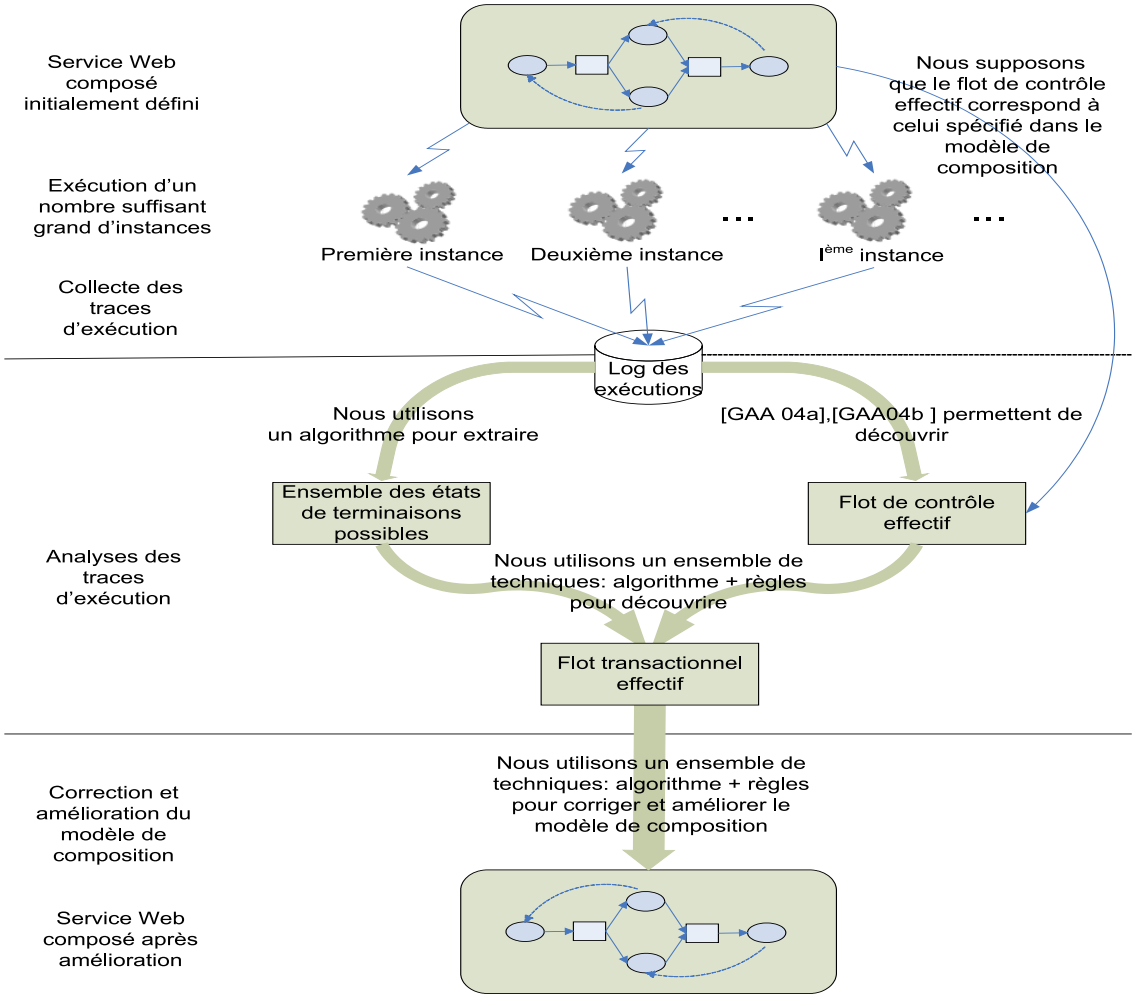


Cette approche, par ré-ingénierie du SCT, permet de détecter de telles anomalies et de corriger les erreurs induites. Pour ce faire, nous procédons en deux étapes principales. La première consiste à découvrir le flot transactionnel effectif du SCT à partir de ses traces d'exécution. Ensuite nous utilisons un ensemble de règles pour améliorer et corriger le modèle de composition. La figure 5.11 présente une vue globale de cette approche.

- **Collecte des traces d'exécution du SCT :** Cette phase consiste à garder les traces d'exécution du service composé. Nous supposons que le nombre d'exécutions est suffisant grand pour prendre en compte tous les cas possibles. La trace d'une exécution capte les différents événements générés. Nous stockons les traces d'exécutions dans un fichier log.
- **Découverte du flot transactionnel effectif :** Le but de cette phase est de découvrir le flot transactionnel effectif du SCT. Pour ce faire, nous avons besoin de l'ensemble des états de terminaison possibles et du flot du contrôle effectif du SCT. Il est possible, à partir du log des exécutions, d'extraire l'ensemble des états de terminaison possibles et de découvrir le flot de contrôle effectif [GBG04b; GBG04a]. Nous supposons que le flot de contrôle spécifié au niveau du modèle de composition correspond au flot de contrôle effectif. A partir de l'ensemble des états de terminaison possibles et du flot de contrôle, nous utilisons un ensemble de techniques pour découvrir le flot transactionnel du SCT.
- **Correction et amélioration du SCT :** A partir des informations découvertes lors de l'étape précédente, nous utilisons un ensemble de règles pour corriger et améliorer le flot transactionnel du SCT.

Dans la suite, nous détaillons chacune de ces étapes. Dans le paragraphe 5.2.1, nous détaillons comment nous trouvons l'ensemble des états de terminaison d'un SCT à partir du fichier log. Nous montrons ensuite dans le paragraphe 5.2.2 comment nous procédons pour découvrir le flot transactionnel d'un SCT à partir de sont flot de contrôle et de l'ensemble de ses états de terminaison. Enfin, dans le paragraphe 5.2.3 nous présentons les règles que nous utilisons pour améliorer le flot transactionnel découvert.

Figure 5.11 Une vue globale de notre approche par re-ingénierie du modèle de composition.



5.2.1 Extraction de l'ensemble des états de terminaison

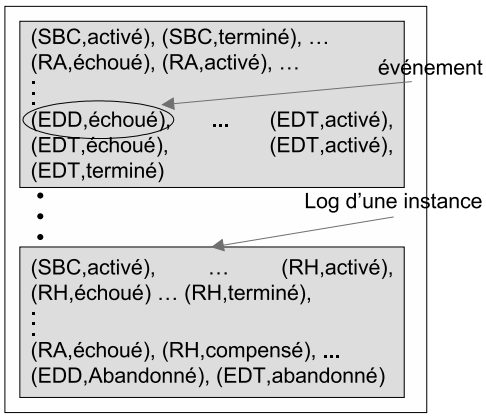
Avant de présenter comment nous procédons pour extraire l'ensemble des états de terminaison possibles d'un SCT à partir de son log d'exécutions. Nous avons besoin de présenter tout d'abord la structure de données que nous utilisons pour garder les traces d'exécutions d'un SCT donné : le `log d'exécutions`.

Plusieurs exécutions peuvent être instanciées selon le même modèle de composition. Le `log d'exécutions` d'un service composé rassemble les traces des exécutions de ses instances. La trace d'une exécution capte les différents événements générés. Un événement marque le changement d'état d'un service composant. Nous définisson un événement généré au cours de l'exécution d'un SCT sc par le couple $ev = (s \in services(sc), e \in états(s))$ où s est un service composant de sc et e un état de s. Nous notons $\mathcal{E}vénement$ l'ensemble de tous les événements générés. Nous notons $\mathcal{E}vnement_{sc}$ l'ensemble de tous les événements générés par le SCT sc.

- Nous définissons la fonction $service : \mathcal{E}vénement \longrightarrow \mathcal{SWT}$ qui permet de retourner le service qui a généré un événement donné.
- Nous définissons la fonction $état : \mathcal{E}vénement \longrightarrow \mathcal{E}tats$ qui permet de retourner l'état correspondant à un événement donné.

Le log d'une instance, $inst_i$, d'un SCT sc est l'ensemble, $log_{inst_i} = \{e \in \mathcal{E}vnement_{sc}\}$ captant tous les événements générés lors de l'exécution de $inst_i$. Enfin, le fichier log d'un SCT regroupe toutes les traces d'exécutions de ses instances. Le log d'un SCT, sc est l'ensemble $log_{sc} = \{log_{inst_i} \mid inst_i$ est une instance de sc $\}$ (voir figure 5.12).

Figure 5.12 Une vue schématique d'un exemple de log d'exécutions du service d'organisation de voyage en ligne.



Nous utilisons l'algorithme 2 pour extraire l'ensemble des états de terminaison d'un SCT à partir de son log d'exécutions. Cet algorithme extrait l'état de terminaison de chaque instance (ligne 3 à la ligne 11). Nous considérons les états de terminaison non redondants (ligne 12 et 13). Pour extraire l'état de terminaison d'une instance, l'algorithme parcours son log de la fin au début (ligne 5, 6 et 11). A chaque événement ev rencontré et si il reste encore des services composant dont l'état de terminaison est non connu (ligne 7), il associe au service correspondant ($service(ev)$) l'état correspondant ($etat(ev)$) comme état de terminaison (ligne 9) et retire ce service de la liste de service dont l'état de terminaison n'est pas encore déterminé (ligne 10).

Exemple Nous supposons que l'application de cet algorithme nous permet d'extraire l'ensemble des états de terminaison illustré par la figure 5.13.

Figure 5.13 Ensemble d'états de terminaison extrait à partir du log d'exécutions.

services / et	SBC	RV	RH	PL	EDF	EDD	EDT
et_1	(terminé,	terminé,	terminé,	terminé,	terminé,	initial,	initial)
et_2	(terminé,	terminé,	terminé,	terminé,	initial,	terminé,	initial)
et_3	(terminé,	terminé,	terminé,	terminé,	initial,	initial,	terminé)
et_4	(terminé,	compensé,	échoué,	abandonné,	abandonné,	abandonné,	abandonné)
et_5	(terminé,	annulé,	échoué,	inhibé,	inhibé,	abandonné,	abandonné)
et_6	(terminé,	terminé,	terminé,	terminé,	échoué,	terminé,	initial)
et_7	(terminé,	terminé,	terminé,	terminé,	initial,	initial,	échoué)

```
Entrées : services[] : liste des services composants
log : le fichier log du SCT
Sorties : eet[] : ensemble d'états de terminaison et[]
Données : listeServices : la liste de services dont l'état de terminaison n'est pas
          encore déterminé
ev : événement courant lu dans le log de l'instance courante
et[] : l'état de terminaison de l'instance courante
1  début
2      pour chaque log d'une instance faire
3          eet[] ⟵ ∅
4          listeServices ⟵ services[]
5          positionner la lecture de l'instance courante de la fin au début
6          ev ⟵ événement suivant dans le log de l'instance courante
7          tant que listeServices ≠ null et ev ≠ null faire
8              si service(ev) ∈ listeServices alors
9                  et[service(ev)] ⟵ état(ev)
10                 listeServices ⟵ listeservices - service(ev)
11             ev ⟵ événement suivant dans le log de l'instance courante
12         si et[] ∉ eet[] alors
13             ajouter et[] à eet[]
14     retourner eet[]
15 fin
```

Algorithme 2 : Extraction de l'ensemble des états de terminaison à partir du fichier log.

5.2.2 Découverte du flot transactionnel d'un SCT

Le flot transactionnel effectif du SCT est en fait le flot transactionnel induit par l'ensembles des états de terminaison découvert. Ainsi il suffit d'appliquer les règles 5.1 et l'algorithme 1 à l'ensemble des états de terminaison découverts pour déduire le flot transactionnel effectif du SCT.

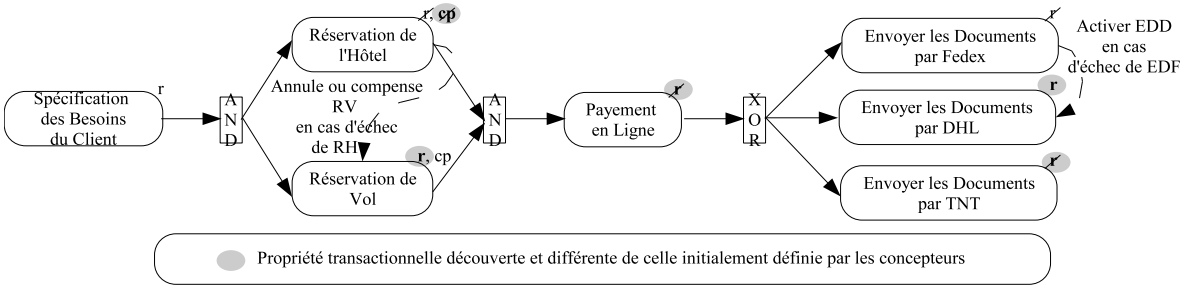
Exemple Dans notre exemple, l'ensemble des états de terminaison découvert est illustré par la figure 5.13. L'application des règles 5.1 nous permettent de découvrir que les services SBC, RV et EDD sont rejouables, que les services RH, EDF, EDT et PL ne sont pas rejouables, que le service RV est compensable et que les autres services ne le sont pas. L'algorithme 1 nous permet de découvrir les dépendances transactionnelles suivantes : RV sera annulé ou compensé suite à l'échec de RH ($CondCps(RV) = RH.échoué$ et $CondAnl(RV) = RH.échoué$) et EDD est une alternative pour EDF ($CondAlt(EDD) = EDF.échoué$). La figure 5.14 illustre le flot transactionnel découvert.

5.2.3 Amélioration du flot transactionnel d'un SCT

Cette approche repose sur le concept de `flot transactionnel intuitivement valide` pour améliorer le flot transactionnel d'un SCT. Un `flot transactionnel intuitivement valide` est, comme son nom l'indique, un flot transactionnel qui respecte certaines conditions intuitives de bon sens. Ces conditions peuvent être résumées dans les règles de bonne conduite suivantes :

R1 : lors de l'échec d'un service il faut toujours essayer d'exécuter une alternative si elle existe,

Figure 5.14 Le flot transactionnel effectif découvert.



R2 : lors de l'échec d'un service causant l'abandon du service global, il faut compenser le travail partiel déjà effectué,

R3 : lors de l'échec d'un service causant l'abandon du service global, il faut annuler toute exécution en cours.

Exemple Le service découvert (figure 5.14) n'est pas intuitivement valide puisqu'il ne respecte pas la règle R1 pour le service EDT (puisque lors de son échec, aucun des ses alternatives, EDF ou EDT, n'est activée) ni la règle R2 pour le service PL (puisque lors de son échec, les réservations d'hôtels et du vol sont maintenues). Cependant, le service illustré par la figure 5.15 est intuitivement valide puisqu'il respecte ces trois règles de bonne conduite pour chacun de ses services composants.

Il est important de noter que la règle R1 implique que la condition de compensation potentielle d'un service suivi par un ensemble d'alternatives (après un *XOR-split*) devient l'échec de toutes ces alternatives. Pour notre exemple, la condition de compensation potentielle du service PL est l'échec des services EDF, EDD et EDT.

Définition 5.7 (Service composé intuitivement valide)
Soit sc un service composé, sc est dit intuitivement valide *si et seulement si*
$\forall$ service composant s

C'1 : $\forall \diamond CondPtAlt_i(s), CondPtAlt_i(s) \in CondAlt(s)$

C'2 : $\forall \diamond CondPtCps_i(s)$

(a) s est compensable

(b) $CondPtCps_i(s) \in CondCps(s)$

C'3 : $\forall \diamond CondPtAnl_i(s), CondPtAnl_i(s) \in CondAnl(s)$

Avec $CondPtCps(s)$, $CondPtAnl(s)$ et $CondPtAlt(s)$ les conditions potentielles respectivement de compensation, d'annulation et d'alternative, du service composant s, définies par le flot transactionnel potentiel du flot de contrôle.

La première condition C'1 postule que toute condition d'alternative d'un service s éventuellement vraie est une condition d'alternative de s. Cette condition permet de respecter la règle R1. La condition C'2 postule que toute condition de compensation d'un service s éventuellement vraie est une condition de compensation de s (évidement s devra être dans ce cas compensable). Cette condition permet de respecter la règle R2. La condition C'3 postule que toute condition d'annu-

lation d'un service s éventuellement vraie est une condition d'annulation de s. Cette condition permet de respecter la règle R3.

