# Adaptation et restructuration de modèles de processus Worflow dans un contexte inter-organisationnel

Saida Boukhedouma

UNIVERSITE DE NANTES
FACULTE DES SCIENCES ET DES TECHNIQUES

**ÉCOLE DOCTORALE STIM**
**Sciences et Techniques de l'Information et des Mathématiques**

Année 2015

# Adaptation et Restructuration de Modèles de Processus Workflow dans un Contexte Inter-Organisationnel

THÈSE DE DOCTORAT

Discipline : Informatique

Spécialité : Informatique

*Présentée par*

**BOUKHEDOUMA Saida**

*Le 02 Octobre 2015, devant le jury ci-dessous*

Rapporteurs : Dominique RIEU Professeur, Université Pierre Mendes - Grenoble
Yamine Ait-Ameur Professeur, INPT-ENSEEIHT- Toulouse
Examinateur : Mohamed Ahmed-Nacer Professeur, USTHB- Alger

*Directeur de thèse: Prof. Oussalah Mourad (Université de Nantes)*
*Co-directrice: Prof. Alimazighi Zaia (USTHB)*
*Co-encadrant: Mme Tamzalit Dalila (Université de Nantes)*

# Remerciements

**A** l'issue de ce projet de thèse, j'aimerais exprimer mes sincères remerciements aux personnes qui ont contribué à son aboutissement et son évaluation.

**J**e tiens à remercier Mme Zaia Alimazighi, professeur à l'université des sciences et technologies Houari Boumedienne (USTHB) d'Alger qui a accepté de me diriger pour la réalisation de mes travaux de thèse et qui m'a orientée vers un domaine de recherche si riche et intéressant. Je la remercie de m'avoir initiée à la recherche au sein de l'équipe ISI, du laboratoire LSI de l'USTHB depuis 15 ans déjà. Je la remercie pour sa disponibilité, son écoute, ses conseils et ses encouragements.

**J**e tiens à remercier et exprimer ma reconnaissance à M. Mourad Oussalah, professeur à l'université de Nantes pour sa grande et précieuse contribution dans l'orientation des travaux de cette thèse. Je le remercie pour l'accueil qu'il m'a toujours réservé durant mes séjours au sein du laboratoire LINA dans le cadre du projet CMEP-Tassili et autre, je le remercie pour ses orientations, ses encouragements et pour toutes les séances de travail dont j'ai bénéficié.

**J**e remercie aussi Mme Dalila Tamzalit, maître de conférences à l'université de Nantes, pour sa contribution au démarrage des travaux de cette thèse, ses orientations et sa rigueur, je la remercie aussi pour sa sympathie et ses encouragements.

**J**e tiens à remercier M. Mohamed Ahmed-Nacer, professeur à l'USTHB pour l'honneur qu'il m'a fait en acceptant d'évaluer ce travail, chaque année régulièrement depuis son démarrage et de m'avoir fait l'honneur d'examiner ce travail à la fin. Je le remercie pour sa rigueur, son esprit scientifique, ses appréciations positives, ses encouragements et sa disponibilité.

**U**n grand MERCI à Mme Dominique Rieu, professeur à l'université Pierre Mendes de Grenoble, de m'avoir fait l'honneur d'expertiser ce travail de thèse, je la remercie pour ses remarques constructives et ses appréciations que j'ai trouvées très positives et très encourageantes pour la suite.

**U**n grand MERCI également à M. Yamine Ait-Ameur, professeur à l'INPT de Toulouse, de m'avoir fait l'honneur d'expertiser ce travail de thèse, je le remercie pour ses différentes appréciations que j'ai trouvées également très positives et très encourageantes pour la présentation de ce travail.

**J**e tiens à remercier M. Pierre Cointe directeur du laboratoire LINA, de l'université de Nantes de m'avoir accueillie dans son laboratoire à chaque fois que cela était nécessaire pour l'avancement de ce travail. **J**e remercie également tout le staff administratif du LINA pour leur sympathie, leur efficacité et leurs services rendus.

**M**es remerciements vont également à mes collègues de l'USTHB pour leur aide précieuse, leur grande sympathie et leur soutien dans les moments difficiles. Je remercie infiniment tous les collègues qui ont répondu « présent » à chaque fois que j'avais besoin d'un remplacement pédagogique pendant mon séjour au LINA.

**J**e tiens à remercier les membres de ma famille pour leur compréhension, leur patience et leurs encouragements toutes les fois où l'aboutissement de ce travail devenait incertain. Qu'ils trouvent ici l'expression d'une grande reconnaissance, particulièrement ma chère mère et mon frère Hocine qui m'a accompagnée partout.

**J**e remercie Hakim, mon mari pour sa grande patience, sa compréhension et ses encouragements pour la poursuite de ce travail.

**E**nfin, je remercie mes adorables nièces Nada, Nihad et Sarah qui pour moi, sont une source d'amour, de courage et d'espoir.

## Résumé

Cette thèse s'inscrit dans le domaine de la coopération B2B, particulièrement la coopération planifiée supportée par les concepts et les outils de WorkFlow Inter-Organisationnel (WFIO). Un ensemble de schémas de coopération identifiés dans la littérature constitue notre point de départ dans ce travail de thèse. Ceux-ci définissent différents modes de partitionnement, de contrôle d'exécution et d'interaction pour l'interconnexion de processus WF issus de différentes organisations. Néanmoins, leur inconvénient majeur est leur rigidité et donc la difficulté d'adapter les modèles de processus WFIO supportés, pouvant être sujets à des changements occasionnels ou fréquents.

De ce fait, nous nous sommes focalisés sur deux problématiques étroitement liées : (i) la définition de nouveaux schémas de coopération équivalents à ceux proposés dans la littérature avec un degré de flexibilité supplémentaire, visant à supporter aisément les changements sur les modèles de WFIO, (ii) la prise en charge de différents aspects de flexibilité de ces modèles.

Pour répondre à la première problématique, nous avons proposé une démarche de restructuration et d'interconnexion de modèles de WF basée sur le paradigme SOA (Architecture Orientée Service); nous avons introduit un nouveau concept appelé *Patron de Coopération à Base de Services* (PCBS) permettant de caractériser un modèle de processus WFIO à base de services selon trois dimensions principales. Aussi, nous avons procédé à une conceptualisation des différents patrons de coopération proposés et une formalisation de chancun d'eux en définissant un opérateur de coopération approprié.

Quant à la flexibilité des modèles de WFIO obéissant aux nouveaux patrons de coopération définis, nous la percevons sous trois aspects complémentaires : adaptabilité, évolutivité et réutilisabilité. A cet effet, nous avons proposé et formalisé à l'aide d'opérateurs dédiés, un ensemble de patrons d'adaptation. Pour les aspects d'évolution et de réutilisation de modèles, nous avons introduit les concepts de patron de coopération *généralisé* et patron de coopération *composite* qui décrivent un aspect de réutilisation des modèles de WFIO, pour la construction de nouveaux modèles de WFIO plus complexes.



## Mots clés

*Coopération B2B, Workflow Inter-organisationnel (WFIO), SOA, Service, Restructuration, Interconnexion, Patron de coopération à base de services (PCBS), Patron d'adaptation, Patron de coopération généralisé, Patron de coopération composite, Flexibilité, Adaptation, Evolution, Réutilisation.*

## Abstract

The work presented in this thesis is related to the field of B2B cooperation; it deals particularly with planned cooperation that is supported by the concepts and tools of Inter-Organizational WorkFlow (IOWF). We start our work from a set of cooperation schemas (IOWF-architectures) identified in the litterature, which express different partinoning, interaction and control modes between WF fragments provided by several business partners. However, in their initial form, these architectures suffer from rigidity inducing a difficulty to adapt IOWF processes to changes which can occur occasionally or frequently.

From that, we focus on two issues closely linked to each other: (i) the definition of new cooperation schemas which are equivalent to the ones initially defined, but with an additional degree of flexibility to easily support changes on the IOWF process models supported. (ii) The support of different aspects of flexibility on the IOWF models newly defined.

For the first issue, we have proposed a process of restructuring and interconnecting WF models based on the SOA (Service Oriented Architecture) paradigm; we have introduced a new concept called *Service Based Cooperation Pattern* (SBCP) that characterizes a given IOWF architecture based on services through three main dimensions. Then, we proceed to the conceptualization of the different cooperation patterns proposed and their formalization by defining a set of cooperation operators.

Regarding the second issue of our work which is the *flexibility* support of IOWF models obeying to the SBCP proposed, we define flexibility of IOWF process models around three complementary aspects: *adaptability*, *evolutivity* and *reusability*. So, for adaptation, we have proposed and formalized a set of adaptation patterns and for evolution and reuse aspects, we have introduced the concepts of *generalized* cooperation patterns and *composite* cooperation patterns which define a manner to build more complex IOWF models by reusing existing ones.

## Key words

*B2B Cooperation, Inter-organisationnel Workflow (IOWF), SOA, Service, Restructuration, Interconnection, Service Based Cooperation Pattern (SBCP), Adaptation Pattern, Generalized cooperation pattern, Composite cooperation pattern, Flexibility, Adaptability, Evolutivity, Reusability.*

# Table des Matières

## Partie II : Contributions et Bilan

# Liste des Figures

# Liste des Tableaux

# CHAPITRE I

# Introduction et Contexte

## Introduction

L'évolution croissante et rapide des technologies de l'information et de la communication (TIC) durant les deux dernières décennies, s'est directement reflétée sur l'évolution des méthodes de travail dans les organisations, induisant de suite des changements considérables dans les procédures de travail internes et dans les relations avec l'environnement extérieur. De plus les facteurs économiques dus à la mondialisation et la globalisation ont fortement agit sur l'instabilité des entreprises et leurs processus métiers. Aussi, toute entreprise voulant rester compétitive dans son domaine d'activité s'est tenue d'une part à suivre les évolutions technologiques et d'autre part à revoir continuellement ses processus métiers, afin de les adapter aux changements imposés par l'environnement et par les exigences de ses clients potentiels en termes de délai, qualité et coût.

Cette évolution croissante dans le milieu des entreprises a donné naissance à de nouvelles formes d'organisation impliquant plusieurs entreprises où la relation entre elles peut varier d'une dépendance totale ou partielle (fusions-acquisitions, prise de participation, *etc.*) à des accords ponctuels sous forme contractuelle (Boukadi, 2009). Des espaces coopératifs dans lesquels les entreprises travaillent et réagissent ensemble ont alors émergé sous diverses formes : entreprise virtuelle, réseau d'entreprises, entreprise réseau, *etc.* La constitution et la gestion de ce type d'organisation s'appuient souvent sur des plateformes techniques et de partage d'information qui jouent un rôle de support et de facilitation de la coopération (Grefen et al., 2009). On parle alors de la coopération B2B (Business to Business), soutenue par des technologies et des infrastructures permettant une circulation de l'information entre partenaires distants.

Le concept B2B a été initialement supporté par les outils de Workflow, notamment le WF inter-organisationnel (Van der Aalst, 1999), (Van der Aalst, 2000) et plus récemment promu par l'apparition des architectures orientées services (SOA) (Brown et al., 2002), (Papazoglou et al., 2007) et les services Web (Alonzo et al., 2004) qui constituent l'instanciation la plus importante de la SOA, dans le domaine des processus métiers.

Dans le cadre de cette thèse, nous nous intéressons à la coopération inter-organisationnelle donnant lieu à une relation durable pour la réalisation de projets à long terme, ce type de coopération appelé *coopération planifiée* se trouve répandu dans le domaine de la « supply-chain management » et les « consortiums d'entreprises ». Il s'agit d'un groupement de partenaires implémentant chacun son processus métier et gérant ses propres ressources de manière autonome, mais qui se met d'accord avec d'autres partenaires pour coordonner les différents fragments de processus afin de répondre à un objectif métier commun, dans le cadre d'une politique « gagnant-gagnant ». Le présent travail consiste à combiner les concepts et outils du Workflow et du paradigme SOA en vue de supporter l'interconnexion entre processus métiers d'organisations différentes. Nous mettons particulièrement en avant, la contrainte de *flexibilité des modèles* de processus métiers inter-organisationnels obéissant aux patrons de coopération à base de services que nous définissons. Pour cela, nous proposons des mécanismes permettant de supporter les différents changements susceptibles d'affecter ces modèles tant au niveau intra-organisationnel qu'au niveau inter-organisationnel.

## I.1 La coopération B2B : Enjeux, principes et défis

La notion de coopération présente une certaine proximité et ressemblances avec d'autres notions comme la coordination, la sous-traitance et surtout la collaboration. Selon (Benali, 2005), la coopération est définie comme suit: "*Travailler ensemble par le biais de la coopération, de la collaboration ou de tout autre moyen, signifie se mettre en groupe et former une organisation particulière à court, moyen ou long terme. Cette organisation facilitera les échanges et la*

*circulation des flux de tout genre, accroîtra la synergie, et permettra d'instaurer un climat de confiance entre les partenaires".*

La coopération est désormais un véritable enjeu pour les entreprises puisqu'elle est devenue une source de valeur ajoutée et constitue dans certains cas, un avantage concurrentiel et un défi dans le fonctionnement des entreprises (Chaari, 2008). Des études comme celles présentées dans (Seong-Jong et al., 2010), (Bigne-Alcaniz et al., 2009) menées sur des échantillons d'entreprises affirment que beaucoup d'organisations envisagent de continuer à coopérer si une coopération est déjà établie, une bonne partie d'autres organisations envisagent de bâtir des relations de partenariat, notamment des externalisations de services. Le gain de toute entreprise est de se concentrer sur le cœur de son métier mais le principal défi c'est le coût de l'externalisation d'une manière particulière et plus généralement le coût de la coopération.

Par ailleurs, dans un contexte de concurrence accrue et d'ouverture sur le marché mondial, la coopération B2B reste un choix stratégique et souvent inévitable. En effet, les entreprises se trouvent confrontées à de nouveaux défis dont l'amélioration de leur productivité, leur rentabilité et leur réactivité face aux exigences de leurs clients potentiels, qui deviennent de plus en plus difficiles à satisfaire avec les contraintes : réduction de coût, réduction des délais et amélioration de la qualité. Face à ces nouveaux défis, un bon nombre d'entreprises ont compris l'importance de la coopération et se sont organisées an groupes afin de répondre à un objectif métier global tenant compte des trois paramètres imposés (coût, délai et qualité). Cette nouvelle forme d'entreprise exige l'interconnexion des processus métiers de plusieurs entreprises donnant lieu à un processus métier inter-organisationnel.

### I.1.1 Processus métier inter-organisationnel

Un processus métier inter-organisationnel est un nouveau processus métier construit à partir de plusieurs processus d'entreprises interconnectés selon des contraintes et des règles d'interconnexion bien précises. Dans notre cas, nous définissons un processus métier inter-organisationnel comme suit :

- **Définition I.1- Processus métier inter-organisationnel**

*Un processus métier inter-organisationnel est un processus métier qui implique au moins deux partenaires, il est soutenu par un modèle qui définit les règles d'interaction, d'échange de données et de contrôle d'exécution entre les fragments de processus fournis par les entreprises partenaires* ».

Les règles d'interaction définissent le mode de communication (synchrone ou asynchrone), le contrôle d'exécution (centralisé, décentralisé, hiérarchisé ou mixte) définit le rôle de chaque système partenaire dans l'exécution d'une instance de processus.

Il faut noter que la gestion de ces processus métiers inclut les concepts, les méthodes et les techniques nécessaires pour leur conception, administration, configuration, exécution et analyse (Weske, 2007).

### I.1.2 Coopération planifiée versus coopération dynamique

Dans le cadre de la coopération B2B et d'un point de vue stratégique, deux types de coopération ont émergé : la coopération *planifiée* et la coopération *dynamique*.
La coopération planifiée (désignée aussi sous le nom de coopération *statique*) suppose une connaissance à priori des partenaires impliqués dans la coopération, elle convient aux relations de coopération durables, particulièrement adaptées à la réalisation de projets importants et de longue durée. Dans ce cas les partenaires se mettent d'accord dès la phase de conception sur un schéma de coopération (règles d'interaction, contrôle d'exécution,…) clairement défini. Parmi, les travaux

les plus répandus dans ce domaine, nous pouvons citer le projet ACE-Flow (Stricker, 2000), l'approche WISE (Lazcano, 2000) et l'approche « Point de synchronisation » (Perrin et al., 2004).

Dans une coopération dynamique, les partenaires ne sont pas connus à priori, ils sont découverts dans un registre de publication où ils publient leurs offres de services (processus), avec un degré d'abstraction leur permettant de préserver leur autonomie et leur savoir-faire. Avant l'établissement d'un quelconque contrat, une phase de négociation est parfois nécessaire afin de se mettre d'accord sur les modalités de coopération. Ce type de coopération est plus adéquat dans le cas de relations occasionnelles pour la réalisation de projets de durée limitée dans le temps. Beaucoup de travaux se sont penchés sur ce type de coopération, parmi lesquels nous pouvons citer les systèmes e-Flow (Casati et al., 2001), CrossFlow (Grefen et al., 2001), CrossWork (Mehandjiev et al., 2005), CoopFlow (Chebbi, 2007) et l'approche CSOMA (Boukadi, 2009). Le tableau I.1 met en évidence les principales différences existant entre les deux types de coopération selon des critères liés à la nature de la coopération et aux processus métiers impliqués dans la coopération.

**TAB I.1.** *Coopération planifiée versus coopération dynamique*

| *Type de coopération* / *Critère* | ***Coopération planifiée*** | ***Coopération dynamique*** |
|---|---|---|
| ***Partenaires*** | Connus a priori | Recherchés dynamiquement dans un annuaire |
| ***Modèles de processus locaux*** | Définis a priori | Définis a priori |
| ***Possibilité de changement dans les processus locaux*** | Oui | Non |
| ***Schéma de coopération*** | Défini a priori | Construit dynamiquement selon les objectifs et les offres |
| ***Durée de la coopération*** | Durable | Occasionnelle |
| ***Domaines d'application*** | Projets à long et moyen terme et à grande valeur ajoutée | Projets à court terme |

### I.1.3 Coopération structurée versus coopération ad-hoc

Du point de vue de la structure du processus métier inter-organisationnel, deux types de coopération peuvent être distinguées : la coopération *structurée* et la coopération *ad-hoc*.
La coopération structurée (Kiepuszewski et al., 2000), (Eder et al., 2002) repose sur un schéma de processus inter-organisationnel clairement défini dès la phase de conception, c'est-à-dire que le flux des activités ainsi que les interactions entre les processus des différents partenaires sont clairement définis. Aussi, les instances de processus s'exécutent toutes conformément au modèle de processus défini.

Contrairement à la coopération structurée, dans une coopération ad-hoc (Voorhoeve et al., 97), le schéma de processus n'est pas complètement défini dès la phase de conception, il peut l'être partiellement et certaines parties du processus ne seront définies qu'à la phase d'exécution (au niveau Runtime, à la volée), on parle souvent de modélisation ad-hoc ou tardive (Saikali, 2001). Ce type de modélisation est souvent nécessaire pour la prise en compte d'exécutions exceptionnelles qui dépendent d'événements externes ou du résultat d'exécution d'une partie du processus. Ainsi, d'une exécution à l'autre, les instances de processus ne suivent pas forcément le même modèle. Beaucoup de travaux se sont focalisés sur la modélisation ad-hoc ou tardive

(Voorhoeve et al., 97), (Meng et al., 2000), (Anzböck et al., 2004), (Polyvyanny et al., 2009), un problème qui prend toute son ampleur dans un environnement métier en perpétuel changement soumis à des événements imprévisibles. Notons que la coopération structurée est mieux adaptée dans le cas d'une coopération planifiée car le schéma de coopération et les règles d'interaction sont clairement définis dès la phase de conception alors que la coopération ad-hoc s'avère plus appropriée dans le cas d'une coopération dynamique. En effet, dans ce dernier cas, les partenaires métier peuvent participer à des coopérations différentes et de différentes manières, les changements de partenariats sont fréquents et induisent de suite, une instabilité au niveau du modèle de processus inter-organisationnel.

Bien que la coopération B2B devienne un choix stratégique et inévitable pour l'entreprise, la prise d'une telle décision demeure difficile. En effet, malgré les évolutions technologiques aidant à mettre en place des solutions techniques de plus en plus adaptées aux concepts de l'inter-organisationnel, la mise en place d'une solution B2B durable et efficace est soumise à un ensemble de contraintes dont il faut tenir compte.

## I.2 Contraintes de la coopération B2B

L'interconnexion des processus métiers des entreprises dépend fortement de la capacité des systèmes d'information à participer à un scénario de coopération ; cette capacité se définit par deux caractéristiques essentielles : la flexibilité (appelée aussi agilité) du système d'information (SI) et la capacité d'inter-opérer avec un système d'information extérieur. Les contraintes de la coopération inter-organisationnelle se résument dans le manque de flexibilité du SI, la réticence à l'ouverture, la nécessité de préserver l'autonomie et le savoir-faire de chaque organisation et la nécessité de maintenir la confidentialité sur les données et les processus locaux.

### I.2.1 Contrainte de flexibilité du SI

La flexibilité constitue depuis plusieurs années et jusqu'à l'heure actuelle, une préoccupation majeure des entreprises voulant rester réactives et efficaces dans un environnement métier en perpétuelle évolution. Ce dernier est caractérisé principalement par l'augmentation de la concurrence sur des aspects multiples, simultanés et potentiellement contradictoires tels que la diversité, la quantité, les coûts, les délais et la qualité (Sarkis, 2001), (Pal et al., 2005). Dans un tel environnement, les besoins des entreprises sont sujets à des changements assez fréquents. La flexibilité devient donc un critère essentiel permettant de gagner en compétitivité en s'adaptant le plus rapidement possible aux changements du marché (Baouab, 2013). La flexibilité (ou agilité) désigne la capacité de l'entreprise à agir et à répondre aux changements d'une manière rapide et efficace dans un environnement instable et imprévisible (McCoy et al., 2006). Le système d'information qui est au cœur du fonctionnement de l'entreprise doit développer une capacité à :

- S'aligner avec les nouvelles exigences métier,
- Envisager et supporter des scénarios d'évolution et des demandes de modification,
- Répondre efficacement (i.e rapport temps de réponse/coût) aux demandes d'adhésion à des scénarios de coopération.

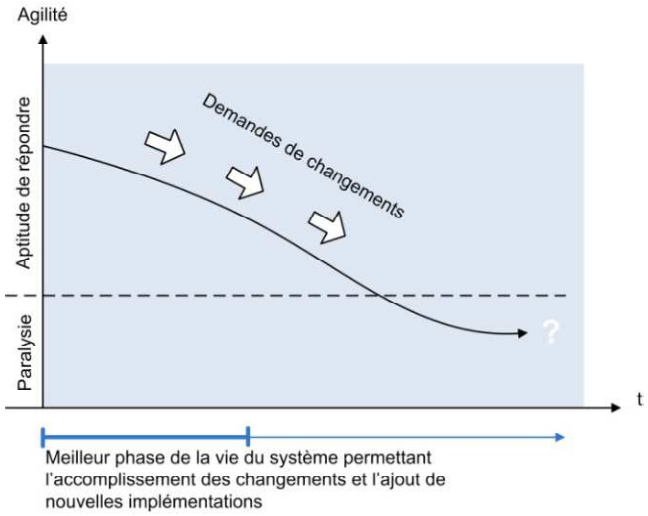



**Figure I.1.** *Diminution de l'agilité d'un SI avec le temps* (Krafzig et al., 2004)

Cependant, un bon nombre d'entreprises se trouvent freinées dans l'établissement d'une coopération à cause des architectures de leurs systèmes d'information déjà mis en place. En effet, en examinant le schéma d'évolution d'un SI typique d'une entreprise (voir Figure I.1), on se rend compte que son agilité diminue avec le temps puisqu'il présente au début une phase initiale durant laquelle les demandes de changements sont accomplies, d'une manière relativement rapide et efficace. Toutefois, après un certain seuil, la capacité du SI à accueillir de nouveaux changements et implémentations devient très limitée et la maintenance du système devient une tâche extrêmement difficile et coûteuse. L'absence de solution architecturale efficace pour résoudre ce problème a plongé les SI dans une situation de blocage vis-à-vis des nouvelles exigences métier (Chaari, 2008). L'enjeu consiste donc à rendre le système d'information le plus réactif possible aux évolutions du métier tout en préservant le patrimoine informationnel de l'entreprise. En d'autres termes, il s'agit de donner à l'entreprise l'agilité nécessaire pour évoluer dans un environnement économique concurrentiel et dynamique (Giachetti et al., 2003), (Sharifi et al., 2000), (Yusuf et al., 2003), (Chen et al., 2009a).

### I.2.2 Réticence à la coopération

Les SI des organisations ont été initialement développés pour fonctionner comme une entité à part entière, où l'interaction avec l'extérieur est régie par des formes d'échanges d'information bien définies et fixes. Par conséquent, dans une perspective de coopération, l'interopérabilité entre des SI hétérogènes est devenue un problème qui a suscité l'intérêt d'un bon nombre de chercheurs ; des solutions à ce problème ont été proposées dans (Izza, 2006), (Touzi, 2007), (Berre et al., 2007), par exemple. Mais, la coopération ne se limite pas à un problème d'interopérabilité, les partenaires métiers sont souvent amenés à revoir leurs processus afin de les adapter aux besoins de la coopération. Ceci peut induire dans certains cas, des coûts de changements considérables et donc, une véritable réticence à coopérer. Enfin, cette contrainte est fortement liée à la contrainte d'agilité ou de flexibilité du SI.

### I.2.3 Contraintes liées à l'interconnexion des processus métiers

La mise en place d'un processus métier inter-organisationnel commence par le choix des partenaires. Les entreprises peuvent proposer leurs ressources ou les résultats de leurs processus métiers comme offres. De plus, ces entreprises doivent spécifier les contraintes, les conditions et les moyens nécessaires qui définissent l'accès à leurs processus et ressources. Dans le cas d'une coopération planifiée, les partenaires sont connus à priori et peuvent s'entendre sur les modalités

de coopération dès la phase de conception du processus inter-organisationnel. Tout changement nécessaire à la coopération au niveau d'un processus local peut être pris en charge par le partenaire qui en est responsable. Dans une coopération dynamique, la question est plus complexe car les entreprises ne sont pas prêtes à apporter des changements au niveau de leur processus locaux, afin de réaliser une coopération donnée et les annuler à la fin de la coopération pour en adopter d'autres. Les changements dans ce cas doivent être dynamiques et pris en charge par le système de gestion de manière autonome et les processus métiers doivent être adaptatifs (s'auto-adapter) (Reichert et al., 98), (Han et al., 98).

Dans tous les cas, toute organisation impliquée dans une coopération, souhaite préserver son *autonomie* et garantir la *confidentialité* de ses données et de sa logique métier ; ces deux points constituent deux contraintes majeures pour l'établissement d'une coopération.

- **Autonomie et Confidentialité**

L'autonomie est une caractéristique bien identifiée de la coopération B2B qui en constitue aussi une contrainte. En effet, cette caractéristique exige que chaque organisation participant à un processus métier inter-organisationnel puisse décider par elle-même, les conditions de la coopération c'est-à-dire quand, comment et avec qui elle coopère. De plus, chaque organisation souhaite préserver sa propre logique métier et le contrôle sur les tâches qu'elle réalise localement. De ce fait, l'interconnexion de processus doit assurer une cohérence du processus global sans pour autant imposer des contraintes particulières sur le déroulement des fragments de processus privés de chaque partenaire.

Bien que les entreprises soient conscientes du besoin et de la nécessité de participer à une coopération, elles souhaitent protéger leurs savoir-faire afin d'assurer une confidentialité suffisante sur leurs données locales et leurs processus locaux. Par conséquent, les processus des partenaires doivent être décrits avec un niveau d'abstraction élevé leur permettant de dévoiler uniquement le strict minimum de leur logique métier qui soit nécessaire à l'interconnexion avec d'autres processus. De plus, chacun des partenaires devra recevoir uniquement les données nécessaires au bon déroulement du fragment de processus dont il est responsable.

## I.3 Le paradigme SOA

Partant des contraintes énoncées dans la section précédente, le principal challenge des entreprises durant cette dernière décennie, est donc de construire un SI d'entreprise capable à la fois d'assurer l'agilité de l'entreprise et de favoriser et faciliter l'interconnexion des processus d'entreprises, tout en préservant leur autonomie et la confidentialité sur leurs processus locaux et leurs données. Le paradigme SOA (Service Oriented Architecture) apparu au début des années 2000, semble apporter une solution intéressante à ce challenge.

### I.3.1 Principe de la SOA

L'architecture orientée services (SOA : *Service Oriented Architecture*) est un nouveau style architectural qui s'est rapidement répandu suite à son succès grandissant, et a été largement accepté comme une architecture support du système d'information de l'entreprise grâce à son concept pivot qui est le *service*. Ce dernier constitue la pierre angulaire d'une SOA et se définit d'après (Papazoglou, 2003), comme suit:

> Un service est *un ensemble d'applications modulaires auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le Web.*

Un service peut effectuer des actions allant de simples requêtes à des processus métiers complexes. Les services permettent d'intégrer des systèmes d'information hétérogènes en utilisant des protocoles et des formats de données standardisés, autorisant ainsi un faible couplage et une grande souplesse vis-à-vis des choix technologiques effectués (Papazoglou et al., 2003).

Un service est un composant logiciel représenté par deux éléments séparés : son interface qui permet de définir les modalités d'accès au service (nom du service et les paramètres des opérations publiques définissant les signatures des opérations) et son implémentation. Les services sont interconnectés via l'échange de messages.

Le concept de service attire de plus en plus l'attention dans le développement économique, aussi bien dans les entreprises de services traditionnelles, que dans les entreprises manufacturières et les prestataires publics de services. On parle souvent de *l'économie de service* dans laquelle les entreprises offrent leurs produits sous forme de services. Particulièrement, les *services Web* (Alonzo et al., 2004) ont émergé comme l'instanciation la plus importante du modèle SOA dans le domaine industriel (Teway et al., 2013). En effet, les entreprises encapsulent les tâches automatiques comme des services logiciels pour les rendre visibles sur Internet. Les services Web sont accessibles à l'intérieur et à l'extérieur d'une entreprise. Par conséquent, ils permettent à l'entreprise d'intégrer ses applications hétérogènes ainsi que ceux des entreprises partenaires, indépendamment des environnements techniques et des langages sur lesquels tournent ces applications. Cette spécificité présente un atout considérable, favorisant ainsi la mise en place des coopérations B2B.

### I.3.2 Avantages de la SOA pour la coopération B2B

Dans une perspective d'intégration ou d'interopérabilité de modèles de processus, les entreprises cherchent en premier lieu, à réduire leurs coûts d'intégration, s'ouvrir à leur environnement, gagner en flexibilité, afin de s'adapter facilement face à l'évolution du marché et réagir rapidement aux changements pouvant affecter leurs processus métiers. La SOA offre beaucoup d'avantages dans ce contexte. En effet, la SOA permet à un système de *gagner en flexibilité* puisqu'elle facilite l'interaction entre processus métiers. Elle permet d'interconnecter des services en se basant sur leurs interfaces indépendamment de leurs implémentations. SOA est une architecture ouverte, basée sur un ensemble de standards industriels pour le développement, le déploiement et l'intégration d'applications Internet, de ce fait elle facilite *l'interopérabilité* entre systèmes hétérogènes.

De plus, la SOA permet *l'urbanisation du SI* puisque ce dernier est découpé en services. La SOA permet une organisation (ou réorganisation) et une structuration efficaces du SI, en considérant les multiples composants d'un SI comme des services d'entreprise. Grâce à cette urbanisation orientée service, la SOA garantit l'*uniformisation* des composants du système tous représentés sous forme de services. Ce qui facilite leur intégration et leur interopérabilité (Chebbi, 2007).

Grâce à la principale caractéristique de couplage faible des services, une SOA permet *une intégration à faible couplage et à faible coût* de systèmes hétérogènes. Dans le domaine du Workflow ou dans un contexte B2B, la SOA contribue à intégrer des processus interentreprises, en permettant à différents logiciels de communiquer entre eux, en utilisant un mécanisme de composition (orchestration ou de chorégraphie) de services. Cette intégration à l'aide de technologies classiques s'avère coûteuse puisqu'elle nécessite la conception de ponts logiciels ad-hoc coûteux et à fort couplage. De plus, l'entretien de telles solutions nécessite une forte cohésion entre les différents acteurs impliqués dans l'intégration.

Enfin, la SOA contribue à *l'amélioration de l'agilité du SI* puisqu'elle permet de structurer d'une manière dynamique un système d'information. Cette dynamique est liée à la possibilité de changer facilement une configuration de services (ajouter ou enlever un service, modifier l'ordre de communication des services,… etc.). De plus, il est possible d'intervenir sur le contenu de service (changement interne) sans toucher à son interface d'accès (Touzi, 2007).