Exemple Le service découvert, $cs_{decouvert}$ illustré par la figure 5.14, ne respecte pas la condition C'1 pour le service EDD ni la condition C'2 pour les services RV et RH. En effet, l'échec de EDT est une condition d'alternative du service EDD éventuellement vraie (puisque EDT n'est pas rejouable). Cependant elle n'est pas spécifiée comme condition d'alternative pour EDD. Par la suite $cs_{decouvert}$ ne respecte pas la règle R1 pour le service EDT puisque le service alternative EDD (ou EDF) n'est pas exécuté suite à l'échec de EDT. De même l'échec de PL est une condition de compensation de RV et de RH éventuellement vraie (puisque PL n'est pas rejouable). Or il n'est pas spécifié comme condition de compensation de RH ni de RV. Par la suite la règle R2 n'est pas respectée pour PL dans $cs_{decouvert}$ puisque le service PL peut échouer causant l'abandon du service composé et les services RH et RV ne sont pas compensés. Par conséquence le service $cs_{decouvert}$ n'est pas intuitivement valide. Par comparaison au service $cs_{ameliore}$ décrit à la figure 5.15, la condition C'1 est vérifiée pour le service EDD, puisque l'échec de EDF est spécifié comme condition d'alternative de EDD. Par la suite la règle R1 est respectée pour EDT. De même $cs_{ameliore}$ spécifie l'échec de PL comme condition de compensation de RV et RH respectant ainsi la règle R2 pour PL. C'est pourquoi, entre autres, $cs_{ameliore}$ est intuitivement valide.

Pour améliorer le flot transactionnel découvert, nous utilisons un ensemble de règles qui génèrent un ensemble de suggestions aux concepteurs afin de définir un flot transactionnel intuitivement valide (compte tenu des propriétés transactionnelles découvertes). Pour notre exemple, afin d'améliorer les mécanismes de recouvrement, nous pouvons suggérer les propositions suivantes aux concepteurs :

S_1 : ajouter une dépendance de compensation de PL vers RH et de PL vers RV,

S_2 : ajouter une dépendance de compensation de RH vers SBC

S_3 : ajouter une dépendance d'alternative de EDT vers EDD,

S_4 : ajouter une dépendance d'alternative de EDT vers EDF.

S_5 : ajouter une dépendance d'alternative de EDF vers EDT.

Les concepteurs peuvent ensuite garder les propositions qui leur semblent intéressantes et refuser les autres. Dans notre exemple, les concepteurs peuvent rejeter (i) la suggestion S_2 puisque le service SBC est sans effet, (ii) la suggestion S_4 puisque le service EDF est non rejouable et (iii) la suggestion S_5 puisque EDD est déjà une alternative de EDF. La figure 5.15 illustre le service d'organisation de voyage en ligne après amélioration.

Selon la définition 5.7, pour qu'un service soit intuitivement valide, il faut et il suffit que chacun de ses services composants vérifie les conditions C'1, C'2 et C'3. Nous utilisons les règles suivantes pour générer les suggestions appropriées afin d'assurer ces conditions pour chaque service composant **conformément aux propriétés transactionnelles découvertes**[19] :

[19]Rappelons que chaque service possède un ensemble de conditions de compensations. Puisque ces conditions sont exclusives, nous les regroupons dans une seule condition écrite sous forme normal exclusive disjonctive.

RÈGLE 5.3
$\forall$ service composant s

1. $\forall\ CondPtCps_i(s) \in CondPtCps(s)$
   si $(\Diamond\ CondPtCps_i) \bigwedge (CondPtCps_i \notin CondCps(s))$ alors suggérer que
   (a) s doit être compensable et que
   (b) $CondCps(s) = CondCps(s) \bigoplus CondPtCps_i(s)$
2. $\forall\ CondPtAnl_i(s) \in CondPtAnl(s)$
   si $(\Diamond\ CondPtAnl_i) \bigwedge (CondPtAnl_i \notin CondAnl(s))$ alors suggérer que
   $CondAnl(s) = CondAnl(s) \bigoplus CondPtAnl_i(s)$
3. $\forall\ CondPtAlt_i(s) \in CondPtAlt(s)$
   si $(\Diamond\ CondPtAlt_i) \bigwedge (CondPtAlt_i \notin CondAlt(s))$ alors suggérer que
   $CondAlt(s) = CondAlt(s) \bigoplus CondPtAlt_i(s)$

La première règle vise à assurer la condition C'1 pour chaque service composant. Elle postule que toute condition de compensation potentielle d'un service s $CondPtCps_i(s)$ éventuellement vraie et qui n'est pas déjà une condition de compensation de s soit ajoutée aux conditions de compensation de s (s devra être par conséquence compensable).

La deuxième règle vise à assurer la condition C'2 pour chaque service composant. Elle postule que toute condition d'annulation potentielle d'un service s, $CondPtAnl_i(s)$, éventuellement vraie et qui n'est pas déjà une condition d'annulation de s soit ajoutée aux conditions d'annulation de s.

La troisième règle vise à assurer la condition C'3 pour chaque service composant. Elle postule que toute condition d'alternative d'un service s, $CondPtAlt_i(s)$ éventuellement vraie et qui n'est pas déjà une condition d'alternative de s soit ajoutée aux conditions d'alternative de s.

Exemple : Voyons comment nous pouvons appliquer ces règles pour améliorer le flot transactionnel du service découvert :

- La condition de compensation potentielle de RH (respectivement RV), $PL.échoué$, est éventuellement vraie (puisque PL n'est pas rejouable) et n'est pas une condition de compensation de RH (respectivement RV). En appliquant la première règle, nous générons la suggestion S_1 ci-dessus.
- La condition d'alternative potentielle de EDD (respectivement de EDF), $EDT.échoué$ est éventuellement vraie (puisque EDT n'est pas rejouable) et n'est pas une condition d'alternative de EDD (respectivement de EDF). En appliquant la troisième règle, nous générons les suggestions S_3 et S_4 ci-dessus.

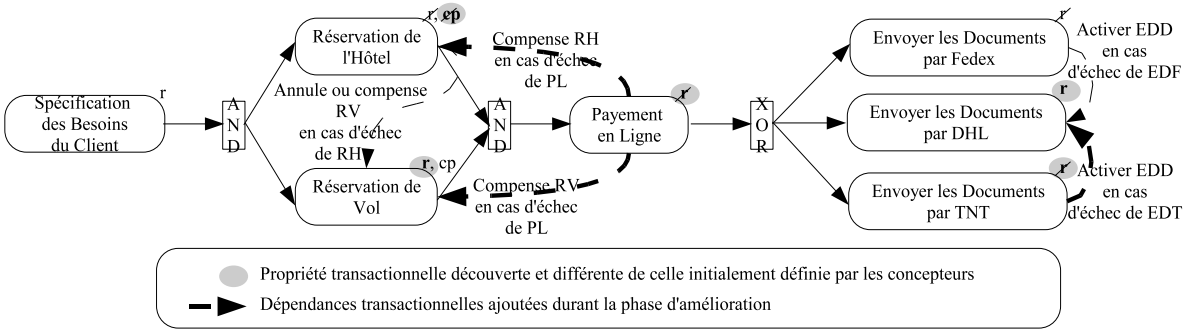
5.2.4 Synthèse

Dans cette section, nous avons présenté notre approche par ré-ingénierie du modèle de composition. Cette approche permet d'améliorer les mécanismes de recouvrement d'un SCT après observation d'un nombre suffisant grand de ses exécutions. Nous procédons principalement en deux étapes. La première étape consiste à découvrir le flot transactionnel effectif du SCT. Ensuite, nous utilisons un ensemble de techniques pour améliorer ce flot découvert.

L'avantage de cette approche est qu'elle repose sur des faits réels (par observation des exécutions) et non sur des hypothèse pouvant ne pas coïncider avec la réalité.

Cette approche améliore le flot transactionnel d'un SCT en supposant certaines règles de bonne conduite. Cependant pour pouvoir tenir compte des contextes spécifiques (où les concep-

Figure 5.15 Service Web composé d'organisation de voyage en ligne après amélioration.



teurs peuvent violer certaines de ces règles) nous procédons de façon interactive avec les concepteurs. En effet, nous générons un ensemble de suggestions permettant d'assurer ces règles de bonne conduite conformément au flot transactionnel découvert. Ensuite, les concepteurs peuvent appliquer les propositions qui leur semblent adéquates.

Néanmoins, ceci ne permet de prendre en compte la spécifité des contextes que partiellement. En effet, les suggestions proposées sont générées conformément à ces règles de bonne conduite et non conformément aux besoins spécifiques des concepteurs. En conséquence certaines anomalies (par rapport aux besoins des concepteurs) peuvent passer inaperçues.

Pour combiner l'avantage de cette approche (le fait de reposer sur des faits réels) avec celui de la précédente (le fait de prendre en compte les besoins spécifiques des concepteurs), nous pouvons appliquer les techniques de découverte pour déterminer le flot transactionnel effectif et utiliser ensuite les propriété de validation générée conformément aux besoins des concepteurs pour améliorer ce flot découvert.

5.3 Approche par composition de patrons transactionnels

Dans cette approche, nous introduisons le concept de `patron transactionnel` et nous montrons comment nous l'utilisons pour définir des SCT fiables. Un `patron transactionnel` combine la flexibilité des workflows et la fiabilité des modèles transactionnels. Nous définissons un service composé en connectant simplement des `patrons transactionnels`. Nous utilisons un ensemble de règle pour assurer la cohérence entre les patrons au sein d'un SCT.

Dans le paragraphe 5.3.1, nous introduisons le concept de `patron transactionnel`. Ensuite nous montrons, dans le paragraphe 5.3.2, comment nous utilisons ce concept pour définir des SCT fiables.

5.3.1 Patrons transactionnels

Un patron de composition, *pat* permet de définir, à partir d'un ensemble de service S, un `flot de contrôle` $pat(S)$. *pat* possède une fonction «potentiel» $potentiel_{pat}$ qui définit le `flot transactionnel potentiel` $potentiel_{pat}(S)$ du `flot de contrôle` $pat(S)$. La figure 5.16.a illustre une instance du patron AND-join et la figure 5.16.b illsutre le flot transactionnel potentielle correspondant. **Un patron transactionnel** dérivé d'un patron de composition *pat* est une instance de *pat* (un flot de contrôle) enrichie par un flot transactionnel *inclus simplement* dans son flot transactionnel potentiel.

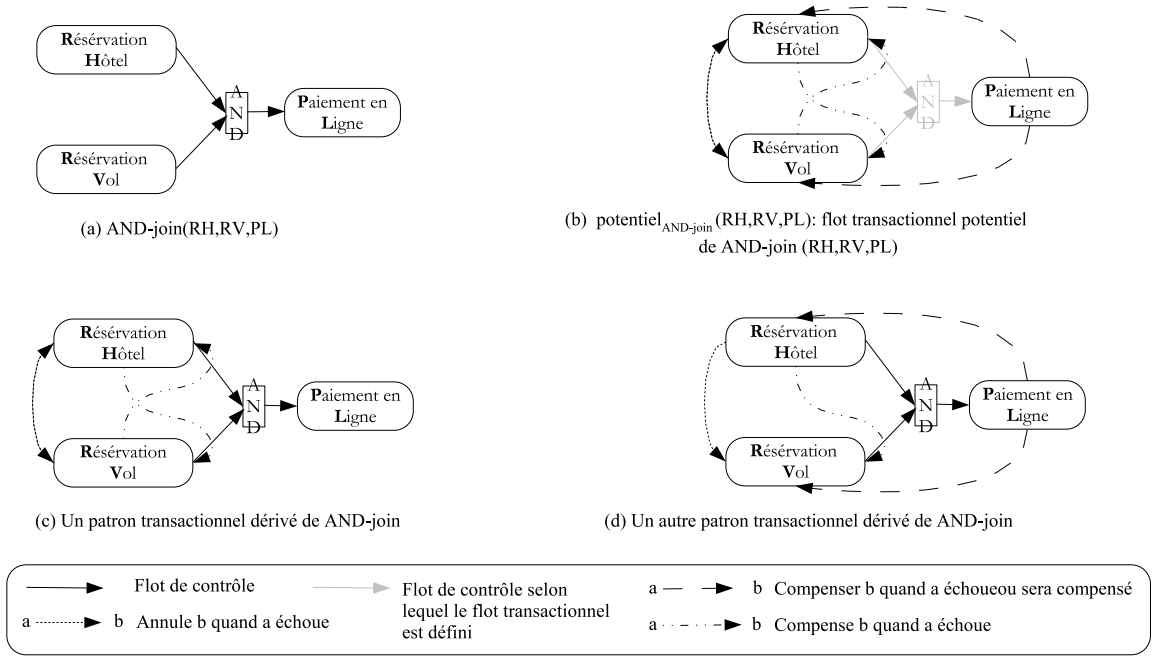
DÉFINITION 5.8 (PATRON TRANSACTIONNEL)
Soit un patron de composition *pat*, nous appelons `patron transactionnel` dérivé de *pat* tout SCT *sc* tq :

$$getFContrôle(sc) = pat(services(sc)) \text{ et}$$
$$getFTransactionnel(sc) \subseteq potentiel_{pat}(services(sc))$$

avec $services(sc)$ est l'ensemble des services composants de *sc*.

Plusieurs `patrons transactionnels` peuvent être dérivés du même patron de composition. La figure 5.16 illustre deux `patrons transactionnels` (figures 5.16.c et 5.16.d) dérivés du patron de composition *AND-join*. Chacun de ces deux patrons étend le flot de contrôle défini par le patron *AND-join*(figure 5.16.a) par un flot transactionnel *inclus simplement* dans le flot transactionnel potentiel correspondant (figure 5.16.b).

Figure 5.16 Deux patrons transactionnels dérivés du patron de composition *AND-join*.

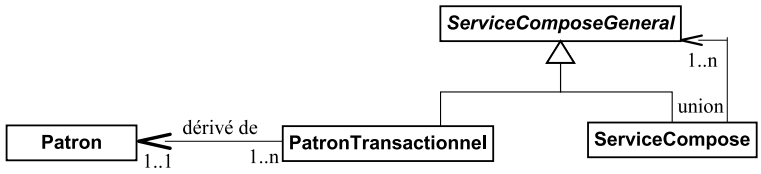


5.3.2 Composition de patrons transactionnsels

Dans cette approche, nous spécifions un SCT comme l'union de `patrons transactionnels` (voir diagramme UML de la figure 5.17). Plus formellement :

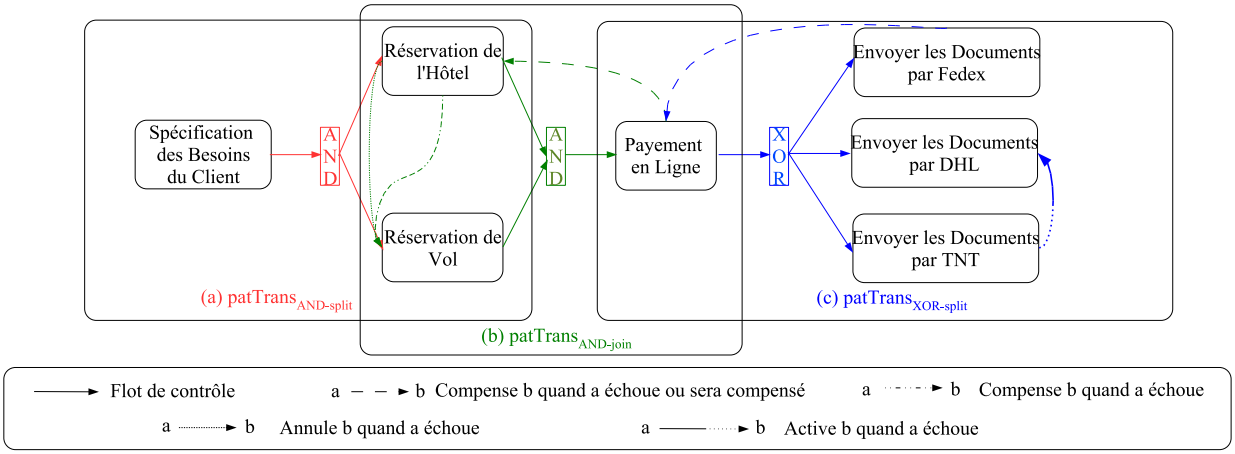
$$\forall \text{ SCT } sc = (ES, \mathcal{P}rec) \; \exists \text{ un ensemble de patrons de compositions } \{pat_1, \ldots, pat_n\} \text{ et une partition } S \text{ de } ES : S = \{S_1, \ldots S_n\} \text{ (avec } ES = \bigcup_{1 \leq i \leq n}(S_i) \mid$$
$$getFContrôle(sc) = \bigcup_{1 \leq i \leq n} pat_i(S_i) \text{ et}$$
$$sc = \bigcup_{1 \leq i \leq n} patTrans_i \text{ avec } patTrans_i \text{ est un patron transactionnel dérivé de } pat_i \text{ tq}$$
$$services(patTrans_i) = S_i$$

Figure 5.17 Selon cette approche, un SCT est défini comme l'union de patrons transactionnels.