## I.4 Point de départ et état de la question

Notre cadre de travail est lié à la coopération planifiée dans laquelle les partenaires sont connus à priori, ce type de coopération est très répandu dans les relations de partenariat de longue durée (alliances interentreprises, consortium d'entreprises, …) et sont plus réalistes et plus adéquates pour la réalisation de grands projets dans divers domaines tels que l'industrie auto-motive, la production pharmaceutique/parapharmaceutique, l'industrie électronique (production de circuits intégrés), … etc. Dans ce type de coopération, l'interconnexion entre processus métiers est réalisée selon un schéma de coopération bien défini, en fonction des offres de chaque partenaire dépendant de leurs compétences respectives et de l'objectif métier global à atteindre.

Ce type de coopération a été initialement supporté par les outils de WF inter-organisationnels (WFIO), des schémas génériques de coopération (appelés aussi architectures de WFIO) ont été clairement identifiés dans la littérature (Van der Aalst, 99), (Van der Aalst, 2000) et reconnus comme des modèles de coopération assez répandus dans le domaine du B2B, utilisés à l'origine dans les applications de e-commerce. Il s'agit du « Partage de charge », l'«Exécution chainée », la « Sous-traitance », le « Transfert de cas », le « Transfert de cas étendu » et le WF « Faiblement couplé ». Ces schémas de coopération définissent chacun, une architecture de WFIO caractérisée par trois dimensions principales : le partitionnement du processus, le contrôle d'exécution (centralisé, décentralisé, hiérarchisé ou mixte) et la structure d'interaction (synchrone ou asynchrone) entre les processus impliqués dans la coopération. Ils s'adaptent parfaitement à une coopération planifiée et plus ou moins à une coopération dynamique. Ces schémas de coopération ont été largement adoptés dans les premières plateformes de coopération pour la réalisation des projets tels que le projet NIIP, le projet TEAM, le projet ACE-Flow (Striker, 2000), CrossFlow (Grefen et al., 2001) et dans les approches Wise (Lazcano, 2000), Crosswork (Mehandjiev et al., 2005), le point de synchronisation (Perrin et al., 2004), … etc.

**Hypothèse :** *Les schémas de coopération de Van Der Aalst définissent les différentes manières avec lesquelles les partenaires métiers peuvent communiquer, coopérer et coordonner leurs processus et couvrent un large éventail de processus métiers existants, donc nous les considérons comme des schémas génériques de coopération qui constituent le point de départ de notre travail de thèse.*

Ces schémas génériques présentent l'avantage de couvrir un grand nombre de processus métiers existants puisqu'ils implémentent différents scénarios de coopération. Toutefois, leur principal inconvénient réside dans la rigidité des modèles de processus WFIO supportés impliquant de suite, une difficulté d'adaptation au changement. Ainsi, toute adaptation s'avère coûteuse surtout si elle affecte le schéma de coopération ou la structure d'interaction, nécessitant d'apporter des modifications au niveau des différents processus impliqués dans la coopération. De là, nous avons jugé nécessaire de nous pencher sur la question de redéfinition de ces schémas de coopération de manière à leur apporter un degré de flexibilité supplémentaire et donc obtenir des modèles de processus WFIO aisément adaptables. De ce fait, notre premier objectif est de proposer des schémas de coopération équivalents à ceux proposés dans (Van der Aalst, 99), (Van der Aalst,

2000) en mettant en avant la contrainte de *flexibilité des modèles* de processus WFIO supportés par ces nouveaux schémas de coopération.

**Objectif 1** : *Proposer des schémas de coopération équivalents à ceux proposés dans (Van der Aalst, 99), (Van der Aalst, 2000) en mettant en avant la contrainte de flexibilité des modèles de processus WFIO supportés par ces nouveaux schémas de coopération.*

La contrainte de flexibilité des modèles de processus constitue notre principale problématique. En effet, une fois le processus inter-organisationnel mis en place, rien ne garantit sa stabilité et sa durabilité dans un environnement instable où les lois organisationnelles peuvent changer occasionnellement ou plus ou moins fréquemment, des disfonctionnement peuvent surgir, des ruptures de contrats entre partenaires, les exigences des clients et la concurrence du marché ne cessent d'augmenter. Face à cela, les entreprises en coopération se trouvent dans des situations embarrassantes qui les conduisent forcément, à revoir continuellement leurs processus internes et en interaction avec d'autres processus, afin d'apporter les modifications nécessaires, à moindre coût et sans perturber la cohérence du processus global. Ces changements peuvent aller d'une simple modification dans un fragment de processus local, à une modification de la structure d'interaction entre deux processus interconnectés, au remplacement d'un fragment de processus par un autre et jusqu'à l'évolution d'un processus inter-organisationnel pour impliquer de nouveaux partenaires. A partir de là, notre deuxième objectif est d'offrir des mécanismes d'adaptation, d'évolution et de réutilisation de modèles de processus WFIO obéissant aux schémas de coopération nouvellement définis.

**Objectif 2 :** *Offrir des mécanismes d'adaptation, d'évolution et de réutilisation de modèles de processus WFIO obéissant aux schémas de coopération nouvellement définis.*

## I.5 La flexibilité des modèles de processus WFIO

Nous percevons la flexibilité des modèles de WFIO sous trois aspects complémentaires : adaptabilité, évolutivité et réutilisabilité et que nous définissons intuitivement comme suit :

L'*adaptabilité d'un modèle définit sa capacité à supporter aisément des modifications, ajouts ou suppressions au niveau des entités qui le composent sans altérer la cohérence et le fonctionnement global du processus et sans impact sur l'aspect coopération.*

*L'évolutivité d'un modèle de WFIO désigne sa capacité à supporter une expansion de sa fonctionnalité globale et/ou de sa coopération (ouverture du processus métier à de nouveaux partenaires). Notons que les deux perspectives ne sont pas exclusives, l'expansion de la coopération induit forcément une expansion de la fonctionnalité globale.*

*La réutilisabilité d'un modèle de WFIO définit sa capacité à être intégré à d'autres modèles et manipulé comme une entité à part entière, afin de construire des modèles de WFIO plus complexes.*

## I.6 Problématiques

A travers les points cités dans l'état de la question, nous pouvons dégager les questions suivantes : comment structurer (ou restructurer) les processus métiers de manière à faciliter leur intégration avec d'autres processus? Comment obtenir un processus métier inter-organisationnel obéissant à un schéma de coopération bien défini? Quels sont les types de changements intra et inter-organisationnels susceptibles d'affecter la structure d'un processus métier? Quels sont les

mécanismes adéquats pour la prise en charge de ces changements? Quels mécanismes pour l'évolution et la réutilisation de modèles de processus métiers inter-organisationnels ?

A partir de là, les sous-problématiques suivantes sont dégagées :

(1) Construction de processus métiers inter-organisationnels suffisamment flexibles et obéissant à des schémas de coopération bien identifiés.

(2) Adaptation des processus métiers inter-organisationnels, prenant en compte les changements internes à une entreprise et les changements interentreprises, notamment au niveau de la structure d'interaction, tout en maintenant la cohérence du processus global.

(3) Evolution des processus métiers inter-organisationnels dans une perspective d'ouverture vers d'autres partenaires éventuellement, sans perturber la cohérence du processus WFIO déjà existant.

(4) Réutilisation de modèles de processus métiers inter-organisationnels pour la construction de nouveaux modèles de processus plus complexes et à forte valeur ajoutée.

## I.7 Approche globale et Contributions

Compte tenu des principales caractéristiques de l'approche SOA et ses avantages d'interopérabilité, de flexibilité et de réutilisabilité, particulièrement bénéfiques pour les processus métiers inter-organisationnels, nous avons adopté une approche *à base de services* pour réaliser l'interconnexion de processus. Les gains de cette approche sont évidents pour la flexibilité des processus métiers, nous en citons les principaux dans ce qui suit :

### I.7.1 Apports du paradigme SOA pour la flexibilité des processus métiers

- **Adaptabilité** : la SOA offre un meilleur développement et une meilleure adaptabilité des processus métiers grâce à la possibilité d'implémenter ou de modifier les processus plus rapidement et à un moindre coût. En effet, les services sont fournis avec un niveau d'abstraction élevé, seules leurs interfaces sont visibles de l'extérieur, leur utilisation dans une application métier est basée sur des interactions faiblement couplées et ne dépend pas de leur implémentation interne. De ce fait, il est plus facile de substituer un service par un autre, ajouter ou supprimer un service, … etc.
- **Evolutivité** : l'encapsulation de la complexité des processus métiers dans des services plus réduits simplifie l'évolution des services eux-mêmes et l'évolution des processus qui les utilisent, ce qui induit donc une maintenabilité et une évolutivité relativement aisée.
- **Réutilisabilité** : SOA permet de favoriser la mutualisation et la réutilisation par variantes de services. Il est donc plus facile de réutiliser un service puisqu'il n'est pas directement lié à d'autres services. De plus, la composition de services est un mécanisme qui permet de combiner les services fournis par les partenaires pour offrir d'autres services à forte valeur ajoutée.

Notons que l'utilisation des services pour la définition et l'interconnexion des processus métiers n'est pas nouvelle, beaucoup de travaux ont adopté une approche à base de services dans le cadre des coopérations interentreprises, notamment dans un contexte de coopération dynamique. Aussi, beaucoup d'approches de coopération se sont basées sur la composition de services vue souvent comme une orchestration de services (ensemble de services orchestrés à l'aide d'un coordinateur central) (Belhajjame et al., 2005), (Mayer et al., 2008), (Chaari, 2008), (Boukadi, 2009),(Esper, 2010), (Heorhi, 2012), (Baina et al., 2006), (Casati et al., 2001); d'autres approches se sont focalisées sur la chorégraphie de services, implémentant un contrôle décentralisé des services (Charif, 2007), (Baouab, 2013), (Ahmed et al., 2014).

Nos contributions dans le cadre de cette thèse, se divisent en deux principaux volets : (i) proposer une approche de restructuration et d'interconnexion de processus métiers pour obtenir des modèles de WFIO suffisamment flexibles et (ii) proposer des mécanismes support de la flexibilité de ces modèles. La figure I.2 décrit le schéma global de nos contributions.

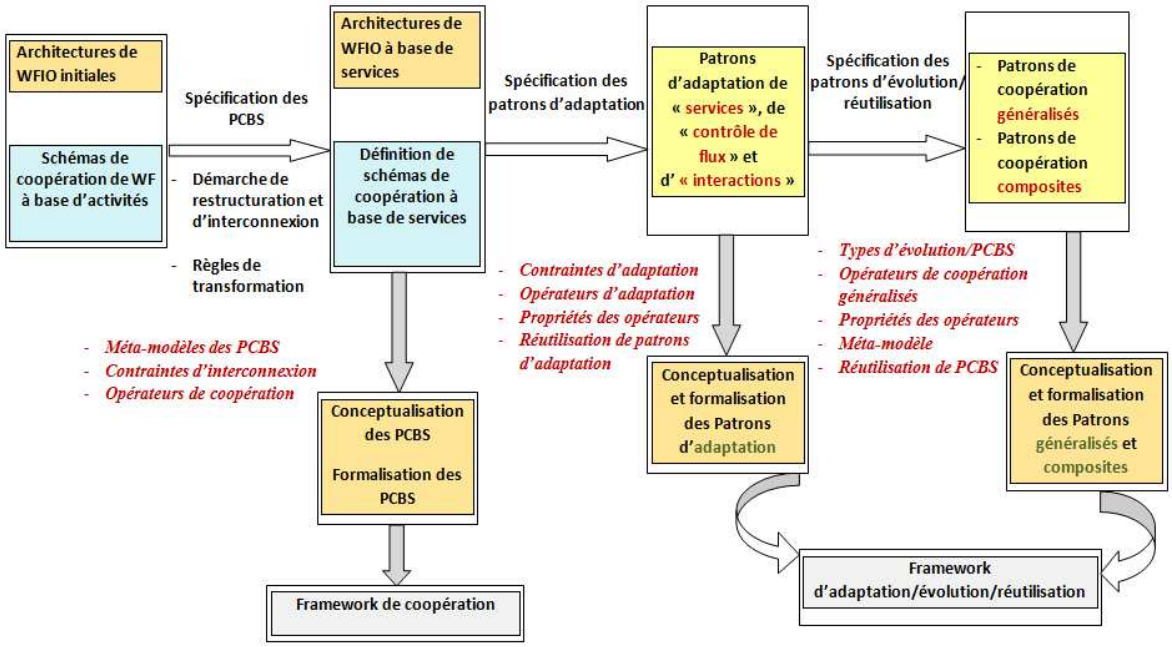



**Figure I.2.** *Schéma global de notre approche*

Outre les nouveaux concepts et les formalisations que nous introduisons dans cette thèse, notre approche d'interconnexion diffère des approches existantes par le fait que les processus métiers que nous considérons ne sont pas appréhendés comme des « boites noires » mais doivent dans certains cas (selon le schéma de coopération à réaliser), exhiber certains détails nécessaires à l'interconnexion appelés points d'interaction. *La règle générale est de présenter les modèles de processus WF avec le maximum d'abstraction et le minimum de visibilité requise pour l'interconnexion.*

A cet effet, nous avons d'abord proposé une démarche de restructuration des modèles de processus WF à base d'activités en modèles de WF à base de services, en vue de réaliser leur interconnexion via les invocations de services. Cette restructuration s'avère nécessaire d'une part, dans le cas où les processus métiers des partenaires sont déjà mis en place selon une approche de WF à base d'activités ; et d'autre part pour mettre en évidence la manière de découper les processus des partenaires en services de telle sorte à respecter les contraintes d'abstraction requises pour chaque schéma de coopération à réaliser.

Pour la phase d'interconnexion, nous avons proposé et implémenté un ensemble de patrons de coopération que nous avons conceptualisés à l'aide de méta-modèles, accompagnés d'un ensemble de règles de restructuration et d'interconnexion et un ensemble de contraintes liées aux flux de données dans le processus global. Nous avons ensuite procédé à une formalisation des patrons de coopération définis, via l'introduction d'opérateurs de coopération spécifiques.

Outre l'aspect de restructuration et d'interconnexion, nous nous sommes focalisés sur les aspects de flexibilité des modèles de processus WFIO. Après avoir établi les définitions des différents aspects de flexibilité (adaptabilité, évolutivité, réutilisabilité), nous avons introduit de nouveaux concepts et des opérateurs spécifiques support de l'adaptation, l'évolution et la

réutilisation des modèles de processus WFIO. Nous avons conçu et implémenté des patrons d'adaptation/évolution pour la prise en charge des changements de différents types pouvant affecter les modèles de processus WFIO obéissant aux différents patrons de coopération que nous avons nouvellement définis.

**Remarque :** Notons que dans notre cas, la réutilisabilité est étroitement liée à l'évolutivité des modèles de WFIO. En effet, nous proposons un mécanisme d'évolution basé sur la réutilisation de modèles ; on peut dire qu'il s'agit d'une *réutilisation pour l'évolution* ou encore *une évolution par réutilisation*. Ceci sera illustré dans le chapitre VI de ce document.

### I.7.2 Une approche à base de patrons

Depuis plusieurs années, l'approche à base de patrons connait un réel succès dans divers domaines, commençant par l'ingénierie des logiciels (Gamma, 1995), **(**Font-Conte et al., 1999), le workflow (van der Aalst et al., 2003), (Russel et al., 2006), (Dohring et al., 2010), l'ingénierie dirigée par les modèles (IDM) (Khadra, 2010) et la vérification de performances de systèmes (Xitong et al., 2010) , (Ait-Cheikh, 2012). L'avantage de l'approche à base de patrons réside dans le fait que le patron fournit une unité de raisonnement très modulaire. En effet, chaque patron existe pour répondre à un problème type. Cette modularité facilite la maintenabilité d'une solution à base de patrons et favorise la réutilisation de patrons élémentaires existants, pour l'élaboration de patrons plus complexes. C'est ce que nous utilisons pour la réalisation de patrons de coopération composites afin de supporter l'évolution de modèles par réutilisation.

Dans notre travail, nous avons commencé par caractériser les architectures de base de WFIO par trois dimensions principales : la distribution des services, le contrôle d'exécution et la structure d'interaction. Nous considérons que les services sont distribués sur les sites des partenaires qui les fournissent, le contrôle d'exécution peut être centralisé, décentralisé ou hiérarchisé et même parfois mixte (une combinaison des modes précédents). Les interactions peuvent être de type synchrone ou asynchrone. A ce niveau, nous introduisons le concept de « *patron de coopération à base de services* » (PCBS).

*Intuitivement, un **PCBS** définit la manière dont les services sont répartis sur les sites des partenaires, la structure d'interaction entre ces services et leur contrôle d'exécution, donnant lieu soit à une orchestration soit à une chorégraphie de services.*

Conceptuellement, chaque schéma de coopération correspondant à une architecture donnée de WFIO et est supporté par un PCBS définissant les règles de distribution de services, de contrôle d'exécution et d'interaction entre ces services qui diffèrent d'un schéma de coopération à l'autre. Ainsi, pour chaque patron de coopération, les informations suivantes sont fournies: une description générale comportant le nom, la référence, le cas d'utilisation, un schéma générique du patron, un méta-modèle, un ensemble de règles d'abstraction et un ensemble de règles et contraintes d'interconnexion. Celles-ci sont fournies pour le guidage de l'implémentation du patron.

### I.7.3 Une approche à base de méta-modèles

Pour la description des PCBS, nous avons adopté une approche par méta-modèles sous forme de diagrammes de classes UML, afin de ressortir les principaux concepts et les liens entre eux et garantir par suite, la construction de modèles de processus WFIO conformes aux méta-modèles définis. En effet, partant d'un méta-modèle générique d'un WFIO à base de services, nous avons procédé à son raffinement afin de l'adapter aux spécificités de chaque PCBS. Nous avons ensuite

formalisé ces patrons de coopération via l'introduction d'opérateurs spécifiques que nous avons appelés « opérateurs de coopération ». Aussi, pour les patrons de coopération composites, nous avons établi un méta-modèle spécifique afin de montrer le lien entre deux PCBS formant un patron de coopération composite.

L'avantage d'une approche par méta-modèle est d'abord d'assurer la conformité d'un modèle par rapports aux concepts ressortis sur le méta-modèle. De plus, si le méta-modèle est dérivé à partir d'un formalisme standardisé tel que UML (définition de profils UML), une transformation des modèles si nécessaire serait d'autant plus simple en utilisant des outils de transformation appropriés comme EMF[1] et ATL[2] (Mbarki et al., 2010).

### I.7.4 Mécanismes pour la flexibilité des modèles de WFIO

Une fois les patrons de coopération décrits et implémentés, nous nous sommes focalisés sur la question de flexibilité (adaptabilité, évolutivité, réutilisabilité) des modèles de WFIO à base de services.

Pour le premier aspect de flexibilité (i.e adaptabilité d'un modèle), nous avons proposé et implémenté des patrons d'adaptation classés en trois catégories selon les trois dimensions de définition d'un PCBS : patrons d'adaptation de *services*, patrons d'adaptation de *contrôle de flux* et patrons d'adaptation des *interactions*. A noter que cette dernière catégorie est étroitement liée au patron de coopération auquel obéit le modèle de WFIO. Des opérateurs spécifiques d'adaptation ont été définis pour la formalisation des patrons d'adaptation de base.

*Un patron d'adaptation (**PA**) décrit les opérations à effectuer afin d'apporter une modification bien définie sur un modèle de WFIO source pour arriver à un modèle de WFIO cible, tout en maintenant la cohérence du modèle global.*

Pour la question d'évolutivité des modèles, nous avons montré que pour une expansion des fonctionnalités, il suffit d'utiliser les patrons d'adaptation de services. Par contre pour une expansion de la coopération, nous avons introduit deux nouveaux concepts : les patrons de coopération *généralisés* et les patrons de coopération *composites*. Notons que l'utilisation de patrons de coopération généralisés ou composites constitue un aspect de réutilisation des modèles de processus WFIO.

*Un patron de coopération <u>généralisé</u> est un patron de coopération de base (PCBS) appliqué à l'interconnexion de trois processus ou plus.*

*Un patron de coopération <u>composite</u> est un patron de coopération qui définit l'interconnexion entre deux WFIO déjà existants obéissant à deux PCBS différents.*

En résumé, les contributions et les caractéristiques de notre approche se résument dans:

(i) Proposition d'une démarche de restructuration et d'interconnexion des modèles de WF en passant du concept d'activité vers le concept de service et en introduisant le concept de patron de coopération à base de services.

[1] https://fr.wikipedia.org/wiki/Eclipse_Modeling_Framework#EMF dernière visite le 15.08.2015
[2] Atlas Transformation Language

(ii) Définition de nouveaux concepts (patrons de coopération composites et patrons de coopérations généralisés) pour soutenir l'évolution et la réutilisation de modèles de processus existants, en vue d'une éventuelle expansion de la coopération.

(iii) Formalisation des patrons de coopération et d'adaptation à l'aide d'opérateurs spécifiques.

(iv) Les processus WF que nous considérons sont perçus comme une composition de services avec différents niveaux d'abstraction, permettant d'exhiber uniquement les détails nécessaires à l'interconnexion.

(v) L'utilisation d'une approche à base de patrons aux niveaux conceptuel et technique permet une maintenabilité et une extensibilité aisée des frameworks de coopération et d'adaptation/évolution développés.

(vi) Notre approche d'interconnexion, d'adaptation et d'évolution de processus couvre un large éventail de processus métiers existants puisque nous considérons différents schémas de coopération largement répandus dans le domaine du B2B.

## Synthèse

Dans ce premier chapitre, nous nous sommes intéressés à cerner le contexte de notre travail, à poser les différentes problématiques de cette thèse, à décrire globalement l'approche que nous adoptons pour atteindre les objectifs que nous nous sommes fixés, en vue de répondre aux problématiques posées.

Après avoir expliqué les différences existant entre les différentes formes de coopération (coopération planifiée et dynamique, structurée et ad-hoc), nous avons mis l'accent sur les contraintes pesant sur la coopération B2B en particulier la contrainte de *flexibilité* des SI. Celle-ci devient une nécessité favorisant d'une part, la participation à des relations de partenariat et facilitant d'autre part, la prise en charge des changements susceptibles d'affecter les processus métiers qui évoluent dans un environnement instable et dynamique, et qui doivent de suite suivre cette dynamique. Nous avons ensuite rappelé que l'avènement des architectures orientées services (SOA) est un véritable atout qui a considérablement favorisé l'ouverture des organisations à leur environnement. Par conséquent, les travaux dans le domaine de la coopération B2B se sont encore multipliés, notamment dans le cadre d'une coopération dynamique.

Partant du constat que la coopération planifiée (et donc forcément structurée) obéit à des schémas de coopération bien identifiés (Van der Aalst, 99), (Van der Aalst, 2000) largement répandus dans le domaine du B2B et que ces schémas de coopération implémentent différents types de contrôle et différents types de partitionnement, nous les considérons comme point de départ de ce travail de thèse. Le principal inconvénient de ces schémas de coopération est leur rigidité face aux changements et la difficulté d'adapter les modèles de processus métiers sous-jacents. A partir de là, nous nous sommes fixés un premier objectif qui consiste à définir de nouveaux schémas de coopération équivalents aux premiers et supportant des modèles de processus suffisamment flexibles.

Partant d'un deuxième constat que la flexibilité d'un modèle dépend fortement des entités qui le composent et des liens entre ces entités, nous avons opté pour une approche à base de services pour la définition de ces nouveaux schémas de coopération. Le but est de tirer profit des

caractéristiques d'interopérabilité et de couplage faible des services. Ainsi, nous introduisons un nouveau concept appelé « Patron de Coopération à Base de Services » (PCBS) pour supporter l'interconnexion de processus métiers à base de services. Globalement, nous adoptons une approche à base de patrons supportée par des méta-modèles de processus. Une fois, les modèles de coopération définis, nous nous focalisons sur la question de flexibilité, notre second objectif qui consiste à fournir des mécanismes d'adaptation, d'évolution et de réutilisation de modèles de processus WFIO. Pour cela, nous définissons des patrons d'adaptation, des patrons de coopération généralisés et des patrons de coopération composites.

Notons que les différents patrons proposés ont été formalisés via l'introduction d'opérateurs de coopération et d'adaptation dédiés. Côté implémentation, les différents patrons ont été implémentés dans des frameworks de coopération et d'adaptation/évolution pouvant s'interfacer entre eux.

Enfin, la question de correction des modèles après interconnexion, adaptation, évolution ou réutilisation doit être prise en compte. A l'état actuel, nous garantissons une cohérence de ces modèles via les règles et les contraintes que nous avons posées dans la conceptualisation des différents patrons. Néanmoins, ceci n'étant pas suffisant surtout pour les modèles de WFIO complexes pour lesquels une vérification formelle s'impose.

### Organisation du document

Le reste du document est structuré en trois parties : « Etat de l'art », « Contributions et Bilan » et « Annexes ».

La partie **« Etat de l'art »** comporte trois chapitres :

- Le **chapitre II** « Workflow, SOA et la Coopération B2B » : présente les bases de la technologie Workflow et ses principaux atouts dans le domaine de la coopération B2B, l'accent est mis particulièrement sur le WFIO, notamment les différentes architectures de coopération. Ce même chapitre décrit les principes des architectures orientées services (SOA) afin de montrer l'apport de ce paradigme et des standards sous-jacents dans le domaine des processus métiers, notamment dans la coopération B2B.

- Le **chapitre III** « Modélisation et Composition de Workflows » : est orienté vers la modélisation et la composition des modèles de processus WF. La question de composition de modèles dans l'IDM (Ingénierie Dirigée par les Modèles) étant abordée, en mettant en évidence les critères qui caractérisent les approches de composition dans l'IDM, afin de situer notre approche de composition (ou d'interconnexion) de modèles de WF par rapport à ces critères.

- Le **chapitre IV** « La Flexibilité du Workflow » : est consacré à la notion de flexibilité dans les WF. Dans ce chapitre, nous rappelons les différentes taxonomies de la flexibilité des processus métiers. Nous rappelons aussi les différentes approches de flexibilité proposées dans la littérature que nous classons dans différentes catégories. De plus, nous procédons à une comparaison de ces approches selon un ensemble de critères que nous avons dégagés.

La partie **« Contributions et Bilan»** comporte quatre chapitres :

- Le **chapitre V** « Notre Approche pour l'Interconnexion des Modèles de WF » : est consacré à la description de notre approche pour l'interconnexion des WF à base de services. Ce chapitre met en évidence les concepts de WF à base d'activités et de WF à

base de services en vue d'établir une correspondance entre eux. Sur la base de ces concepts, nous présentons une démarche de restructuration et d'interconnexion de WF qui s'articule autour de trois phases principales : une phase de « mapping », une phase d'abstraction et une phase d'interconnexion. Nous présentons ensuite la conceptualisation et la formalisation des différents patrons de coopération (PCBS) que nous proposons.

- Le **chapitre VI** « Notre Approche Support de la Flexibilité des Modèles de WFIO » : est dédié comme son titre l'indique, à la description de notre approche support de la flexibilité des modèles de WFIO. A ce niveau, nous rappelons les différentes définitions des différents aspects de flexibilité que nous considérons. Pour chacun des aspects, nous décrivons les mécanismes adéquats que nous proposons sous forme de patrons d'adaptation et patrons d'évolution de modèles de WF. Nous présentons à la fin du chapitre quelques facteurs dont la majorité sont qualitatifs liés à l'évaluation de notre approche d'adaptation/évolution.

- Le **chapitre VII** « Frameworks de Coopération et d'Adaptation/Evolution » : décrit dans sa première partie, notre framework de coopération (S-IOFLOW) dédié à la construction de modèles de WFIO obéissant à l'un des patrons de coopération (PCBS) définis. En effet, nous présentons l'architecture de déploiement de notre framework, son découpage conformément au patron de conception MVC ainsi que certains détails sur chacune des perspectives (Modèle, Vue et Contrôleur). La même démarche est reconduite, dans la deuxième partie du chapitre pour décrire les principaux éléments d'implémentation de notre framework d'adaptation/évolution.

- Le **chapitre VIII** « Bilan et Perspectives » : rappelle le contexte du travail, les différentes contributions et met en évidence les limites de ce travail et un ensemble de perspectives de recherche qui y sont liées.

Nous terminons ce document par un ensemble d'annexes complémentaires.

- L'annexe A présente les standards des services Web ainsi que des éléments sur le langage BPEL.
- L'annexe B décrit l'environnement de développement de nos frameworks de coopération et d'adaptation/évolution.
- L'annexe C illustre quelques éléments d'implémentation et des exemples de scénarios d'exécution de nos frameworks.
- L'annexe D montre le code BPEL d'un processus exemple.

# Partie I

# Etat de l’Art

# CHAPITRE II

# Workflow, SOA et la Coopération B2B

## Introduction

Depuis plusieurs années, la technologie Workflow a été largement utilisée pour l'automatisation des processus métiers des entreprises (Levan, 2000), (Van der Aalst, 2002). Grâce à leurs bénéfices remarquables dont la réduction des coûts opérationnels, le gain en temps et l'augmentation de la qualité de travail, les systèmes Workflow ont conduit à une amélioration considérable des processus d'entreprises.

Cependant, dans un contexte d'ouverture sur le marché mondial, face à la concurrence et dans le but d'améliorer leur productivité, les entreprises expriment depuis plusieurs années déjà, un grand besoin d'ouverture et de coopération à l'échelle mondiale. Elles ont besoin de s'allier à d'autres entreprises de compétences complémentaires afin de coopérer avec et offrir des services qui ne sont pas à la portée d'une seule entreprise (par exemple, fusions de groupes, extensions à l'international des structures d'entreprises, externalisations intensives de services, etc.). Ces entreprises coopératives mettent alors en commun, leurs processus métiers respectifs afin de bâtir un nouveau processus dit *processus métier inter-organisationnel*. Ce dernier nécessite des outils spécifiques de définition, de configuration, d'implémentation et de suivi d'exécution, incluant notamment la prise en charge des échanges inter-organisationnels. La coopération B2B a été initialement, supportée par les outils de workflow inter-organisationnels (WFIO) (Van der Aalst, 2000), (Van der Aalst et al, 2001), un concept qui trouve ses origines dans le domaine du e-commerce (Van der Aalst, 1999).

Notons que l'émergence de la technologie Workflow a considérablement servi de support technique au concept d'entreprise virtuelle (Lefebvre et al, 1999), notamment l'entreprise virtuelle coopérative qui consiste en un groupement d'entreprises devant communiquer via le web afin d'échanger des données et faire coopérer des procédés, dans le but d'atteindre un objectif métier commun.

Cependant, les outils de WFIO se sont avérés parfois insuffisants vis-à-vis des besoins d'interopérabilité et surtout de flexibilité des processus métiers, qui se font fortement ressentir dans un contexte inter-organisationnel. En effet, mettre en commun des processus métiers issus de plusieurs organisations peut nécessiter des adaptations internes au niveau local des processus, des adaptations au niveau de la structure d'interaction, des évolutions dans une optique d'ouverture vers de nouveaux partenaires ou une réutilisation de WFIO existants en vue de construire des WFIO plus complexes. Ces différents aspects fortement contraints par les limites des outils de WFIO semblent trouver un support adéquat avec l'avènement des architectures orientées services (SOA), un paradigme qui apporte justement, un pouvoir d'interopérabilité, d'adaptabilité, d'évolutivité et de réutilisabilité d'applications grâce aux caractéristiques de son concept pivot qui est le *service* et aux mécanismes de composition de services.

Ce chapitre est consacré à la présentation de trois points essentiels, le premier point aborde le concept de l'entreprise virtuelle (EV), ses différentes formes et caractéristiques. Le second point présente les concepts de base de la technologie Workflow en mettant l'accent sur le workflow inter-organisationnel (WFIO), support de l'entreprise virtuelle coopérative. Le troisième point présente les principes et les concepts de base des architectures orientées services en mettant en évidence leurs avantages appréciables dans le domaine des processus métiers, notamment dans la coopération B2B.

## II.1 Le Concept d'Entreprise Virtuelle (EV)

L'entreprise virtuelle sous sa forme la plus simple consistait en des firmes ayant choisi Internet comme moyen de ventes, de promotion et de logistique pour la distribution et pour les transactions financières : elles sont par conséquent virtuelles puisque le consommateur final ne sait pas nécessairement où elles se situent.

Pour des entreprises plus complexes, la chaîne de valeur représente un moyen fort simple de visualiser les activités (par exemple : conception de procédés, conception de produits, fabrication, …etc.) à valeur ajoutée d'une entreprise qu'elle soit manufacturière ou dans le secteur des services. Lorsque la majorité des activités d'une entreprise peuvent être réalisées en mode électronique ou virtuel, l'entreprise est dite virtuelle. Les exemples de ce type d'entreprises se retrouvent dans tous les secteurs de l'économie, certaines peuvent devenir complètement virtuelles, d'autres partiellement.

Sous l'impulsion des TIC et le besoin d'échange d'information entre les entreprises, le concept d'entreprise virtuelle a de plus en plus émergé, dans le cadre des collaborations et des partenariats stratégiques ; on parle alors d'*entreprise virtuelle coopérative*.

### II.1.1 Définitions

Il n'existe pas de définition académique stricte soumise à un consensus précis, pour définir les entreprises virtuelles. Nous en citons les suivantes :

- **Définition II.1 – EV**

«*Une organisation virtuelle correspond à une communauté de douzaines voire de centaines d'entreprises, chacune concentrée sur ce qu'elle sait faire le mieux, toutes reliées par un réseau électronique qui leur permet d'opérer de façon flexible, sans se soucier de leurs emplacements respectifs.* » (Upton & Mc Affe, 1996)

- **Définition II.2- EV**

« *L'entreprise virtuelle est un mode d'organisation où la totalité des fonctions est confiée à des partenaires, chaque partenaire étant susceptible d'être remplacé à tout moment si l'on trouve une solution plus satisfaisante.*» (Bartoli, 1996).

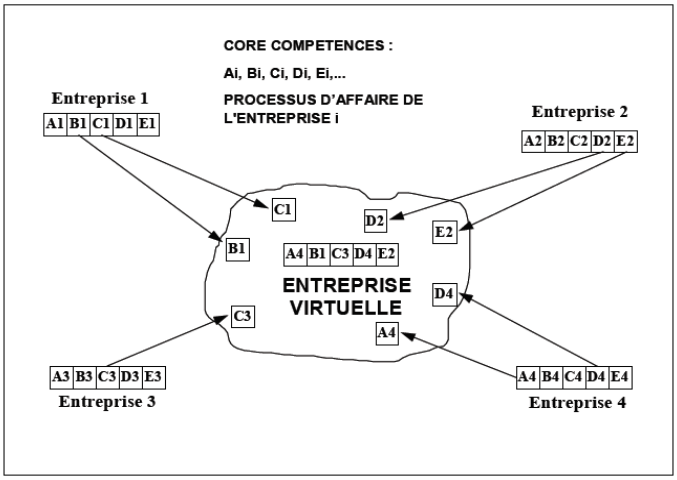



**Figure II.1.** *Schéma d'une entreprise virtuelle coopérative*

- **Définition II.3- EV**

« Une entreprise virtuelle est un *regroupement de plusieurs intervenants fonctionnant sur une même plate-forme informationnelle, et ce, pour la durée du projet ou de la réalisation du produit* ». (Lefebvre, 1999).

### II.1.2 Types d'Entreprises Virtuelles

Selon la nature et la durée de coopération, plusieurs types d'entreprises virtuelles sont identifiés (Kanzow, 2004) :

- ***Entreprise Virtuelle à court terme***

Ce type d'entreprise supporte une collaboration basée sur une affaire unique où les activités sont souvent asynchrones. Le système de gestion de processus inter-organisationnel contrôle l'activité globale mais les détails des activités sont implémentés localement au niveau de chaque partenaire, l'échange d'informations et la synchronisation des activités se limitent à des contacts informels du type mail.

- ***Entreprise Virtuelle basée sur l'entreprise étendue***

Ce type d'entreprise implique des collaborateurs permanents selon une *hiérarchie claire*. Le système de gestion de processus doit assurer l'intégration étroite des partenaires, notamment pour la gestion de la chaine logistique et nécessite l'utilisation d'outils de gestion de processus permettant l'intégration inter-organisationnelle des différents processus. Ainsi, L'EV étendue intègre à ses systèmes informatiques et ses processus métiers, ceux de ses clients, fournisseurs et partenaires.


**Figure II.2.** *Schéma d'une EV étendue*

- ***Entreprise Virtuelle sous forme de Consortium***

Ce type d'entreprise supporte les alliances de plusieurs entreprises d'un même secteur exerçant la même activité. Une EV de ce type définit une collaboration permanente entre les participants basée sur une structure non hiérarchique où chaque participant garde un degré d'indépendance ; cela exige une très grande flexibilité et réactivité. Tout en travaillant pour le même objectif, les membres d'une entreprise virtuelle peuvent être concurrents.


**Figure II.3.** *Schéma d'une EV sous forme de Consortium*

### II.1.3 Caractéristiques de l'EV

Bien que les définitions académiques de l'entreprise virtuelle diffèrent, leurs modèles présentent des caractéristiques métiers et techniques communes (Kanzow, 2004) dont:

- *L'ignorance des frontières entre participants :* une collaboration peut avoir lieu par l'union de compétences fondamentales et/ou la combinaison de méthodes de production utilisées par les participants.
- *La complémentarité des compétences fondamentales et le partage des ressources :* la relation de partenariat permet de livrer un produit ou de réaliser un projet à travers la collaboration et le partage des ressources.
- *L'égalité des participants et leur dynamique :* chaque participant dans la collaboration joue son propre rôle en contribuant à l'amélioration du produit final, il fait partie de la chaine indépendamment de l'emplacement dans le processus de l'entreprise virtuelle. De plus, les participants dans une entreprise virtuelle ne sont pas forcés de rester dans l'entreprise à long terme, ils peuvent former une entreprise virtuelle avec d'autres entreprises. Le nombre de partenaires dans l'entreprise virtuelle peut être statique, ou bien dynamique selon les besoins et les exigences des partenaires impliqués.
- *La confiance :* l'EV repose principalement sur la confiance entre les partenaires, cette confiance est garantie par des procédures et des contrats.
- *La temporalité :* L'entreprise virtuelle est considérée temporaire par le fait que sa vie dure le temps de l'achèvement d'un projet. Mais elle pourrait aussi avoir une durée indéterminée; l'entreprise restera fonctionnelle tant que la demande des clients existe et/ou les participants constatent que leur collaboration est avantageuse.
- *La dispersion géographique des participants et l'utilisation des TIC*: l'emplacement géographique des participants n'a plus de signification puisque la communication entre eux est réalisée à travers les TIC. Ainsi, le courrier électronique, la messagerie vocale et la communication à travers vidéoconférence permettent l'établissement rapide de contact entre les différents partenaires. Les logiciels de travail en groupe (Groupware) offrent aussi des possibilités de collaboration et de communication efficaces, croisant les frontières de temps et d'emplacements géographiques.

Le Workflow qui représente une des applications du groupware, est une technologie largement utilisée pour supporter l'automatisation des processus métiers. Dans la suite, nous présentons brièvement les bases de cette technologie.

## II.2 Le Workflow

L'utilisation des workflows est devenue depuis plusieurs années déjà, un atout majeur fournissant aux entreprises un avantage concurrentiel, comme étant un moyen pour réduire les coûts, automatiser les processus et réduire le temps de réponse. Correctement implémentées, les applications basées sur les workflows permettent aux entreprises de restructurer et de rationaliser leurs processus métiers.

### II.2.1 Définitions

- **Définition II.4 – Workflow**

« U*n workflow est une représentation d'un procédé métier dans un format interprétable par la machine. Il est constitué d'un réseau d'activités et de 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.), les activités et les procédés ont des données en entrée et en sortie.* » (WFMC, 95).

- **Définition II.5 – Workflow**

« *Un Workflow est un processus d'une organisation, gérable par un outil Workflow, dont le but est d'automatiser tout ou une partie d'un processus d'entreprise, au cours duquel l'information circule d'une tâche à l'autre, c'est-à-dire d'un participant à l'autre, pour une action en fonction d'un ensemble de règles de gestion* » (WFMC, 99).

- **Définition II.6 – Processus**

*« Un processus est un ordonnancement de tâches séquentiels, parallèles ou alternatives, soumises à des conditions de transition et devant être exécutées selon un ensemble de règles bien définies. Ces tâches peuvent-être automatiques, semi-automatiques ou manuelles.»* (Saikali, 2001)

- **Définition II.7 – Schéma de Processus**

**«** *Une définition de processus ou schéma de processus décrit l'aspect comportemental d'un Workflow. En d'autres termes, un schéma de processus décrit les tâches qui le composent, leur enchaînement et dépendances d'exécution, les critères de lancement et d'achèvement et les informations relatives aux tâches (participants, ressources, données). »* (Saikali, 2001)

Les spécialistes du Workflow distinguent généralement trois types de processus (Saikali, 2001) :

***a)*** **Les processus matériels** : se caractérisent par la manipulation, l'assemblage, la livraison, la transformation, la mesure et le stockage d'objets physiques. Les processus matériels lient entre elles des activités humaines ou automatisées, localisées dans le monde physique. Il ne s'agit donc pas d'activités administratives ou intellectuelles.

***b)*** **Les processus informationnels** : relient entre elles des activités automatiques ou semi-automatiques. Ces activités créent, traitent, génèrent et fournissent des informations.

***c)*** **Les processus métiers** : appelés aussi processus d'entreprise ou processus opérationnels représentés par une collection d'activités consommant des entrées, pouvant être de nature matérielle ou informationnelle, et délivrant un ou plusieurs résultats à orientation économique. Les processus métiers sont directement liés au métier et aux objectifs principaux de l'entreprise qui les met en œuvre.

Parmi ces trois classes de processus, les processus métiers sont des processus candidats au Workflow. La figure II.4 montre l'exemple d'un processus WF (sous forme d'un diagramme d'activités UML) lié aux demandes de clients pour les raccordements Internet. Le processus fait intervenir deux organismes respectivement, « Eepad » et « AT ».

Le partenaire « Eepad » implémente un workflow de traitement de requêtes clients et doit sous-traiter les requêtes de raccordement Internet auprès du partenaire AT. Une requête client arrive au niveau de l'agent Hotline d'Eepad, elle est aussitôt vérifiée. S'il s'agit d'un problème technique, il est diagnostiqué par l'agent Hotline qui le prend en charge localement dans le cas d'un problème solvable. Dans le cas contraire, la requête est transmise au superviseur Hotline qui édite une réclamation et l'envoie au technicien approprié. Ce dernier prend en charge la requête du client afin de résoudre le problème. Si la requête du client est une demande de raccordement à Internet, elle est transmise au partenaire AT qui la prend en charge à travers son WF. Le WF de raccordement d'AT est fourni tel un service Web et envoie les résultats du traitement à Eepad qui se charge de terminer la procédure à son niveau et informer le client du résultat de sa requête de raccordement.

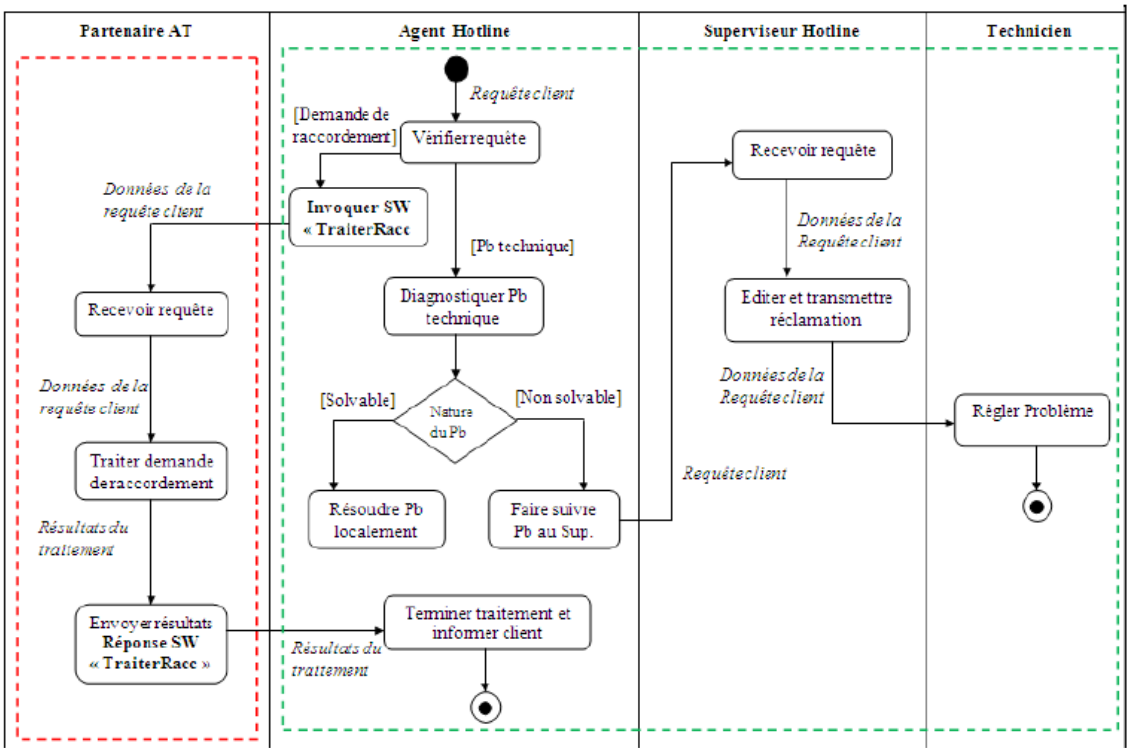



**Figure II.4**. *Exemple d'un processus workflow* (Boukhedouma et al., 2010a)

### II.2.2 Concepts de base du Workflow

Pour une meilleure illustration des concepts accompagnant la gestion de processus par un système de Workflow, la WFMC propose un méta-modèle faisant ressortir les principaux concepts de définition d'un processus Workflow et les liens entre eux.

Un workflow est composé d'un ensemble d'activités. Une Activité est une étape d'un processus au cours de laquelle une action élémentaire (c'est-à-dire non décomposable) est exécutée. Un Acteur (ou participant) représente une entité (personne ou matériel), membre d'une organisation, chargé de réaliser les activités qui lui sont assignées, à travers ses rôles. Un Rôle décrit la liste des attributs, des compétences et du savoir-faire d'un acteur du Workflow (participant) et qu'il met en pratique. Ce rôle définit la position de l'acteur dans une organisation, un rôle peut être tenu par plusieurs acteurs. Une *Donnée* représente l'objet

manipulé pendant la réalisation d'une activité ; la donnée peut être utilisée en entrée comme elle peut aussi être générée par l'activité.

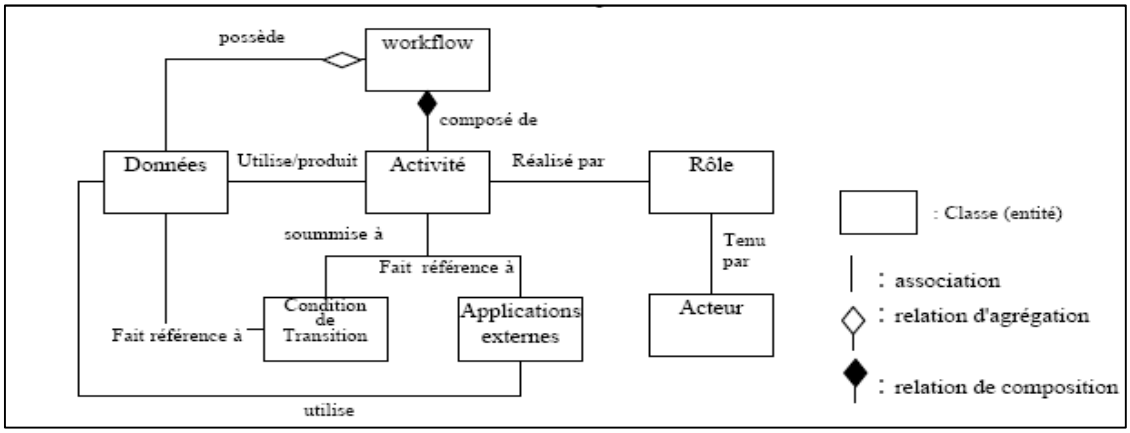



**Figure II.5.** *Méta-modèle de WF pour la définition de processus* (WFMC, 99)

Une *condition de transition* définit le critère de changement d'état d'une activité ou de passage à l'activité suivante; elle s'exprime sous forme d'évènement ou d'expression logique. Une condition peut s'exprimer sous la forme d'une expression logique en utilisant les opérateurs de contrôle de flux des activités tels que : And-Join, And-Split, Or-Join, Or-Split, XOR-Join, XOR- Split. Une *application externe* est une application informatique que l'on appelle pour réaliser une activité. Si l'application n'est pas uniquement informatique, on parle de ressource.

D'autres concepts dits complémentaires viennent s'ajouter aux concepts de base, nous en citons:

- *Les Règles :* décrivent le principe de comportement que l'acteur doit suivre pour exécuter une activité. En cours ou à la suite d'une opération, sont mises en jeu des règles de gestion qui définissent la forme prise par l'information et le cheminement ultérieur vers les opérations suivantes.

- *Instance de processus :* c'est un cas qui correspond à une exécution donnée d'une définition de processus. Une définition peut être instanciée plusieurs fois, et plusieurs instances peuvent s'exécuter concurremment.

- *Le moteur de Workflow:* c'est un service logiciel qui fournit tout ou une partie de l'environnement d'exécution d'un Workflow.

- *Le système de gestion de Workflow (SGWF):* c'est un système complet qui sert à définir, gérer et exécuter des procédures grâce à des programmes dont l'ordre d'exécution est prédéfini dans une représentation informatique de la logique de ces procédures.

L'évolution des processus organisationnels de l'entreprise a favorisé l'utilisation des outils de Workflow dans divers domaines d'applications. En effet, l'utilisation de cette technologie joue un rôle considérable dans les entreprises du monde financier comme les systèmes bancaires, les assurances et peut même être étendue à tout processus de travail cyclique dans le monde de l'entreprise. Le workflow s'intéresse aussi au processus de développement d'un logiciel ; en intégrant l'aspect travail coopératif au sein du Workflow, on peut lier l'intégration progressive des éléments d'un logiciel avec l'organisation prévue. Les Workflows peuvent également être utilisés dans des organisations autres que l'entreprise, dans le monde médical par exemple, ou dans l'éducation (processus de e-learning) ou encore dans l'automatisation des procédures de l'administration publique (processus de e-administration, e-gouvernance).

### II.2.3 Classification des Workflows

Il n'existe pas de classification commune des systèmes Workflow dans la littérature et qui soit reconnue par l'ensemble de la communauté Workflow. Ceci étant essentiellement dû au nombre important de critères de classification qu'il est possible de retenir, parmi ces critères de classification on a : le domaine d'application, les objectifs, le degré de structuration du Workflow… etc. Seulement la classification par domaine d'application est la plus répandue (McCready, 1992), elle propose de distinguer quatre catégories de systèmes Workflow selon deux axes : *approche* et *structure*, comme le montre la figure II.6.

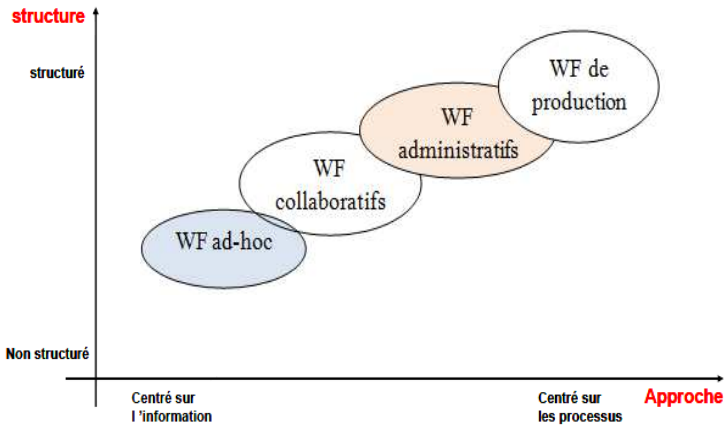



**Figure II.6.** *Classification des Workflows*

#### II.2.3.1 Le Workflow de production

Appelé aussi « WF procédural » ou « case-based » centré sur les processus et présente le plus haut niveau de structuration. L'utilisation de ce type de Workflow a pour but d'optimiser des processus répétitifs et bien maitrisés, afin d'augmenter la productivité et la qualité des articles produits.

#### II.2.3.2 Le Workflow administratif

Les systèmes WF administratifs ont pour objectif de décharger les ressources d'une entreprise, des tâches administratives. En effet, ces procédures sont répétitives, fortement prédictibles et les règles d'enchaînement des tâches sont très simples et clairement définies et donc aisément automatisables. Ce type de procédures possède une structure statique, sont rarement assujetties à des modifications car elles possèdent une longue durée d'utilisation (Scheer, 1997). Les WF administratifs sont aussi centrés processus et sont bien structurés. En particulier, ce type de WF permet d'apporte une valeur ajoutée considérable aux organismes de l'administration publique dont le métier est centré sur des procédures administratives.

#### II.2.3.3 Le Workflow ad-hoc

Le WF ad-hoc est un ensemble d'activités ordonnées selon un ensemble de règles procédurales pour réaliser un objectif précis au sein d'un groupe de personnes. Cet ensemble de règles est destiné à des situations spécifiques (causées par des exceptions) où la logique de déroulement à suivre est définie durant l'exécution. Il forme une solution hybride rassemblant des caractéristiques d'administration, de production et de collaboration. Généralement, ce type de procédé ne possède pas de structure prédéfinie, l'étape ultérieure à suivre est déterminée

essentiellement par les participants au procédé. Ce sont des WF peu structurés et centrés sur l'information.

#### II.2.3.4 Le Workflow collaboratif

Les WF collaboratifs appelés aussi WF coopératifs, sont dédiés au support du travail de groupe, tel que la conception, la gestion de projet ou la résolution des problèmes faisant appel à différents niveaux d'expertise. Ces WF permettent de réunir un certain nombre d'intervenants dans le but d'atteindre un objectif commun. Les clients du processus y sont souvent associés. Les tâches gérées sont généralement complexes et les acteurs impliqués dans le processus doivent être très compétents (dans la spécialité qui est la leur). Ces WF sont faiblement structurés et peu répétitifs, ils sont centrés sur l'information. Par contre, ils sont à forte valeur ajoutée par rapport à l'entreprise qui les met en place.

En résumé, un grand nombre d'entreprises ont eu recours à l'utilisation de solutions Workflow durant les années 90, chacune dans son domaine d'activité et selon ses besoins. Néanmoins, l'inconvénient majeur des solutions proposées réside dans leur rigidité et leur fermeture, donc l'inadaptation de ces solutions à supporter les interactions coopératives. Par ailleurs, l'émergence d'une économie mondiale et la dynamique du marché, le développement des réseaux et l'avènement d'Internet sont des facteurs ayant conduit les organisations à mettre en commun leurs compétences et à partager des ressources pour faire face à une demande de produits et de services de meilleure qualité, disponibles dans des délais minimaux, moins chers et personnalisés. Dans le but de suivre ces évolutions et ces nouvelles exigences, le WF a subi lui aussi des changements considérables. En effet, il s'est avéré nécessaire de mettre en commun et faire coopérer les processus de plusieurs organisations réparties, autonomes et hétérogènes; ce qui a donné naissance au Workflow Inter-Organisationnel (WFIO) qui fait partie des technologies support de l'entreprise virtuelle coopérative.

## II.3 Le Workflow Inter-Organisationnel (WFIO)

### II.3.1 Définitions

- **Définition II.8 –** WFIO

*« Le WFIO peut être vu comme une collection de WF locaux (ou composants distribués), éventuellement hétérogènes, ayant chacun une activité propre plus ou moins autonome, et travaillant de manière coopérative pour répondre à un objectif commun. L'objectif ultime d'un WFIO est de composer/coordonner les différents WFs locaux avec une contrainte forte imposée par chaque partenaire qui veut maintenir le contrôle sur son WF local ».* (Chebbi, 2007)

- **Définition II.9 – WFIO**

Le WFIO peut être défini comme *« un gestionnaire d'activités faisant intervenir deux ou plusieurs workflows de partenaires, autonomes, interopérables et éventuellement hétérogènes.»* (Bernauer et al, 2003).

### II.3.2 Caractéristiques et Contraintes du WFIO

Dans le cadre d'une coopération inter-organisationnelle, un ensemble de caractéristiques importantes doivent être prises en considération pour l'élaboration du WFIO et son bon fonctionnement (Chebbi, 2007).

***a) Autonomie*** : un partenaire impliqué dans une coopération avec d'autres partenaires souhaiterait préserver au maximum son savoir-faire, son modèle de WF préétabli, ses données locales et sa logique métier dans l'exécution des instances de processus.

***b) Hétérogénéité*** : les workflows impliqués dans une coopération sont souvent hétérogènes. L'hétérogénéité peut être perçue au niveau : (i) des modèles de processus puisque les processus WF sont spécifiés avec des langages de définition de processus métiers différents, (ii) au niveau des plates-formes d'exécution de workflow, c'est-à-dire les systèmes de gestion de workflow (SGWF) et (iii) au niveau des données échangées où l'on distingue une hétérogénéité syntaxique et sémantique.

***c) Interopérabilité*** : afin que des partenaires puissent inter-opérer à travers leurs systèmes autonomes et hétérogènes, il est nécessaire de prévoir des interfaces d'échange permettant une compréhension commune des données et des processus ou services impliqués dans la coopération.

D'autres caractéristiques telles que la *distribution* et l'*ouverture* n'apparaissent pas explicitement dans la définition mais sont liées à tout système de WFIO.

***d) Distribution*** : les différents WF formant un WFIO sont implémentés chacun au niveau d'un site distant, ils manipulent des définitions de processus locaux et des données stockées dans des bases de données locales. Toutefois, il existe des architectures de WFIO qui maintiennent une définition centralisée du processus WF global et/ou manipulent des données dans une BD centralisée ; c'est le cas de l'architecture «Partage de charge » que nous verrons plus loin dans ce chapitre.

***e) Ouverture*** : l'ouverture d'un système vers l'environnement extérieur nécessite la prise en compte des facteurs de sécurité des échanges, de confiance entre partenaires, de dissimulation des données et des services privés et enfin la légalité des échanges.

Enfin, une caractéristique importante devant être vérifiée par tout système de WFIO est la *flexibilité* permettant ainsi de le qualifier de système de WFIO flexible (ou agile).

***d) Flexibilité*** : les entreprises sont sans cesse soumises à de nouveaux défis et de nouvelles contraintes issues du marché. L'environnement des entreprises ainsi que les processus métiers qui décrivent leur comportement sont dynamiques par nature. En effet, un processus est perpétuellement sujet à des modifications car soumis à l'influence de facteurs externes ou internes, remettant en cause sa structure et sa validité. *Un processus est donc une entité qui évolue, se modifie et s'adapte.* Les entreprises les plus performantes sont donc celles qui s'adaptent le plus rapidement, en améliorant la qualité et le temps de réponse de leurs services ainsi que l'adéquation de leurs produits aux besoins des clients. Dans le domaine du WF, la caractéristique de flexibilité a été initialement étudiée dans le cas des WF intra-organisationnels (appelés aussi WF traditionnels) et ensuite élargie aux WF inter-organisationnels. Selon le niveau et le degré de flexibilité des systèmes, on parle de WF flexibles, dynamiques, adaptables ou adaptatifs (Ellis et al, 98), (Liu et al, 98), (Kiepuszewski et al, 2000), (Saikali, 2001), (Muller, 2002), (Muller et al, 2004).

Concrètement, la flexibilité d'un WF consiste d'une part à absorber tous les événements pouvant inquiéter la cohérence et l'intégrité des données mais également de conserver la cohérence du déroulement du processus en toute circonstance, ainsi que d'intégrer les

modifications qui peuvent avoir lieu dans l'organisation de l'entreprise. Dans la pratique, on constate cependant que la flexibilité du WF se reflète principalement au niveau des processus (et des interactions), bien qu'elle implique et s'appuie également sur les autres niveaux – données, organisation. Par ailleurs, du point de vue des mécanismes support de la flexibilité d'un WF, on distingue deux niveaux complémentaires:

*Au niveau des systèmes*, la flexibilité définit la capacité des systèmes de gestion de WF (SGWF) à faire face aux situations imprévues et erronées telles les exceptions et à gérer de manière flexible l'allocation des ressources (les différents types d'acteurs de WF) dans l'exécution des instances de processus, au niveau « Runtime ».

*Au niveau des modèles de processus*, la flexibilité peut être vue sous différents aspects dont l'*adaptabilité*, l'*évolutivité et la réutilisabilité* des modèles comme nous l'avons déjà souligné dans le chapitre précédent. Notons que nous reviendrons sur cette caractéristique et ses différents aspects plus en détail dans le **chapitre IV** de ce document.

L'ensemble des caractéristiques de WFIO énoncées plus haut induisent des contraintes au niveau métier et/ou au niveau technique des WF mis en coopération. Le tableau II.1 résume les principales contraintes liées aux caractéristiques du WFIO.

**TAB II.1.** *Caractéristiques et contraintes du WFIO*

| **Caractéristique** | **Contraintes d'un point de vue métier** | **Contraintes d'un point de vue technique** |
|---|---|---|
| ***Autonomie*** | – Protection du savoir-faire<br>– Préservation de l'existant<br>– Choix des partenaires | – Respect de la logique métier dans le modèle de processus<br>– Contrôle décentralisé |
| ***Hétérogénéité*** | | – Intégration des SGWF des différents partenaires |
| ***Interopérabilité*** | – Choix de l'architecture de coopération répondant à l'objectif métier | – Offrir des interfaces d'échange<br>– Protocole de communication à mettre en place<br>– Interopérabilité syntaxique et sémantique des données échangées |
| ***Distribution*** | | – Nature du contrôle : centralisé, décentralisé ou mixte |
| ***Ouverture*** | - Degré de confiance entre les partenaires | - Sécurité des échanges et confidentialité |
| ***Flexibilité*** | – Gestion des situations imprévues (rupture de contrat, besoin de nouvelles ressources,…)<br>– Amélioration du processus (nouvelles exigences métier)<br>– Possibilité de coopération dynamique | – Flexibilité des SGWF : gestion des exceptions et des cas d'erreurs.<br>– Flexibilité des modèles de processus : Adaptabilité, Evolutivité et Réutilisabilité |

Notons que les contraintes du niveau métier se résument dans la protection du savoir-faire, la préservation des WF préétablis, le choix des partenaires, la possibilité d'une coopération dynamique, l'amélioration du processus,… etc. Les contraintes d'ordre technique sont liées aux

problèmes d'interopérabilité, d'intégration, de contrôle d'exécution, la gestion des exceptions et les changements à apporter aux modèles de processus.

### II.3.3 Architectures génériques du WFIO

Lors de l'exécution des instances de processus WFIO, quatre types d'interopérabilité peuvent être distingués : une interopérabilité *hiérarchique*, une interopérabilité *chaînée*, une interopérabilité *point à point* et une interopérabilité *parallèle*. Ces types d'interopérabilité ont été identifiées dans (Van der Aalst, 99), (Van der Aalst, 2000) pour définir les différents schémas de coopération (appelés architectures de WFIO), pouvant lier les processus WF de partenaires métier dans une coopération B2B, pour la réalisation d'un objectif métier commun et selon une politique gagnant-gagnant.

Cette section passe en revue ces différentes architectures en mettant en évidence les particularités de chacune d'elles ainsi que les avantages et les insuffisances qu'elles présentent. La figure II.7 schématise ces différentes architectures. Sur la figure, WF1, WF2 et WF1_X désignent des workflows implémentés au niveau des sites de partenaires.

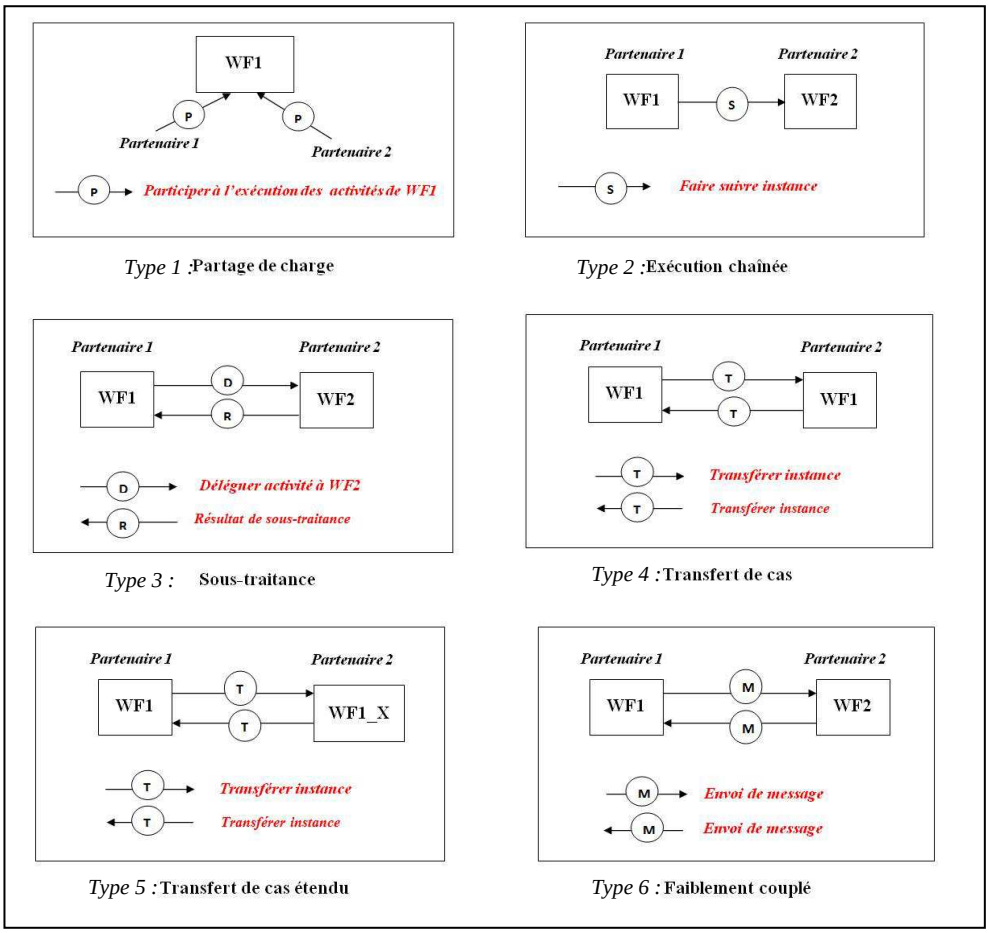



**Figure II.7.** *Schémas d'architectures génériques de WFIO*

Sur la figure II.7, les flèches ont des significations différentes (donc des symboles différents) et représentent les possibilités d'interaction entre les WF impliqués dans la coopération. Chaque WF est doté d'un système de gestion de WF (SGWF) ayant servi à son implémentation et permettant le suivi et le contrôle d'exécution local des instances de processus qui lui sont associées.