La figure 5.18 illustre un service composé d'organisation de voyage en ligne spécifié comme l'union de patrons transactionnels : $patTrans_{AND-split} \bigcup patTrans_{AND-join} \bigcup patTrans_{XOR-split}$.

Figure 5.18 Service composé d'organisation de voyage en ligne défini comme une union de patrons transactionnels.



Service d'organisation de voyage en ligne défini comme l'union d'instances de patrons transactionnel:

$patTrans_{AND-split} \cup patTrans_{AND-join} \cup patTrans_{XOR-split}$

Cependant, connecter un ensemble de `patrons transactionnels` peut conduire à des incohérences au niveau (du flot) de contrôle et/ou au niveau (du flot) transactionnel entre les patrons d'un SCT.

Une incohérence de flot de contrôle peut surgir, par exemple, quand les patrons transactionnels sont disjoints ou quand une instance du patron *XOR-split* est suivie par une instance du patron *AND-join*.

Une incohérence de flot transactionnel peut surgir, par exemple, quand un service composant peut échouer causant l'abandon du SCT alors que les effets de l'exécution partielle n'est pas compensée. Prenons par exemple le SCT défini à la figure 5.18, nous remarquons d'une part que le service PL peut échouer (puis qu'il est spécifié que le service RH va être compensé quand PL échoue), d'autre part le service RV n'est pas compensé quand PL échoue.

Dans ce qui suit, nous montrons comment nous assurons la cohérence de contrôle et la cohérence transactionnelle entre les patrons transactionnels au sein d'un SCT.

Assurer la cohérence de contrôle. Nous utilisons la grammaire décrite au tableau 4.1 (page 63) pour assurer la cohérence du flot de contrôle. Soit sc un SCT tel que $sc = \bigcup_i patTrans_i(S_i)$,

le flot de contrôle de *sc* est cohérent *si et seulement si* le mot $\bigcup_i pat_i(S_i)$ est un mot généré par cette grammaire (avec pat_i le patron de composition duquel $patTrans_i$ est dérivé et S_i est l'ensemble des services de $patTrans_i$).

Assurer la cohérence transactionnelle. Pour assurer la cohérence du flot transactionnel, nous utilisons l'ensemble de règles 5.3 afin d'assurer que le service défini soit intuitivement valide. Cependant pour appliquer ces règles, nous avons besoin de connaître les propriétés transactionnelles des services composants. Deux cas se présentent. Soit le modèle de composition initialement spécifié par les concepteurs utilise des services Web réels dont les propriétés transactionnelles sont connues. Soit le modèle spécifié est abstrait et les services composants ne correspondent pas à des services réels. Dans ce deuxième cas, où les propriétés transactionnelles ne sont pas connues, nous utilisons l'ensemble de règles suivant (dans l'ordre indiqué) pour extraire les propriétés transactionnelles des services composants à partir du modèle spécifié :

RÈGLE 5.4
1. Chaque service est par défaut rejouable et pivot.
2. Chaque service qui est cible d'une dépendance de compensation est compensable.
3. Chaque service qui est source d'une dépendance de compensation (activé suite à son échec), d'annulation ou d'alternative n'est pas rejouable.

Par exemple, dans le SCT défini à la figure 5.18, la dépendance de compensation de RH vers RV implique que RH n'est pas rejouable (puis qu'il peut échouer) et que RV est compensable. De même, nous pouvons déduire que le service EDD est rejouable puisqu'il n'est source d'aucune dépendance de compensation, d'alternative ou d'annulation, contrairement au service EDF qui n'est pas rejouable. La figure 5.19 illustre les propriétés transactionnelles déduites à partir du modèle initialement spécifié par les concepteurs (illustré par la figure 5.18).

Une fois les propriétés transactionnelles connues, nous utilisons les règles 5.3 pour assurer que le flot transactionnel du service composé est intuitivement valide. Par exemple, l'application de ces règles au service composé illustré par la figure 5.18 permet de le corriger et l'améliorer comme illustré par la figure 5.19. Cet amélioration comprend l'ajout d'une dépendance de compensation du service PL vers le service RV et la spécification du service EDD comme une alternative au service EDF.

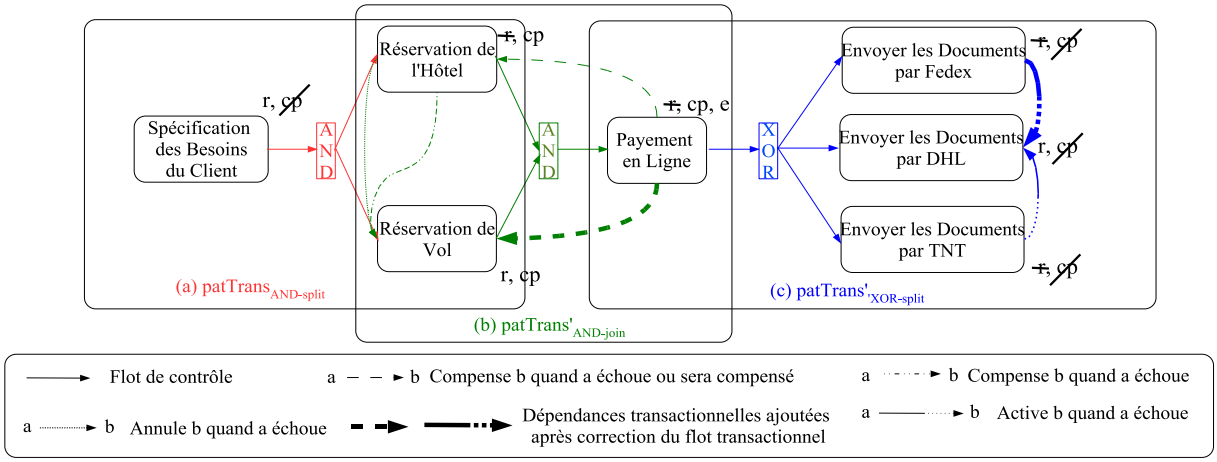
Il est important de noter que ces deux étapes (l'extraction des propriétés transactionnelles et la détection et correction d'incohérence transactionnelle) peuvent être menées d'une façon interactive avec les concepteurs pour prendre en compte leurs besoins spécifiques (qui ne correspondent pas aux règles de bonne conduite que nous supposons).

5.3.3 Synthèse

Dans cette section, nous avons introduit la notion de `patron transactionnel` et nous avons montré comment nous l'utilisons pour définir des SCT faibles. Un `patron transactionnel` est un patron qui combine la flexibilité des workflows et la fiabilité des modèles transactionnels. Nous définissons un SCT en connectant des `patrons transactionnels`. Nous utilisons un ensemble de règles pour assurer la cohérence (au niveau du flot de contrôle et transactionnel) entre les instances d'un SCT.

Comme dans l'approche précédente, nous assurons la cohérence transactionnelle en respectant certaines règles de bonnes conduites. De plus, pour tenir compte des besoins spécifiques des concepteurs, nous procédons d'une façon interactive. En effet, les corrections induites par ces

Figure 5.19 Service Web composé d'organisation de voyage en ligne après correction.



règles de cohérence ne sont pas faites d'une façon automatique mais plutôt interactive avec les concepteurs pour qu'ils puissent préciser leurs besoins spécifiques (violer certaines de ces règles de bonne conduite).

L'avantage majeur de cette approche est sa simplicité relative. En effet, d'une part elle permet de tenir compte des spécifités du contexte tout en évitant aux concepteurs de specifier l'ensemble d'états de terminaison acceptés. D'autre part, elle peut simplement offrir une interface graphique (plus appropriée aux utilisateurs humains qu'une interface textuelle) et procède d'une façon interactive avec les concepteurs.

5.4 Conclusion

Dans ce chapitre, nous avons présenté trois approches pour assurer des compositions fiables de services Web.

Dans la première approche, nous procédons par validation du SCT. Contrairement aux MTA, nous partons des spécifications des concepteurs pour déterminer les mécanismes transactionnels permettant d'assurer des exécutions correctes selon leurs besoins de fiabilité. L'avantage de cette approche est qu'elle permet de prendre en compte les besoins spécifiques des concepteurs.

La deuxième approche procède par ré-ingénierie du SCT. Cette approche permet d'améliorer les mécanismes de recouvrement d'un SCT après analyse de ses traces d'exécution. Par comparaison à la première approche, l'avantage de cette approche est qu'elle repose sur des faits réels et non sur des hypothèses qui peuvent ne pas coïncider avec la réalité. Cependant, elle ne permet de prendre en compte que partiellement les besoins spécifiques des concepteurs.

La troisième approche repose sur le concept de «patron transactionnel». Un patron transactionnel est un patron qui combine la flexibilité des workflows et la fiabilité des modèles transactionnels avancés. Plusieurs connecteurs transactionnels peuvent être dérivés d'un même patron transactionnel. Nous définissons un SCT comme union de patrons transactionnels. Nous utilisons un ensemble de règles de cohérence pour assurer que le SCT composé est fiable. L'avantage de cette approche est sa simplicité relative et son coté pratique. En plus, elle permet de tenir compte des besoins spécifiques des concepteurs.

Chapitre 6

Mise en oeuvre

Table des matières

6.1 Environnement de conception 99
6.2 Présentation de Bonita 101
6.2.1 Vue générale de Bonita 101
6.2.2 Architecture de Bonita 102
6.3 Extension de Bonita via des plug-in 103
6.3.1 Introduction 103
6.3.2 Structure et rôle des plug-in 104
6.3.3 Adaptation du moteur d'exécution 106
6.4 Exemples de plug-in 107
6.4.1 Plug-in rejouable 107
6.4.2 Plug-in compensation 108
6.5 Conclusion 108

Dans ce chapitre, nous illustrons l'implantation de notre approche dans un environnement de conception de services composés (section 6.1). Cet environnement est ensuite intégré dans Bonita un système de workflow développé par l'équipe ECOO (section 6.2). Nous montrons en particulier comment nous l'avons modifiés, via l'utilisation des plug-in, pour qu'il puisse supporter des services Web transactionnels (section 6.3).

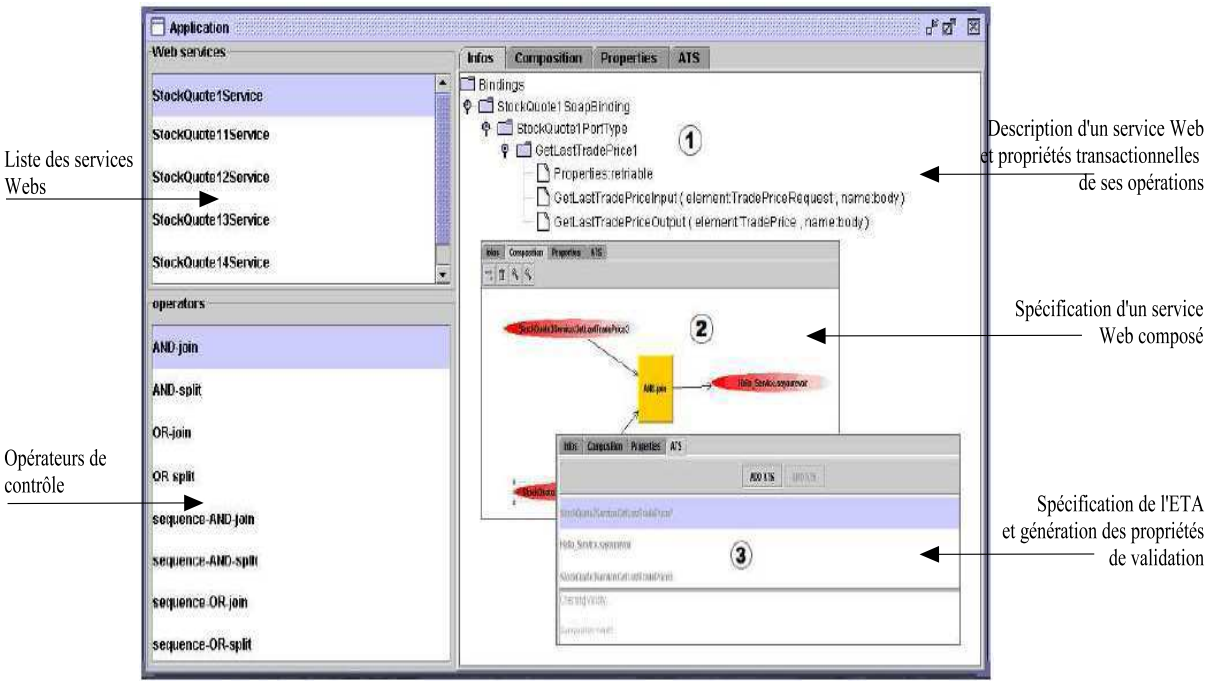
6.1 Environnement de conception

La faisabilité de notre approche est démontrée par un environnement de conception de services Web composés fiables. Cet environnement offre une interface graphique et implémente les différentes techniques que nous avons développées dans notre première approche de validation. Comme illustré par la figure 6.1, le concepteur dispose :

- d'un ensemble de services Web avec leurs propriétés transactionnelles,
- d'un ensemble d'opérateurs permettant la mise en oeuvre des patrons de composition,
- d'une page pour visualiser un service particulier, ses opérations ainsi que leurs propriétés transactionnelles,
- d'une page pour spécifier des services composés. Le concepteur sélectionne les opérations des services Web et définit la logique de leurs invocations en utilisant les opérateurs de contrôle dont il dispose. Il peut aussi enrichir le flot de contrôle spécifié par un ensemble de mécanismes de recouvrement,

– et d'une page pour spécifier l'ensemble des états de treminaison acceptés.

Figure 6.1 Capture d'écran des interfaces graphiques fournies par notre environnement de conception.



Ainsi, le concepteur peut choisir un ensemble de services et les combiner, en utilisant les opérateurs de contrôle, pour définir un service plus complexe à valeur ajoutée, et peut spécifier ses besoins de fiabilité en précisant l'ETA. L'environnement génère alors les propriétés de validation du service composé selon les besoins du concepteur. Si le service initialement défini n'est pas valide, le concepteur spécifiera un autre service en prenant en compte les propriétés de validation générées. Une fois un service valide est atteint, il sera exécuté par le système de workflow BONITA.

Chaque service Web est défini par un fichier WSDL qui spécifie l'ensemble des opérations offertes. Afin d'exprimer les propriétés transactionnelles d'un service, nous avons étendu la description WSDL par un fichier «WSTP»[20]. Un fichier WSTP précise les propriétés transactionnelles des opérations offertes par un service donné. La figure 6.2 illustre un fichier WSTP qui spécifie les propriétés transactionnelles de l'opération «GetLastTradePrice1» du service «stockquote1». La structure d'un fichier WSTP est relativement simple. Le préambule permet de spécifier l'URI[21] du service correspondant. La balise <properties> encapsule les propriétés transactionnelles des opérations du service. La balise <property> définit les propriétés transactionnelles d'une opération donnée où les attributs «operationName» et «transactional» définissent respectivement le nom de l'opération et ses propriétés transactionnelles.

Dans ce qui suit, nous présentons brièvement le système de workflow Bonita et nous montrons comment nous l'avons modifié pour qu'il puisse gérer notre approche transactionnelle.