### II.3.3.1 Architecture de type 1 « Partage de Charge » (Capacity Sharing)

L'architecture de type 1 « Partage de charge » implémente une première forme d'interopérabilité où les partenaires partagent le même modèle de processus qu'ils définissent conjointement. Celui-ci n'est pas dupliqué au niveau de chaque partenaire mais il existe en une seule copie, sur le site d'un partenaire ou d'un tiers externe, accessible par un SGWF centralisé. Ce système se charge de distribuer les instances d'activités constituant le WFIO global, entre les différents partenaires selon les rôles qu'ils jouent dans le processus. Sur la figure II.7, deux partenaires métier accèdent au même modèle de processus *WF1* afin d'exécuter les activités composant le WF global (WF1).

### II.3.3.2 Architecture de type 2 « Exécution Chaînée » (Chained Execution)

Dans une architecture de type « Exécution chainée », les partenaires participent à l'exécution d'un WFIO global, de manière séquentielle. Le WF global est partitionné en fragments disjoints, chaque fragment est implanté localement au niveau du partenaire qui en est responsable et est géré par un SGWF local. Dès que le premier partenaire termine l'exécution de son WF local, le WF du second partenaire est déclenché et ainsi de suite jusqu'au WF du dernier partenaire. Sur la figure II.7, deux partenaires métier possédant chacun, son propre modèle de processus *WF1* et *WF2* respectivement, devant être exécuté selon l'ordre séquentiel défini dans le WFIO global.

### II.3.3.3 Architecture de type 3 « Sous-Traitance » (Subcontracting)

La forme de « sous-traitance » suppose l'existence d'un partenaire principal implémentant localement, son propre modèle de WF géré par un SGWF local, et devant sous-traiter certaines activités du processus, auprès d'autres partenaires. Les activités faisant l'objet de sous-traitance paraissent atomiques au niveau du partenaire principal mais peuvent correspondre à des WF complexes au niveau des partenaires sollicités pour la sous-traitance. Ces derniers hébergent localement leurs WFs et leurs SGWFs respectifs. Dans ce cas de figure le contrôle du processus étant hiérarchisé. La figure II.7 montre le schéma d'un partenaire principal *partenaire1* ayant son propre *WF1* et sous-traitant avec un partenaire *partenaire2* qui implémente localement son *WF2*.

### II.3.3.4 Architecture de type 4 « Transfert de Cas » (Case Transfer)

Dans cette architecture, les partenaires métier partagent tous le même modèle de processus dupliqué au niveau de chacun d'eux. Leur coopération consiste à transférer l'exécution d'une instance de processus d'une localisation à une autre afin de prendre en charge la suite de son exécution. A tout moment, une instance de processus se trouve au niveau d'un seul partenaire, le transfert de l'instance doit prendre en compte son état afin de maintenir la cohérence du système global. Le transfert de cas se fait pour des raisons d'équilibrage de charge entre les partenaires ou d'indisponibilité de ressources adéquates, par exemple. La figure II.7 schématise une coopération entre deux partenaires métiers hébergeant tous les deux le même modèle de processus *WF1*.

### II.3.3.5 Architecture de *type 5* « Transfert de Cas Etendu» (Extended Case Transfer)

Cette forme d'interopérabilité est une variante de la précédente, la seule différence réside dans la possibilité de redéfinitions locales de certaines activités dans le modèle de processus (ajout d'activité, suppression,…etc.), tout en maintenant la logique globale du processus. Sur la figure

II.7, on schématise la coopération entre deux partenaires en supposant qu'une extension du modèle est définie différemment au niveau du *partenaire2*. Ainsi, le *partenaire1* héberge le modèle *WF1* étendu en *WF1-x* au niveau du *partenaire2*.

### II.3.3.6 Architecture de type 6 « Faiblement Couplée » (Loosely Coupled)

L'architecture « Faiblement couplée » offre la possibilité d'exécution parallèle des fragments de WF implémentés et hébergés localement avec les SGWF correspondants, au niveau des différents partenaires. Un protocole de communication public est défini afin de permettre l'échange de données entre les différents WF formant le WFIO global. Les différents WF communiquent de manière asynchrone, par envoi de messages. La figure II.7 montre une coopération faiblement couplée entre deux partenaires hébergeant les modèles de workflow *WF1* et *WF2* pouvant s'exécuter en parallèle avec la possibilité d'interactions asynchrones conformément au protocole de communication défini.

Les architectures de coopération su-décrites ont été caractérisées par deux dimensions principales : la dimension *partitionnement du processus* et la dimension *contrôle d'exécution.* A ces deux dimensions, nous ajoutons la dimension *type d'interaction* qui diffère d'une architecture à l'autre ; en effet le type d'interaction peut être synchrone ou asynchrone et même dans ce cas, on peut distinguer la communication dans un seul sens (One-way) de la communication à double sens (Two-way). De ce fait, cette dimension constitue aussi une caractéristique du WFIO et doit être prise en compte dans sa modélisation.

### II.3.3.7 Dimensions du WFIO

#### A- Partitionnement du processus

Dans un WFIO, la coopération consiste à exécuter conjointement les instances de processus inter-organisationnel, par les différents partenaires impliqués dans la coopération.

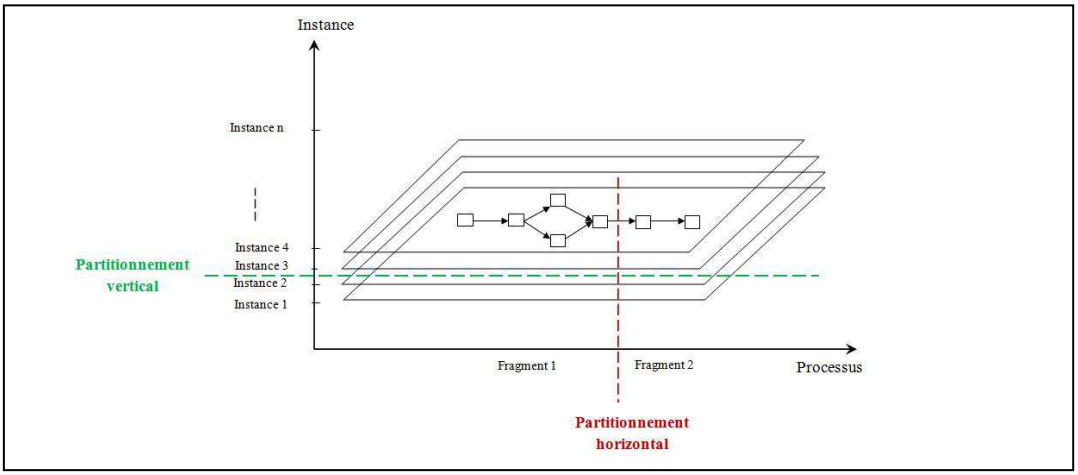



**Figure II.8.** *Axes de partitionnement dans un WFIO*

Ainsi deux types de partitionnement sont distingués : (1) le partitionnement selon la *dimension schéma de processus* si le modèle de processus global est composé de fragments de modèles disjoints implémentés au niveau des partenaires. (2) Le partitionnement selon la *dimension instance* si à tout moment, les instances de processus en cours d'exécution sont réparties de manière disjointe sur les différents sites de partenaires.

Dans (Van der Aalst, 1999), l'auteur parle de partitionnement *vertical* et de partitionnement *horizontal* (voir Figure II.8), dans le sens où les instances d'un processus donné suivent toutes le même modèle de processus et se superposent selon l'axe *vertical*. Ainsi, on parle de *partitionnement vertical* lorsque les instances de processus sont réparties de manière disjointe sur les sites des différents partenaires (à un moment donné, une instance de processus existe au niveau d'un et un seul partenaire). *Le partitionnement horizontal* consiste à fragmenter le modèle de processus et à implémenter chaque fragment de modèle au niveau d'un partenaire. Par conséquent, nous pouvons dire que le partitionnement vertical est supporté par un mécanisme d'*instanciation* du modèle de processus, alors que le partitionnement horizontal est supporté par un mécanisme d'*agrégation* des fragments de modèles de processus afin de construire le processus de WFIO global. Le tableau suivant (TAB II.2) fournit une comparaison entre les deux types de partitionnement (vertical/horizontal) selon un certain nombre de critères que nous avons posés.

**TAB II.2.** *Comparaison des deux types de partitionnement*

| *Partitionnement du processus* | *Implantation du modèle de processus* | *Exécution concurrente* | *Degré de parallélisme* | *Interactions* | *Coût de la coordination* |
|---|---|---|---|---|---|
| *Vertical* | Le même modèle de processus peut être dupliqué sur tous les sites | L'instance se trouve à une seule localisation, pas d'exécution concurrente | Réduit | Simples | Réduit |
| *Horizontal* | Le modèle de processus est fragmenté sur différents sites | L'instance est répartie sur plusieurs sites. Nécessité de gérer les relations causales, les conflits et les exceptions | Elevé | Complexes | Elevé |

### B- Nature du contrôle

Vu le caractère distribué des systèmes de WFIO, le contrôle d'exécution des instances de processus peut être centralisé, décentralisé, hiérarchisé ou mixte.

- ***Centralisé***: le contrôle d'exécution est délégué à l'un des partenaires ou à une entité externe qui contrôle l'exécution de bout en bout d'une instance de processus. Ce type de contrôle est utilisé dans l'architecture de type 1 « Partage de charge ».

- ***Décentralisé***: chaque partenaire possède son propre SGWF qui contrôle l'exécution de l'instance sur le fragment de processus qu'il implémente à son niveau. Un protocole de communication public ou une politique d'interaction est défini(e) afin de coordonner l'exécution de l'ensemble des fragments de processus. Le contrôle décentralisé est utilisé pour les architectures de type 2, type 4, type 5 et type 6 : « Exécution chainée », « Transfert de cas (étendu) » et « Faiblement couplé » du WFIO.

- ***Hiérarchisé*** : l'exécution du WF global est sous le contrôle du système d'un seul partenaire dit principal, mais certaines parties du processus sont déléguées à d'autres partenaires dits secondaires qui contrôlent leurs exécutions, localement. Ce type de contrôle est implémenté dans le cas d'une architecture de type 3 « Sous-traitance ».

- ***Mixte*** : chaque partenaire contrôle l'exécution du WF qu'il implémente (contrôle décentralisé) mais les interactions entre les WF des différents partenaires sont contrôlées de manière centralisée par un système indépendant appelé coordinateur central ; ce qui donne lieu à un contrôle mixte (centralisé/décentralisé). Ce type de contrôle peut être implémenté dans le cas d'une architecture « Transfert de cas » ou « Transfert de cas étendu » lorsque la politique de transfert est complexe et basée sur des règles de transfert non déterministes. En effet dans ce

cas, il est nécessaire de faire intervenir un *coordinateur central* pour la sélection du partenaire devant recevoir l'instance transférée. Cette configuration est proposée dans l'un de nos travaux de recherche (Boukhedouma et al., 2012a).

Notons que le partitionnement horizontal favorise le contrôle décentralisé du processus. Le partitionnement vertical pourrait être lié à un contrôle décentralisé ou centralisé du processus. Cependant, dans un contexte inter-organisationnel le contrôle décentralisé est privilégié car il permet une meilleure préservation de l'autonomie des partenaires.

### C- Type d'Interaction

Les interactions entre les processus des différents partenaires ont lieu dans le but d'échanger les données nécessaires à l'exécution du processus de WFIO. Ces interactions peuvent être de nature synchrone ou asynchrone. En effet, dans l'architecture de type 1 « Partage de charge », les interactions sont de type synchrone entre le système central qui contrôle l'exécution du processus global et les différents participants (les partenaires). Dans l'architecture de type 3 « Sous-traitance », les interactions sont de nature synchrone, puisque le partenaire principal délègue une partie de son processus à un partenaire secondaire qui se charge de l'exécution de cette partie et renvoie le résultat au processus principal pour continuer son exécution. L'architecture de type 2 « Exécution chainée » implémente des interactions asynchrones entre deux processus consécutifs dans la séquence formant le WFIO. L'architecture de type 6 « Faiblement couplée » implémente des interactions asynchrones via un protocole de communication public. Enfin, les architectures de type 4 et type 5 « Transfert de cas » et « Transfert de cas étendu » implémentent aussi un contrôle asynchrone pour le transfert d'instances d'un partenaire à l'autre selon un ensemble de règles spécifiées dans la politique de transfert.

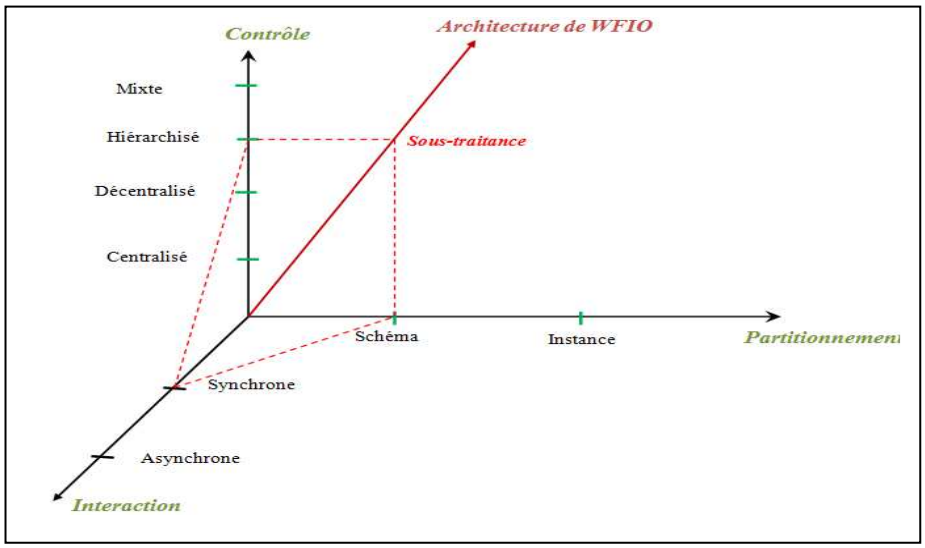



**Figure II.9.** *Dimensions du WFIO*

Comme le montre la figure II.9, l'architecture de « Sous-traitance » par exemple, est définie selon un partitionnement *horizontal* (i.e schéma de processus), un contrôle d'exécution *hiérarchisé* et un mode d'interaction *synchrone*.

### II.3.3.8 Architectures de WFIO et Types de Coopération

Comme nous l'avons déjà souligné, il existe deux types de coopération dans le domaine des WFIO : la coopération planifiée (ou statique) et la coopération dynamique. Dans le chapitre précédent, nous avons établi une comparaison entre les deux types de coopération, selon certains critères qui permettent de les distinguer. Le tableau II.3 montre l'adéquation des différents types d'architectures de WFIO précédemment décrites avec les types de coopération (planifiée/dynamique). En effet, les six architectures s'adaptent parfaitement à une coopération planifiée mais certaines ne peuvent pas s'adapter à une coopération dynamique, c'est le cas des architectures de type 1 et type 4. Les architectures de type 3 et de type 6 s'adaptent bien à la coopération dynamique. Les architectures de type 2 et de type 5 s'adaptent plus ou moins à la coopération dynamique moyennant certaines contraintes relevées sur le tableau.

**TAB II. 3.** *Adéquation des types d'architectures de WFIO avec les types de coopération*

| ***Type d'architecture de WFIO*** | ***Coopération statique*** | ***Coopération dynamique*** |
|---|---|---|
| ***Type 1*** | + | Contrainte de contrôle centralisé et même modèle de processus |
| ***Type 2*** | + | ± Contrainte d'enchaînement séquentiel |
| ***Type 3*** | + | + |
| ***Type 4*** | + | Contrainte de modèle de processus identique (protection du savoir faire) |
| ***Type 5*** | + | ± Contrainte de modèle de processus partiellement identique (protection partielle du savoir-faire) |
| ***Type 6*** | + | + |

### II.3.3.9 Récapitulatif des Architectures de WFIO

Le tableau II.4 résume les différentes architectures de WFIO, en mettant en évidence leurs caractérisations par rapport aux trois dimensions : partitionnement, contrôle et interaction, ainsi que les avantages et les inconvénients de chacune d'elles.

Nous pouvons remarquer que les principaux avantages sont liés à la simplicité de mise en œuvre pour les architectures de type 1 et type 2, à l'équilibrage de charge et l'utilisation rationnelle des ressources pour les architectures de type 4 et type 5, à la possibilité d'exécution parallèle pour l'architecture de type 6 et enfin la préservation du savoir faire pour les architectures de type 2 et type 3. Les principaux inconvénients sont liés à la rigidité de l'architecture (type 1, type 2 et type 6), la non préservation du savoir-faire des partenaires (type 4 et type 5).

**TAB II. 4.** *Récapitulatif des architectures de WFIO*

| ***Type d'architecture*** | **Partitionnement du processus** | **Contrôle** | ***Interactions*** | ***Avantages*** | **Inconvénients** |
|---|---|---|---|---|---|
| *Type 1 : Partage de charge* | | Centralisé | Synchrones | Mise en œuvre simple<br>Contrôle centralisé | Architecture rigide<br>Ne favorise pas la coopération dynamique |
| *Type2 : Exécution chaînée* | Horizontal / Vertical<br>(Schéma/ instance) | Décentralisé | Asynchrones | Interaction simple | Architecture rigide<br>Interactions limitées |
| *Type 3 : Sous-traitance* | Horizontal<br>(Schéma) | Hiérarchisé | Synchrones | Préserve le savoir-faire des partenaires | Interactions limitées |
| *Type 4 : Transfert de cas* | Vertical<br>(Instance) | Décentralisé<br>Ou<br>Mixte | Asynchrones | Favorise l'équilibrage de charge entre partenaires<br>Accélère le temps de réponse<br>Utilisation rationnelle des ressources des partenaires | Ne préserve pas le savoir-faire des partenaires.<br>Nécessité d'augmenter les systèmes de la capacité de transfert des données d'état des instances. |
| *Type 5 : Transfert de cas étendu* | Vertical<br>(Instance) | Décentralisé<br>ou<br>Mixte | Asynchrones | Favorise l'équilibrage de charge entre partenaires<br>Accélère le temps de réponse<br>Utilisation rationnelle des ressources des partenaires | Ne préserve pas totalement le savoir-faire des partenaires.<br>Nécessité d'augmenter les systèmes de la capacité de transfert des données d'état des instances. |
| *Type 6 : Faiblement couplé* | Horizontal<br>(Schéma) | Décentralisé | Asynchrones | Favorise l'exécution parallèle<br>Accélère le temps de réponse | Protocole de communication rigide |

### II.3.4 Méta-modèle de WFIO et perspectives de modélisation

A partir de l'analyse des architectures de WFIO effectuée plus haut, nous proposons un méta-modèle générique de définition de WFIO, comme le montre la figure II.10.

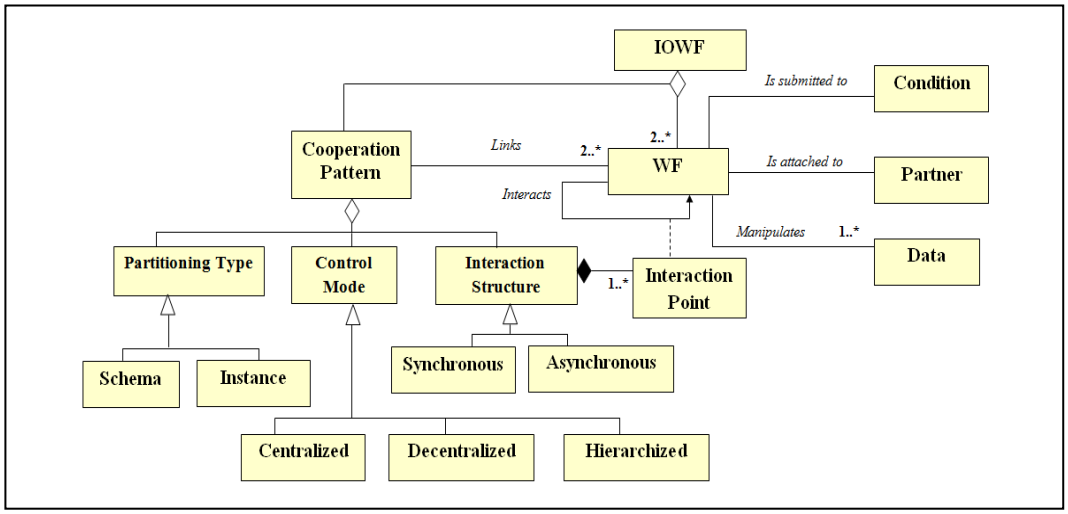



**Figure II.10.** *Méta-modèle générique de WFIO* (Boukhedouma et al., 2014a)

Un WFIO est défini par un ensemble de processus WF (qu'on appellera aussi *fragments de WF*) et un *patron de coopération*. Chaque WF est rattaché à un partenaire métier, manipule des *données* et est soumis à une *condition d'invocation*. Un WF interagit avec un autre WF à travers des points d'interaction définis conformément au patron de coopération. Ce dernier décrit une architecture spécifique de WFIO et est lui-même défini selon les trois dimensions principales du WFIO : le *partitionnement du processus*, le *contrôle d'exécution et la structure d'interaction*.

A travers les concepts illustrés sur le méta-modèle, on peut dire qu'un modèle de processus WFIO couvre quatre axes (ou aspects) principaux: l'axe *processus* (à travers les concepts de WFIO, WF, patron de coopération et condition), *organisation* (le concept de partenaire), *données* (le concept de donnée) et *interaction* (les concepts de structure d'interaction et point d'interaction). Par conséquent, les contraintes de flexibilité ne s'appliquent pas à un seul axe mais couvrent les quatre axes de définition de WFIO.

- **Aspect Organisationnel** : décrit les différents participants au processus inter-organisationnel, il s'agit des différents *partenaires* impliqués dans la coopération, chacun jouant un rôle dans le processus global.
- **Aspect informationnel** : décrit les *données* manipulées par le processus. Dans un contexte inter-organisationnel, ces données peuvent êtres publiques si elles sont nécessaires à l'interaction entre les processus des partenaires ; elles sont privées si elles sont manipulées par un fragment de WF local à un partenaire et doivent rester confidentielles.
- **Aspect interactionnel** : décrit les *points d'interaction* entre les différents fragments de WF impliqués dans une coopération. La communication peut être synchrone ou asynchrone, elle est réalisée à travers l'envoi de messages entre les différents partenaires.
- **Aspect Processus** *:* regroupe les aspects fonctionnel et comportemental, il décrit d'une part le découpage du WF global en *fragments de WF* fournis par les différents partenaires et invoqués conformément à des *conditions*. D'autre part, il décrit l'agencement de ces fragments dans le processus WFIO à travers le *concept de patron de coopération* qui est défini selon les dimensions *partitionnement*, *contrôle* et *interaction*.

### II.3.5 Architecture des Systèmes de Gestion de WF (SGWF)

Selon la WFMC, un SGWF est composé d'un ensemble de modules qui permettent la définition, l'interprétation et le suivi d'exécution des instances de WF (WFMC, 1999). Ces différents modules aux fonctionnalités complémentaires donnent lieu à un modèle de référence des architectures de WFIO, tel montré sur la figure II.11 ci-dessous. Sur cette architecture, on peut distinguer trois parties complémentaires comportant chacune un certain nombre de composants qui doivent communiquer entre eux à travers des interfaces dédiées.

(1) **Une partie responsable de la modélisation de workflow** (Outils de définition de processus) : il s'agit d'outils informatiques basés sur une interface graphique permettant la modélisation d'un WF sous une notation standard (par exemple : jPDL, BPEL, BPMN) ou propriétaire. De tels outils permettent de générer des modèles de WF exploitables. L'interface 1 entre les outils de définition et le moteur de WF permet à ce dernier d'interpréter la définition d'un processus WF au moment de l'exécution des instances.

(2) **Une partie responsable de la gestion complète des modèles réalisés** (Moteur de Workflow , Outils d'administration et de pilotage et l'application cliente de WF ): cette gestion comprend l'exécution des instances de WF, la distribution des tâches aux rôles appropriés, la mise à disposition de l'ensemble des données et des outils nécessaires, la supervision et le contrôle de la cohérence. Le moteur de WF représente le système nerveux de l'architecture de référence ; l'interface 2 permet au moteur de WF de communiquer avec les applications clientes afin de présenter les bons de travail aux acteurs chargés de leur exécution, selon les rôles qu'ils tiennent. L'interface 5 assure l'échange de données d'exécution entre le moteur de WF et les outils d'administration et de pilotage qui se chargent du suivi d'exécution des instances.

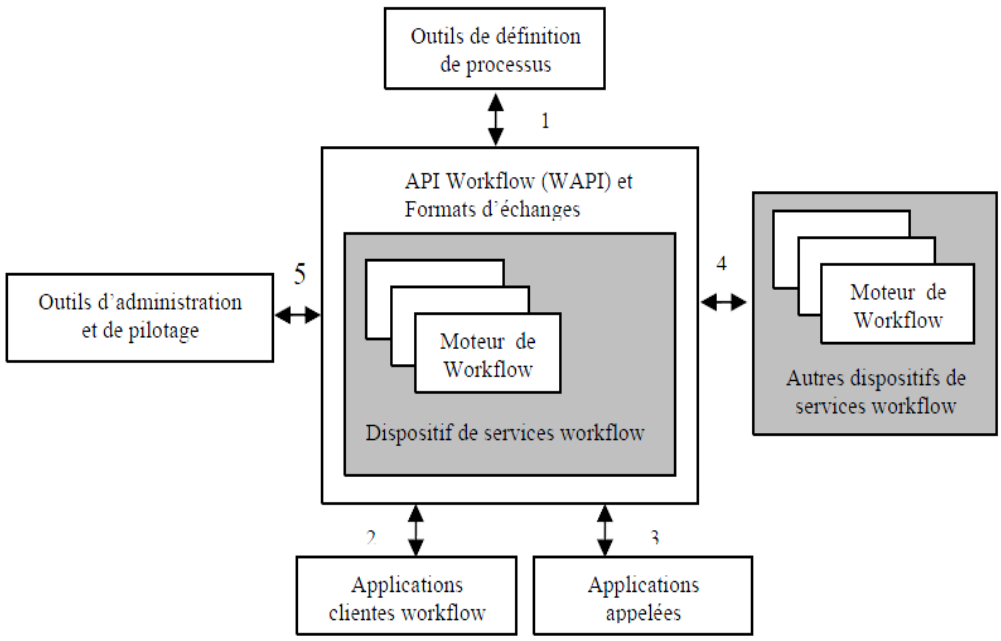



**Figure II.11.** *Architecture d'un SGWF (WFMC, 1999)*

(3) **Une partie responsable de la gestion des connexions externes avec d'autres applications ou modules** (Applications appelées, Autres moteurs de WF) : cette partie contient des fonctions API qui assurent ce genre de connexions. Ces fonctions permettent d'adapter le système et ses services en fonction de leurs besoins spécifiques. L'interface 3 permet au moteur de WF

d'appeler des applications externes (service de messagerie, application de production, envoi d'une télécopie,… etc.) afin de compléter l'exécution des activités. L'interface 4 assure l'interopérabilité entre systèmes de WF hétérogènes.

En résumé, si la technologie WF a considérablement servi à l'automatisation des processus d'entreprises dès l'aube des années 90, en leur permettant des gains considérables en matière de temps, délais et coûts, elle a montré son inadéquation pour la prise en charge des changements qui peuvent affecter les processus métiers, évoluant dans un environnement dynamique. De là, sont nés des outils de WF *flexibles* permettant de prendre en charge des changements au niveau des modèles de processus ou des instances en cours d'exécution. Néanmoins, ces outils restent propriétaires et valables surtout pour la prise en charge des changements *intra-organisationnels* affectant les processus d'une seule entreprise (Saikali, 2001). Cependant, la caractéristique de flexibilité des WF s'avère d'autant plus importante dans le cas de processus métiers inter-organisationnels où les besoins d'interopérabilité, d'adaptabilité et d'évolutivité se font ressentir davantage puisqu'il s'agit de mettre en commun des processus métiers issus de différentes organisations autonomes, distribuées et hétérogènes. Dans un tel contexte, les outils de WFIO présentent des limites vis-à-vis des contraintes d'interopérabilité, de protection du savoir-faire et surtout de la contrainte de *flexibilité* qui nécessite des opérations coûteuses à chaque changement.

Heureusement, les années 2000 ont connu l'apparition d'un nouveau style architectural permettant de répondre plus efficacement aux contraintes su-citées : il s'agit des architectures orientées services (SOA) qui ont permis, grâce à leurs bénéfices remarquables, une véritable relance des outils et des plateformes permettant la mise en oeuvre d'applications collaboratives, notamment dans le domaine de la coopération B2B. Par suite, un ensemble de travaux de recherche tels que (Leymann et al., 2002), (Crusson, 2003), (Gorton et al., 2009) se sont focalisés sur la combinaison de la technologie WF, du BPM[3] et des architectures orientées services pour proposer des approches support de solutions métier inter-organisationnelles, en tirant profit aussi bien des avantages du WF que ceux des SOA.

## II.4 L'Architecture Orientée Service (SOA)

L'architecture orientée service (Service Oriented Architecture – SOA) est née du besoin d'intégrer les applications en éliminant les contraintes de contrôle centralisé et de sécurité représentant l'inconvénient majeur des « middlewares » (Serain, 97), particulièrement problématique dans le cas des applications B2B. Le paradigme SOA prend de plus en plus d'ampleur dans l'ingénierie des procédés ainsi que dans la technologie des logiciels (Ferguson, 2005). Entre autres, la SOA représente une nouvelle manière d'intégrer et de manipuler les différentes briques et composants applicatifs d'un système informatique et de gérer les liens qu'ils entretiennent. Elle repose sur la réorganisation des applications en ensembles fonctionnels appelés *services* (Justin et al., 2002) et présente des informations détaillées sur ces services pour qu'ils soient facilement utilisés par les clients. La SOA présente plusieurs avantages grâce aux principales caractéristiques d'interopérabilité et de couplage faible des services. De ce fait, cette architecture s'est imposée aujourd'hui comme un thème majeur pour les systèmes d'information d'entreprise. Plus qu'une nouvelle technologie ou méthode, c'est la convergence de plusieurs approches existantes, et l'émergence d'un style d'architecture et de gouvernance des systèmes d'information. Dans la suite, nous passons en revue les principes et les concepts de base du paradigme SOA et nous mettons l'accent sur ses avantages notamment bénéfiques dans le cas d'une coopération B2B.

[3] Business Process Management

## II.4.1 Principes et Concepts d'une SOA

### II.4.1.1 Principes de base d'une SOA

L'architecture orientée service se base sur les principes suivants (Raymond, 2011):

- *Diviser pour régner* : substituer la découpe strictement applicative par une structuration en composants plus réduits et potentiellement plus simples à faire évoluer.
- *Alignement métier* : construire et organiser le système à partir des réalités métier, qui doivent se retrouver dans ses constituants.
- *Neutralité technologique* : assurer une indépendance totale entre les interfaces et les implémentations. L'application qui utilise un service ne doit pas être contrainte ni par sa technologie d'implémentation, ni par sa localisation (potentiellement distribuée).
- *Mutualisation* : favoriser la réutilisation de services métiers par plusieurs lignes métier ou applications. Permettre la construction de services de haut niveau par combinaison de services existants.
- *Automatisation des processus métiers* : isoler la logique des processus métiers sur des composants dédiés, qui prennent en charge les enchaînements et les échanges de flux d'information.
- *Echanges orientés document* : les informations échangées par les services possèdent une structure propre, guidée par les besoins métiers. On privilégie la transmission de contenus complets et utilisables au profit d'accès direct aux structures de type objet ou relationnel.

### II.4.1.2 Définitions

Une architecture orientée service rassemble les grandes applications de l'entreprise en services interopérables et réutilisables. En effet, l'architecture orientée service décrit les composants d'un système et la manière de leur interaction à un haut niveau. L'idée de base est donc de construire une architecture logicielle globale décomposée en services qui correspondent aux processus métiers de l'entreprise. L'objectif majeur de cette architecture est d'autoriser les applications à communiquer et à travailler ensemble quelle que soient leurs plates-formes respectives, et ce par le biais des services.

- **Définition II.10-**SOA

*« L'architecture orientée service constitue un style d'architecture basé sur le principe de séparation de l'activité métier en une série de services. Ces services peuvent être assemblés et liés entre eux selon le principe de couplage lâche pour exécuter l'application désirée. Ces services sont définis à un niveau supérieur de la traditionnelle approche composants»* [4].

D'autres définitions existent telles que :

- **Définition II.11-**SOA

*« L'architecture orientée services (SOA) est le terme utilisé pour désigner un modèle d'architecture pour l'exécution d'applications logicielles réparties. Son principe consiste à offrir des services qui réalisent chacun une tâche bien précise, et des applications leur faisant appel si elles en ont besoin. Ces applications elles-mêmes peuvent être déployées comme des services réalisant un grand rôle* » (Walid, 2011).

[4] http://www.Gartner.com

L'objectif est donc de décomposer une fonctionnalité en un ensemble de fonctions basiques, fournies par des composants (services) et de décrire finement le schéma d'interaction entre ces services. Ces applications-services sont exécutées sur des plateformes hétérogènes et sur des réseaux d'information distribués, et fournissent des fonctionnalités à d'autres entités du réseau.

### II.4.1.3 Le Concept de Service

Le service est le concept clé de la SOA. Il rassemble une ou plusieurs activités métier réelles que l'on peut associer à des opérations, c'est-à-dire des fonctionnalités offertes par le service. Un service peut être défini comme suit :

- **Définition II.12- Service**

*« Un Service est un composant logiciel distribué, exposant des fonctionnalités à forte valeur ajoutée d'un domaine métier »*[5].

Un service est fourni par le fournisseur de services, qui ne connait pas forcément ses utilisateurs (les consommateurs de services). Il est composé d'un ensemble de politiques et de contrats qui représentent les contraintes posées sur son utilisation. Ce sont des contraintes posées sur les conditions d'invocation des opérations, sur leurs effets ainsi que sur les caractéristiques non fonctionnelles du service (qualité de service, sécurité, etc.). Ces différentes contraintes sont réparties dans l'interface du service et la description du service. L'interface du service définit comment déclencher l'exécution des opérations tandis que la description du service regroupe les contraintes qui garantissent la bonne exécution de ces opérations.

### II.4.1.4 Caractéristiques d'un Service

L'adoption d'une architecture orientée service en entreprise ne nécessite pas le remplacement total de l'architecture déjà en place, car elle permet de réutiliser l'existant et de le transformer en des services plus agiles qui doivent répondre aux critères suivants (Heubès, 2009) :

1. *Contrat standardisé* : entre le fournisseur de service et le consommateur de service. Le contrat de service détaille les conditions d'utilisation du service sous forme de pré et post conditions, protocoles, et contraintes. Il en existe 3 types :
   - Contrat lié à la syntaxe du service
   - Contrat lié à la sémantique du service, les règles et contraintes d'usage.
   - Contrat lié la qualité de service
2. *Couplage lâche* : la communication doit se faire à travers des messages, suivant un protocole de communication unifié, offrant ainsi un couplage faible permettant de changer facilement la configuration d'un service.
3. *Abstraction* : le fonctionnement interne du service doit être invisible aux utilisateurs, il fonctionne donc sous le principe de la « boite noire ».
4. *Découvrabilité* : le fournisseur de services doit déposer et mettre à jour ses services dans un annuaire, l'enrichir par un ensemble de métadonnées dans le but de faciliter la recherche aux consommateurs de services.
5. *Réutilisabilité* : un service doit être conçu de manière à ce qu'il puisse être utilisé dans différentes applications.

[5] http://blog.xebia.fr/2009/03/04/soa-du-composant-au-service-le-contrat-standardise/ Dernière visite le 20.08.2015.

6. *Autonomie* : un service doit disposer de l'ensemble des informations nécessaires à son exécution et ne doit dépendre d'aucun autre service (couplage lâche).
7. *Sans état* : entre deux instanciations du même service par le même client, aucune donnée ou valeur entrée n'est sauvegardée, autrement dit il n'y a pas de conservation d'information, et cela afin de minimiser au maximum la consommation des ressources.
8. *Composition de services* : un service doit fonctionner de manière modulaire et non pas intégrée et si ce dernier est complexe il faut le décomposer en sous-services moins complexes pour assurer son autonomie.

### II.4.1.5 Les couches d'une SOA

L'architecture SOA se caractérise par une répartition des processus métiers, des présentations, des logiques applicatives et des données en trois couches distinctes et faiblement couplées :

- *La Couche Présentation :* c'est la partie de l'application visible à l'utilisateur et qui lui permet de manipuler des données ou effectuer les opérations qu'il souhaite. elle peut être un Service Web (Monfort et al., 2003), une Servlet ou une page JSP.
- *La couche métier* : cette partie décrit le fonctionnement d'une application, elle implémente la logique des opérations effectuées sur les données en fonction des requêtes des utilisateurs.
- *La couche accès aux données :* elle contrôle l'accès aux données stockées dans les fichiers, bases et entrepôts de données du système, ces données peuvent être issues d'un autre système interagissant avec le premier.

### II.4.1.6 Composants d'une SOA

Les principaux composants et standards de la SOA sont illustrés dans la figure II.12 ci-dessous (Fournier-Morel et al., 2006):

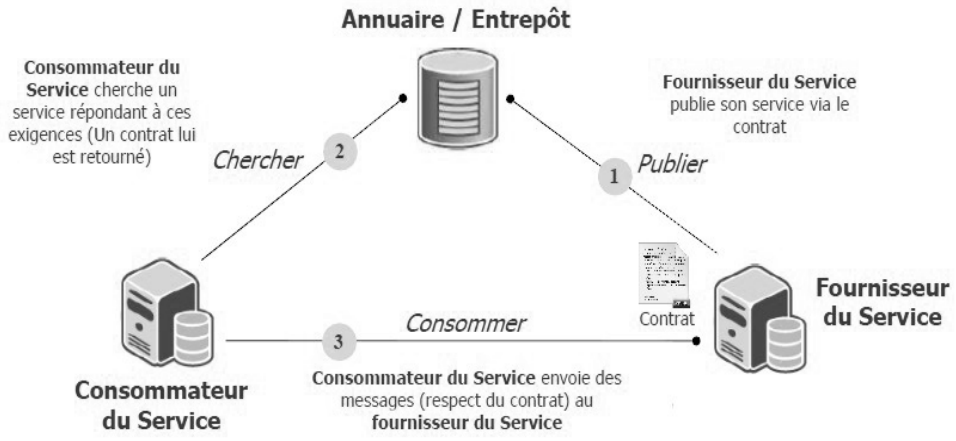



**Figure II.12.** *Composants d'une SOA*

- *Le fournisseur du service* : appelé aussi « prestataire de service », désigne d'un point de vue conceptuel, la personne ou l'organisation responsable juridiquement du service. D'un point de vue opérationnel, les fournisseurs de services peuvent désigner le ou les serveurs qui hébergent les services déployés.
- *Le client* (consommateur du service ou demandeur de services) : représente une personne ou une organisation, cliente potentielle de services. Du point de vue opérationnel, il représente l'application cliente qui invoque le service.

- *L'annuaire de services* : appelé aussi registre, rassemble les descriptions de tous les services publiés par les fournisseurs de services, afin de faciliter la tâche de recherche du service adéquat au client.

## II.4.2 Les Services Web

Les services Web constituent l'instanciation la plus importante d'une SOA dans le domaine industriel (Baghdadi, 2012). En effet, les entreprises encapsulent les tâches automatiques comme des services logiciels pour les rendre visibles sur Internet. Les services Web sont accessibles à l'intérieur et à l'extérieur d'une entreprise. Ils communiquent via des standards de communication, éliminant de suite, les frontières des plates-formes technologiques ou des logiciels en se reposant sur des méthodes de communication virtuellement disponibles sur tous les systèmes.
Selon, le W3C (World Wide Web Consortium), un service Web est un système logiciel conçu pour supporter les interactions entre applications à traves le réseau. D'autres définitions existent comme :

- **Définition II.13-**Service Web

**«** *Un service Web est une agrégation de fonctionnalités publiées pour être utilisées, il utilise Internet comme conduit pour réaliser une tâche, il est aussi semblable à un processus métier virtuel qui définit des interactions au niveau application.* **»** (Justin & al, 2002)

- **Définition II.14-**Service Web

*« Les services Web sont des applications auto-descriptives modulaires et faiblement couplées qui fournissent un modèle simple de programmation et de déploiement d'applications, basé sur des normes, et s'exécutant au travers de l'infrastructure web. Les services Web réalisent des fonctions allant des simples requêtes aux processus métiers sophistiqués »* (Monfort et al., 2003).

### II.4.2.1 Caractéristiques des Services Web

Un certain nombre de caractéristiques représentant en même temps des atouts de la technologie des services Web sont recensés dans la littérature à travers une panoplie de travaux, nous rappelons celles présentées dans (Babin et al., 2003).

- *L'interopérabilité :* c'est la capacité des services Web à interagir avec d'autres composants logiciels via des éléments XML et utilisant des protocoles de l'Internet.
- *La simplicité :* les services Web réduisent la complexité des branchements entre les participants. Cela se fait en ne créant la fonctionnalité qu'une seule fois plutôt qu'en obligeant tous les fournisseurs à reproduire la même fonctionnalité à chacun des clients selon le protocole de communication supporté.
- *Le couplage faible:* l'architecture modulaire des services Web, combinée au faible couplage des interfaces associées, permet l'utilisation et la réutilisation de services qui peuvent être facilement recombinés à différentes autres applications.
- *L'hétérogénéité :* les services Web permettent d'ignorer l'hétérogénéité entre les différentes applications. En effet, ils décrivent comment transmettre un message (standardisé) entre deux applications, sans imposer comment construire ce message.
- *L'auto-descriptivité :* les services Web ont la particularité d'être auto-descriptifs, c'est à dire capables de fournir des informations permettant de comprendre comment les manipuler. La capacité des services à se décrire par eux-mêmes permet d'envisager l'automatisation de l'intégration de services.

### II.4.2.2 Apports des Services Web pour le B2B et limites

Pour Le W3C, la modélisation des processus métiers correspond à l'évolution naturelle des services Web. Un processus métier est considéré alors comme un assemblage de services Web techniques et métiers. On parle de « composition » de services Web, « d'orchestration » d'activités et parfois de « chorégraphie » de services. Pour le W3C, le concept de Workflow décrit uniquement l'enchaînement ou représente la composition des services Web.
Afin de faciliter les échanges B2B entre des partenaires commerciaux à travers le Web, le W3C a défini différentes couches composant un service Web telles que la couche de découverte de services, la couche de description de services et la couche la plus haute, appelée **« couche métier »** représentée par ebXML, BPML, BPEL ou BizTalk. Elle permet de mutualiser les fonctions utiles à certains métiers de l'entreprise. Les principaux avantages tels énoncés dans (Richardson et al., 2007) sont :

- **Les services Web permettent d'automatiser facilement les processus métiers** : ils permettent d'intégrer, gérer et automatiser rapidement les processus métiers intra et interentreprises en échangeant des informations au format XML. L'assemblage des différents services peut être orchestré par un moteur de WF.
- **Les services Web facilitent l'interopérabilité entre systèmes et plates-formes hétérogènes :** ceci découle immédiatement de l'indépendance de services, des plates-formes sur lesquelles ils s'exécutent.
- **Les services Web facilitent l'intégration d'applications et de services :** cette intégration a pour objectif de décloisonner les différentes applications informatiques du système d'information de manière à :
  - Fluidifier les flux entre les applications ;
  - Réduire le temps de latence de ces flux ;
  - Faciliter l'évolutivité de ces flux.

Il faut noter qu'à cet ensemble d'avantages des services Web, s'ajoutent quelques limites dont : (i) la non prise en compte de l'aspect sémantique, en effet les services Web ne traitent que la syntaxe puisqu'ils utilisent le standard XML qui permet de décrire seulement l'aspect structurel d'un document. (ii) la difficulté de composer des services complexes avec d'autres services existants. (iii) La nécessité de maintenir les démarches commerciales auprès des partenaires. En effet, pour des fonctions clés d'une organisation, on ne peut se contenter de la découverte automatique d'une relation commerciale. (iv) Le problème de sécurité et confiance reste posé.

## II.4.3 La composition de services

La composition de services est l'un des atouts les plus importants de l'architecture SOA. En effet, il s'agit d'un mécanisme permettant de construire des services à forte valeur ajoutée dans un domaine donné, par agencement et réutilisation de services existants. D'un point de vue opérationnel, la composition de services est un processus de raffinement permettant de passer d'une spécification abstraite de la composition vers une description exécutable. Les étapes pour la création d'une composition de services ont été définies dans (Yang et al., 2004) comme suit :

**1.** Une phase de définition permettant de spécifier d'une manière abstraite la composition de services. Dans cette phase, il faut identifier en premier la fonctionnalité devant être fournie par la composition ainsi que, la fonctionnalité devant être apportée par les différents participants. Finalement, les interactions entre les participants sont spécifiées.

**2.** Une phase de planification servant à déterminer comment et à quel moment les services seront exécutés. Dans cette phase, la conformité et la compatibilité des services doivent être vérifiées.

**3.** Une phase de construction fournissant une composition concrète et sans ambiguïtés prête à s'exécuter. Uniquement les services potentiellement disponibles à l'exécution, restent à déterminer.

**4.** Une phase d'exécution implémentant les liaisons avec les services disponibles et exécutant la composition ensuite.

Différentes approches de composition de services ont été définies dans la littérature, chacune concentrée sur un critère donné. En effet, (Foster et al., 2003) considèrent le degré d'automatisation comme principal critère et classent la composition de services selon trois types : manuelle, semi-automatique et automatique. D'autres auteurs comme (Casati et al., 2001) considèrent le degré de flexibilité comme principal critère de la composition et distinguent deux approches de composition: une approche statique et une approche dynamique. Selon (Ferguson, 2005), le contrôle d'une composition de services par rapport au flux de services peut être extrinsèque ou intrinsèque à ces derniers. Par rapport à cela, deux styles de composition de services sont distingués : une composition structurelle ou une composition orientée procédé de laquelle découlent les deux grandes approches de composition, à savoir l'orchestration et la chorégraphie de services. La figure suivante résume ces différents types de composition.

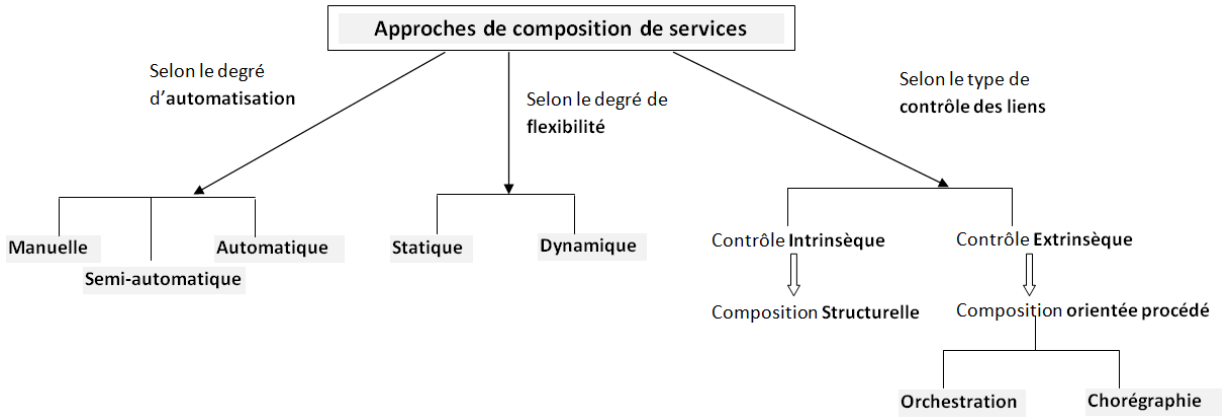



**Figure II.13.** *Approches de composition de services*

### II.4.3.1 Classification par degré d'automatisation

Dans cette catégorie, sont distinguées trois types de composition :

- **La composition manuelle :** suppose que l'utilisateur ne se sert d'aucun outil dédié et génère la composition à la main via un éditeur de texte.

- **La composition semi-automatique :** parfois, les services qui participent à une composition ne sont pas obligatoirement fixés à l'avance. Par exemple, la non-disponibilité d'un service nécessite de le remplacer par un autre assurant les mêmes fonctionnalités. La composition de services est alors guidée par des suggestions dont l'utilisateur peut en tenir compte dans la définition de son propre processus, en utilisant des outils graphiques pour la modélisation et la conception de services Web composites. Les travaux traitant de la composition semi-automatique de services sont par exemple (Arenaza, 2006) et (Lopez-Velasco, 2009).

- **La composition automatique** : prend en charge tout le processus de composition et le réalise automatiquement, sans la moindre intervention de l'utilisateur. Durant cette dernière décennie, de nombreux travaux comme (Traverso et al., 2004), (Pistore et al., 2005), (Benhassine et al., 2006), (Bourdon, 2007), (Elfidoussis et al., 2014) ont porté sur l'automatisation de la composition des services.

### II.4.3.2 Classification par degré de flexibilité

Dans cette catégorie, on trouve deux types de composition

- **La composition statique** : passe par une phase de spécification durant laquelle les services sont identifiés, interconnectés, compilés et déployés pour être utilisés. Cela devrait fonctionner tant que l'environnement du service (par exemple, la disponibilité) n'a pas changé. Par contre, si de nouveaux services sont disponibles ou remplacés par d'autres, des incohérences seront relevées. Dans ce cas, il est inévitable de changer ou modifier l'architecture du système et de le relier à nouveau à d'autres services, ou, dans le pire des cas, de changer la définition du processus et concevoir à nouveau le système (Ait-Cheikh-Bihi, 2012).

- **La composition dynamique** : prend en compte les services disponibles, leurs fonctionnalités et l'objectif à atteindre que ce soit avant ou pendant l'exécution des services. La composition dynamique de services Web est la technologie clé de l'implémentation de l'architecture SOA. Cependant, la sélection de services représente un réel problème dans ce type de composition. La complexité de la sélection des services inclut trois facteurs principaux, le premier concerne le grand nombre de changements dynamiques des instances de services disponibles pour composer un service. Le deuxième facteur concerne les différentes possibilités d'intégration des composants d'une instance de service, au processus d'un service complexe. Tandis que le troisième facteur correspond aux diverses exigences de performance d'un service complexe (c.-à-d. introduction de critères de la qualité de service tels que le délai, le prix et la fiabilité) (Qiqing et al., 2009), (Tholkappia et al., 2014).

Les deux compositions, statique et dynamique, dépendent du paramètre temporel. En effet, dans une composition statique, les services sont connus au préalable, c'est-à-dire au moment de la conception, contrairement à la composition dynamique où les services sont découverts et sélectionnés au moment de l'exécution.

### II.4.3.3 Classification selon le contrôle des liens

Cette dernière catégorie considère le contrôle des liens entre les services comme principal critère de la composition, (Ferguson, 2005) distingue deux types de composition : une composition structurelle où le contrôle des liens est intrinsèque aux services et une composition orientée procédé où le contrôle de liens est extrinsèque aux services.

#### A. La composition structurelle

Dans ce mode de composition, les composants qui fournissent les services sont clairement identifiés, chaque composant définit explicitement ses interfaces fournies comme celles requises. La composition structurelle exprime des liens entre couples d'interfaces fournies et requises par deux composants. Le formalisme pour indiquer cet assemblage dépend de l'approche suivie, il est connu comme langage de description d'architecture ADL (*Architecture Description Language*).
La logique de contrôle, exprimant comment et à quel moment les opérations des services composés doivent être invoquées est implicite et répartie entre les différents composants. La figure II.14 ci-après, présente un exemple de composition structurelle.


**Figure II.14.** *Schéma d'une composition structurelle* (Pedraza, 2009)

La logique qui régit le contrôle de flux indiquant comment sont consommés les services fournis par les composants C2 et C3, se trouve à l'intérieur de l'implémentation du composant C1 (C1.java).

### B. La composition orientée procédé

Dans cette approche, la composition est spécifiée en utilisant un modèle de procédé décrivant la logique de coordination et d'exécution des services utilisés par le service composite. Le principal avantage de cette approche est le fait que la logique soit externe par rapport aux services composés car elle est décrite en dehors de ces services. Dans ce cas, la logique de contrôle de la composition est plus explicite.

Le modèle de procédé est représenté généralement par un graphe, où chaque nœud correspond à l'invocation d'une opération d'un service, et les arcs servent à exprimer l'ordre d'enchaînement de ces invocations. La spécification de la composition est faite dans un langage qui est interprété par un moteur d'exécution. La responsabilité du moteur est d'invoquer les services dans l'ordre spécifié, de faire le routage des données, de maintenir et gérer l'état des activités et du composite, ainsi que de gérer les situations d'exception (Pedraza, 2009).

Ce mode de composition est subdivisé en deux catégories qui sont l'*orchestration* de services et la *chorégraphie* de services ; il faut noter que la différence majeure entre l'orchestration et la chorégraphie réside dans le fait que l'orchestration offre une vision centralisée de la composition, alors que la chorégraphie offre une vision globale et plus collaborative de la coordination.

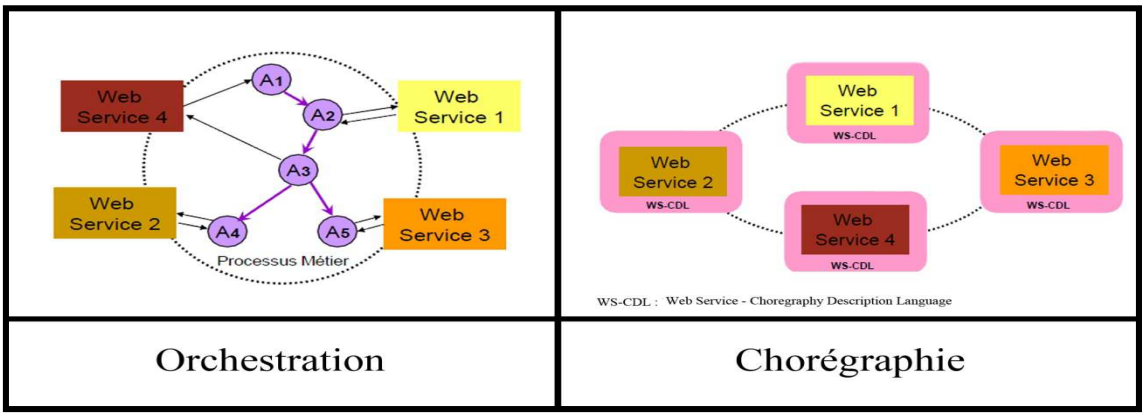



**Figure II.15.** *Orchestration et Chorégraphie de services*

### B.1 L'orchestration de services

Une orchestration de services permet de décrire l'enchaînement des services selon un canevas prédéfini, et de les exécuter. Généralement on parle d'orchestration quand il y a un processus principal (l'orchestrateur) qui prend le contrôle du déroulement de la composition et coordonne les différentes opérations des différents services. L'orchestration est une méthode très utilisée, car il est relativement simple d'écrire un processus centralisé gérant l'invocation de services. Dans le cas d'une réutilisation de services basiques, seule la partie centrale est à développer. L'orchestration

décrit donc l'enchainement et les interactions entre plusieurs services afin de fournir une nouvelle fonctionnalité offerte par le nouveau service dit « composite ». Parmi les langages utilisés pour l'orchestration de services, on peut citer :

**WSFL** (*Web Service Flow Language*) (Havey, 2005) proposé par IBM, permet de spécifier comment mettre en oeuvre un modèle de processus métier en utilisant l'architecture des services Web. En effet, WSFL s'intègre avec UDDI et WSDL pour la sélection des services Web. Le processus métier est décrit en utilisant WSDL et représenté par un modèle de flux XML, qui définit les activités implémentées sous le format des services Web, et définit également les séquences de flux de données entre les activités.

**XLANG** (Monfort et al., 2003) est un autre langage de composition de flux développé initialement par Microsoft pour la création des processus métiers et l'interaction entre les fournisseurs de services. Ce langage utilise WSDL pour décrire l'interface du service du processus. Il étend le langage WSDL en ajoutant des capacités de comportement (opérateurs de contrôle de flux : séquence, parallélisme, choix) pour définir l'enchainement et l'ordre d'exécution des activités dans le processus.

**BPML** (Business Process Modeling Language) (Havey, 2005): dialecte XML qui permet de décrire tous les paramètres techniques nécessaires à l'exécution d'un processus au sein d'un moteur d'orchestration, fondé sur le concept de machine transactionnelle à états finis. BPML est un métalangage voué à la modélisation des processus métiers, tout comme XML est un métalangage dédié à la description des données.

**BPEL4WS** (Jordan et al., 2006) : développé par un ensemble de firmes telles qu'IBM, BEA et Microsoft, est une dérivée à la fois de WSFL et XLANG. C'est un langage permettant de décrire les interactions des services Web comme étant un processus métier. La définition de ce dernier implique la spécification précise du comportement visible des messages échangés de chaque partie impliquée dans ce processus, sans révéler leur mise en oeuvre interne. De cette façon, BPEL4WS définit comment les interactions multiples des services sont coordonnées afin d'atteindre les objectifs métiers, ainsi que l'état et la logique nécessaire pour cette coordination. Il suppose que les services sont décrits par un langage basé sur XML comme WSDL.

### B.2 La Chorégraphie de services

La chorégraphie décrit une collaboration entre services pour accomplir un certain objectif. A l'inverse de l'orchestration, la chorégraphie ne repose pas sur un coordinateur central. C'est une coordination décentralisée où chaque participant est responsable d'une partie du workflow, et connaît sa propre part dans le flux de messages échangés. Suivant cette approche, l'ensemble des activités est achevé comme la composition d'interactions *peer-to-peer* (P2P, qui est un modèle de réseau informatique proche du modèle client-serveur mais où chaque client est aussi un serveur) entre les services participants. Plus précisément, la chorégraphie définit les règles et les interactions de collaboration entre deux ou plusieurs services. De ce fait, la chorégraphie est plus collaborative que l'orchestration et permet à chaque service appelé de décrire son rôle dans la composition ou l'interaction. Les langages définis pour cette approche sont décrits dans ce qui suit :

**WS-CDL** (*Web Service Choreography Description Language*) (Kavantzas et al., 2005) est l'un des langages de description de chorégraphie pour les services et permet la spécification des protocoles P2P où chaque partie est autonome et n'a pas d'hiérarchie vis-à-vis des autres parties.

WS-CDL peut être considéré comme un complément à un langage d'orchestration tel que BPEL en lui apportant un modèle global nécessaire pour assurer la cohérence des interactions des services partenaires, et pour prendre en considération les éventuelles situations de compétition entre partenaires. Plus précisément, WS-CDL est une collection d'activités qui peut être effectuée par un ou plusieurs participants. Il existe trois types d'activités dans WS-CDL, les activités de contrôle de flux (séquence, parallèle et choix), les activités de l'unité de travail et les activités de base. L'unité de travail décrit une exécution répétée possible ou conditionnelle d'une activité. Les activités de base dans WS-CDL, décrivent les points dans la chorégraphie quand un rôle exécute une action qui n'affecte pas le reste de la chorégraphie.

**WSCI** (*Web Service Choreography Interface*) (Havey, 2005) est un langage de description basé sur XML qui permet de chorégraphier le flux des messages entre les services Web. Ce langage décrit le flux des messages échangés par un service Web dans un processus particulier. Il permet notamment de décrire les messages collectifs échangés entre les services Web qui interagissent, en fournissant une vue globale du processus impliquant les multiples services Web. WSCI est un langage basé sur une structure qui permet de traiter l'observable externe plutôt que la définition interne du comportement du service, qui est exprimée en termes de dépendances temporelles et logiques entre les messages échangés.

**BPMN** (Business Process Modeling Notation) (BPMS, 2005): est une norme de notation pour la modélisation de processus. BPMN est soutenu par l'OMG/BMPI *(Object Management Group/ Business Process Management Initiative)* depuis leur fusion en 2005. Son objectif est de fournir un cadre permettant de décrire un processus d'une manière commune à tous les utilisateurs et ce, indépendamment de l'outil utilisé et supportant la norme. Les objets de base de BPMN sont : les tâches (service, tâche manuelle,…), les évènements et les connecteurs (branchements) en plus des objets de connexion (messages, associations,…), les couloirs (swimlanes) et les artefacts (données, groupes, annotations). La norme BPMN définit deux concepts pour organiser les processus : les orchestrations et les chorégraphies. La version 2.0 de BPMN fournit des diagrammes d'interaction qui consistent à décrire le comportement des participants au sein d'un processus collaboratif, et ce, en se focalisant sur les interactions coordonnées entre deux ou plusieurs participants. BPMN 2.0 dispose de deux diagrammes pour modéliser les interactions d'une chorégraphie : le diagramme de collaboration et le diagramme de chorégraphie. Le premier est disponible en BPMN 1.x et a été renforcé dans la deuxième version, tandis que le second est apparu dans la nouvelle version.

**BPEL4Chor** (Decker et al., 2007), (Decker et al., 2009) est un langage de chorégraphie formant une couche additionnelle au dessus du standard BPEL (OASIS, 2007). BPEL4Chor spécifie les chorégraphies de services orchestrés (i.e les chorégraphies de workflows), suivant l'approche du modèle de comportement d'interfaces interconnectées. Le langage BPEL4chor définit trois types d'artefacts :

- Une description du comportement des participants « The *Participant Behavior Descriptions* (PBD) » : utilise le procédé BPEL abstrait pour spécifier le comportement de communication entre les participants de la chorégraphie. Seules les activités permises conformément au « *Process Profile for Observable Behavior* » spécifié dans le standard BPEL (OASIS, 2007) peuvent être utilisées pour la description du comportement de communication. De plus, BPEL4Chor interdit l'utilisation des attributs « partnerLink », « portType » et « operation »; ce qui permet de découpler les PBD et la description des messages échangés de leur réalisation technique concrète.

- Une description des aspects structurels de la chorégraphie dans l'artefact typologie des participants (Participant Typologie) qui renferme la description des types de participants, leurs références, et les liens de messages « *Participant Types*, *Participant References*, *Message Links* ». « Participant Types » sont spécifiés dans le procédé BPEL abstrait lié à la description du comportement du participant. « Participant References » désignent la représentation de la chorégraphie des participants dans BPEL4Chor. « Message Links » spécifie les participants qui peuvent échanger des messages entre eux. Par conséquent, « Participant Topology » relie les différents PBD dans une chorégraphie.

- Le troisième artefact de BPEL4chor est le « *Participant Grounding* ». BPEL4Chor offre la possibilité de séparer les informations techniques telles que les formats de messages, les ports types et les opérations du comportement abstrait de la communication, c'est-à-dire de « Participant Behavior Description » et « Participant Topology ». Les informations techniques sont spécifiées dans le fichier de connaissances (Grounding file).

Il faut noter que ces langages ne permettent pas de réaliser une composition dynamique car tous ne supportent pas l'aspect sémantique de la composition, nécessaire dans le cas de la découverte de services (Benatallah et al., 2005). Pour cela, des langages basés sur le web sémantique tels que DAML-S et OWL-S ont été proposés. DAML-S (DARPA Agent Markup Language for Web Services)[6] est un langage basé sur XML et permet de faciliter l'automatisation des tâches des services Web. Il met à disposition des fournisseurs de services un ensemble de moyens et de constructions basés sur le langage de balisage pour la description des propriétés et des capacités de leurs services Web, d'une façon non ambiguë et interprétable par l'ordinateur. OWL-S (*Ontology Web Language*)[7] est une extension de DAML-S visant à mettre en place un cadre sémantique pour les services Web ; il se base sur l'ontologie pour fournir la description des services Web, réaliser la découverte, l'invocation, la composition et l'interopérabilité automatique des services.

### II.4.4 Avantages et Limites de la SAO pour les processus métiers

Comme nous l'avons déjà souligné dans le chapitre I, dans un contexte d'intégration ou d'interopérabilité, la SOA contribue à l'évolutivité et la pérennité d'un système d'information et offre beaucoup d'avantages, notamment pour le développement d'applications B2B. Les principaux avantages se résument dans la flexibilité, l'ouverture, la réutilisation, l'urbanisation et l'uniformisation du système d'information qui devient structuré en services, en plus d'une intégration aisée faiblement couplée et à faible coût. Ce qui permet finalement, une encapsulation de la complexité des processus métiers et des applications dans des composants réduits que sont les services et donc une amélioration de l'agilité du SI.

Par ailleurs, comme n'importe quelle approche, l'adoption du paradigme SOA en entreprise, présente des inconvénients liés principalement aux coûts de conception et de développement initiaux, qui s'avèrent conséquents, en plus de la nécessité d'appréhender de nouvelles technologies qui exige un temps supplémentaire d'adaptation. A cela, s'ajoute l'intégration (ou migration) de modules logiciels non SOA déjà existants et la lenteur d'exécution due principalement à l'ajout d'une couche logicielle supplémentaire et au niveau de granularité du service. Enfin, le problème de sécurité dépend de l'infrastructure elle-même (pour le cas des services Web, il est conseillé d'utiliser HTTPS comme protocole de communication).