[20] WSTP est l'acronyme de Web Service Transactional Properties.
[21] URI est l'acronyme de Uniform Resource Identifier

Figure 6.2 Fichier WSTP précisant les propriétés de l'opération «GetLastTradePrice1» du service «stockquote1».

```
<?xml version="1.0"?>
<definitions name="file1"

targetNamespace="http://example.com/stockquote1.wstp"
        xmlns:tns="http://example.com/stockquote1.wstp"
        xmlns:xsd1="http://example.com/stockquote1.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wstp/soap/"
        xmlns="http://schemas.xmlsoap.org/wstp/">

  <properties>
   <property operationName="GetLastTradePrice1"
transactional="retriable"/>
  </properties>

</definitions>
```

6.2 Présentation de Bonita

6.2.1 Vue générale de Bonita

Bonita [MC] est un système de workflow coopératif permettant de spécifier, d'exécuter et de contrôler des procédés coopératifs. Les principaux caractéristique de Bonita sont :

- C'est un moteur de workflow de troisième génération permettant l'anticipation des exécutions des activités avant que leurs prédécesseurs se terminent [GCG01].
- Il est implémenté en utilisant J2EE Enterprise Java Beans [Mica].
- Il offre la possibilité d'exécuter le code du coté serveur pour différent événements (e.g commencer et annuler des activités) à l'aide des «hooks». Les «hooks» peuvent être des programmes Java, ou des scripts BeanShell [Nie] et peuvent être attribués aux événements du procédé et des nœuds[22].
- Il offre une interface utilisateur graphique pour concevoir le flot de contrôle des procédés basé sur JFC/Swing de Java [Micb].
- Il fournit une interface Web pour contrôler les procédés de workflow. Les méthodes de workflow sont publiées comme des services Web.

Pour supporter ces caractéristiques, le système repose sur deux composants : le composant de modélisation et le composant d'exécution.

- Composant de modélisation : dans Bonita, l'application *GraphEditor* permet au concepteur de représenter et de visualiser le procédé de workflow en spécifiant les activités et les connecteurs entre elles. L'utilisateur peut spécifier les attributs d'une activité (type, description, mode d'exécution, date limite, rôle, etc.). Pour supporter les concepts de cette nouvelle génération de workflow, le composant de modélisation de Bonita permet la définition et la visualisation du procédé par plusieurs utilisateurs en cours d'exécution.
- Composant d'exécution : le moteur d'exécution de Bonita offre une gestion flexible des données et une exécution flexible du workflow. La gestion flexible des données permet aux activités d'échanger des résultats intermédiaires. L'exécution flexible permet l'anticipation des activités.

[22] Dans Bonita un nœud représente une étape dans un procédé qui peut être une activité dans un contexte workflow ou un service dans un contexte de services Web.

Bonita inclut aussi un navigateur permettant de gérer et de contrôler l'exécution des procédés d'une manière interactive. En plus, l'utilisation de l'application «Java Web start» et des technologies des services Web permettent aux concepteurs et aux organisations de manipuler le système de workflow Bonita via le Web.

6.2.2 Architecture de Bonita

Bonita est implémenté en utilisant la spécification «J2EE Enterprise Java Beans» (EJB) qui fournit un environnement plus flexible et portable pour les applications distribuées. La figure 6.3 illustre l'architecture générale de Bonita :

Couche de persistance de Bonita Les principaux éléments de la couche persistance de Bonita sont les *projets*. Un *projet* représente un procédé de workflow. Chaque procédé de workflow va contenir essentiellement des informations relatives (nom du procédé, l'utilisateur créateur, l'état de l'exécution du procédé, etc.) et un grand nombre de données associées :les activités, les connecteurs entre elles, les données du procédé, les propriétés du procédé, les participants, etc.

Afin de séparer les données du *projet* des éléments qui composent le *projet* lui même, nous avons développés un bean portable qui assomme les relations avec les autres entités qui composent la représentation d'un workflow dans Bonita : activités, connecteur, utilisateur, rôles, propriétés et «hooks».

Moteur d'exécution de Bonita Le moteur d'exécution de Bonita est conçu pour augmenter la flexibilité d'exécution d'un procédé. Cette flexibilité permet aux activités de partager les résultats intermédiaires pendant l'exécution, s'exécutant ainsi d'une façon non isolée. L'exécution flexible d'un workflow de Bonita est basée sur le principe d'anticipation [GCG01], qui permet à une activité de violer le modèle de synchronisation traditionnel terminé-activé. De la même façon, un utilisateur peut annuler l'exécution d'une activité ou changer le mode d'exécution d'une activité (avec ou sans anticipation) en cours d'exécution.

Le composant d'exécution de Bonita est constitué du Bean «EngineSession», des modes d'exécution et de la liste des tâches. Le bean «EngineSession» définit toutes les opérations d'exécution d'un procédé : commencer, terminer, et annuler une activité, et terminer l'exécution d'un procédé. Le moteur d'exécution est responsable de gérer les différents modes d'exécution. Les participants d'un procédé de workflow peut modifier le mode d'exécution de chaque activité en mettant l'anticipation à vraie ou à faux, permettant de spécifier si'ils veulent l'exécuter avec un mode conventionnel ou avec anticipation.

Le moteur d'exécution de Bonita introduit aussi le concept de «hook» d'activité. Les «hooks» dans Bonita sont des unités de code source qui peuvent être exécutées pendant l'exécution par le moteur. Ils sont associés aux activités des procédés de workflow. Les «hooks» doivent être écrites en Java ou dans l'un des langages scripts objet disponible dans Bonita (Tcl, BeanShell). L'utilisateur peut associer des «hooks» à différents moments du cycle de vie de l'exécution d'une activité : avant de commencer, après avoir commencé, lors de la création, avant de se terminer, après la terminaison et pendant l'anticipation (anicipating).

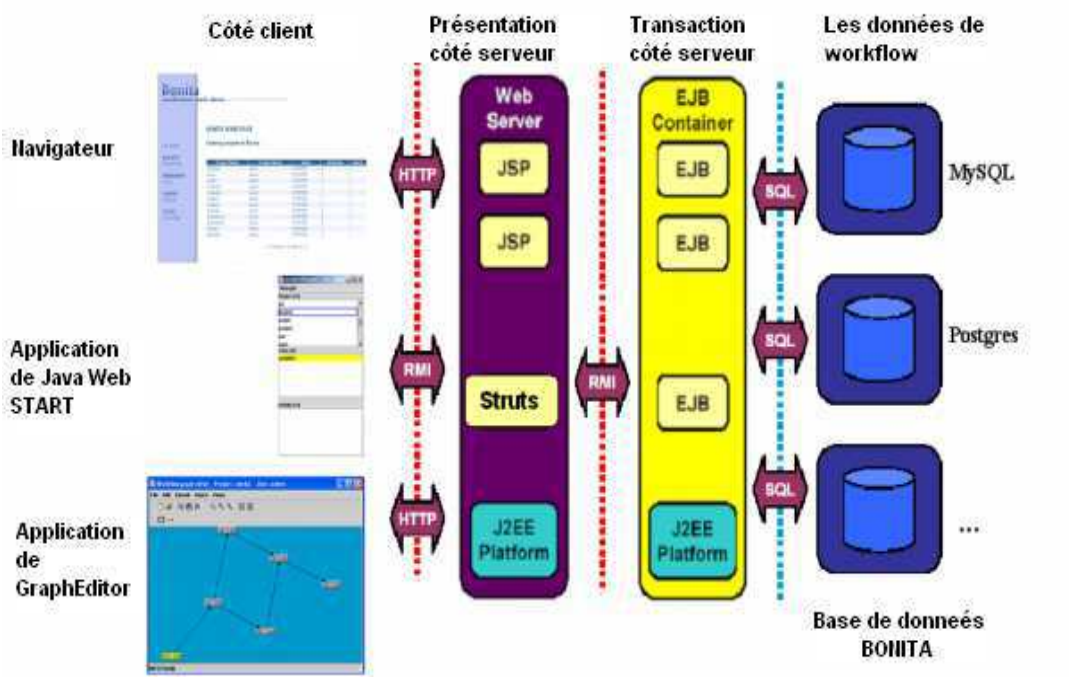
Les services communs de Bonita Bonita intègre un grand nombre de services de contrôle qui simplifient plusieurs aspects coopératifs :

- Le service de messagerie JMS de Bonita notifie la définition et l'exécution des changements dans un procédé de workflow. Chaque interaction d'un utilisateur est notifiée au noyau de Bonita et lance un événement JMS.
- Bonita utilise JMX[23] pour définir des dates limites des activités. Ce qui permet de notifier

[23] Java Management Extensions

à l'utilisateur qu'une activité ne se termine pas à sa date attendue.
– Le service de notification «Jabber» de Bonita permet aux utilisateurs de recevoir des notifications en temps réel et échanger différents types de messages.

Figure 6.3 Architecture générale du système de workflow Bonita.



6.3 Extension de Bonita via des plug-in

6.3.1 Introduction

Initialement, Bonita a été conçu pour supporter des exécutions flexibles, en particulier l'échanges des résultats intermédiaires et l'anticipation. Afin d'implémenter notre approche transactionnelle, nous devons adapter/étendre Bonita pour qu'il puisse prendre en compte les propriétés et les dépendances transactionnelles.

Une première approche, consistait à utiliser les «hooks» puisqu'ils permettent d'exécuter un code arbitraire du coté serveur. Notre première idée était de généraliser l'utilisation des «hooks» pour toutes les opérations possibles (e.g *start() terminate()*, *cancel()*). Cependant, cette approche ne prend pas en considération les différents modèles de procédés qui peuvent être envisagés dans un contexte plus général. Chaque modèle utilise un ensemble particulier d'états et d'opérations. La généralisation des «hooks», permettra la personnalisation du comportement d'un service, mais pas l'implémentation d'un modèle de procédé particulier.

Prenant en compte cette considération, nous avons adopté une autre approche. Elle consiste à identifier les aspects du modèle de procédé de Bonita qu'on peut personnaliser et de les séparer dans un plug-in. Chaque modèle de procédé doit définir les aspects spécifiques lui correspondant et les implémenter dans un plug-in. Le moteur doit être changé pour qu'il soit le plus indépendant possible des modèles. Il doit demander au plug-in les informations dépendantes d'un modèle particulier.

Selon ce principe, l'idée était de changer Bonita le moins possible afin de tirer profit de la stabilité de la version courante. En plus, puisque Bonita évolue constamment, ayant le moins de modifications possibles dans le moteur aidera à changer la version la plus récente afin de supporter la nouvelle approche des plug-in.

Dans ce qui suit, nous décrivons la nouvelle architecture en détaillant les plug-in et les modifications à apporter au moteur de Bonita.

6.3.2 Structure et rôle des plug-in

Dans cette section, nous étudions la version initiale de Bonita qui intègre le modèle d'anticipation. Nous séparons en particulier les éléments de base (qui sont définis par défaut indépendamment des plug-in) des éléments spécifique à l'anticipation (qui sont initialement gérés par le moteur) qui doivent être délégués à un plug-in d'anticipation. Nous présentons ensuite la structure de base d'un plug-in.

6.3.2.1 Éléments à déléguer aux plug-in

Nous commençons par étudier les différents composants de Bonita afin de déterminer quels éléments seront délégués au plug-in et quels éléments doivent rester dans le moteur. Le plug-in détermine quelle est l'état suivant d'un nœud[24] et doit être capable d'ajouter de nouveaux états à ceux qui sont définis par défaut. Dans les paragraphes suivantes, nous allons présenté les états des nœuds, les types d'opérations ainsi que les états des connecteurs actuels. Dans chaque paragraphe, nous précisons les éléments de base (qui vont exister indépendamment du plug-in utilisé) et les éléments nécessaires pour construire un plug-in afin d'imiter le comportement actuel de Bonita (qui est l'anticipation et la possibilité d'avoir des activités automatiques).

Types de nœuds. Les types de nœuds de base sont *AND-join* et *OR-Join*. Bonita définit aussi deux variations de ces deux types de base qui permettent l'exécution automatique des nœuds. Le premier est appelé *AND-join automatique*. Le deuxième est appelé *OR-join automatique*. L'activité ou le service relatif à ces types de nœud est activé et est terminé automatiquement quand sa condition d'activation sera vraie.

Etats des nœuds. Les états des nœuds de base sont :

- *initial* : Initialement un nœud est dans l'état *initial*. Dans cet état, sa condition d'activation n'est pas vérifiée. Tant que cette condition est évaluée à faux, le nœud reste dans cet état *initial*.
- *ready* : Un nœud atteint cet état quand sa condition d'activation devient vraie.
- *executing* : Un nœud atteint cet état quand il sera choisi pour être exécuté après avoir été activé.
- *suspended* : Un nœud atteint cet état quand il sera suspendu en cours d'exécution. Son exécution peut être reprise par la suite.
- *terminated* : Un nœud atteint cet état quand il se termine avec succès.
- *failed* : Un nœud atteint cet état quand son exécution échoue.
- *expired* : Un nœud atteint cet état quand sa date limite s'est expirée.
- *dead* : Un nœud atteint cet état quand il sera abandonné.

Les états initialement définis par Bonita et qui sont spécifiques à l'anticipation sont :

- *Anticipating* : Un nœud, automatique, atteint cet état quand il peut anticiper son exécution alors que ses activités prédécesseurs sont encore en cours d'exécution (se trouvent dans l'état *«executing»*).

[24] rappelons qu'un nœud correspond à une activité ou un service

- *Anticipatable* : Un nœud, qui n'est pas automatique, atteint cet état si il peut anticiper son exécution et sa condition d'anticipation devient vraie. Ce nœud reste dans cet état jusqu'à un participant le choisit et le fait ainsi transiter vers l'état *«anticipating»*.
- *Anticipation Suspended* : Un nœud atteint ce état quand il est suspendu alors qu'il est dans l'état *«anticipating»*. Son exécution anticipée peut être reprise en appelant l'opération *«resume»*.

Opérations des nœuds. Les opérations des nœuds de base sont :

- *start()* : Cette opération permet d'initialiser Un nœud. L'état d'un nœud après un appel à cette opération est *«initial»*.
- *activate()* : cette opération permet d'activer un nœud quand sa condition d'activation devient vraie. Son appel le fait transiter vers l'état *«ready»*.
- *suspend()* : Cette opération permet de suspendre l'exécution d'un nœud. Son appel le fait transiter vers l'état *«suspended»*.
- *resume()* : cette opération permet de reprendre l'exécution d'un nœud suspendu.
- *terminate()* : cette opération permet de terminer l'exécution d'un nœud avec succès. Son appel le fait transiter vers l'état *«terminated»*.
- *deactivate()* : cette opération est appelée quand la condition d'activation d'un nœud est évaluée à faux.
- *cancel()* : cette opération permet d'annuler un nœud quand sa condition d'activation ne sera jamais vérifiée. Son appel le fait transiter vers l'état *«dead»*.
- *Change to Failed* : c'est une opération spéciale qui est appelée par le moteur quand l'exécution d'un nœud échoue. Par défaut, l'appel à cette opération fait passer le nœud correspondant à l'état *«failed»*.

Pour gérer l'anticipation, Bonita fournit l'opération *«Activate Anticipation»*. Cette opération est appelée sur un nœud quand sa condition d'anticipation devient vraie. Son appel le fait transiter vers l'état *«anticipating»* si le nœud est automatique et vers l'état *«anticipatable»* sinon.

Comportements des nœuds. Le comportement d'un nœud est décrit par un tableau à transitions d'états. Il n'y a pas de comportement de base (c.à.d tout les comportements doivent être définis dans le plug-in). Selon la version actuelle de Bonita, il y a quatre comportements possibles qui sont le résultat de la combinaison de l'anticipation et de l'exécution automatique d'activité.

États des connecteurs. Les états des connecteurs de base sont *«initial»* (l'activité source n'été pas exécutée ou elle n'est pas encore terminée), *«active»* et *«dead»*. Pour gérer l'anticipation, Bonita considère, en plus, l'état «anticipating» d'un connecteur. Un conncteur atteint cet état si le nœud source est en cours d'exécution et le nœud destination peut anticiper son exécution.

Propriétés personnalisées. En plus des propriétés générales des workflows, le plug-in doit être capable de gérer des propriétés personnalisées pour tout un workflow ou pour une activité particulière. Les propriétés personnalisées peuvent être utilisées pour sauvegarder des données correspondantes au comportement spécifique du workflow ajouté par le plug-in.