[6] http://www.daml.org/services/daml-s/0.9/daml-s.html Dernière visite le 20.08.2015
7 http://www.ai.sri.com/daml/services/owl-s/1.2/overview/ Dernière visite le 20.08.2015

Au final, il faut noter que malgré les inconvénients que nous venons de citer, l'apport du paradigme SOA dans le domaine des processus métiers, agissant notamment sur leur interopérabilité, flexibilité et réutilisabilité reste tellement appréciable qu'une panoplie d'approches supportant des processus métiers inter-organisationnels, basées sur l'approche SOA ont vu le jour. Dans la suite, nous décrivons quelques unes de ces approches en mettant l'accent sur les spécificités de chacune d'elles.

## II.4.5 Quelques Approches de WFIO à Base de Services

Les approches présentées dans cette section partent du constat que la composition de services possède plusieurs similarités avec la technologie Workflow (Wil, 2003). Les deux ont pour but de spécifier un processus métier par la composition d'entités autonomes, de simple et de forte granularité ; leur différence réside dans la nature de l'entité. Dans le cas d'un Workflow, les entités sont des activités, des tâches, des sous-processus ou des applications conventionnelles invoquées, alors que dans celui des services, les entités sont des services.

- **eFlow**

Le modèle e-Flow (Casati et al., 2001) est une plateforme qui permet la spécification, l'exécution et la gestion des services composites. Un service composite est décrit comme un schéma de processus qui combine des services composites et des services élémentaires.
Un processus est modélisé sous forme d'un graphe qui définit l'ordre d'exécution des nœuds. Les nœuds peuvent être de trois types différents : service, décision ou événement. Le nœud *service* représente l'invocation d'un service basique ou composite, choisi au moment de l'exécution. E-Flow propose un module de choix des services qui peut être remplacé par un module spécifique en fonction du domaine d'utilisation. Le nœud *décision* spécifie les alternatives et les règles qui contrôlent le flux d'exécution. Le nœud *événement* permet au service d'envoyer ou de recevoir plusieurs types d'événements : données temporelles, événements produits par le processus ou notifications spécifiques des applications externes.
Pour faciliter la mise en place des processus, e-Flow propose un registre qui contient des processus complets et des patrons de processus dont certaines parties sont spécifiées comme des services abstraits, pas encore définis. Pour supporter l'hétérogénéité des services, e-Flow définit des adaptateurs de services qui supportent plusieurs protocoles d'interaction B2B comme par exemple *OBI* ou *RosettaNet*.

L'architecture d'e-Flow (voir Figure II.16) est composée de trois entités: le moteur e-Flow, le système de découverte des services (Broker) et les services élémentaires (service operation Monitor, Service process composer … etc). Le rôle du moteur d'e-Flow est d'exécuter les instances de processus, mettre à jour les états des nœuds de services, décider les nœuds qui sont actifs dans une instance, communiquer avec le Broker pour découvrir les services. Ce modèle e-Flow présente certaines caractéristiques qui sont :

- *Un processus adaptatif de services (Adaptative service process)* **:** la spécification du nœud d'exécution inclut la description du service qui sera invoqué, ainsi que la spécification de la règle pour choisir le service, il permet une découverte dynamique des services. L'utilisateur peut aussi remplacer le service broker par un nouveau qui est le plus adapté à ses besoins.

- *Une modification dynamique des processus de services (Dynamic service process modification) :* le schéma de processus peut être changé pendant que le service fonctionne. Pour le changement, tout d'abord un nouveau schéma doit être défini, ensuite c'est une migration du « service process instance » du schéma courant vers le nouveau schéma.

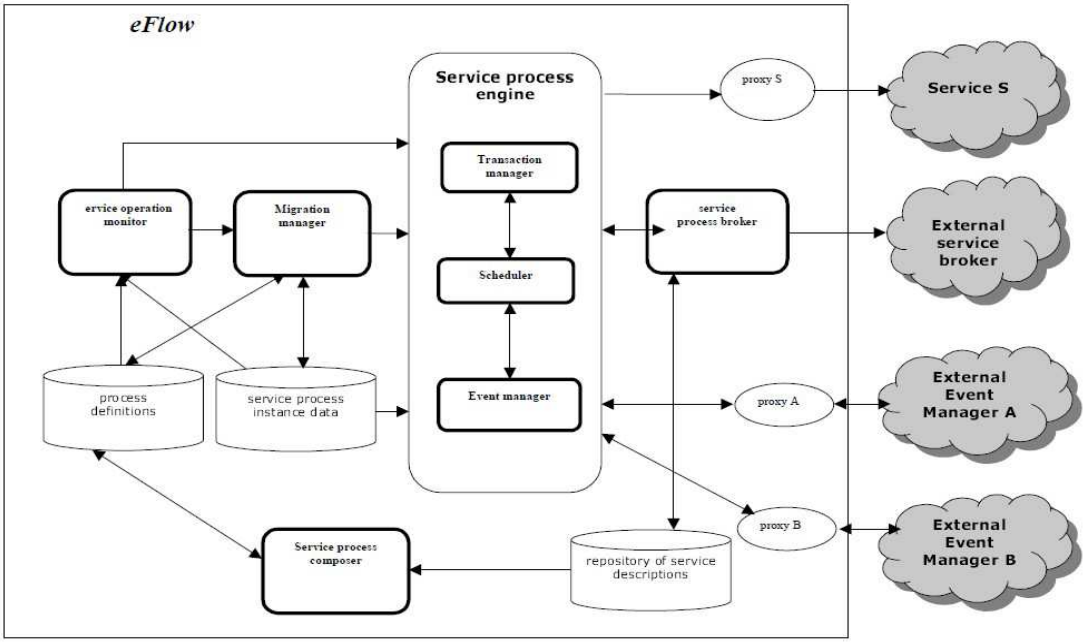



**Figure II. 16.** *Architecture de e-Flow* (Casati et al., 2001)

- **Self-Serv**

Le projet « Self-Serv » (*Composing Web Accessible Information and Business Services*) est un projet réalisé par (Sheng et al., 2002) qui consiste en une plateforme de composition dynamique de services Web interentreprises, dans un environnement pair à pair (*Peer to Peer*). L'architecture proposée par Self-Serv est illustrée dans la figure II.17, elle montre deux parties fondamentales: le *gestionnaire de services* (Service Manager) et le *Pool de Services.* Le gestionnaire de services permet de stocker les services, les déployer et aussi les découvrir. Il effectue les fonctionnalités d'un registre dans l'architecture classique des services Web et il est connecté à l'UDDI. Quant au *pool de services*, il se charge de gérer la composition des services.

« Self-Serv » distingue trois types de services: les services *élémentaires*, les services *composés* et les *communautés de services*. Ces dernières représentent l'agrégation d'un ensemble de services substituables. Une composition de services au sein de la plateforme « Self-Serv » est exprimée à l'aide d'un langage déclaratif basé sur les diagrammes à état. Afin de mettre en œuvre une composition de services, « Self-Serv » propose des agents logiciels appelés coordinateurs qui se chargent de la coordination d'exécution d'un service composé. Un coordinateur (hébergé par le prestataire du service correspondant) est généré pour chacun des services composés ainsi que pour les composants. Les coordinateurs sont des planificateurs légers qui reçoivent les notifications de terminaison des autres coordinateurs et invoquent le service correspondant. Une fois le service accompli, le coordinateur transmet les résultats obtenus à celui qui a invoqué le service et envoie une notification de terminaison aux coordinateurs des états suivants.

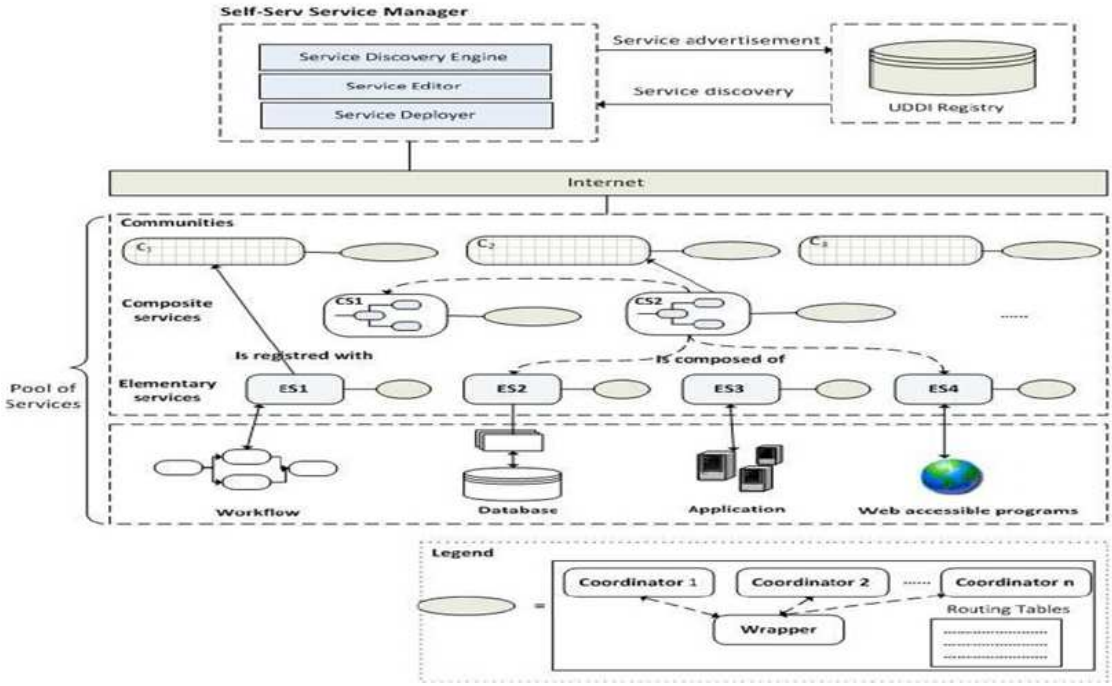



**Figure II.17.** *Architecture de Self-Serv* (Sheng et al., 2002)

- **PYROS**

PYROS (Belhadjjame et al., 2005) est un environnement pour la construction et l'orchestration de services ouverts représentés par des WF coordonnant les appels à leurs méthodes, l'architecture générale de PYROS est montrée dans la figure II.18. Les contributions de cette approche se résument en quatre aspects : (i) les activités composant un service donné ainsi que leur ordre d'exécution sont visibles, (ii) l'approche permet la personnalisation et la modification de services selon les besoins des utilisateurs, (iii) l'approche offre un contrôle de la progression et de fin d'exécution de services afin de gérer leurs dépendances, (iv) A partir d'un ensemble de services et de la spécification de leur orchestration, l'approche permet la génération de gestionnaires responsables de leur exécution.

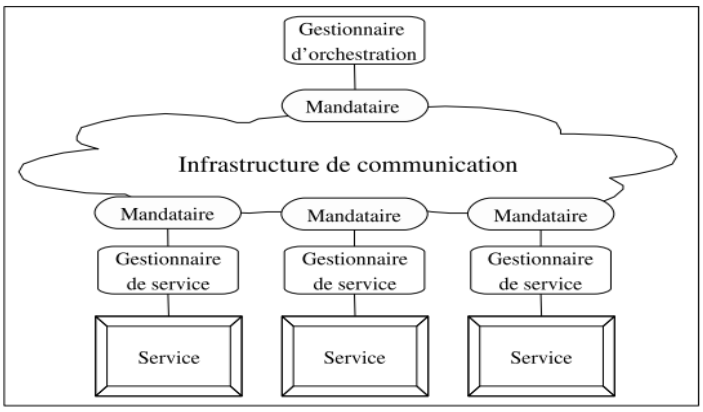



**Figure II.18.** *Architecture d'une application sous PYROS* (Belhadjjame et al., 2005)

Afin de modéliser des services ouverts et adaptables, la spécification d'un service est effectuée en trois étapes :

- Le fournisseur de services exporte ses *API*s contenant l'ensemble des méthodes qu'il offre ainsi que leurs dépendances (précédence, conjonction, exclusion) permettant ainsi de diriger le concepteur lors de la spécification de services. Celui-ci définit un WF de services où les activités

correspondent aux différents appels de méthodes du fournisseur en question et les flots de contrôle spécifient l'ordre d'invocation de ces méthodes. Cette manière de découpler l'interface du WF ordonnant les appels des méthodes, permet une personnalisation du WF de services selon les besoins des utilisateurs.

- La seconde étape consiste à vérifier que le WF ainsi construit est bien structuré, chaque *and-split* (respectivement *or-split*) est complété par un *and-join* (respectivement *or-join*), l'association des méthodes et le respect des dépendances.

- La dernière étape est la génération du gestionnaire de services à partir du schéma de services. Le gestionnaire de services est formé de quatre modules : (i) un *ordonnanceur* responsable de l'exécution de flots de contrôle d'un WF de services, (ii) un *module d'invocation* pour l'appel des méthodes du fournisseur de services, (iii) un *module de trace* pour l'insertion de résultats dans une base de données et la récupération d'information sur la progression du service et (iv) un *module d'exécution de services* pour l'exécution du service en question.

- **CoopFlow**

CoopFlow (Chebbi, 2007) est une approche ascendante pour la coopération de workflows interentreprises. CoopFlow est inspirée de l'architecture orientée services qui est basée sur trois opérations : la publication, la recherche et la connexion. Les fournisseurs de services publient les différents services qu'ils offrent. Les demandeurs de services cherchent les différents services répondant à leurs besoins. Une fois que les services sont trouvés, ces demandeurs s'y connectent afin d'exécuter un ensemble d'opérations offertes par les services. De même, cette approche est composée de trois étapes : (i) Publication de parties de WF d'entreprises pouvant être exploitées par d'autres entreprises, (ii) Interconnexion de WF et (iii) Coopération et surveillance du WF global conformément à un ensemble de politiques de coopération (contraintes d'interactions).

Cette approche est basée sur le modèle SOA et est considérée comme flexible. Elle préserve l'autonomie et le savoir-faire des entreprises. L'application des politiques de coopération dans une approche CoopFlow peut être implémentée de deux manières différentes, selon le degré de confiance qu'ont les partenaires entre eux:

- Les entreprises ont suffisamment confiance les unes aux autres : dans ce cas, les politiques de coopération sont complètement distribuées.
- Il n'existe pas de confiance directe entre les entreprises : dans ce cas, les interactions sont assurées par un tiers de confiance agissant comme intermédiaire et donnant lieu à une architecture centralisée de Coopflow, comme le montre la figure II.19.

Cette architecture comporte :

- un ensemble de partenaires identifiés, possédant des compétences complémentaires et souhaitant coopérer ensemble
- un *abstracteur* pour l'abstraction et la publication de WF des partenaires
- un *MatchMaker* pour l'interconnexion des abstractions de WF
- un ensemble d'adaptateurs jouant le rôle d'interface entre les environnements internes et externes des entreprises
- un *proxy* (mandataire) pour la gestion de la coopération
- un contrôleur pour la gestion de la coopération
- une base de données interne
- un éventuel annuaire abritant les profils des WF des entreprises coopératives.
- un intergiciel assurant la communication entre les systèmes des entreprises participantes.

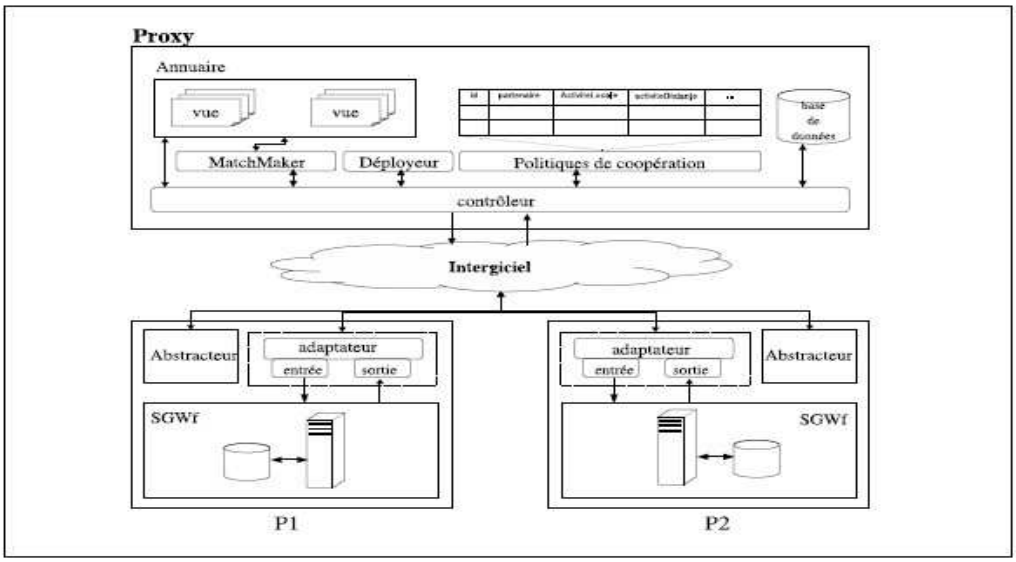



**Figure II.19.** *Architecture centralisée de CoopFlow* (Chebbi, 2007)

- **FrEOs**

FrEOs (*Framework de définition de l'Entreprise Orientée Services*) (Chaari, 2008) propose un framework pour la construction de l'entreprise orientée services dont l'objectif est d'assurer l'interconnexion des processus d'entreprise afin d'aboutir à un processus de coopération porteur de valeur ajoutée. Ainsi, le point de départ du framework consiste en une phase de réingénierie de l'entreprise, une étape qui vise à réorganiser et à restructurer l'entreprise de manière qu'elle soit plus agile afin de garantir un alignement entre le niveau SI, d'une part, et les besoins métiers de l'entreprise d'autre part. Pour cela, l'approche adoptée se base sur l'Architecture Orientée services (SOA).

Les processus métiers de l'entreprise serviront de contexte métier pour le développement de services afin d'aboutir concrètement à une entreprise axée sur l'architecture SOA. Dans cette optique, le niveau métier de l'entreprise est perçu comme un ensemble de processus métiers (des séquences d'activités ordonnées partiellement et synchrones) et de services métiers (des activités ou ensemble d'activités en ligne et asynchrones). Le service métier est un ensemble de fonctionnalités offertes par un fournisseur de services (une unité organisationnelle par exemple). Il peut être invoqué à l'intérieur ou à l'extérieur de l'entreprise. Sa différence majeure avec une activité métier ordinaire de l'entreprise est qu'une activité ne peut fonctionner qu'au sein d'un processus métier; par contre un service métier peut être invoqué tout seul comme il peut être sollicité au sein d'un processus métier. Les services métiers agissent comme une couche d'abstraction entre le niveau métier et le niveau informatique de l'entreprise qui, à son tour, sera transformée en un ensemble de services informatiques.

Cette restructuration de l'entreprise aboutit à une nouvelle architecture d'entreprise plus agile appelée l'Entreprise Orientée Services (EOS). Une fois ce but atteint, l'interconnexion des processus est assimilée à un problème de composition de services d'entreprise qui seront publiés dans des registres publics. Les services de l'entreprise sont décrits de manière à favoriser une collaboration dynamique et à la demande. Le framework de construction de processus collaboratifs tient compte des paramètres fonctionnels et non fonctionnels des services.

- **FOCAS**

FOCAS (*Framework for Orchestration, Composition and Aggregation of Services*) (Pedraza, 2009) propose un canevas extensible pour la création d'applications orientées procédés, c'est une approche qui s'intéresse à la conception, la spécification et l'exécution d'applications orientées procédé en général, et plus particulièrement à l'orchestration de services. Le canevas FOCAS est basé sur la technologie WF et utilise une ingénierie dirigée par les modèles (IDM) pour la spécification abstraite d'une orchestration, qui permet de structurer une application en différents points de vue (les données, le contrôle et les services) autour de l'abstraction du procédé. Dans FOCAS, la description abstraite de l'application est découplée des services (ou applications) supportant son exécution. Cette description permet d'abord, une indépendance de la technologie utilisée pour l'implémentation des services, ainsi qu'une liaison dynamique à l'exécution, aux services disponibles. Le canevas propose entre autres un noyau basé sur les concepts de l'orchestration de services. L'approche propose également des mécanismes permettant d'étendre le canevas dans différents domaines ainsi que pour supporter des aspects non-fonctionnels. L'architecture du Runtime de FOCAS (Figure II.20) comporte les principaux éléments suivants :

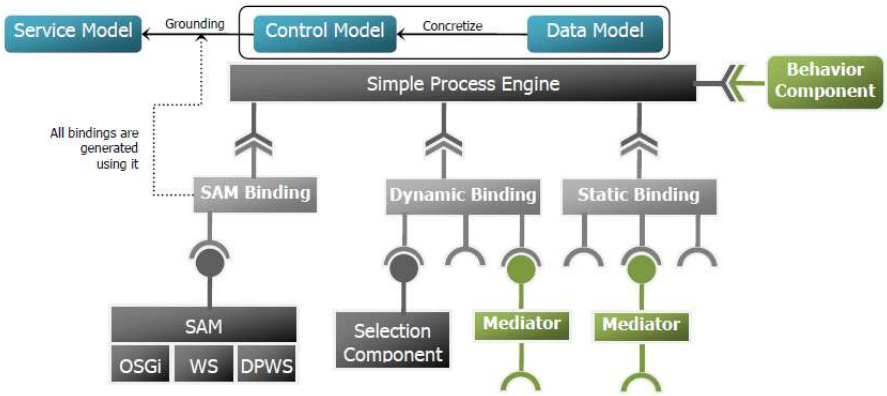



**Figure II.20.** *Architecture de FOCAS* (Pedraza, 2009)

- les composants fournis par le développeur des orchestrations. Spécifiquement, le composant d'orchestration abstraite qui est spécifié à travers les modèles de chacun des trois points de vue (*Service model, Data model, Control model*) et leurs relations.
- les composants partiellement générés par l'environnement de spécification pour une orchestration particulière. Par exemple, les classes qui implémentent le comportement pour une orchestration sont générées par l'environnement et complétées par le développeur. Dans cette catégorie, se trouvent le composant de comportement et les médiateurs (*Behaviour component, mediator*).
- Les composants complètement générés par l'environnement de spécification pour une orchestration particulière. Le composant de liaison (*Binding*) est généré à partir des modèles spécifiés par le développeur. La génération de ce composant dépend de choix techniques comme l'utilisation d'une liaison statique (déterminée à la conception) ou d'une liaison dynamique (déterminée à l'exécution).
- Les composants génériques du canevas qui supportent l'exécution de toutes les applications orientées procédé. Dans cette catégorie, on trouve le moteur simple d'exécution de procédés (*Simple Process Engine*) et la machine SAM (*Service Abstract Machine*).

- **Orchestration Multi-Contextuelle**

Dans (Esper, 2010), une nouvelle approche d'orchestration de services a été proposée, en tenant compte du contexte d'exécution des services (l'auteur parle de « contextualisation » de services).

A cet effet, cette nouvelle approche est basée sur l'intégration d'une architecture orientée services allant du niveau technologique au niveau métier. L'auteur propose de redéfinir les processus de manière abstraite au niveau de la logique métier en exprimant les différents rôles à jouer. Ceci permet de contourner la vision « top-down » de la modélisation traditionnelle en apportant une vision de composition incrémentale par composition de services abstraits en fonction des besoins.

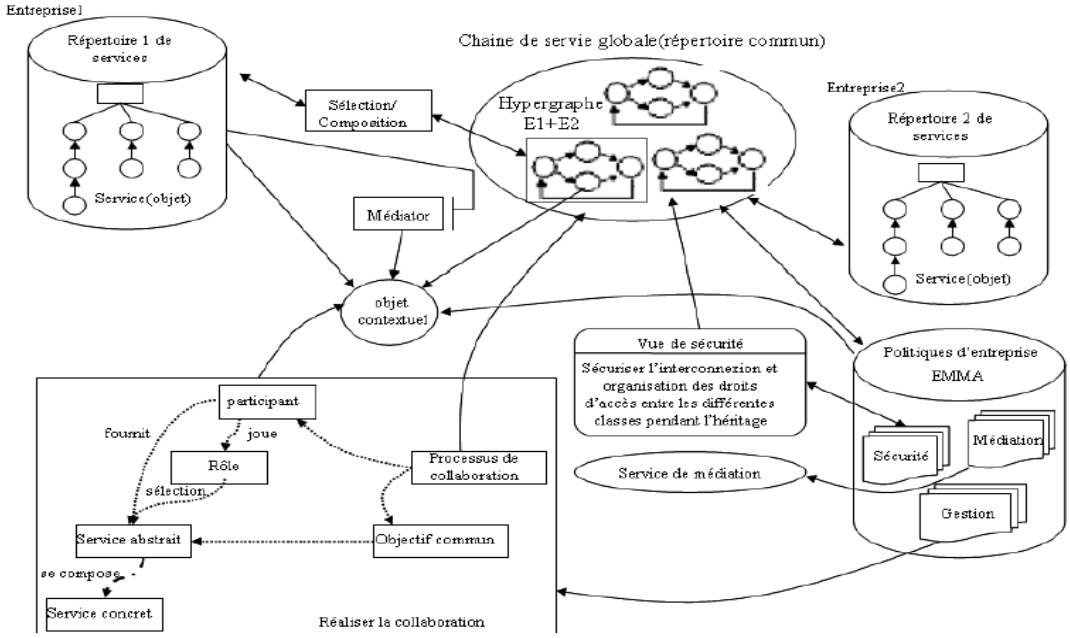



**Figure II.21.** *Architecture de l'approche « Orchestration multi-contextuelle »* (Esper, 2010)

Ces processus abstraits sont utilisés pour sélectionner et interconnecter les services technologiques, selon une logique d'assemblage permettant d'intégrer différentes logiques de médiation, de sécurité…etc. en composant des services technologiques adaptés au contexte. Pour cela, une nouvelle logique d'instanciation est proposée: il ne s'agit plus seulement de dériver un modèle générique pour le particulariser puis transformer ce modèle particulier en un code exécutable, mais d'adopter une logique de composition de services exécutables à partir de *modèles abstraits* ou d'informations *contextuelles*.

L'approche proposée utilise une vision en hypergraphe permettant de tirer partie à la fois des avantages de l'approche objet (héritage, agrégation,…), de l'ingénierie des modèles (logique de transformation) et de la composition de services. En plus de l'interconnexion des services et les liens établis entre les services concrets et abstraits, l'architecture proposée permet d'intégrer un «tissage» de modèles de sécurité, de médiation, des relations de similarité…etc. qui permettront d'organiser au mieux le processus «concret» support de la collaboration, en guidant les différentes opérations de composition.

- **Astro-CaptEvo**

L'objectif de l'approche Astro-CaptEvo (Heorhi, 2012) est de proposer un framework de composition de services qui :

(i) permet de manipuler des services "stateful" et non déterministes interagissant en mode asynchrone.

(ii) permet un contrôle riche et prend en compte les exigences de composition des flux de données qui demeurent indépendantes des détails d'implémentation.

(iii) exploite des techniques avancées de planification pour le raisonnement automatisé.

(iv) utilise une méthodologie de modélisation applicable en environnement dynamique.

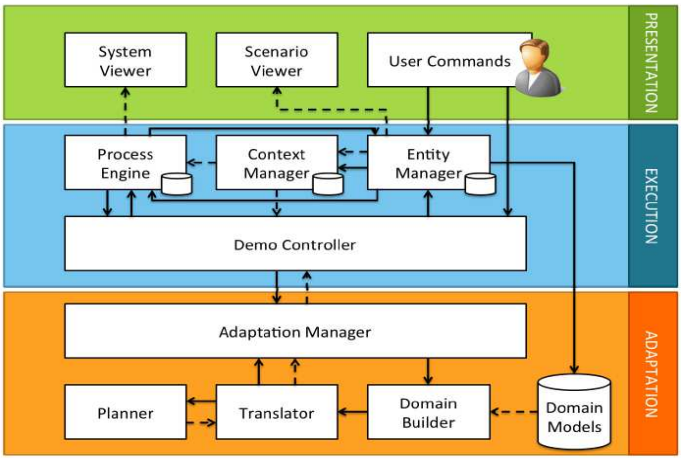



**Figure II.22.** *Architecture de ASTRO-CaptEvo* (Heohri, 2012)

L'idée de base du framework est d'expliciter le modèle du contexte qui abstrait les exigences de composition et les contraintes indépendamment des détails d'implémentation des services. En reliant les services au modèle du contexte d'une part, et en exprimant les exigences de la composition et les contraintes dans le modèle du contexte d'autre part, l'approche proposée offre un cadre formel dans lequel les exigences abstraites et les contraintes peuvent toujours rejoindre les implémentations de services disponibles. De plus, le framework offre la possibilité de déplacer la plupart des activités humaines au niveau « buidtime » afin de rendre la gestion du cycle de vie de la composition complètement automatique. Le framework proposé prend en compte aussi le problème de l'adaptation des processus métiers à base de services, la solution à ce problème est basée sur l'approche de composition proposée.

Le tableau II.5 résume les différentes approches su-décrites, mettant en évidence leurs caractérisations par rapport à trois critères que nous avons jugés intéressants, à savoir : le type de coopération supporté, les architectures de WFIO prises en charge et les mécanismes de flexibilité offerts par chacune des approches.

**TAB II.5.** *Récapitulatif des approches de WFIO à base de services*

| *Approche* | Type de coopération | Description | Architectures de WFIO supportées | | | | | | Aspects de flexibilité pris en charge |
|---|---|---|---|---|---|---|---|---|---|
| | | | Type 1 | Type 2 | Type 3 | Type 4 | Type 5 | Type 6 | |
| *e-Flow* | Dynamique | Publication de services web<br>Composition et exécution de services web en utilisant un moteur central. | ✓ | | | | | | Sélection dynamique des services<br>Adaptation du modèle de processus en cours d'exécution conformément au protocole défini.<br>Utilisation d'activités génériques.<br>Réutilisation de patrons de processus |
| *Self-Serv* | Dynamique | plateforme de composition dynamique de services web interentreprises, dans un environnement P2P.<br>Utilisation d'agents coordinateurs pour le suivi d'exécution. | | ✓ | | | | ✓ | Réutilisation de services élémentaires et composites.<br>Adaptation en cours d'exécution, par substitution de services. |
| *Pyros* | Dynamique | Publication de services.<br>Orchestration de services via un WF global qui joue le rôle de coordinateur central. | ✓ | | | | | | Le modèle d'orchestration est adaptable, adaptation du contrôle de flux des services.<br>Possibilité d'adaptation interne (personnalisation selon les besoins) des services fournis par les partenaires.<br>Réutilisation de services |
| *CoopFlow* | Dynamique | Inspirée de l'approche SOA.<br>Basée sur la publication, l'interconnexion et le contrôle d'exécution des WF de différents partenaires. | ✓ | | | | | ✓ | Sélection dynamique des WF.<br>Réutilisation de WF fournis par les partenaires. |
| *FOCAS* | Dynamique | Approche basée sur l'orchestration de services, pour la construction d'applications orientées procédés<br>Permet un contrôle centralisé et distribué.<br>Pas spécialement dédiée aux applications B2B mais supporte certaines de ses architectures. | ✓ | ✓ | ✓ | | | ✓ | Adaptation dynamique grâce à une description abstraite du procédé.<br>Possibilité d'extensions fonctionnelles et non fonctionnelles des services.<br>Réutilisation de services. |
| **FrEOS** | Dynamique | Framework pour la construction de l'entreprise orientée services, basé sur une composition dynamique des services.<br>Sélection automatique et composition semi-automatique des services. | ✓ | ✓ | ✓ | | | | Sélection dynamique de services basée sur leurs caractéristiques fonctionnelles et non fonctionnelles.<br>Réutilisation de services. |
| **Multi-contextual orchestration** | Dynamique | Basée sur les paradigmes SOA et Objet<br>Construction d'un processus collaboratif commun pour supporter l'interopérabilité des services fournis par différents partenaires dans différents contextes. | ✓ | | | | | | Sélection dynamique de services et construction flexible du processus collaboratif (à la volée).<br>Adaptation des services au contexte<br>Composition de services abstraits selon les besoins.<br>Réutilisation de services |
| **Astro-CaptEvo** | Dynamique | Framework pour la composition dynamique de services avec communication asynchrone, en plus de mécanismes d'adaptation de processus métier à base de services. | | | | | | ✓ | Sélection dynamique de services.<br>Adaptation de processus métier au contexte.<br>Réutilisation de services. |

Nous pouvons remarquer que toutes les approches relevées sur le tableau, sont destinées à une coopération dynamique où les partenaires métiers sont découverts dans un annuaire de publication de services ; ceci met en évidence l'apport de l'architecture SOA dans la coopération B2B dynamique (occasionnelle ou spontanée). Nous remarquons aussi que toutes les approches présentent déjà une flexibilité dans la sélection des services et offrent des mécanismes pour l'adaptation du processus global. Par exemple, e-Flow offre des mécanismes pour l'adaptation du modèle global en fonction d'un protocole prédéfini, l'approche Self-Serv offre une adaptation par substitution de services, PYROS permet une adaptation du contrôle de flux et une personnalisation des services selon les besoins des utilisateurs. Les approches FOCAS et Orchestration Multi-Contextuelle utilisent le concept de procédé (ou service) abstrait pour supporter l'adaptation de l'application, Astro-CaptEvo offre des mécanismes pour l'adaptation des processus métiers au contexte. Les aspects d'évolution peuvent être déduits de l'association de nouveaux partenaires (ajout de nouveaux services par exemple, lors d'une adaptation du processus). Quant aux aspects de réutilisation, ils concernent la réutilisation de services basiques dans toutes les approches, la réutilisation de patrons de processus dans e-Flow, la réutilisation de WF dans l'approche CoopFlow et la réutilisation de services composites dans Self-Serv.

## Synthèse

Dans ce chapitre, nous nous sommes focalisés sur les concepts et les technologies clés du Workflow et de l'architecture orientée services (SOA), permettant la mise en œuvre de l'entreprise virtuelle coopérative, à travers la réalisation d'une coopération B2B entre partenaires métiers. Nous avons rappelé les bases de la technologie WF ainsi que les concepts et les outils de WF inter-organisationnels (WFIO). Nous nous sommes arrêtés particulièrement, sur les schémas de coopération qui constituent le point de départ de notre travail de thèse, identifiés dans (Van der Aalst, 99), (Van der Aalst, 2000) et qui définissent les différentes manières de faire coopérer des processus WF entre eux, afin d'atteindre un objectif métier global. Nous avons rappelé les avantages de la technologie WF et ses gains considérables (en matière de temps, délais, qualité et coûts) pour les processus métiers d'entreprises.

Cependant, cette technologie a montré son inadéquation pour la prise en charge des changements pouvant affecter les processus métiers, particulièrement dans un contexte de coopération B2B. Ceci s'avère très contraignant car dans un contexte inter-organisationnel, la caractéristique de *flexibilité* revêt une grande importance. En effet, les besoins d'interopérabilité, d'adaptabilité et d'évolutivité se font ressentir davantage lorsqu'il s'agit de mettre en commun des processus métiers issus de différentes organisations autonomes, distribuées et hétérogènes.

L'avènement des architectures orientées services (SOA) a levé cette contrainte grâce aux avantages de ce paradigme dont l'indépendance des plateformes d'implémentation, le couplage faible des services, la facilité d'adaptation, d'évolution et de réutilisation. Dans ce paradigme, l'interconnexion de processus métiers issues de différentes organisations, se fait grâce à des mécanismes de composition de services. Nous avons présenté les différentes approches de composition proposées dans la littérature, que nous avons classées selon les critères : degré d'automatisation, degré de flexibilité et contrôle des liens entre les services. Suite aux avantages apportés par le paradigme SOA, une panoplie d'approches visant à interconnecter des processus métiers à base de services, dans un contexte inter-organisationnel ont vu le jour. Nous avons présenté quelques une des approches que nous avons rencontrées dans la littérature, ainsi qu'une analyse par rapport à trois critères que nous avons jugés significatifs, à savoir : les architectures de WFIO supportées, le type de coopération et les aspects de flexibilité offerts par chacune des approches.

Nous avons déjà souligné au chapitre I, que la réalisation d'une coopération B2B implique la construction d'un *processus métier inter-organisationnel* obtenu par interconnexion des processus métiers de plusieurs organisations. Comme les processus métiers sont décrits via des modèles de processus (spécifiant les entités qu'ils renferment et les liens entre eux), l'interconnexion de processus revient à une interconnexion des modèles qui les décrivent, une question qui relève de la composition des modèles dans l'IDM (Ingénierie Dirigée par les Modèles). Dans notre cas, nous faisons la distinction entre *composition* et *interconnexion* dans le sens où la première donne lieu à un seul modèle cible global dit *composite*, à partir de modèles sources et la seconde préserve les modèles sources et établit des *liens* entre eux pour l'interconnexion.

Dans le prochain chapitre, nous allons nous pencher sur la question de modélisation des WF et la composition de modèles. Nous allons examiner les aspects de la composition de modèles dans l'IDM et les critères permettant de caractériser les techniques de composition afin de pouvoir situer notre approche de composition (ou d'interconnexion) de WF par rapport à ces critères.

# CHAPITRE III

# Modélisation et Composition de WF

## Introduction

La modélisation est un élément clé dans le développement de systèmes Workflow, elle est basée sur des concepts théoriques invariants, sur lesquels peuvent s'appliquer avec plus ou moins de succès des formalismes et des méthodes de modélisation existants. De ce fait, la plupart des systèmes de WF commerciaux proposaient des méthodes de modélisation et des formalismes propriétaires inspirés en grande partie des formalismes existants (Saikali, 2001).

Dans une étude de (Joosten et al., 96), des fondements pour la modélisation de WF ont été extraits, en se basant sur les critères retenus par les entreprises et ceux proposés par les spécialistes théoriciens. Ces fondements ont été repris par les développeurs de systèmes WF et augmentés par des chercheurs dans le domaine. En effet, les principaux aspects à modéliser dans un WF restent invariants (déjà cités dans le chapitre II), il s'agit des aspects fonctionnel, organisationnel, comportemental et informationnel (Joosten et al., 96), (Barthelmess et al., 95) pour répondre respectivement, aux questions suivantes : quelles sont les activités à réaliser (*quoi*) ? Quelles sont les compétences nécessaires (*qui*) ? Quand faut-il les réaliser (*quand*)? Quels sont les outils et les informations nécessaires à leur réalisation (*comment*)?. Dans un contexte de WF inter-organisationnel, vient s'ajouter un aspect supplémentaire visant à modéliser les interactions entre les WF mis en coopération, il s'agit de l'aspect *interactionnel*. Dans ce cas, il s'agit de composer (ou interconnecter) deux ou plusieurs modèles de processus WF pour former un modèle de processus WFIO global.

Notons que la modélisation se fait dans plusieurs buts, notamment dans un but d'analyse, de compréhension, de simulation, de re-conception des processus métiers ou d'amélioration du fonctionnement de ces processus (BPR[8] ou CPI[9]) (Scheer, 98), (Vernadat, 96).

Dans ce chapitre, nous nous penchons sur la question de modélisation des WF, nous commençons par donner les définitions des concepts de base liés à la modélisation ainsi que les liens qui les unissent. Nous présentons la définition de la composition de modèles dans l'IDM (Ingénierie dirigée par les modèles) et les critères permettant de différencier les techniques de composition afin de pouvoir situer notre approche de composition (ou d'interconnexion) de WF par rapport à ces critères. Nous rappelons ensuite, les différents types de modélisation de WF ainsi que les différents formalismes de modélisation; nous nous arrêtons sur la modélisation de WF en utilisant deux langages de modélisation qui à l'origine, ne sont pas dédiés à la modélisation de WF mais qui ont été adaptés à cet effet ; il s'agit de la notation unifiée UML et du formalisme des Réseaux de Petri (RdP). Nous nous focalisons ensuite sur la question de composition de WF qui doit s'effectuer à travers la composition (ou l'interconnexion) de deux ou plusieurs modèles de processus WF fournis par les partenaires métiers en coopération. Ces modèles de processus doivent être présentés avec un niveau d'abstraction permettant d'une part de préserver l'autonomie et la confidentialité souhaitée par chaque partenaire et d'autre part, exhiber les détails nécessaires pour la composition (ou l'interconnexion) des différents modèles. Enfin, nous passons brièvement en revue les techniques de composition de WF établies dans la littérature.

### III.1 Notions de base de la modélisation

Dans cette section, nous rappelons les définitions des concepts liés à la notion de modélisation, principalement, les concepts de modèle, méta-modèle et langage de modélisation. Nous évoquerons aussi la notion de composition de modèles dans l'IDM.

[8] *Business Process Reingeneering*
[9] *Continuous Process Improvement*

### III.1.1 Modèle

Bien que possédant un grand nombre de définitions, un modèle dans l'ingénierie dirigée par les modèles (IDM) est l'abstraction d'un objet sujet de l'acte de modélisation. Son but est de permettre une étude plus simple dans un contexte maitrisé autre que le contexte réel.

En d'autres termes, *un modèle représente l'abstraction d'un objet du monde réel, en vue de manipuler, analyser ou utiliser cet objet dans un domaine d'application donné.* Le modèle peut représenter un aspect particulier de l'objet ou intégrer différents aspects ; il doit permettre de représenter fidèlement les aspects de l'objet modélisé sur lesquels porte l'analyse ou l'utilisation. Un modèle se concentre en général uniquement sur certains des aspects du sujet, qui présentent un intérêt pour le concepteur du modèle (Muller, 98). Selon (Saikali, 2001), un modèle peut être défini comme suit :

- **Définition III.1-** Modèle

*Un modèle est une abstraction d'un certain sujet, matériel (par exemple, une voiture) ou immatériel (par exemple, un processus) existant ou susceptible d'exister. Par exemple, un modèle d'un processus métier peut correspondre à un processus existant ("as is") ou à un processus auquel on veut aboutir ("to be").*

Mais pourquoi a-t-on besoin de modéliser ? La modélisation d'un sujet (ou objet) qu'il soit matériel ou immatériel permet d'obtenir une représentation observable, manipulable et évaluable de l'objet ; cette représentation aide à comprendre l'objet modélisé et à l'étudier sans avoir à l'affecter. D'autre part, la modélisation permet de simplifier l'objet en question en faisant ressortir uniquement les aspects significatifs et en faisant abstraction des autres aspects qui peuvent être plus significatifs dans une autre étude. Au fait, les aspects à modéliser dépendent de l'utilisation que l'on veut faire du modèle représentant l'objet.

### III.1.2 Modèle de processus

- **Définition III.2-** Modèle de processus

*Un modèle de processus représente l'enchainement chronologique d'un ensemble d'activités ayant des entrées et des sorties et devant être réalisées par des ressources (humaines, matérielles ou logicielles).*

Un modèle de processus possède des spécificités aidant d'une part à comprendre le fonctionnement du processus (modélisation descriptive) et d'autre part à supporter l'automatisation de ces processus (modélisation active). Les spécificités d'un modèle de processus ont été résumées dans (Berry, 98) comme suit :

- Facilite la compréhension du fonctionnement de l'entreprise au travers de ses processus,
- Facilite la communication entre les différents acteurs concepteurs, participants et utilisateurs du processus.
- Permet l'amélioration des processus (CPI) ou leur re-conception (BPR),
- Définit une base à l'automatisation de l'exécution des processus,
- Supporte la gestion des processus (grâce à leur automatisation).

Par ailleurs, (Saikali, 2001) reprend les trois types de modèles de processus identifiés dans la littérature:

*(a) Les modèles passifs* : sont des modèles qui capturent la structure d'un système à un moment donné. Une fois le modèle créé, il devient indépendant du sujet qu'il représente, c'est à dire qu'il n'est pas affecté par des évolutions ou des modifications du sujet. Ce type de modèle est

celui qui est le plus communément utilisé dans les systèmes d'information tels les bases de données.

*(b) Les modèles dynamiques passifs* : ce sont des modèles passifs qui permettent de représenter le comportement d'un système. Ils sont principalement utilisés à des fins de simulation, pour l'analyse et la validation des systèmes modélisés. Ils peuvent également servir de support pour une exécution automatisée.

*(c) Les modèles actifs* : utilisent des techniques de modélisation leur permettant de rester liés au sujet (dans notre cas, le processus WF) avec lequel ils deviennent synchronisés. Toute modification du sujet est donc automatiquement répercutée sur le modèle qui reflète à tout moment l'état et la structure du sujet. Deux types de liens unissent un modèle actif à son sujet (processus) : les liens "réactifs" et les liens "actifs". Les premiers permettent aux modèles actifs de contrôler le comportement du sujet, grâce à la réception de « stimuli ». Les liens actifs quant à eux permettent aux modèles actifs d'évoluer en fonction de l'état du sujet (processus). Cet aspect s'avère extrêmement intéressant dans la modélisation des processus métiers qui sont soumis à des changements fréquents dans un environnement métier dynamique.

### III.1.3 Langage de modélisation

Un *langage de modélisation* sert à décrire un système, standard ou méthodologique, général ou spécifique à un domaine et/ou un contexte par ses composants et leurs relations.

Un langage de modélisation peut être représenté par des mots standardisés reliés au domaine de l'activité modélisée ou par des symboles graphiques décrivant les liens entre les composants du modèle représenté. Par exemple, le diagramme d'activité d'UML représente un langage graphique de modélisation de l'aspect comportemental d'un processus métier.

### III.1.4 Méta-modélisation et Méta-modèle

Un acte de méta-modélisation a les mêmes objectifs qu'un acte de modélisation avec pour seule différence l'objet de la modélisation. Il s'agit dans tous les cas, de supporter tout ou partie du cycle de vie d'un modèle : spécification (informelle ou formelle), conception, implantation.

Dans certains cas, l'acte de modélisation peut être plus un instrument de dialogue qu'une technique d'ingénierie; cela est par exemple le cas des manuels de méthodes de conception qui décrivent de plus en plus leurs différents modèles par « instanciation » d'un méta-modèle graphique. Dans le domaine de la méta-modélisation, trois tendances sont distinguées (Oussalah et al., 2014):

- la méta-modélisation comme technique de réflexivité : la méta-modélisation est un acte de modélisation appliqué sur un modèle. Dans le cas des modèles réflexifs, la méta-modélisation permet à un modèle de s'auto-décrire : il est à la fois l'enjeu et le moyen de modélisation.
- la méta-modélisation comme technique de dialogue : les concepteurs de systèmes ont de plus en plus recours à cette technique pour expliquer, commenter, documenter, comparer des modèles, en particulier les modèles semi-formels utilisés dans les méthodes de conception. Il s'agit de décrire un modèle par son schéma conceptuel, résultat d'une étape de spécification utilisant un méta-modèle qui est souvent semi-formel. Ce schéma constitue alors un document d'explication et/ou de documentation du modèle. Il peut également servir de moyen de comparaison et d'unification de modèles.
- la méta-modélisation comme technique d'ingénierie : il s'agit de documenter, d'expliquer ou d'unifier des modèles « semi-formels », la langue naturelle, si elle n'était pas aussi ambiguë, pourrait être utilisée comme méta-modèle. Il s'agit d'appliquer à nos modèles nos propres

techniques d'ingénierie. On peut modéliser un modèle pour les mêmes raisons que l'on modélise des systèmes applicatifs.

- **Définition III.3 – méta-modèle**

*Un méta-modèle est une définition formelle d'un modèle qui aide à le comprendre et qui facilite le raisonnement sur sa structure, sa sémantique et son usage* (Damak et al., 2011).

- **Définition III.4– méta-modèle**

*Un méta-modèle est l'entité concrétisant informatiquement, un contexte de modélisation pour la conception et la manipulation de modèles* (Jeusfeld et al., 2009).

Le méta-modèle représente l'abstraction du langage de modélisation des modèles. Un lien s'établit alors entre le modèle et le méta-modèle: la conformité du premier pour le second. Généralement, la pratique de la méta-modélisation consiste à spécifier des méta-modèles contemplatifs qui reflètent la structure statique des modèles, c.à.d. les concepts et les liens entre ces concepts (Jeusfeld et al., 2009). Selon le formalisme utilisé (UML, par exemple), un tel méta-modèle peut être complété par la spécification de certaines contraintes (en langage OCL, par exemple).

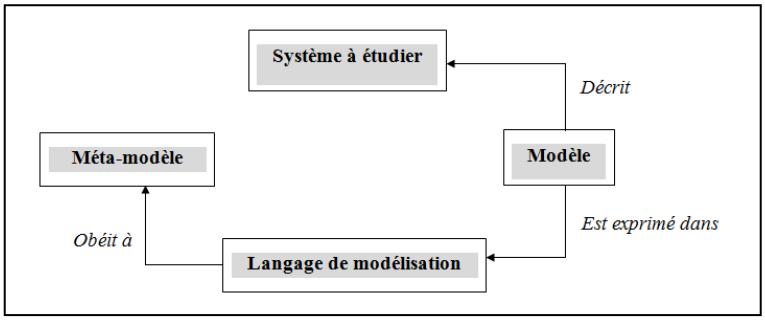



**Figure III.1.** *Liens entre les éléments de modélisation*

Afin de cerner la relation entre modèle, méta-modèle et langage de modélisation (voir Figure III.1), on peut dire que le méta-modèle décrit les éléments du langage de modélisation et les liens entre eux, le modèle est décrit dans un langage de modélisation qui doit rester conforme au méta-modèle.

L'organisme de standardisation OMG (OMG, 2005) distingue quatre niveaux d'abstraction dans la modélisation d'un système.

*Au premier niveau(M0)*, se trouve le système à étudier, il peut s'agir d'un quelconque système à automatiser entre autres, un processus métier.

*Au second niveau(M1)*, se trouve le modèle qui décrit certains aspects du système que l'on veut étudier: il peut s'agir d'un diagramme d'activités UML, d'un modèle conceptuel de traitement MERISE, un réseau de Petri ou de tout autre schéma qui représente une vue abstraite des objets modélisés.

*Au troisième niveau(M2),* se trouve le langage de modélisation ou méta-modèle, par exemple les définitions des concepts d'un diagramme d'activités UML ou les concepts du modèle réseau de Perti. C'est ce langage qui doit faire l'objet d'une spécification formelle. La méta-modélisation est très utilisée dans le domaine de l'ingénierie des systèmes d'information et particulièrement dans l'ingénierie des modèles et des méthodes (Rolland, 2005), (Rolland, 2007). Dans l'ingénierie des méthodes, c'est un outil conceptuel indispensable pour définir et raisonner sur de nouveaux modèles. Dans l'ingénierie des outils, les méta-modèles sont une définition formelle nécessaire pour concevoir et construire les outils pour éditer, vérifier, transformer et éventuellement exécuter des instances de ces modèles.

*Au quatrième niveau(M3)*, se trouve le méta-méta-modèle, il s'agit d'un langage qui doit être assez générique pour définir les différents langages de modélisation existants. Le MOF[10] (Meta Object Facility) est, pour l'OMG, le méta-méta-modèle unique servant de base à la définition de tous les méta-modèles.

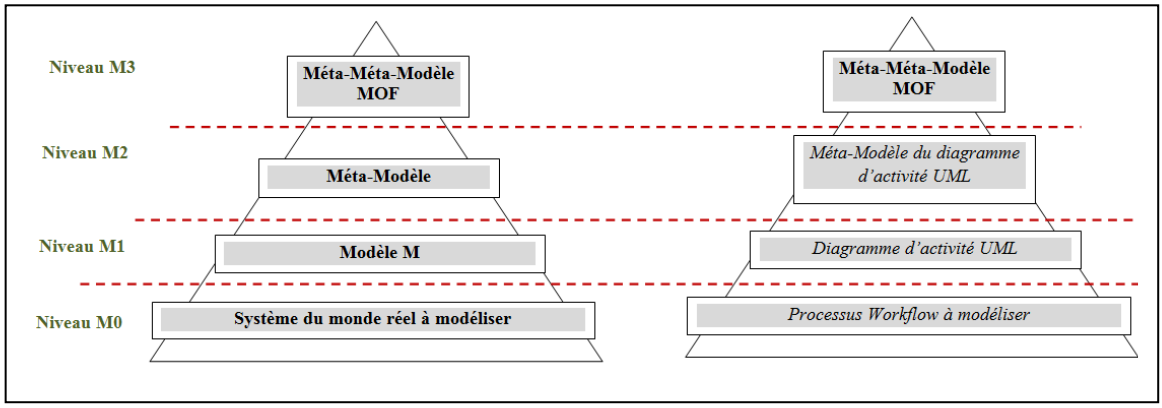



**Figure III.2.** *Niveaux d'abstraction dans la modélisation*

Par exemple, Le diagramme d'activités UML de la figure III.3 décrit un processus de WFIO destiné à la conception et la réalisation de circuits intégrés (CI).

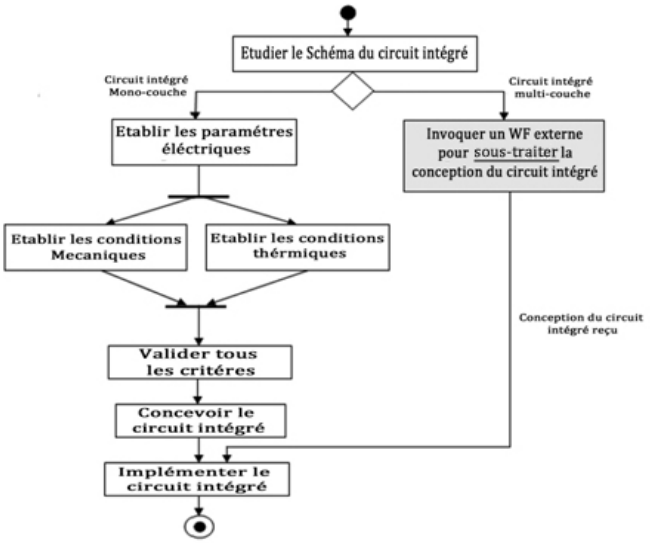



**Figure III.3.** *Modèle du processus « Réalisation de Circuits Intégrés »* (Boukhedouma et al., 2012b)

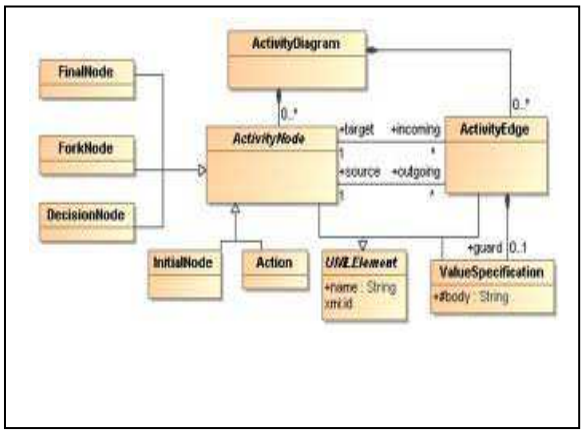



**Figure III.4.** *Méta-modèle simplifié du diagramme d'activité UML*

Sur la figure III.3, sont exposées d'une manière générique, les phases les plus importantes du processus. Les différentes phases représentent des sous-processus pouvant être complexes et composés de plusieurs activités. La figure III.4 représente un méta-modèle simplifié du diagramme d'activité UML, qui est dans ce cas, le langage de modélisation ayant servi à la modélisation du processus de la figure III.3.

[10] *www.**omg**.org/spec/**MOF**/2.4.2*

## III. 2 La composition de modèles

La composition de modèles est un thème de recherche en pleine expansion dans l'IDM (ingénierie dirigée par les modèles). Une opération de composition est définie autour de trois points essentiels : la description des modèles à composer, les mécanismes utilisés pour réaliser la composition et la description du (ou des) modèle (s) obtenu (s) par composition.

### III.2.1 Définitions

Informellement, la composition de modèles se définit comme suit:

- **Définition III. 5–** composition de modèles

*"Model composition is an operation that combines two or more models into a single one."* (Didonet et al., 2006).

- **Définition III.6 –** composition de modèles

*"Model composition in its simplest form refers to the mechanism of combining two models into a new one."* (Baudry et al., 2007).

On peut donc dire qu'une composition de modèles est une opération qui prend deux ou plusieurs modèles en entrée et les combine afin de fournir un seul modèle en sortie qualifié de modèle composite.

### III.2.2 Critères liés à la composition de modèles en IDM

Dans la littérature, un certain nombre d'approches de composition de modèles ont été définies, elles se distinguent par un ensemble de critères qu'elles posent sur les modèles en entrée et en sortie et sur l'opération de composition elle-même, nous reprenons les critères recensés dans (Nuygen, 2008).

- **Critères sur les modèles en entrée/sortie :** ces critères se résument en cinq points essentiels qui sont :

*(a) Le nombre de modèles en entrée* : deux ou plusieurs.

Un processus de composition peut recevoir deux ou plusieurs modèles en entrée. Dans la plupart des cas, chaque composition prend deux modèles, mais il existe des cas comme les éditeurs EMF (Budinsky et al., 2003) où on peut composer plusieurs modèles à la fois (par l'importation des modèles sous forme de ressources au même éditeur).

*(b) Le rôle des modèles composés* : symétrique ou asymétrique.

Certaines approches dites symétriques ne font pas la différence entre les modèles composés (symétriques) (Bernstein, 2003) mais certaines autres distinguent les modèles de base et les modèles d'aspects, il s'agit d'approches asymétriques (Reddy et al., 2006).

*(c) Le type des modèles composés* : structurel ou comportemental.

Certaines approches se focalisent sur la composition de modèles structurels (schémas de base de données (Bernstein, 2003), diagrammes de classes UML (Clarke, 2002) etc.) ; certaines autres s'intéressent aux modèles comportementaux (modèles de réseaux de Petri, diagrammes d'état/transition UML (Nejati, 2007), etc.). Notons que, dans la première catégorie, certaines approches comme (Bernstein, 2003) et (Didonet et al., 2006) ne précisent pas les types des modèles composés, ceux-ci peuvent être un ensemble d'objets, les schémas de base de données, des modèles UML,… etc., alors que d'autres se restreignent à un seul type de modèles par exemple des modèles UML (Clarke, 2002).

*(d) L'hétérogénéité des modèles source* : composition endogène ou exogène

Lorsque les modèles à composer sont conformes à un même méta-modèle, on parle de composition *endogène* ; dans le cas contraire (méta-modèles différents), on parle de composition *exogène*. Dans ce dernier cas, la composition des modèles doit se faire à travers la composition des méta-modèles qui consiste à établir des relations entre les concepts des méta-modèles. Ces relations expriment le lien sémantique entre les méta-modèles dans le sens où la sémantique de l'un est étendue/ou complétée par l'autre et vice versa. Un principe de base pour réaliser une composition de méta-modèles est qu'il faut bien distinguer les concepts de la syntaxe abstraite de ceux de la sémantique du méta-modèle et établir les bonnes relations entre les parties correspondantes.

*(e) Le nombre de modèles en sortie* : un ou plusieurs.

Le résultat final de l'opération de composition peut être soit un modèle composite (Clarke, 2002), (Reddy et al., 2006), soit un ensemble comprenant les modèles d'entrée plus une autre partie qui les intègre (Bouzitouna, 2004).

- **Critères sur l'opération de composition :** les principaux critères liés à l'opération de composition sont les suivants :

*(a) Le type de composition* : par opérateurs ou par relations.

Dans (Nguyen, 2008), les auteurs distinguent deux types de composition, par opérateurs ou par relations. La composition par opérateurs repose comme son nom l'indique sur des opérateurs de composition tels que la fusion, le remplacement, l'union, le tissage,… etc. Dans ce cas, la composition est réalisée en exécutant des opérations sur les modèles sources. La composition par relations quant à elle, est basée sur des relations telles que l'association, l'agrégation et l'héritage. Dans ce cas, les modèles sources sont composés en utilisant les relations pour les connecter. La différence est que les opérateurs ne sont pas une partie du modèle final ; tandis que les relations font vraiment partie de ce modèle (Nguyen, 2008).

*(b) Les éléments de la composition* : en général, un processus de composition a besoin de connaître d'une part, la nature des éléments (classe, attribut, méthode, …) des modèles à composer et leurs correspondances (dans le cas d'une composition exogène) et d'autre part, les mécanismes servant à la composition (comment la composition sera effectuée). Selon les approches, ces informations peuvent être modélisées par une ou plusieurs spécifications de composition à la fois. Par exemple, dans EML (Kolovos et al., 2006), ces informations sont décrites dans les spécifications des règles de composition (règles de correspondance, règles de fusion et règles de transformation), tandis que dans AMW (Didonet et al., 2006), elles sont réparties dans les modèles de tissage et les transformations.

*(c) L'effet de la composition* : la structure des modèles sources peut être transformée ou préservée, suite à l'opération de composition.

*(d) Le langage de composition* : les éléments de composition ont besoin de formalismes pour les exprimer. Selon les modèles à composer, il peut s'agir d'un langage de tissage (Didonet et al., 2006), un méta-modèle des règles de composition (Kolovos et al., 2006), un profil UML pour la composition de modèles (Clarke, 2002), ou un langage de composition de services comme BPEL, XLANG, …etc.

### III.2.3 Phases du processus de composition

Généralement, le processus de composition de modèles peut être divisé en trois phases : avant, pendant et après la composition.

- **Phase I - Avant la composition** : il est probablement nécessaire de transformer (ou adapter) les modèles sources pour que la composition puisse être réalisée sans conflit.

- **Phase II – Pendant la composition**: cette phase comporte deux étapes :

**(i)** *Etape de correspondances* : établit les correspondances entre les éléments des modèles. La découverte des correspondances (syntaxiques et sémantiques) peut être automatique (Kolovos et al., 2006), Fleurey et al., 2007), manuelle (Bezivin et al., 2006) ou semi-automatique (Nejati et al., 2007).

**(ii)** *Etape de fusion* : intègre les éléments en correspondance. Il est probable d'avoir les conflits comme une incompatibilité de types entre des éléments en correspondance. La résolution de conflits est soit automatique, basée sur certaines stratégies de résolution prédéfinies, soit manuelle (par exemple, faire apparaître un panel pour que l'utilisateur résolve le conflit).

- **Phase III - Après la composition** : pendant cette phase, il est nécessaire de restaurer la cohérence du modèle final, on peut dire que c'est une phase de vérification et de validation de la composition.

Cependant, dans la réalité, l'implémentation des approches de composition ne respecte pas toujours les trois phases su-décrites. Certaines approches proposent explicitement l'adaptation avant la composition (Fleurey et al., 2007), contrairement à certaines autres telle que l'approche présentée dans (Kolovos et al., 2006).

## III.3 La modélisation WF

Dans cette section, nous nous concentrons sur la modélisation WF en mettant l'accent sur les perspectives de modélisation d'un WFIO, les types de modélisation dans le domaine du WF et les formalismes de modélisation, notamment ceux que nous utilisons dans le cadre de cette thèse.

### III.3.1 Perspectives de modélisation dans le WFIO

La figure III.5 résume les différents aspects de modélisation d'un WFIO, en mettant au centre l'aspect fonctionnel qui est au cœur de la modélisation WF. L'exécution d'un WF doit permettre de répondre aux questions suivantes : quelles sont les activités ou les services à réaliser (*quoi*) ? Quand faut-il les réaliser (*après…, avant…, sous quelles conditions, suite à quels événements*) ? Quelles sont les compétences nécessaires à la réalisation des activités ou des services (*qui*) ?
Notons que dans un contexte intra-organisationnel, les compétences représentent les acteurs internes à l'organisation, leur rattachement aux unités organisationnelles de l'entreprise et leurs rôles dans le processus, tandis que dans un contexte inter-organisationnel, il s'agit de décrire les partenaires impliqués dans la coopération (sans se soucier de l'organisation interne au niveau de chaque partenaire), chacun avec le ou les rôles qu'il remplit dans le processus de WFIO global. Une autre question qui doit trouver sa réponse dans l'exécution d'un WF est : quelles sont les ressources nécessaires à l'exécution des activités ou des services (Comment ?). Ces ressources sont les données d'entrée, la description des applications qui prennent en charge l'exécution des activités, … etc.

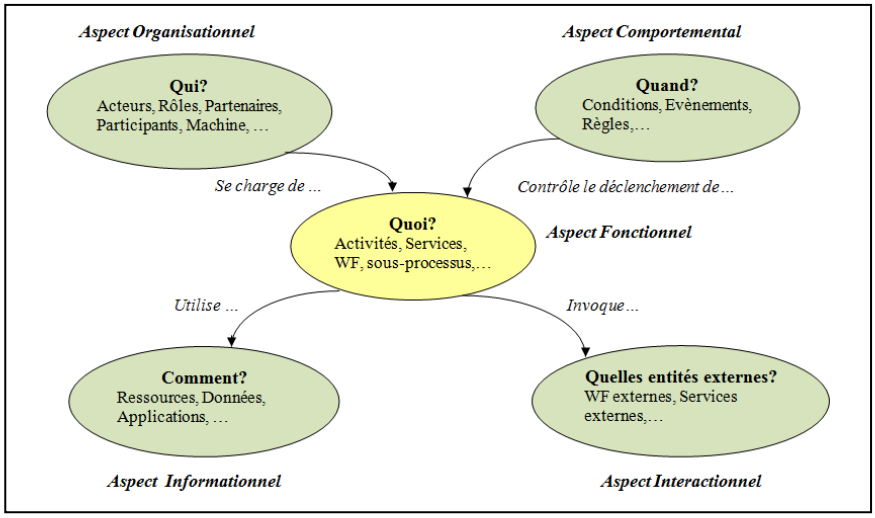



**Figure III.5.** *Aspects de modélisation d'un processus WFIO*

Dans un contexte inter-organisationnel, les données nécessaires à l'interaction entre les différents fragments sont publiques mais les données nécessaires à l'exécution locale d'un fragment de WF restent privées. Un autre aspect de la modélisation WF doit être pris en compte dans le cas d'un WFIO, il s'agit de l'aspect interactionnel (Quelles entités externes ?) ; cet aspect correspond aux invocations d'entités externes (services, WF,…) en vue de décrire les interactions d'un fragment de WF local avec un fragment de WF externe, fourni par un autre partenaire impliqué dans la coopération.