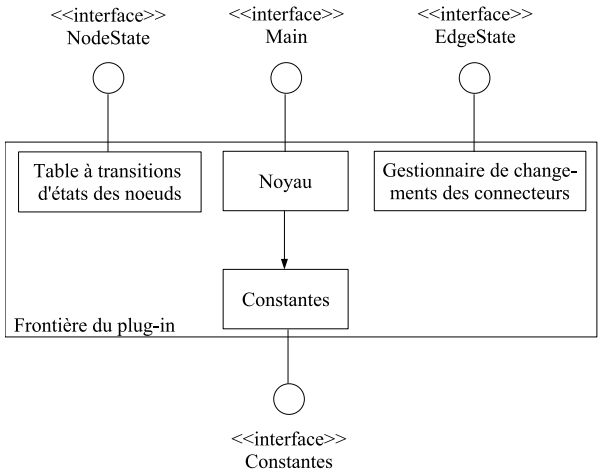
6.3.2.2 Architecture d'un plug-in

L'architecture du plug-in de base est illustrée à la figure 6.4 et elle est détaillée à la figure 6.5. Il est important de noter que la structure interne d'un plug-in n'est pas restreinte à cette architecture. La seule contrainte à respecter est de préserver les interfaces externes et leurs sémantiques que nous expliquons ci-dessous.

Dans ce plug-in, nous pouvons distinguer 4 modules principaux :

- **Le noyau.** Ce module permet au moteur d'accéder à tous les éléments d'un plug-in. Afin d'intégrer un modèle particulier, le moteur doit obtenir une instance du noyau du plug-in correspondant dont l'interface est connue. Cet instance permettra au moteur d'exécution de Bonita d'accéder aux éléments spécifiques à ce modèle. Les éléments retirés par le noyau peuvent être gérés par le moteur parce qu'ils implémentent des interfaces connues.
- **Les tableaux à transition d'états des nœuds.** Les types de comportement des nœuds sont représentés par des tableaux à transition d'états. L'interface *«NodeState»* définit une opération qui retourne l'état du nœud suivant après l'application d'une opération donnée sur ce nœud. L'implémentation correcte de l'interface *«NodeState»* pour un type de nœud particulier est obtenue via l'élément noyau en exécutant des méthodes dont les signatures sont pré-définies et connues.
- **Le gestionnaire des changements d'états des connecteurs.** Ce module aide le moteur d'exécution à déterminer l'opération à appliquer à un nœud quand ses connecteures entrants changent d'états. Le changement d'état des connecteurs entrants a un impact sur la condition d'activation, qui peut nécessiter l'application d'une opération sur le nœud. Comme le module des tableaux à transition d'états, l'interface *«EdgeState»* fournit une méthode qui indique l'opération nécessaire auprès d'un nœud quand ses connecteurs changent d'états.
- **Les constantes.** Ce module définit les états, les opérations, les types et les comportements des nœuds. Les deux modules précédents de l'architecture s'intéressent au comportement du modèle de procédé. Ce module s'intéresse aux définitions statiques du modèle de procédé.

Figure 6.4 Architecture de base du plug-in.



L'interface d'un plug-in est un groupe de classes abstraites qui définit tous les états, les opérations et les types de base. Elle définit aussi les méthodes pour faciliter l'accès à ces concepts par le moteur d'exécution. Tout plug-in doit étendre ces classes abstraites en ajoutant les concepts spécifiques lui correspondant et en réécrivant les méthodes de base.

6.3.3 Adaptation du moteur d'exécution

Certains éléments du moteur doivent être modifiés afin de supporter les plug-ins.

Les «hooks». Les états peuvent être redéfinis par un plug-in. Ainsi, puisque les «hooks» sont étroitement liés aux changements d'états, leur gestion doit être redéfinie. Il est possible de

définir la relation entre les états et les «hooks». Pour chaque état d'un nœud, il y aura deux «hooks» qui seront exécutés respectivement avant et après que l'état du nœud change.

Par exemple, la gestion des échecs peut être faite à l'aide d'un «hook» qui s'exécute avant que le nœud passe à l'état échoué (nous appelons ce hook *«beforeChangingToFAILED»*). Ce hook peut exécuter une compensation de l'activité du nœud ou il peut le rejouer (selon les propriétés de ce nœud).

Afin de contrôler l'exécution d'un «hook», des pre et des post conditions sont définies. Elles visent à contrôler l'état avant et après l'exécution d'un «hook» afin d'interdire certains changements (e.g les changement d'état d'un nœud durant les «hooks» *beforeChangingTO* qui peuvent interdire l'exécution d'un «hook» *afterChangingTo*). Si une condition d'un «hook» n'est pas vérifiée ou une exception se déclenche durant son exécution, l'état du nœud passe à l'état *«Failed»*.

Gestion de plusieurs plug-in dans la même installation de Bonita. Bien qu'une installation de Bonita puisse gérer plusieurs modèle de procédé (plusieurs plug-in), un projet est associé qu'à un seul plug-in. Ceci permet d'éviter les problèmes d'intégration des plug-ins. Si un projet utilise des concepts offerts par plusieurs plug-ins, un nouveau plug-in est défini. Ce nouveau plug-in agira comme une façade pour les autres plug-ins qui doivent être implémentés. Un plug-in peut être défini en étendant les classe d'un autre plug-in. Un plug-in de base est défini par défaut. Il devra être étendu par un autre plug-in si nécessaire.

6.4 Exemples de plug-in

Nous avons développé deux plug-in (qui étendent le plug-in de base) pour que Bonita puisse supporter notre approche transactionnelle. Ces deux plug-in sont : le plug-in rejouable et le plug-in compensation.

6.4.1 Plug-in rejouable

Ce plug-in permet de gérer la propriété *«rejouable»*. Cette propriété est définie en spécifiant le nombre maximale et la date limite de réitération (c.à.d après cette date limite, il n'est plus possible de rejouer le service correspondant). Ces information sont sauvegardées comme deux propriétés d'un nœud.

Par rapport au plug-in de base, le plug-in «rejouable» n'ajoute pas de nouveaux états ni de nouvelles opérations. Cependant, il modifie le tableau à transitions d'états d'un service *rejouable*. Dans le plug-in de base, l'opération *«changeToFAILED»* fait transiter un service vers l'état *«Failed»*. Cependant, dans le plug-in «rejouable» cette opération peut conduire à un échec ou à une ré-activation du service (c.à.d passer à l'état *«Ready»*) selon ses propriétés de réitération.

Nous définissons le plug-in «rejouable» en étendant le plug-in de base afin de modifier le tableau à transition d'états et de garder toutes les autres fonctionnalités. Ainsi, la méthode *«computeState»* de l'interface *«NodeState»* est réécrite pour qu'elle prenne en considération les deux propriétés de réitération d'un service lors du calcul de son nouveau état suite à l'application de l'opération *«changeToFAILED»*.

La figure 6.6 présente le diagramme de classe du plug-in «rejouable». Elle illustre comment ses classes (encadrées par le rectangle à gauche) étendent les classes du plug-in de base (encadrées par le rectangle à droite).

6.4.2 Plug-in compensation

Ce plug-in permet de gérer la propriété de compensation. Chaque service peut être défini comme compensable. Un service compensable possède un ensemble de services de compensation, dont la cardinalité varie de 0 à N (zero signifie que le service n'est pas compensable). Un service qui a plusieurs services de compensation est dit sélectif. Par exemple, un service A peut être défini comme compensable et son ensemble de compensation est $Comp_A = \{B, C, D\}$.

Par rapport au plug-in de base, le plug-in «compensation» ajoute deux nouveaux états *«compensating»* et *«compensated»*. Un service atteint l'état *«compensating»* quand un de ses services de compensation est exécuté. Un service est compensé quand il se termine avec succès et un service ultérieur échoue (nécessitant un mécanisme de compensation). Deux opérations peuvent être appelées quand un service est dans l'état *«compensating»*. La première est l'opération *«terminate»*. L'appel de cette opération désigne la terminaison de la compensation avec succès. Le service passe ainsi à l'état *«compennsated»*. La deuxième opération est *«cancel»*. L'appel de cette opération désigne l'échec de la compensation. Le service passe dans ce cas à l'état *«Failed»*.

Pour les services sélectifs (c.à.d avec plusieurs services de compensation), c'est l'utilisateur qui doit choisir le service de compensation à exécuter dans le cas ou la compensation est nécessaire. Le résultat d'un service peut être fixé par l'utilisateur. Dans ce cas, le service devient un service *pivot*, ce qui signifie qu'il ne peut pas être compensé. Le mécanisme de recouvrement qui peut impliquer la compensation de plusieurs activités dans l'ordre inverse de leurs exécutions, s'arrête quand il rencontre un service *pivot*.

Par exemple, supposons un service A_1 dont l'ensemble des ses services de compensation est $Comp_{A1} = \{B_1\}$. De même, nous considérons les services A_2, A_3, B_2, et B_3 où $Comp_{A2} = \{B_2\}$ et $Comp_{A3} = \{B_3\}$. La séquence d'exécution [A_1, A_2, *fix*, A_3, *«compensate»*] exécute A_1, A_2, et A_3 et ensuite le service de compensation B_3. Les services de compensation B_1 et B_2 ne sont pas exécutés parce qu'ils ont été annulés par l'instruction *fix*. Un service de compensation peut être aussi compensable. Ceci permet d'intégrer les compensations imbriquées.

Afin de supporter les propositions actuelles de protocoles de coordination transactionnelles de services Web (WS-coordination et WS-BusinessActivity), nous avons introduit la notion de «contexte» (scope). Un «contexte» regroupe un ensemble de services Web. Il définit l'étendu d'un mécanisme de compensation suite à l'échec d'un service. Seuls les services appartenant au même contexte sont compensés. Un mécanisme de compensation peut être arrêter par l'instruction *fix*.

Par exemple, reprenons les services A_1 et A_2 définis à l'exemple précédent. La séquence d'exécution suivante : A_1, [A_2, *«compensate»*] va exécuter la séquence A_1, A_2 et B_2. A_1 est à l'extérieur du «contexte» de l'instruction de compensation et ainsi son service de compensation B_1 n'est pas invoqué.

Nous pouvons définir aussi des «sous-contextes». Un «sous-contexte» n'est pas compensé lors d'une compensation au sein de son contexte père. Il est considérés comme une boîte noir. Un «sous-contexte» est traité comme une service non compensable au sein de son «contexte» père est compensé. Un «contexte» peut être éliminé (ou à la limite ignoré) une fois il est fermé.

6.5 Conclusion

Dans ce chapitre, nous avons présenté une implémentation de notre approche par validation des services composés transactionnels. Nous avons illustré un environnement de conception de services composés fiables selon les besoins des concepteurs. Nous avons ensuite montré comment nous avons adapté le système de workflow Bonitan, via des plug-in, pour qu'il puisse supporter

plusieurs modèles de procédés. Nous avons détaillé en particulier les plug-in rejouable et compensation que nous avons définis pour que Bonita puisse supporter notre approche transactionnelle et gérer par la suite l'exécution des services composés définis par l'environnement de conception.

Figure 6.5 Diagramme de classe de plug-in.

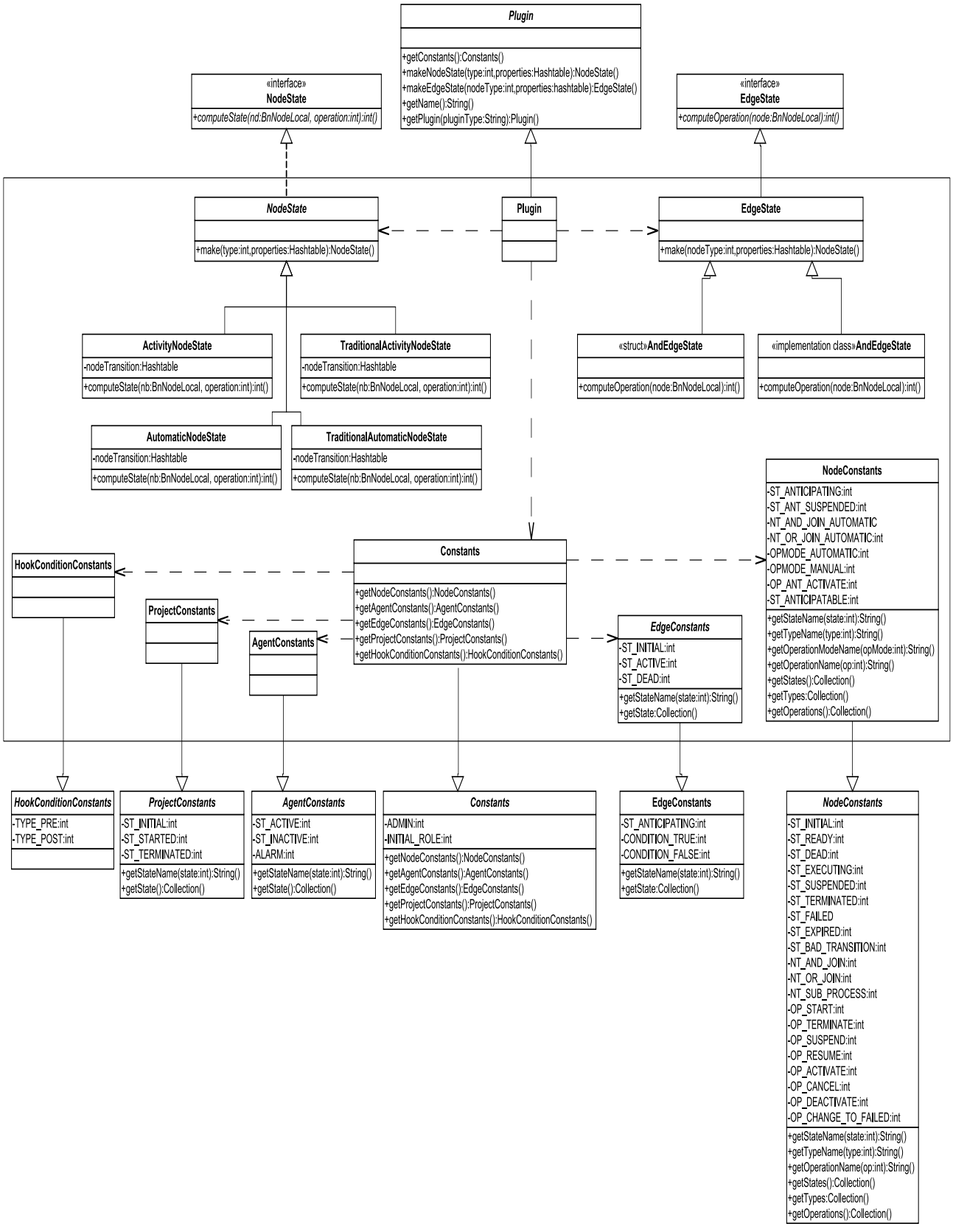
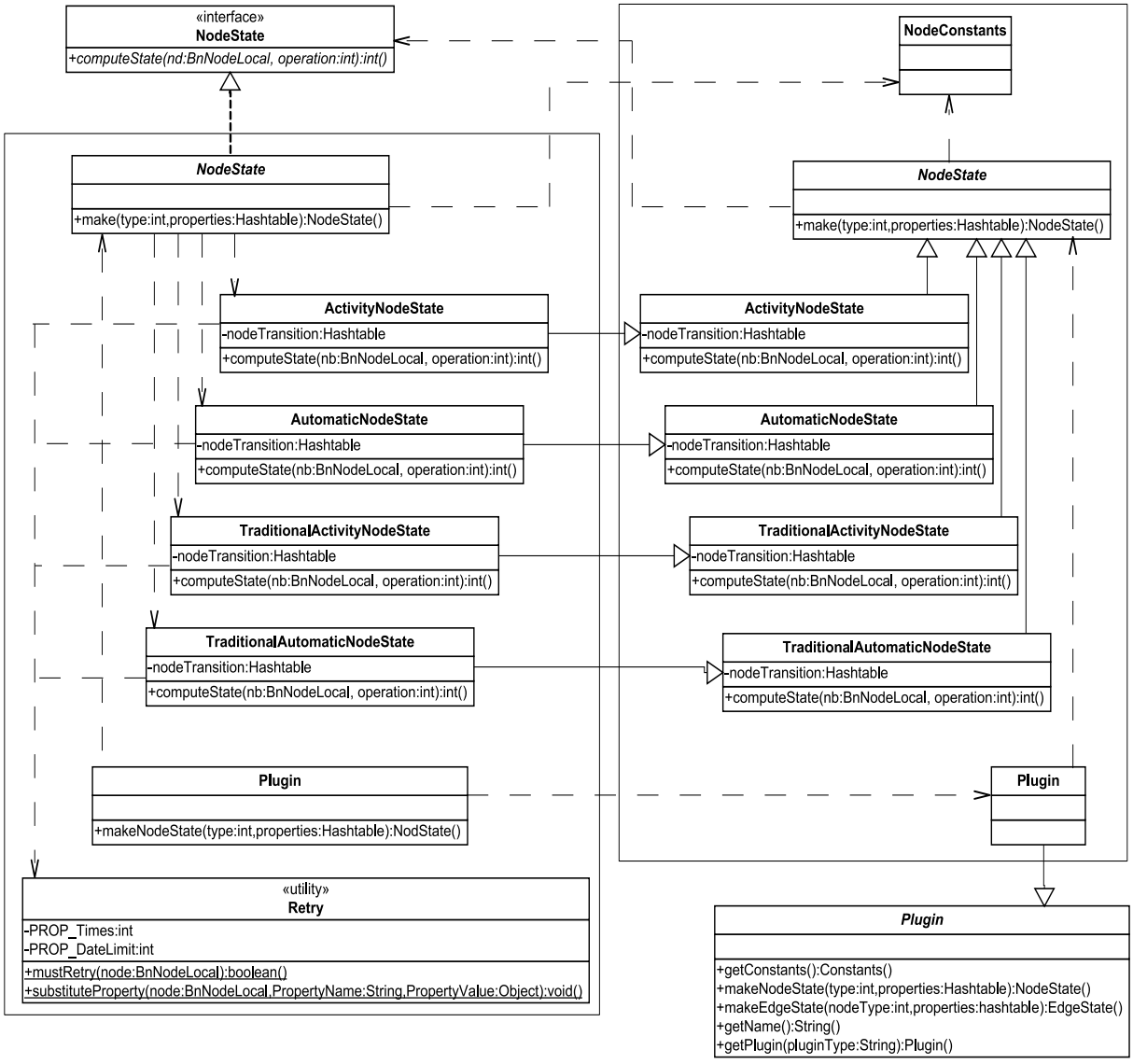


Figure 6.6 Le plug-in rejouable implémenté comme extension du plug-in de base.



Chapitre 7

Bilan et perspectives

Dans cette thèse, nous nous sommes intéressés à assurer des compositions fiables de services Web. Pour cela, nous avons identifié quatres dimensions à prendre en considération afin d'atteindre notre objectif.

- **flexibilité** : contrairement au modèles transactionnels avancés (MTA), il faut laisser la liberté aux concepteurs de spécifier leurs besoins en terme de structure de contrôle et de fiabilité,
- **correction** : contrairement aux workflows, il faut pouvoir assurer des exécutions correctes conformément aux besoins des concepteurs,
- **besoins des procédés métiers** : il faut que l'ensemble des techniques développées et des sémantiques utilisées soient approprié au contexte des procédés métiers,
- **contexte des services Web** : il faut respecter le contexte des services Web en particulier le couplage faible et la gestion décentralisée.

Les technologies actuelles des services Web ne permettent pas de résoudre d'une manière efficace ce problème. Ces technologies se sont principalement inspirées des (i) MTA en ce qui concerne la coordination transactionnelle et (ii) de l'approche «workflow» en ce qui concerne l'orchestration et la chorégraphie de services Web. Or ces deux approches présentent des limites pour intégrer les dimensions présentées ci-dessus. D'une part, les MTA assurent un bon niveau de correction au détriment des dimensions «flexibilité» et «procédés métiers». D'autre part, les systèmes de workflow assurent un bon niveau de flexibilité et s'adaptent bien aux contextes de procédés métiers. Cependant, ils ignorent relativement la dimension «correction». Les technologies actuelles de services Web ont adopté les concepts de ces deux approches et les ont adaptés au contexte de services Web. En conséquence, elles ont hérité de certaines de leurs limites.

7.1 Travail réalisé et contributions

Ayant ces quatres dimensions comme principes directeurs, nous avons proposé un modèle de services Web transactionnel au dessus duquel nous avons développé un ensemble de techniques pour assurer des exécutions correctes de services composés.

Notre modèle permet (i) d'enrichir la description de services Web pour mieux exprimer leurs propriétés transactionnelles et (ii) de modéliser un service Web composé dont les services composants manifestent des propriétés transactionnelles. Nous avons distingué en particulier entre le flot de contrôle et le flot transactionnel d'un service Web composé transactionnel (SCT). Le flot de contrôle définit l'ordre d'invocation des services composants. Le flot transactionnel définit les mécanismes de recouvrement en cas d'échecs.

L'originalité de notre modèle est qu'il a pu réconcilier les systèmes de workflow et les MTA. D'une part, un SCT peut être considéré comme un flot de services autonomes et hétérogènes. D'autre part, il peut être considéré aussi comme une transaction structurée où les services composants sont les sous-transactions et les interactions sont les dépendances transactionnelles. Par rapport à l'approche «workflow», notre modèle permet d'intégrer le flot transactionnel, un concept qui a été (relativement et volontairement) ignoré. Par comparaison au MTA, notre modèle permet plus de flexibilité.

Nous avons développé trois approches pour assurer des compositions fiables de services Web. Dans la première approche, nous procédons par validation du SCT. Selon cette approche, les concepteurs spécifient leurs besoins en terme de structure de contrôle et de fiabilité. Ensuite, nous utilisons un ensemble de techniques pour générer des propriétés de validation. Ces propriétés permettent de valider les SCT composés. Un SCT est valide conformément aux spécifications des concepteurs *si et seulement si* il respecte ces propriétés de validation.

L'originalité de cette première approche est la flexibilité que nous offrons aux concepteurs pour spécifier leurs besoins en terme de structure de contrôle et de fiabilité. Contrairement aux MTA, nous partons des spécifications des concepteurs pour déterminer les mécanismes transactionnels permettant d'assurer des exécutions fiables selon leurs besoins.

La deuxième approche procède par ré-ingénierie du SCT. Cette approche permet d'améliorer les mécanismes de recouvrement d'un SCT après analyse de ses traces d'exécutions. L'avantage de cette deuxième approche est qu'elle repose sur des faits réels et non sur des hypothèses qui peuvent ne pas coïncider avec la réalité.

La troisième approche repose sur le concept de «patron transactionnel», un nouveau concept que nous avons introduit. Un patron transactionnel est un patron qui combine la flexibilité des workflows et la fiabilité des MTA. Nous définissons un SCT en connectant des patrons transactionnels. Nous utilisons un ensemble de règles de cohérence pour assurer que le SCT composé est fiable. L'avantage de cette troisième approche est sa simplicité et son coté pratique. En plus comme la première approche, elle permet de prendre en compte les besoins spécifiques des concepteurs.

7.2 Perspectives

Actuellement, nous entrevoyons trois perspectives au travail présenté. Une première direction concerne l'extension de notre approche afin qu'elle puisse supporter des structures de contrôle plus complexes et gérer des propriétés transactionnelles dynamiques dans le temps. Ceci implique principalement pouvoir gérer les retours en arrière, les chevauchements entre les patrons et le changement dynamique des propriétés transactionnelles.

La deuxième perspective vise à l'intégration de notre travail aux efforts de standardisation existants. Nous distinguons à ce niveau entre deux sous objectifs. Le premier consiste à intégrer notre approche aux technologies de coordination transactionnelles actuelles à savoir WS-Transaction et WS-TXM, et aux langages d'orchestration et de chorégraphie BPEL et WS-CDL. Le but est de permettre (i) des protocoles de coordination ayant des structures de contrôle plus complexes et (ii) de fournir à BPEL et WS-CDL un ensemble de techniques pour assurer un certain niveau de fiabilité.

Le deuxième objectif consiste à situer notre travail par rapport à la thématique de recherche des services Web sémantiques. Les services Web sémantiques se situent à la convergence de deux domaines de recherche importants qui concernent les technologie de l'Internet, à savoir le Web sémantique et les services Web [KT03]. L'objectif ultime est d'automatiser les processus de dé-

couverte et de composition de services Web. L'idée de base est d'enrichir les technologies actuelles par une une couche sémantique afin de permettre la manipulation automatique des services Web. Notre approche peut être appliquée pour assurer des compositions fiables d'une façon automatique. La première étape consiste à enrichir la description de services Web par leurs propriétés transactionnelles. Ceci est faisable dans OWL-S [Coa03] en précisant les propriétés transactionnelles d'un service Web comme des attributs non fonctionnels dans son profil. Ensuite, nous pouvons appliquer nos techniques pour assurer des compositions fiables. Une autre application envisageable de notre approche est d'étendre les constructeurs de contrôle du modèle de service de OWL-S avec les patrons transactionnels. Nous pouvons alors utiliser l'ensemble de règles de cohérentes développées pour assurer des compositions fiables.

Enfin, la troisième perspective vise à adapter/combiner notre approche à/avec des domaines connexes telles que la découverte de procédés et la qualité de services de services Web.

La découverte de procédés permet de générer un modèle de procédé à partir des traces d'exécutions. Cette approche présente deux points intéressants. Premièrement, elle permet la modélisation automatique d'un procédé à partir de ses traces d'exécutions. Deuxièmement, le modèle découvert reflète bien les faits réels (par comparaison aux modèles conçus qui font des hypothèses pouvant ne pas coïncider avec la réalité). Une perspective intéressante à ce niveau est de combiner la découverte de procédés avec notre approche pour automatiser le processus de correction et d'amélioration des mécanismes de recouvrement à partir des traces d'exécutions.

Dans le contexte de qualité de services, notre approche permet d'assurer une qualité non fonctionnelle qui est la fiabilité. Notre travail peut à notre sens être facilement adapté pour assurer d'autres types de propriétés, de qualité de service, d'assurance ou de sécurité, à partir du moment où ces propriétés peuvent être exprimées sous forme d'états, et que leurs préconditions sont exclusives et stables dans le temps.

Annexe A

Patrons de composition

Dans cet annexe, nous présentons les patrons de composition *OR-split*, *OR-join*, *XOR-join* et *m-out-of-n*, et nous donnons leurs définitions formelles.

A.1 Patron OR-split

[vdABtHK00] définit un patron *OR-split* comme un point dans le procédé d'un workflow où un processus de contrôle se divise, selon certaines conditions, en un ou plusieurs processus de contrôle.

Selon notre approche, un patron *OR-split* spécifie qu'un ou plusieurs services, selon certaines conditions, seront activés après la terminaison d'un autre service.

DÉFINITION A.1 (PATRON *OR-split*)
Nous définissons le patron de composition *OR-split* comme la fonction suivante :

$$\text{OR-split} : \mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FC}ontrôle$$
$$\{s_0, s_1, \ldots, s_n\} \longmapsto fc = (ES, \mathcal{P}rec) \text{ tq}$$

- $ES = \{s_0, s_1, \ldots, s_n\}$,
- $\mathcal{P}rec(s_0.activer()) = \textit{événement externe à } fc$,
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = s_0.\textit{terminé} \bigwedge c_i$.

A.2 Patron OR-join

[vdABtHK00] définit un patron *OR-join* comme un point dans le procédé d'un workflow où un ou plusieurs processus de contrôle en parallèle convergent et se synchronisent en un seul processus de contrôle.

Selon notre approche, un patron *OR-join* spécifie qu'un service sera activé après la terminaison (avec succès ou échec) de plusieurs services s'exécutant en parallèle (dont au moins un service s'est terminé avec succès).

DÉFINITION A.2 (PATRON *OR-join*)
Nous définissons le patron de composition *OR-join* comme la fonction suivante :
OR-join : $\mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FC}ôntrole$
$\{s_1, \ldots, s_n, s_0\} \longmapsto fc = (ES, \mathcal{P}rec)$ tq
- $ES = \{s_1, \ldots, s_n, s_0\}$,
- $\mathcal{P}rec(s_0.activer()) = \bigoplus_{1 \leq i \leq n}(s_i.terminé \bigwedge_{1 \leq j \leq n, j \neq i}(s_j.terminé \bigvee s_j.échoué))$,
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = \textit{événement externe à fc.}$

A.3 Patron XOR-join

[vdABtHK00] définit un patron *XOR-join* comme un point dans le procédé d'un workflow où un seul processus de contrôle, parmi plusieurs s'exécutant en parallèle, sera choisi.

Selon notre approche, un patron *XOR-join* spécifie qu'un service sera activé suite à la terminaison d'un service parmi plusieurs s'exécutant en parallèle.

DÉFINITION A.3 (PATRON *XOR-join*)
Nous définissons le patron de composition *XOR-join* comme la fonction suivante :
XOR-join : $\mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FC}ôntrole$
$\{s_1, \ldots, s_n, s_0\} \longmapsto fc = (ES, \mathcal{P}rec)$
- $ES = \{s_1, \ldots, s_n, s_0\}$,
- $\mathcal{P}rec(s_0.activer()) = \bigoplus_{1 \leq i \leq n}(s_i.terminé \bigwedge c_i)$ | il y a toujours un seul c_j $(1 \leq j \leq n)$ évalué à vrai,
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = \textit{événement externe à fc}$

A.4 Patron m-out-of-n

[vdABtHK00] définit un patron *m-out-of-n* comme un point dans le procédé d'un workflow où m processus de contrôle parmi n s'exécutant en parallèle convergent et se synchronisent en un seul processus d'exécution.

Selon notre approche, un patron *m-out-of-n* spécifie qu'un service sera activé après la terminaison de m services parmi n s'exécutant en parallèle.

DÉFINITION A.4 (PATRON *m-out-of-n*)
Nous définissons le patron de composition *m-out-of-n* comme la fonction suivante :
m-out-of-n : $\mathcal{P}(\mathcal{SWT}) \longrightarrow \mathcal{FC}ôntrole$
$\{s_1, \ldots, s_n, s_0\} \longmapsto fc = (ES, \mathcal{P}rec)$
- $ES = \{s_1, \ldots, s_n, s_0\}$,
- $\mathcal{P}rec(s_0.activer()) = \bigoplus_{S \in \mathcal{P}(ES) | card(S) = m}\ (\bigwedge_{s_i \in S}\ s_i.terminé)$ avec $\mathcal{P}(ES)$ est l'ensemble des sous ensembles de ES et $card(S)$ la cardinalité de S.
- $\forall i,\ 1 \leq i \leq n\ \mathcal{P}rec(s_i.activer()) = \textit{événement externe à fc.}$

Annexe B

Fonctions «potentiel» des patrons de composition

Dans cet annexe, nous présentons les fonctions «potentiel» des patrons de composition *OR-split*, *OR-join*, *XOR-join* et *m-out-of-n*.

B.1 Flot transactionnel potentiel du patron OR-split

La fonction «potentiel» du patron *OR-split*, $potentiel_{OR-split}$ définit à partir d'un ensemble de services ($\{s_0, s_1, \ldots, s_n\}$) les dépendances transactionnelles suivantes : il y a une dépendance de compensation de s_i ($1 \leq i \leq n$) vers s_0 conformément à la politique de synchronisation des services $s_1, \ldots, s_n$.

Définition B.5 (fonction «potentiel» d'un *OR-split*)
La fonction «potentiel» du patron *OR-split* est définie comme suit :

$$\begin{array}{rccc} potentiel_{OR-split} : & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FT}ransactionnel \\ & S = \{s_0, s_1, \ldots, s_n\} & \longmapsto & ft = (ES, \mathcal{P}rec) \end{array}$$

avec ft le `flot transactionnel potentiel` du `flot de contrôle` *OR-split(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_0, s_1, \ldots, s_n\}$:
- $CondPtAlt(s_i) = faux$ $\forall 0 \leq i \leq n$,
- $CondPtCps(s_0) = \bigoplus_{1 \leq i \leq n} CondPtCps(s_i)$,
- $\forall\ 1 \leq i \leq n$ $CondPtCps(s_i) =$ *défini par le patron de synchronisation*,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n$ $CondPtAnl(s_i) =$ *défini par le patron de synchronisation.*

B.2 Flot transactionnel potentiel du patron OR-join

La fonction «potentiel» du patron *OR-join*, $potentiel_{OR-join}$ définit à partir d'un ensemble de services ($\{s_1, \ldots, s_n, s_0\}$) les dépendances transactionnelles suivantes : tout service s_i ($1 \leq i \leq n$) sera annulé ou compensé (selon si'il est encore en cours d'exécution ou il s'est déjà terminé) quand tous les autres services échouent. Le service s_0 va compenser tout les autres services en cas d'échec ou de compensation.

DÉFINITION B.6 (FONCTION «POTENTIEL» D'UN *OR-join*)
La fonction «potentiel» du patron *OR-join* est définie comme suit :

$$\begin{array}{rccc} potentiel_{OR-join} : & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FT}ransactionnel \\ & S = \{s_1, \ldots, s_n, s_0\} & \longmapsto & ft = (ES, \mathcal{P}rec) \end{array}$$

avec ft le flot `transactionnel potentiel` du `flot de contrôle` *OR-join(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_1, \ldots, s_n, s_0\}$:
- $CondPtAlt(s_i) = faux\ \forall 0 \leq i \leq n$,
- $CondPtCps(s_0) =$ *événement externe à ft*,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) = s_0.échoué \bigoplus s_0.compensé$
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) = faux$

B.3 Flot transactionnel potentiel du patron XOR-join

La fonction «potentiel» du patron *XOR-join*, $potentiel_{XOR-join}$ définit à partir d'un ensemble de services ($\{s_1, \ldots, s_n, s_0\}$) les dépendances transactionnelles suivantes : Le service s_0 va compenser le service qui l'a activé en cas d'échec ou de compensation.

DÉFINITION B.7 (FONCTION «POTENTIEL» D'UN *XOR-join*)
La fonction «potentiel» du patron *XOR-join* est définie comme suit :

$$\begin{array}{rccc} potentiel_{XOR-join} : & \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FT}ransactionnel \\ & S = \{s_1, \ldots, s_n, s_0\} & \longmapsto & ft = (ES, \mathcal{P}rec) \end{array}$$

avec ft le flot `transactionnel potentiel` du `flot de contrôle` *XOR-join(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_1, \ldots, s_n, s_0\}$:
- $CondPtAlt(s_i) = faux\ \forall 0 \leq i \leq n$,
- $CondPtCps(s_0) =$ *événement externe à ft*,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) = s_0.échoué \bigwedge c_i \bigoplus s_0.compensé \bigwedge c_i$, avec c_i les conditions définies déjà par le patron $XOR - join$,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) = faux$.

B.4 Flot transactionnel potentiel du patron m-out-of-n

La fonction «potentiel» du patron *m-out-of-n*, $potentiel_{m-out-of-n}$ définit à partir d'un ensemble de services ($\{s_1, \ldots, s_n, s_0\}$) les dépendances transactionnelles suivantes : tout service s_i sera annulé ou compensé (selon si'il est encore en cours d'exécution ou il s'est déjà terminé) quand $n - m + 1$ services échouent. Le service s_0 va compenser tout les autres services (s'ils sont terminés avec succès) en cas d'échec ou de compensation.

DÉFINITION B.8 (FONCTION «POTENTIEL» D'UN *m-out-of-n*)
La fonction «potentiel» du patron *m-out-of-n* est définie comme suit :

$$potentiel_{m-out-of-n} : \begin{array}{ccc} \mathcal{P}(\mathcal{SWT}) & \longrightarrow & \mathcal{FT}ransactionnel \\ S = \{s_1, \ldots, s_n, s_0\} & \longmapsto & ft = (ES, \mathcal{P}rec) \end{array}$$

avec ft le flot `transactionnel potentiel` du `flot de contrôle` *m-out-of-n(S)*.
$\mathcal{P}rec$ définit pour chaque service $s \in \{s_1, \ldots, s_n, s_0\}$:
- $CondPtAlt(s_i) = faux\ \forall 0 \leq i \leq n$,
- $CondPtCps(s_0) =$ *événement externe à ft*,
- $\forall\ 1 \leq i \leq n\ CondPtCps(s_i) = s_0.échoué \bigoplus s_0.compensé \bigoplus_{S \in \mathcal{P}(ES) | card(S) = n-m+1} (\bigwedge_{s_i \in S} s_j.échoué)$,
- $CondPtAnl(s_0) = faux$,
- $\forall\ 1 \leq i \leq n\ CondPtAnl(s_i) = \bigoplus_{S \in \mathcal{P}(ES) | card(S) = n-m+1} (\bigwedge_{s_i \in S} s_j.échoué)$.

Annexe C

Algorithmes de calcul du comportement transactionnel induit par un ETA

Dans cet annexe, nous présentons les algorithmes permettant d'extraire les conditions d'annulation et d'alternative, d'un service s, induite par l'ETA.

C.1 Algorithme d'extraction de la condition d'annulation d'un service s induite par l'ETA

L'algorithme 3 permet d'extraire pour chaque service s sa condition d'annulation induite par l'ETA : $CondCpsETA(s)$. Le principe est le suivant : une condition d'annulation potentielle d'un service s devient une condition d'annulation induite par l'ETA si elle est vérifiée dans un état de terminaison accepté dans lequel l'état du service s est *annulé*. Ainsi l'algorithme va parcourir l'ETA état par état (ligne 4 à la ligne 14). Pour chaque état de terminaison accepté dont l'état du service s est *annulé* (ligne 5), l'algorithme va chercher la condition d'annulation potentielle de s qui est vérifiée dans cet état (ligne 6 à la ligne 13). Les lignes 7 et 13 permettent de parcourir les conditions d'annulation potentielles de S. La variable booléenne «satisfé» (lignes 6 et 11) permet de marquer si la condition d'annulation potentielle de s courante est vérifiée ou non dans l'état de terminaison courant (variable «eta»)(ligne 9). Une condition d'annulation potentielle qui se trouve vérifiée dans «eta» elle sera considérée comme condition d'annulation de s induite par l'ETA (ligne 10). Cette condition sera retirée de la liste des conditions d'annulation potentielles de s pour ne pas être re-vérifiée (ligne 12) dans les autres états de terminaison.

C.2 Algorithme d'extraction de la condition d'alternative d'un service s induite par l'ETA

L'algorithme 4 permet d'extraire pour chaque service s sa condition d'alternative induite par l'ETA : $CondAltETA(s)$. Le principe est le suivant : une condition d'alternative potentielle d'un service s devient une condition d'alternative induite par l'ETA si elle est vérifiée dans un état de terminaison accepté dans lequel l'état du service s est *terminé* ou *compensé*. Ainsi l'algorithme va parcourir l'ETA état par état (ligne 4 à la ligne 14). Pour chaque état de terminaison accepté dont l'état du service s est *terminé* ou *compensé* (ligne 5), l'algorithme va chercher la condition d'alternative potentielle de s qui est vérifiée dans cet état (ligne 6 à la ligne 13). Les lignes 7 et 13 permettent de parcourir les conditions d'alternative potentielles de S. La variable booléenne

Entrées : ETA : l'ETA exigé par les concepteurs
CondPtAnl(s) : La condition d'annulation potentielle de s
Sorties : CondAnlETA(s) : La condition d'annulation de s induite par l'ETA
Données : eta :l'état de terminaison accepté courant (dans l'ETA)
$CondPtAnl_i(s)$: la condition d'annulation potentielle courante dans CondPtAnl(s)
satsifé : une variable booléenne mis à vrai si $CondPtAnl_i(s)$ est satisfaite dans le courant eta

```
1  début
2      CondAnlETA(s) ⟵ ∅
3      eta ⟵ eta suivant dans l'ETA
4      tant que eta ≠ null faire
5          si l'état de s dans eta est annulé alors
6              satisfé ⟵ faux
7              CondPtAnl_i(s) ⟵ CondPtAnl_i(s) suivant dans CondPtAnl(s)
8              tant que non satisfé et CondPtAnl_i(s) ≠ null faire
9                  si CondPtAnl_i(s) est satisfait dans eta alors
10                     CondAnlETA(s) ⟵ CondAnlETA(s) ⊕ CondPtAnl_i(s)
11                     satisfé ⟵ vrai
12                     CondPtAnl(s) ⟵ CondPtAnl(s) - CondPtAnl_i(s)
13                 CondPtAnl_i(s) ⟵ CondPtAnl_i(s) suivant dans CondPtAnl(s)
14         eta ⟵ eta suivant dans l'ETA
15 fin
```

Algorithme 3 : Extraction de la condition d'annulation d'un service s induite par l'ETA

«satisfé» (lignes 6 et 11) permet de marquer si la condition d'alternative potentielle de s courante est vérifiée ou non dans l'état de terminaison courant (variable «eta»)(ligne 9). Une condition d'alternative potentielle qui se trouve vérifiée dans «eta» elle sera considérée comme condition d'alternative de s induite par l'ETA (ligne 10). Cette condition sera retirée de la liste des conditions d'alternative potentielles de s pour ne pas être re-vérifiée (ligne 12) dans les autres états de terminaison.

Entrées : ETA : l'ETA exigé par les concepteurs
CondPtAlt(s) : La condition d'alternative potentielle de s
Sorties : CondAltETA(s) : La condition d'alternative de s induite par l'ETA
Données : eta :l'état de terminaison accepté courant (dans l'ETA)
$CondPtAlt_i(s)$: la condition d'alternative potentielle courante dans CondPtAlt(s)
satsifé : une variable booléenne mis à vrai si $CondPtAlt_i(s)$ est satisfaite dans le courant eta

```
1  début
2      CondAltETA(s) ⟵ ∅
3      eta ⟵ eta suivant dans l'ETA
4      tant que eta ≠ null faire
5          si l'état de s dans eta est terminé ou compensé alors
6              satisfé ⟵ faux
7              CondPtAlt_i(s) ⟵ CondPtAlt_i(s) suivant dans CondPtAlt(s)
8              tant que non satisfé et CondPtAlt_i(s) ≠ null faire
9                  si CondPtAlt_i(s) est satisfait dans eta alors
10                     CondCpsETA(s) ⟵ CondCpsETA(s) ⨁ CondPtAlt_i(s)
11                     satisfé ⟵ vrai
12                     CondPtAlt(s) ⟵ CondPtAlt(s) - CondPtAlt_i(s)
13                 CondPtAlt_i(s) ⟵ CondPtAlt_i(s) suivant dans CondPtAlt(s)
14         eta ⟵ eta suivant dans l'ETA
15 fin
```

Algorithme 4 : Extraction de la condition d'alternative d'un service s induite par l'ETA

Annexe D

Demonstrations des lemmes

Dans cet annexe, nous donnons les démonstrations des trois lemmes que nous avons introduit au chapitre 5

D.1 Démonstration du lemme 5.1

Énoncé du lemme
Un service composé sc est valide *si et seulement si* $calculET_{avecEchecs}(getFTransactionnel(sc)) \subseteq ETA_{avecEchecs}$.

Démonstration
Selon la définition 5.4

$$\text{un service composé } sc \text{ est valide} \Leftrightarrow calculET(sc) \subseteq ETA$$

Ceci signifie que

$$(1)\ sc \text{ est valide} \Leftrightarrow calculET_{sansEchec}(getFControle(sc)) \bigcup calculET_{avecEchecs}(getFTransactionnel(sc)) \subseteq ETA_{sansEchec} \bigcup ETA_{avecEchecs}$$

Or puisque sc et ETA sont définis selon le même flot de contrôle (qui définit l'ensemble des états de terminaison sans échec) alors

$$(2)\ calculET_{sansEchec}(getFControle(sc)) = ETA_{sansEchec}$$

compte tenu de (1) et (2) (et puis que un état de terminaison est d'un seul type), nous déduisons que

$$sc \text{ est valide} \Leftrightarrow calculET_{avecEchecs}(getFTransactionnel(sc)) \subseteq ETA_{avecEchecs}$$

D.2 Démonstration du lemme 5.2

Énoncé du lemme
Soit fc un flot de contrôle, $ft2$ et $ft1$ deux flots transactionnels définis selon fc. $ft2$ est inclus fortement dans $ft1 \Longleftrightarrow calculET_{avecEchecs}(ft2) \subseteq calculET_{avecEchecs}(ft1)$.

Démonstration
Nous appelons historique d'exécution d'un service composé sc une séquence d'événements générés durant l'exécution d'une de ses instances. Deux instances qui ont la même historique d'exécution possèdent le même état de terminaison.

Démontrer ce lemme revient donc à montrer que l'ensemble des historiques des exécutions de $ft2$ est inclus dans celui de $ft1$. Soit $h = \{e_0, e_1, \ldots, e_i, e_{i+1} \ldots e_n\}$ une historique d'exécution de $ft2$, montrons par récurrence que h peut être aussi une historique d'exécution de $ft1$.

Le premier événement possible d'être généré par $ft2$ est soit $s_0.abandonné$, soit $s_0.activé$ (avec s_0 le premier service composant). Cette sous historique est aussi possible dans $ft1$ (puisque $ft2$ et $ft1$ partagent le même flot de contrôle). Supposons maintenant que la sous historique $h1 = \{e_0, e_1, \ldots, e_i\}$ de $ft2$ est possible d'être généré par $ft1$. Montrons que la sous historique $h2 = \{e_0, e_1, \ldots, e_i, e_{i+1}\}$ est aussi possible dans $ft1$. l'événement e_i peut être l'un des événements suivants : *abandonné*, *activé*, *échoué*, *annulé*, *terminé* ou *compensé*. Si l'état e_i est *abandonné* ou *terminé* le prochain événement possible dans $ft2$ est aussi possible dans $ft1$ parce qu'ils partagent le même flot de contrôle. Si l'état e_i est *activé* le prochain événement (*échoué* ou *terminé*) possible dans $ft2$ est aussi possible dans $ft1$ parce que $ft2$ n'accepte que les échecs acceptés par $ft1$ donc si s_i (service générant l'événement e_i) est possible d'échouer dans $ft2$ alors il l'est forcement possible dans $ft1$ (le cas contraire n'est pas vrai car $ft1$ peut accepter des échecs non acceptés par $ft2$). Si s_i est rejouable, nous revenons au cas précédent. Maintenant si e_i est *annulé* ou *compensé*, ça veut dire que cet événement est déclenché dans un mécanisme de recouvrement. Or puisque $ft2$ n'accepte que les échecs acceptés par $ft1$ et adopte les mêmes mécanismes de recouvrement que $ft1$ alors nécessairement l'événement généré après e_i est aussi possible dans $ft1$. Par la suite toute historique d'exécution de $ft2$ est aussi une historique d'exécution de $ft1$. Le cas contraire est faux parce que l'événement e_{i+1} dans $ft1$ peut être l'échec d'un service non accepté dans $ft2$.

D.3 Démonstration du lemme 5.3

Énoncé du lemme
Un service composé sc est valide $\Longleftrightarrow getFTransactionnel(sc) \sqsubseteq calculET^{-1}_{avecEchecs}(ETA_{avecEchecs})$.

Démonstration
Soit sc un service composé tq

$$getFTransactionnel(sc) \sqsubseteq calculET^{-1}_{avecEchecs}(ETA_{avecEchecs})$$

D'après le lemme 5.2, ceci est équivalent à

$$calculET(getFTransactionnel(sc)) \subseteq calculET(calculET^{-1}_{avecEchecs}(ETA_{avecEchecs}))$$

ce qui signifie

$$calculET_{avecEchecs}(getFTransactionnel(sc)) \subseteq ETA_{avecEchecs}$$

Or d'après le lemme 5.1, le service sc est valide.

Annexe E

Acronymes

SWC : Service Web Composé
SWT : Service Web Transactionnel
$\mathcal{SWT}$: ensemble de tous les SWT
$\mathcal{P}rofil$: ensemble des profils de tous les services Web
s^r : s est rejouable
s^{cp} : s est compensable
s^p : s est pivot
$\mathcal{E}tats$: ensemble des états de tous les SWT
$\mathcal{T}ransitions$: ensemble des transitions de tous les SWT
états : fonction qui retourne l'ensemble des états d'un SWT
$\mathcal{T}ransExt$: ensemble des transitions externes de tous les SWT
$\mathcal{T}ransInt$: ensemble des transitions internes de tous les SWT
source : fonction qui retourne l'état source d'une transition
cible : fonction qui retourne l'état cible d'une transition
tex : fonction qui retourne l'ensemble des transitions externes d'un SWT
tin : fonction qui retourne l'ensemble des transitions interne d'un SWT
SCT : Service Web Composé Transactionnel
$\mathcal{SCT}$: ensemble de tous les SCT
$\mathcal{P}recond$: fonction qui définit une condition d'activation pour chaque transition externe d'un service composant au sein d'un SCT
$depAct(s_1, s_2)$: dépendance d'activation du service s_1 vers le service s_2
$CondAct(s)$: condition d'activation du service s
$depAlt(s_1, s_2)$: dépendance d'alternative du service s_1 vers le service s_2
$CondAlt(s)$: condition d'alternative du service s
$depAbn(s_1, s_2)$: dépendance d'abandon du service s_1 vers le service s_2
$CondAbn(s)$: condition d'abandon du service s
$depCps(s_1, s_2)$: dépendance de compensation du service s_1 vers le service s_2
$CondCps(s)$: condition de compensation du service s
$depAnl(s_1, s_2)$: dépendance d'annulation du service s_1 vers le service s_2
$CondAnl(s)$: condition d'annulation du service s
$\mathcal{FC}ontrôle$: ensemble des flots de contrôle de tous les SCT
getFContrôle : fonction qui retourne le flot de contrôle d'un SCT
$\mathcal{FT}ransactionnel$: ensemble des flots transactionnels de tous les SCT
getFTransactionnel : fonction qui retourne le flot transactionnel d'un SCT

$sc_1 \bigcup sc_2$: union de deux services composés
$\mathcal{P}(E)$: ensemble des sous ensemble de E
$potentiel_{pat}$: fonction «potentiel» du patron pat
$CondPtCps(s)$: condition potentielle de compensation de s
$CondPtAnl(s)$: condition potentielle d'annulation de s
$CondPtAlt(s)$: condition potentielle d'alternative de s
potentielle : fonction qui retourne le flot transactionnel potentiel d'un flot de contrôle
$ft1 \subseteq ft2$: le flot transactionnel $ft1$ est inclus simplement dans $ft2$
$\mathcal{ET}$: domaine des ensembles des états de terminaison de tous les SCT
$calculET$: fonction qui retourne l'ensemble des états de terminaison d'un SCT
$\mathcal{ET}_{sansEchec}$: domaine des ensembles des états de terminaison sans échec de tous les SCT
$calculET_{sansEchec}$: fonction qui retourne l'ensemble des états de terminaison sans échec d'un SCT
$\mathcal{ET}_{avecEchecs}$: domaine des ensembles des états de terminaison avec échecs de tous les SCT
$calculET_{avecEchecs}$: fonction qui retourne l'ensemble des états de terminaison avec échecs d'un SCT
$ft1 \sqsubseteq ft2$: le flot transactionnel $ft1$ est inclus fortement dans $ft2$
$calculET^{-1}_{avecEchecs}$: fonction qui retourne le flot transactionnel induit par un ensemble d'états de terminaison avec échecs
$CondCpsETA(s)$, $CondAnlETA(s)$ et $CondAltETA(s)$: condition de compensation, d'annulation et d'alternative de s induite par l'ETA
$patTrans_{pat}$: un patron transactionnel dérivé du patron de composition pat.

Bibliographie

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 574–581. IEEE Computer Society, 1996.

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert - Special Issue on Cooperative Information Systems*, 12(5), 1997.

[ACKM04] G. Alonso, F. Casati, H. K., and V. Machiraju. *Web Services Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[AFI+] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services composite application framework (ws-caf). http ://www.arjuna.com/standards/ws-caf/.

[AM97] G. Alonso and C. Mohan. Workflow Management Systems : The Next Generation of Distributed Processing Tools. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, pages 35–62. Kluwer Academic Publishers, 1997.

[ARS92] N. Ansari, L. Rusinkiewicz, and A. Sheth. Using flexible transaction to support multi-system telecommunication applications. In *Proc. of the 18th VLDB*, pages 65–76, Vancouver, Canada, 1992.

[BDG+94] A. Biliris, S. Dar, N. Gehani, H. V. Jagadish, and K. Ramamritham. Asset : a system for supporting extended transactions. *SIGMOD Rec.*, 23(2) :44–54, 1994.

[BGP04] S. Bhiri, C. Godart, and O. Perrin. A transaction-oriented framework for composing transactional web services. In *IEEE SCC*, pages 654–663, 2004.

[BGP05] S. Bhiri, C. Godart, and O. Perrin. Reliable web services composition using a transactional approach. In *EEE*, pages 15–21, 2005.

[BGPG05] S. Bhiri, K. Gaaloul, O. Perrin, and C. Godart. Overview of transactional patterns : Combining workflow flexibility and transactional reliability for composite web services. In *à paraître dans Third International Conference on Business Process Management*, Nancy, France, September 2005.

[BM03] IBM BEA and Microsoft. Business process execution language for web services (bpel4ws). 2003.

[BN83] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. In *Proceedings of the ACM Symposium on Operating System Principles*, page 3, Bretton Woods, NH, 1983. Association for Computing Machinery.

[BPG05] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *WWW*, pages 138–147, 2005.

[Bus03] C. Bussler. *B2b Integration*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[CC95] Microsoft Corporation and Digital Equipment Corporation. Object model specification. http ://www.opengroup.org/pubs/catalog/ax01.htm, October 1995.

[CLSMS+00] A. G. Cass, B. S. Lerner, Jr. S. M. Sutton, E. K. McCall, A. Wise, and L. J. Osterweil. Little-jil/juliette : a process definition language and interpreter. In *ICSE '00 : Proceedings of the 22nd international conference on Software engineering*, pages 754–757, New York, NY, USA, 2000. ACM Press.

[Coa96] WorkFlow Management Coalition. Terminology and glossary. technical report wfms-tc-1011. Technical report, Workflow Management Coalition Brussels - Belgium, 1996.

[Coa03] DAML-S Coalition. Daml-s 0.9 draft release. http ://www.daml.org/services/daml-s/0.9/, 2003.

[Cor01] Microsoft Corporation. Message queuing in windows xp. 2001.

[Cor02] Microsoft Corporation. Microsoft biztalk server 2002 enterprise edition. http ://www.microsoft.com, 2002.

[CR94] P. K. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models using ACTA. *ACM Transactions on Database Systems*, 19(3) :450–491, 1994.

[CS01] F. Casati and M. C. Shan. Models and languages for describing and discovering e-services. In *SIGMOD '01 : Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, page 626, New York, NY, USA, 2001. ACM Press.

[cXM03] cXML. cxml. http ://www.cxml.org, 2003.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 204–214. ACM Press, 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A transactional model for long-running activities. In Guy M. Lohman, Amílcar Sernadas, and Rafael Camps, editors, *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings*, pages 113–122. Morgan Kaufmann, 1991.

[DRR03] F. T. Dabous, F. A. Rabhi, and P. K. Ray. Middleware technologies for b2b integration. In *Annual Review of Communications, Vol.56 . International Engineering Consortium*, 2003.

[ea02] Cabrera et al. Web services coordination (ws-coordination). http ://www.ibm.com/developerworks/library/ws-coor/, August 2002.

[ea03] Cabrera et al. Web services atomic transaction (ws-atomictransaction). September 2003.

[ea04] Cabrera et al. Web services business activity framework (ws-businessactivity). January 2004.

[eCO03] eCO. eco. http ://eco.commerce.net, 2003.

[(Ed92] A. Elmagarmid (Ed.). *Transaction Models for Advanced Database Applications.* Morgan-Kaufmann, 1992.

[EDI] EDIFACT. United nations directories for electronic data interchange for administration, commerce and transport. http ://www.unece.org/trade/untdid/welcom.htm.

[Ell99] C. A. Ellis. *Computer Supported Cooperative Work*, chapter Workflow Technology. John Wiley and Sons, 1999.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and Marek Rusinkiewicz. A multidatabase transaction model for interbase. In *Proceedings of the sixteenth international conference on Very large databases*, pages 507–518. Morgan Kaufmann Publishers Inc., 1990.

[FBM02] D. Fensel, C. Bussler, and A. Maedche. Semantic web enabled web services. volume 2348, pages 1–2, June 2002. Invited paper.

[Fis00] L. Fischer, editor. *The Workflow Handbook 2001.* Published in association with the Workflow Management Coalition (WfMC), October 2000.

[GBG04a] W. Gaaloul, S. Bhiri, and C. Godart. Discovering workflow patterns from timed logs. In *EMISA 2004, Informationssysteme im E-Business und E-Government, Beiträge des Workshops der GI-Fachgruppe EMISA*, LNI, pages 84–94, Luxemburg, October 6-8, 2004. GI.

[GBG04b] W. Gaaloul, S. Bhiri, and C. Godart. Discovering workflow transactional behaviour from event-based log. In *12th International Conference on Cooperative Information Systems (CoopIS'04)*, LNCS, Larnaca, Cyprus, October 25-29, 2004. Springer-Verlag.

[GC02] N. Gioldasis and S. Christodoulakis. Utml : Unified transaction modeling language. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering*, pages 115–126. IEEE Computer Society, 2002.

[GCG01] Daniela Grigori, François Charoy, and Claude Godart. Anticipation to enhance flexibility of workflow execution. In *DEXA*, pages 264–273, 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[GHKM94] D. Georgakopoulos, M. F. Hornick, P. Krychniak, and F. Manola. Specification and management of extended transactions in a programmable transaction environment. In *Proceedings of the Tenth International Conference on Data Engineering, February 14-18, 1994, Houston, Texas, USA*, pages 462–473. IEEE Computer Society, 1994.

[GMGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling long-running activities as nested sagas. *IEEE Data Eng. Bull.*, 14(1) :14–18, 1991.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 249–259. ACM Press, 1987.

[Gro95] Object Management Group. The common object request broker : Architecture and specification (revision 2.0). July 1995.

[Gro97] OMG CCM Implementers Group. Corba : Common object services specification. www.omg.org/cgi-bin/doc ?ccm/2002-04-01, 1997.

[Gro99] Object Management Group. Unified modeling language specification (version 1.3). June 1999.

[Gro01] Object Management Group. Event service specification. www.omg.org/technology/documents/formal/event_service.htm, 2001.

[Gro02] Object Management Group. Corba component model tutorial. 2002.

[IBM02] IBM. Websphere mq integrator broker : Introduction and planning. June 2002.

[IBM03] IBM. Websphere. http ://www.ibm.com, 2003.

[Jac99] I. Jacobson. The unified process for component-based development. In *CAiSE '99 : Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, page 1, London, UK, 1999. Springer-Verlag.

[JB96] S. Jablonski and C. Bussler. *Workflow Management : Modeling Concepts, Architecture, and Implementation.* International Thomson Computer Press, 1996.

[KBRL04] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web services choreography description language version 1.0. 2004.

[KS95] N. Krishnakumar and A. P. Sheth. Managing Heterogeneous Multi-system Tasks to Support Enterprise-Wide Operations. *Distributed and Parallel Databases*, 3(2) :155–186, 1995.

[KT03] P. Kellert and F. Toumani. Les web services sémantiques. In *Web sémantique, Action spécifique 32 CNRS/STIC*, October 2003.

[Law97] Peter Lawrence. *Workflow handbook 1997.* John Wiley & Sons, Inc., 1997.

[Ley01] F. Leymann. Web services flow language. version 1.0. In *Technical Report, International Business Machines Corporation (IBM)*, May 2001.

[MAGK95] C. Mohan, G. Alonso, R. Günthör, and M. Kamath. Exotica : A research perspective ob workflow management systems. *IEEE Data Eng. Bull.*, 18(1) :19–26, 1995.

[MBB+03] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions : issues and enabling technologies. *The VLDB Journal*, 12(1) :59–85, 2003.

[MC] V. Miguel and F. Charoy. Bonita : Workflow cooperative system. http ://bonita.objectweb.org.

[Mica] Sun Microsystems. Enterprise javabeans technology. http ://java.sun.com/products/ejb/.

[Micb] Sun Microsystems. Java foudation classes (jfc/swing). http ://java.sun.com/products/jfc/.

[MM99] B. Meyer and C. Mingins. Component-based development : From buzz to spark. *Computer*, 32(7) :35–37, 1999.

[Mos81] E. B. Moss. Nested transactions : An approach to reliable distributed computing. Technical report, Cambridge, MA, USA, 1981.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, ii. *Inf. Comput.*, 100(1) :41–77, 1992.

[MRKS92] Sharad Mehrotra, Rajeev Rastogi, Henry F. Korth, and Abraham Silberschatz. A transaction model for multidatabase systems. In *ICDCS*, pages 56–63, 1992.

[Nie] P. Niemeyer. Beanshell-lightweight scripting for java. http ://www.beanshell.org.

[P. 02] P. F. Pires and M. Benevides and M. Mattoso. WEBTRANSACT : a Framework For Specifying And Coordinating Reliable Web Services Compositions. Technical report, sherry.ifi.unizh.ch/pires02webtransact.html, 2002.

[Pla99] D. S. Platt. *Understanding COM+*. Microsoft Press, Redmond, WA, USA, 1999.

[Pre97] W. Pree. Component-based software development-a new paradigm in software engineering ? In *APSEC*, pages 523–524, 1997.

[Ros00] RosettaNet. *Partner Interface Process (PIP) Release 1.3*. www.rosettanet.org, January 4, 2000.

[ros03] Rosettanet. http ://www.rosettanet.org, 2003.

[RS95] M. Rusinkiewicz and A. P. Sheth. Specification and execution of transactional workflows. pages 592–620, 1995.

[SABS02] H. Schuldt, G. Alonso, C. Beeri, and H. J. Schek. Atomicity and isolation for transactional processes. *ACM Trans. Database Syst.*, 27(1) :63–116, 2002.

[SKM+96] A. P. Sheth, K. Kochut, J. A. Miller, D. Worah, S. Das, C. Lin, D. Palaniswami, J. Lynch, and I. Shevchenko. Supporting state-wide immunisation tracking using multi-paradigm workflow technology. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 263–273. Morgan Kaufmann, 1996.

[SR93] A. P. Sheth and M. Rusinkiewicz. On transactional workflows. *Data Engineering Bulletin*, 16(2) :37–40, 1993.

[Sun03] Sun. Java rmi (remote method invocation). http ://java.sun.com/products/jdk/rmi, 2003.

[UDD00] UDDI.Org. *Universal Description, Discovery and Integration (UDDI) Technical White Paper*. www.uddi.org, September 2000.

[vdABtHK00] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski. Advanced Workflow Patterns. In O. Etzion and Peter Scheuermann, editors, *5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00)*, number 1901 in LNCS, pages 18–29, Eilat, Israel, September 6-8, 2000. Springer-Verlag.

[vdAvH02] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management : models, methods and tools*. Cooperative Information Systems. MIT Press, 2002.

[W3C03a] W3C. Simple object access protocol (soap). http ://www.w3.org/TR/soap, 2003.

[W3C03b] W3C. Universal description, discovery, and integration (uddi). http ://www.uddi.org, 2003.

[W3C03c] W3C. Web services description language (wsdl). http ://www.w3.org/TR/wsdl, 2003.

[Web02] WebMethods. Webmethods enterprise integrator :user's guide. 2002.

[WFM95] WFMC. *Workflow Reference Model*. WFMC (Workflow Management Coalition), www.wfmc.org, January 1995.

[WR92] H. Watcher and A. Reuter. The contract model. pages 219–263, 1992.

[WS92] G. Weikum and H. J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. 1992.

[WS97] D. Worah and A. P. Sheth. Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34. 1997.

[WWWD96] D. Wodtke, J. Weißenfels, G. Weikum, and A. Kotz Dittrich. The mentor project : Steps toward enterprise-wide workflow management. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 556–565. IEEE Computer Society, 1996.

[ZNBB94] A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 67–78. ACM Press, 1994.