## III.3.2 Types de modélisation dans le WF

Dans la littérature du WF et selon les paradigmes utilisés, quatre types de modélisation de WF sont distingués : la modélisation à base d'*activités*, la modélisation basée sur la *communication*, la modélisation à base d'*artefacts* et la modélisation à base de *règles* (voir Figure III.6). Néanmoins, l'utilisation d'un type de modélisation n'exclut pas les autres, en effet deux (ou plus) types de modélisation peuvent coexister dans un même système afin de décrire le processus WF sous différents angles complémentaires. Ces types de modélisation peuvent être combinés à différents niveaux de la modélisation afin d'avoir une vue complète, efficace et claire de la modélisation du processus. Le choix d'un formalisme (et donc d'un type de modélisation) dépend de son expressivité par rapport aux aspects à faire ressortir sur le modèle et son adéquation aux besoins des utilisateurs.

**III.3.2.1 La modélisation à base d'activités** ("activity based modeling") (Berry, 98) : représente le type de modélisation le plus répandu dans les systèmes WF orientés processus. Il consiste principalement à décrire les processus en termes d'activités et de sous-processus en définissant les conditions d'exécution des activités (les événements déclencheurs) et le contrôle de flux (opérateurs de synchronisation, de disjonction,…).

Selon la richesse des formalismes utilisés, les modèles à base d'activités permettent de représenter les aspects fonctionnel et comportemental du WF. Ils peuvent également inclure une partie du modèle organisationnel ou du moins y faire référence grâce aux rôles associés aux activités. Par exemple, les « swimlanes » dans les diagrammes d'activité UML (destinés à la description de la vue comportementale d'un processus) permettent d'indiquer les rôles (un concept de la vue organisationnelle) responsables de l'exécution des activités.

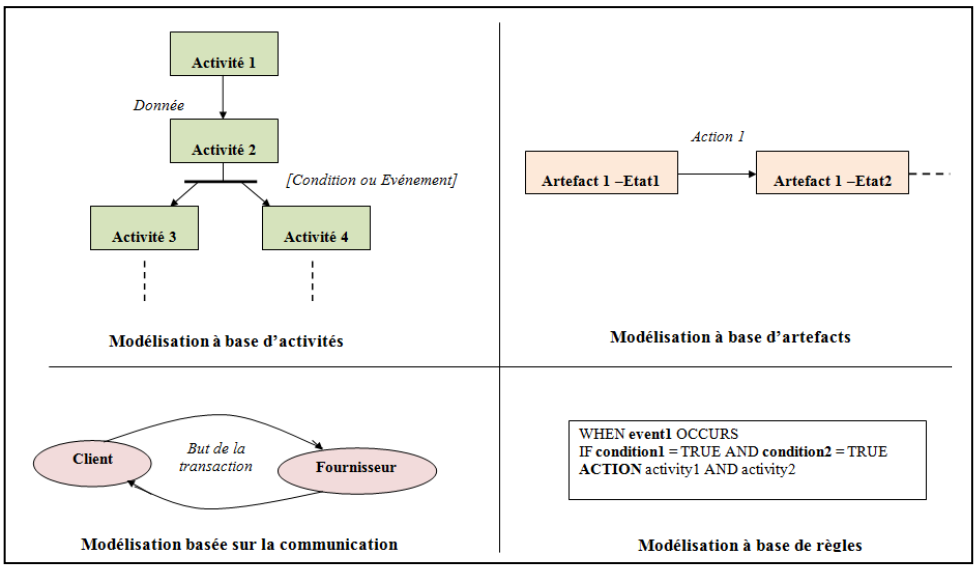



**Figure III.6.** *Types de modélisation dans le WF*

**III.3.2.2 La modélisation basée sur la communication** ("communication based modeling") (Berry, 98), (Nurcan, 96) ou modélisation conversationnelle (Eder, 98), est fondée sur la théorie du discours/action ; ce type de modélisation est basé sur le paradigme de la communication entre un client et un fournisseur. Le modèle décrit des transactions entre ces deux types d'acteurs. Les différentes transactions sont liées entre elles afin de décrire l'ensemble du processus. Ce type de modélisation est utilisé dans les systèmes WF orientés communication et peut être utilisé pour décrire l'aspect interactionnel dans un WFIO.

**III.3.2.3 La modélisation à base d'artefacts** ("artifact based modeling") (Berry, 98) : se concentre sur les artefacts (documents, formulaires, tables de base de données ou objets informatiques métiers) manipulés par les activités formant le processus WF. Ce type de modélisation se focalise sur l'aspect informationnel du processus et décrit son déroulement en termes d'états atteints par les différents objets manipulés tout au long du processus. Le routage est tracé en fonction des états des artefacts qui sont affectés par l'exécution des activités. Le processus est donc basé sur le résultat des activités accomplies sur les artefacts ; le diagramme d'état/transition d'UML permet de supporter ce type de modélisation.

**III.3.2.4 La Modélisation à base de règles** ("Rule based Modeling") (Eder et al., 96), (Casati et al., 96) : ce type de modélisation propose de décrire un processus grâce à des règles. Chaque règle est composée d'événements, de conditions et d'une ou plusieurs actions. L'occurrence des événements permet d'activer la règle. Les conditions si elles sont vérifiées, permettent à la règle d'exécuter les actions qui lui sont associées ("*Event Condition Action rules – ECA rules*"). Ce type de modélisation est directement issu de la notion de déclencheurs ("*Triggers*") liée aux bases de données. En général, le formalisme utilisé est principalement textuel, par exemple un langage de définition de règles.

### III.3.3 Méthodes et formalismes de modélisation de WF

Dans (Saikali, 2001), un panorama détaillé des méthodes et formalismes de modélisation de WF a été décrit. On y retrouve plusieurs méthodes de modélisation qui sont basées soit sur leurs propres formalismes (bien que ces derniers soient inspirés de formalismes existants) soit sur des formalismes standards tels que les réseaux de Petri. Parmi ces méthodes de modélisation, nous

pouvons citer celles qui utilisent une modélisation à base d'activités comme SADT (Marca et al.87), OSSAD (Chappelet et al., 2004), ADONIS (Glassey et al., 2002), IDEF (Mayer et al. 95) et ARIS (Scheer, 98). Dans la famille des méthodes à base de règles (ou à base de déclencheurs), on retrouve La méthode TMW "*Trigger Modelling for Workflow*" (Joosten, 94). La méthode Communication/Action ("*Speech/Action*") a été adaptée au Workflow par T. Winograd et F. Flores (Winograd et al., 90) ; elle se base sur le concept de communication et vise un objectif bien déterminé : la satisfaction du client. Ainsi, au lieu de décrire des processus et des objets circulant dans une organisation, cette méthode décrit plutôt une transaction entre un client et un fournisseur.

Indépendamment des méthodes, un ensemble de langages de modélisation tels que BPMN (Wohed et al., 2006), YAWL (Van der Aalst et al., 2005), UML(Bastos et al., 2002), les réseaux de Petri (RdP) (Van der Aalst et al., 98) ont été proposés pour la modélisation WF ou adaptés à la modélisation WF.

Dans la suite, nous évoquerons l'utilisation d'UML pour la modélisation des processus WF ainsi que l'utilisation du formalisme des réseaux de Petri (RdP), un outil mathématique doté d'une sémantique formelle dédié à la vérification de propriétés comportementales des systèmes qu'il modélise.

### III.3.3.1 UML et la modélisation de WF

La notation unifiée objet UML (Rational, 96), (Muller, 98) est le résultat de l'unification des principaux formalismes liés aux méthodes d'analyse, de conception et de développement Objet. C'est une notation permettant de représenter un système selon différents points de vue grâce au large éventail de diagrammes et d'outils proposés (OMG, 2011). UML n'est pas spécifique au WF, c'est une notation générale dédiée au développement de systèmes informatiques. Grâce à la notion de « stéréotype », UML est une notation ouverte qui accepte l'augmentation et l'adaptation de son méta-modèle à de nouveaux concepts qu'il est nécessaire de représenter dans un domaine spécifique. UML propose trois types de diagrammes : (i) les diagrammes statiques, (ii) les diagrammes dynamiques (dits de comportement) décrivant d'une part le comportement des objets et d'autre part la collaboration entre les objets et (iii) les diagrammes d'architecture (ou d'implémentation). Dans la première catégorie, on retrouve les diagrammes de classe, les diagrammes d'objet, les diagrammes de composants et les diagrammes de structure composite. Les diagrammes de cas d'utilisation, d'interaction, d'état/transition, de séquence, de communication et d'activités s'inscrivent dans la deuxième catégorie et enfin dans la troisième catégorie, on retrouve les diagrammes de déploiement et de packages.

Dans la modélisation WF, les diagrammes de cas d'utilisation sont utilisés pour décrire l'aspect fonctionnel du processus ; les diagrammes d'activité sont utilisés pour représenter les activités du processus, le contrôle de flux et les conditions de transition entre activités, c'est-à-dire l'aspect comportemental. Les diagrammes de séquence peuvent être utilisés pour décrire les interactions (envoi de messages entre les systèmes de WF offerts par des partenaires différents), en particulier ils sont utilisés pour décrire un protocole de communication. Les diagrammes de classe peuvent être utilisés pour représenter les données et les ressources manipulées par les activités.

Les diagrammes d'activités sont les diagrammes les plus intéressants et les mieux adaptés à la modélisation Workflow (Russel et al., 2006a). En effet, ils se prêtent très facilement à la représentation de l'ensemble des éléments composant des modèles de WF à base d'activités, respectivement : les rôles, les activités et leur contrôle de flux, les événements et les flux de données.

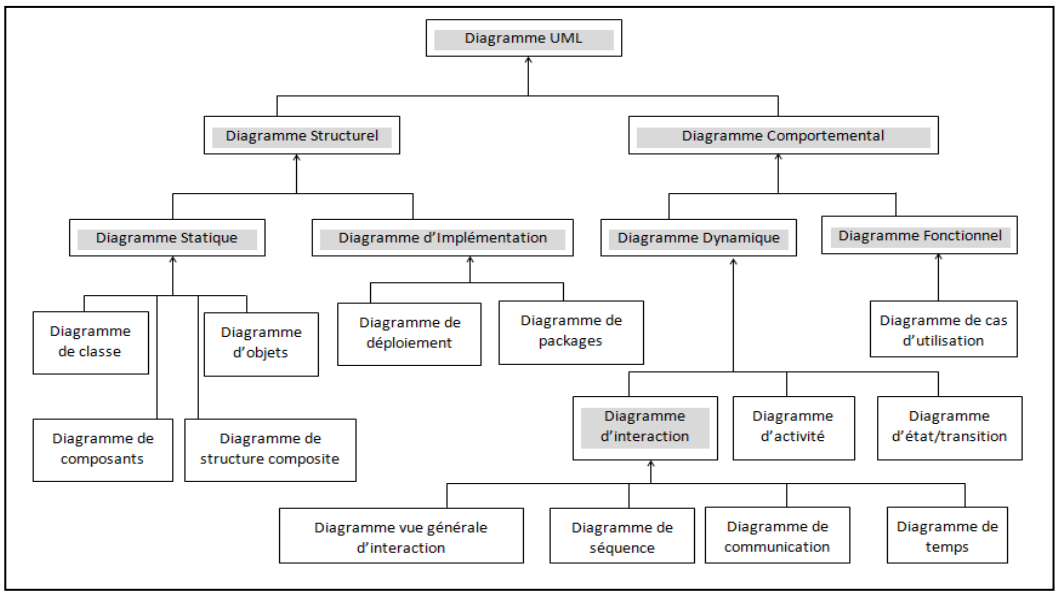



**Figure III.7.** *Vue d'ensemble des diagrammes UML 2.0*

De plus, différents diagrammes UML peuvent s'intégrer pour fournir une vue globale d'un processus WF. Concrètement, chaque entité d'un diagramme d'activité peut être décrite en détail par d'autres diagrammes. Par exemple, on peut décrire les propriétés et le comportement des données à l'aide de diagrammes d'états / transitions, et les relations qui existent entre les données à l'aide de diagrammes de classes. Il en est de même pour les rôles et les acteurs associés aux activités.

- **Avantages d'UML pour la modélisation WF**

Les intérêts que présente UML pour la modélisation de WF peuvent se résumer en six points essentiels:

- *Intégration* : La possibilité d'intégrer des vues complémentaires d'un système en intégrant différents diagrammes UML.
- *Expressivité* : La richesse des formalismes pour la prise en compte des différents aspects à modéliser
- *Extensibilité* : possibilité d'augmenter la notation grâce à la notion de stéréotypes qui permet d'une part d'adapter les concepts au domaine d'étude et d'autre part d'établir des liens avec des concepts empruntés à d'autres formalismes ou méthodes. Dans la partie intitulée "UML for business Modeling" (Rational, 97), ont été introduits des stéréotypes spécifiques à la modélisation de processus, tels les stéréotypes d'*acteur*, d'*activité* ou de *processus métier*. Dans (Amsden, 2004) par exemple, un profil UML a été défini pour la modélisation de processus métiers à l'aide des diagrammes d'activités.

- *Communication* : UML étant une notation générale et peut être utilisée à différents niveaux d'un projet Workflow (Hruby, 98), elle constitue un moyen standard de communication entre les différents intervenants (analystes, concepteurs, programmeurs,…) dans le projet. L'avantage est donc que les différents spécialistes puissent utiliser les mêmes formalismes ou des formalismes appartenant à la même notation, donc compréhensibles par tous.

- *Adaptation et spécialisation* : ces deux propriétés sont directement héritées du paradigme objet.

### III.3.3.2 Les Réseaux de Petri et la modélisation WF

Un grand nombre de méthodes de modélisation dans les systèmes WF utilisent des modèles inspirés des graphes orientés. En général, les noeuds correspondent aux activités du processus modélisé alors que les arcs sont associés à des conditions de transitions ou des flux d'information. Parmi les dérivés des graphes, on retrouve souvent les réseaux de Petri (RdP), un modèle aux fondements mathématiques, introduit par Carl Adam Petri (Petri, 62) pour la modélisation des systèmes concurrents. Un réseau de Petri est un graphe biparti et orienté où les noeuds représentent soit des places soit des transitions. Les places peuvent éventuellement contenir des jetons qui correspondent généralement aux ressources disponibles.

Les réseaux de Petri (RdP) sont largement utilisés dans les modèles qui traitent des problèmes de coordination et de synchronisation. Ils sont très populaires dans le domaine de gestion des processus métiers (BPM) (Van der Aalst, 98). Le formalisme des RdP est très répandu dans la modélisation des WF orientés processus, sous leur forme initiale ou sous une forme modifiée utilisant des variantes du modèle RdP de base. Selon les besoins du modèle et le domaine d'application, une place dans un RdP peut correspondre à une activité dans un processus, à l'état d'un système ou à un événement, à une file d'attente, … etc. Les transitions quant à elles correspondent le plus souvent à des événements ou des conditions à vérifier. Elles peuvent cependant représenter des activités, en particulier si les places jouent le rôle d'événements. La figure III.8 reprend l'exemple du WFIO de la figure III.3 pour le représenter à l'aide du modèle RdP, les transitions correspondent aux activités du processus et les places correspondent aux conditions de transition vers les activités suivantes.

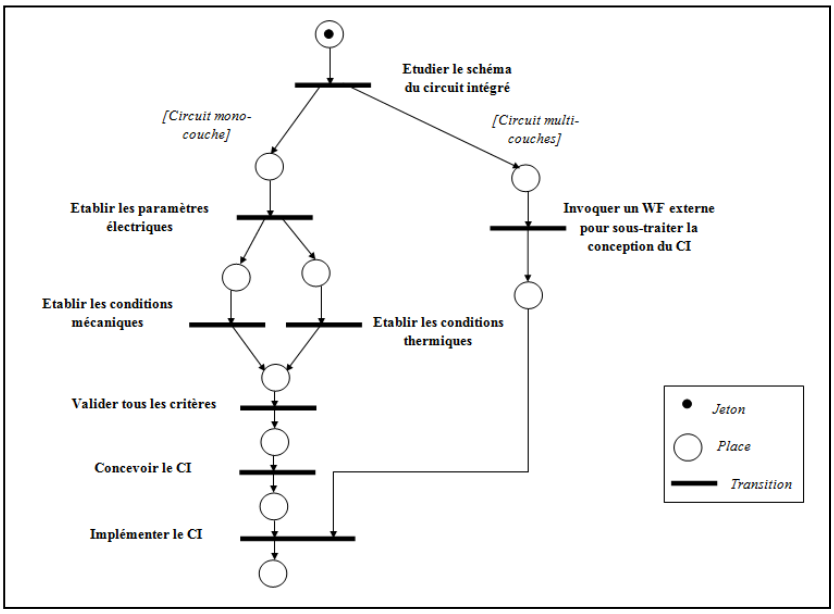



**Figure III.8.** *Modélisation du WFIO « Réalisation de Circuits Intégrés » à l'aide d'un RdP*

Au cours des dernières décennies, les réseaux de Petri ont été étendues, notamment avec les notions de couleur, de temps ou encore de hiérarchie (RdP colorés, temporisés, récursifs, …) (Van der Aalst, 97), (Liu et al., 2002), (Van der Aalst, 2003), (Yang et al., 2005). Les réseaux de Petri utilisant ces extensions sont dits de haut niveau et ils facilitent la modélisation de processus complexes où les données et/ou le temps sont des facteurs importants.

- **Avantages des RdP pour la modélisation WF**

Le succès des RdP dans la modélisation WF tient à plusieurs facteurs dont la simplicité de leurs formalismes, leur base mathématique, la possibilité de représenter la dynamique d'un système et les contrôle de flux (parallélismes, boucles, synchronisations) ; ils sont suffisamment expressifs pour modéliser toutes les primitives requises pour la modélisation de WF et toutes les structures de contrôle prises en charge par les logiciels actuels.

Les fondements mathématiques des réseaux de Petri permettent l'utilisation de nombreuses techniques d'analyse. Ces techniques peuvent être utilisées pour prouver des propriétés telles que l'invariance ou le non-blocage (deadlock-free). Il est aussi possible d'analyser la performance au travers de critères tels que le temps de réponse ou d'attente, en utilisant les RdP temporisés. De plus, la possibilité d'effectuer des simulations et de les visualiser (grâce à la circulation des jetons) permet de détecter les causes de ralentissement et les boucles. Ce qui permet l'amélioration du processus modélisé, dans un contexte de BPI. Enfin, les réseaux de Petri fournissent un langage indépendant de tout outil, n'étant pas basé sur un logiciel spécifique.

Malgré leurs avantages indéniables et l'attachement que leur portent beaucoup de spécialistes, les RdP n'en demeurent pas moins sans inconvénients pour la modélisation WF. Le premier inconvénient provient du fait qu'ils ne peuvent pas décrire certains aspects importants du WF, tels que les rôles responsables de la réalisation des activités, bien qu'il soit possible de les ajouter en tant qu'attributs dans les places, mais qui ne pourront pas figurer sur le modèle. Le deuxième inconvénient est la difficulté d'intégration du modèle comportemental (défini à l'aide d'un RdP) avec les autres modèles due à l'absence d'entités jouant le rôle d'interface entre les différents modèles. Enfin, malgré la dynamique qu'ils expriment, les RdP sont impossibles à reconfigurer dynamiquement, par exemple lors d'une modification du processus correspondant.

## III.4 Workflow et Composition de Services

Comme nous l'avons précisé au chapitre II, l'avènement des architectures orientées services (SOA) a permis d'une part, de restructurer les processus de l'entreprise en un ensemble de services qu'il faut composer entre eux pour former un flux cohérent d'activités. D'autre part, la SOA a considérablement favorisé l'interopérabilité entre processus WF hétérogènes issus d'organisations différentes. Dans ce cas, les services à composer n'appartiennent pas à une seule organisation mais sont fournis par un ensemble de partenaires métiers désirant bâtir une relation de partenariat entre eux. Au fait, l'idée de base est de transformer le concept d'activité en un concept de service (réellement, l'activité devient une invocation à un service) et de bâtir un WF intra-organisationnel à partir de services locaux ou un WF inter-organisationnel par combinaison de services locaux et externes.

Dans le chapitre V de ce document, une correspondance de concepts entre WF à base d'activités et WF à base de services est établie dont l'objectif est d'exhiber les similarités existant entre les deux entités, pour enfin déduire qu'une composition de services n'est autre qu'un WF dont les activités sont des invocations à des services, remplissant des fonctionnalités bien définies.

Ainsi, les méthodes utilisées initialement pour la modélisation de WF à base d'activités restent valables pour modéliser une composition de services. Aussi, les formalismes comme UML, BPMN et les RdP (réseaux de Petri) restent applicables pour la modélisation, la génération de code et la vérification de propriétés sur les WF à base de services qui ne sont autre que des compositions de services. Néanmoins, côté langages d'exécution, ont été développés des langages dédiés à la spécification et l'exécution d'une composition de services. Dans le chapitre II, nous

avons cité quelques langages de composition de services décrivant soit une orchestration soit une chorégraphie de services ; les langages tels que BPEL, XLANG, WSFL, WSCI sont orientés exécution. En effet, ils sont supportés par un éditeur graphique permettant de dessiner le processus et peuvent s'interfacer avec un ou plusieurs moteurs d'exécution qui permettent d'interpréter le code généré et exécuter le processus en question. Ces langages reposent sur des concepts de programmation et manquent de formalisme pour la vérification et la validation de la composition. Par ailleurs, comme nous l'avons déjà évoqué, le but de la modélisation WF ne se limite pas à une description et une compréhension du processus mais possède d'autres intérêts tels que la génération de code à partir de modèles, la vérification formelle des propriétés des modèles et l'évaluation des performances des processus modélisés. Dans une thèse de doctorat (Ait cheikh Bihi, 2012), l'auteur distingue trois catégories d'approches orientées modèles pour la composition de services qui diffèrent selon le critère « objectif de la modélisation »:

**(i)** ***les approches orientées modélisation et génération de code*** telles que BPMN et UML, largement utilisés dans la pratique, proposent de modéliser la composition de services tel un processus métier (Ouvans et al., 2006), (Geambasu et al., 2012). Ces langages proposent une notation graphique, sont assez expressifs et permettent la génération de code à partir des modèles graphiques. Par ailleurs, des travaux orientés transformation de modèles de processus d'un langage vers un autre ont été développés, par exemple les auteurs de (Bendraou et al., 2005) ont proposé un nouveau langage basé sur les diagrammes d'activité UML 2.0 pour la modélisation de la composition de services, permettant aussi la génération de code BPEL à partir des modèles de la composition. Dans (Dumez et al., 2008), UML-S (UML pour les Services) a été proposé pour modéliser la composition de services. Les auteurs ont également introduit des règles de transformation des diagrammes UML-S en BPEL.

**(ii)** ***les approches orientées vérification de propriétés*** : utilisent des formalismes tels que les réseaux de Petri (RdP) avec leurs différentes variantes, les automates avec leurs différentes variantes (Alur et al., 94), (Komenda et al., 2009) qui peuvent être utilisés pour décrire, spécifier et vérifier la composition de services. A ceux-ci s'ajoutent les algèbres de processus comme CSP (Hoare, 83), CCS (Milner, 89) et LOTOS (Bolognesi, 89) basées sur l'émission/réception de messages par un processus. Ces différents modèles munis d'une sémantique formelle ont été adoptés dans le but de vérifier les propriétés comportementales d'un système par simulation, avant toute implémentation. En particulier, ces modèles ont été utilisés pour la vérification de propriétés sur les WF à base d'activités (van der Aalst, 97), (Liu et al., 2002) et plus récemment sur la composition de services (Hamadi et al., 2003), (Ferrara, 2004), (Kaabi, 2007) , (Ouyang et al.,2007). Particulièrement, plusieurs approches ont été développées pour modéliser la composition de services par les RdP ; par exemple dans (Yi et al., 2004), un framework est proposé pour la conception et la vérification de la composition de services basée sur les RdP. Dans (Hinz et al., 2005), une description complète et officielle de la sémantique des RdP pour BPEL est présentée.

**(iii)** ***Les approches orientées évaluation de performances*** : cette troisième catégorie d'approches de composition de services vise à modéliser une composition en vue d'évaluer les performances (temps d'exécution, qualité de service,…) du modèle obtenu. Par exemple, l'approche de (Rainer et al., 2006) basée sur l'utilisation des heuristiques est orientée vers l'évaluation de la qualité de service dans une composition de services.

Rappelons que le premier volet de ce travail de thèse se focalise sur la proposition de patrons de coopération de WF qui consistent à composer (ou interconnecter) des WF de plusieurs partenaires afin de former un WFIO global supportant la coopération. Concrètement, ceci revient à

composer (ou interconnecter) des modèles de WF afin de former un modèle de WFIO global. Dans la suite, nous nous penchons sur la question de composition et d'interconnexion de modèles de workflow.

## III.5 Composition et interconnexion de Workflows

Nous avons vu dans le chapitre II, que la mise en coopération de deux ou plusieurs workflows nécessite la construction d'un processus métier inter-organisationnel. En effet, il s'agit de combiner (composer ou interconnecter) les modèles de processus des différents partenaires afin de créer un modèle de processus WFIO *global* en respectant les règles d'enchainement, d'interaction et d'échange de données entre les différents processus. Aussi, la communauté des WF doit se doter d'outils donnant la possibilité d'assembler différents WF entre eux. Cet assemblage a pour but, d'effectuer des activités qu'un simple WF ne saurait résoudre, c'est la *composition (ou l'interconnexion) des Workflows* qui permet de supporter la coopération, la collaboration et le partage de compétences entre les partenaires. Dans la suite, nous rappelons les définitions proposées dans la littérature pour la composition de WF.

### III.5.1 Définitions

- **Définition III.7 –** Composition de WF

*La composition de WF est un processus par lequel un nouveau WF est créé par orchestration de deux ou plusieurs Workflows. L'objectif principal de cette composition est alors l'intégration des Workflows afin de favoriser la coopération, la coordination et les échanges d'informations entre entreprises partenaires dans un contexte B2B (Business to Business)* (Chebbi, 2007).

- **Définition III.8 –** Composition de WF

*La composition de WF est l'arrangement et éventuellement la fusion des activités de deux ou plusieurs Workflows pour créer un nouveau Workflow appelé Workflow Composite* (Blay-Fornarino, 2009).

- **Définition III.9 –** WF composite

*Un Workflow composite peut être défini par un schéma de processus qui combine des activités composites et des activités atomiques* (Blay-Fornarino, 2009).

Une activité atomique est une activité issue d'un seul processus WF alors qu'une activité composite est le résultat de la fusion d'activités appartenant à deux processus différents.
En d'autres termes, la composition d'activités vise à définir une nouvelle activité composite à partir d'un ensemble d'activités **atomiques** (non décomposables) ou **composites** (composées d'autres activités atomiques). Généralement, la composition d'activités nécessite des mécanismes de « *fusion* **»** dans lesquels des activités se fusionnent pour engendrer de nouvelles activités (par exemple, la fusion de deux invocations à un même service peut donner lieu à une nouvelle invocation dans laquelle les valeurs des paramètres ont été agrégées par appel à une nouvelle activité).

Dans notre contexte de travail, nous proposons les définitions suivantes et nous jugeons nécessaire de faire la distinction entre la *composition* et l'*interconnexion* de workflows.

- **Définition III.10 –** Composition de WF

*La **composition** de WF est une opération qui consiste à fusionner deux ou plusieurs processus WF en un seul, en respectant des règles de contrôle de flux entre les activités, les règles d'interaction et d'échange de données entre les différents WF.*

- **Définition III.11 –** Interconnexion de WF

*L'**interconnexion** de WF est l'opération qui permet d'établir des liens entre deux ou plusieurs processus WF sans pour autant fusionner les processus en un seul mais tout en respectant des règles de contrôle de flux entre les activités, les règles d'interaction et d'échange de données entre les différents WF.*

Comme tout processus WF est décrit par un modèle, la composition (ou l'interconnexion) de WF se fait à travers la composition (ou l'interconnexion) des *modèles* qui les décrivent.
Donc, la principale différence que nous voulons ressortir entre composition et interconnexion de WF, c'est que la première donne lieu à *un seul modèle composite* géré par un coordinateur central et la deuxième (l'interconnexion) maintient les modèles sources en établissant uniquement *des liens* permettant de les raccorder et de contrôler l'exécution d'un processus WFIO global de manière décentralisée.

## III.5.2 Niveaux d'abstraction dans la composition de WF

D'après (Blay-Fornarino, 2009), la composition des WF peut être appréhendée sous deux formes, selon le niveau d'abstraction que l'on pose sur les WF à composer:

- La première en considérant les éléments du système (services et orchestrations) comme des **boîtes noires** (Figure III.9(a)) accessibles uniquement via leurs interfaces. Dans ce cas, la définition du résultat d'une orchestration comme nouveau service permet une composition récursive des orchestrations. Cependant cette composition pose des problèmes d'appels multiples aux services communs, d'absence de partage du contexte, de gestion des erreurs délocalisées, …etc.
- La seconde en considérant les orchestrations comme des **boîtes blanches** (Figure III.9(b)) qu'il s'agit de composer par tissage de code. Dans ce cas, la composition est une tâche plus complexe, qui suppose une bonne connaissance des activités et des orchestrations existantes.

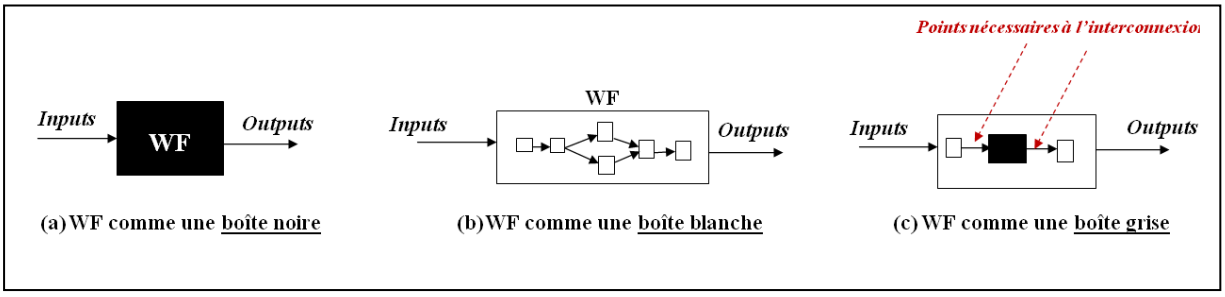



**Figure III.9.** *Niveaux d'abstraction d'un WF*

Dans notre cas, nous considérons une troisième alternative qui se situe entre les deux et qui consiste à percevoir les orchestrations (ie les WF à composer) comme des **boites grises** (Figure III.10(c)), dans ce cas il s'agit d'exhiber dans un modèle de WF, uniquement les points nécessaires à l'interconnexion entre les différents modèles, les autres détails seront encapsulés le plus possible dans une seule entité ( i.e un service composite considéré comme une boite noire).

### III.5.3 Modèles de composition ou d'interconnexion de WF

Dans une composition (ou interconnexion) de WF, on peut envisager selon les cas et les besoins de coopération, des modèles spécifiques inspirés des schémas génériques de coopération proposés par Wil van der Aalst (Van der Aalst, 99). Dans la suite, nous essayons de caractériser ces modèles selon qu'il s'agisse d'une composition proprement dite ou d'une interconnexion de WF et selon le niveau d'abstraction requis pour les WF à combiner.

(a) *Composition par orchestration d'activités*: dans cette approche, il s'agit d'obtenir un modèle de WF composite en combinant les activités de deux ou plusieurs WF à composer. En d'autres termes, il s'agit de définir un nouveau contrôle de flux (orchestration) entre les différentes activités qui seront éventuellement fusionnées ou simplement entrelacées entre elles. En effet, un opérateur de contrôle de flux peut lier deux activités issues d'un même processus ou de deux processus différents, deux activités issues de deux processus différents peuvent être fusionnées pour en former une seule. Ce type de composition donne lieu à un seul modèle de WFIO géré de manière centralisée. Dans cette approche, on retrouve le schéma de coopération « Partage de charge ».

(b) *Composition (ou interconnexion) par enchainement de modèles* : représente l'approche la plus simple de la composition dans le sens où il s'agit de raccorder la dernière activité du premier processus à la première activité du deuxième processus. Selon les cas, si on décide de composer séquentiellement les modèles pour former un seul modèle géré par un orchestrateur central, on parle de *composition* de modèles. Mais, si on décide de relier les modèles et gérer l'exécution du processus de manière décentralisée sans créer un troisième modèle composite, on parle d'*interconnexion* de modèles ; ce deuxième cas de figure se retrouve dans le schéma de coopération « Exécution chainée ». Dans les deux cas, les modèles de processus sont considérées comme des « boites noires », seules leurs entrées/sorties sont nécessaires à l'établissement de la composition ou l'interconnexion.

(c) *Composition (ou interconnexion) par imbrication de modèles (un modèle dans un autre)* : cette approche permet de déléguer l'exécution d'une partie du processus d'un partenaire à un autre partenaire impliqué dans la coopération. Dans ce cas, on peut envisager une imbrication implicite ou une imbrication explicite. Dans le cas d'une imbrication implicite, on parle d'*interconnexion* de modèles car il ne s'agit pas de créer un troisième modèle composite mais juste une invocation à un processus externe, en envoyant les données nécessaires à l'exécution de la partie déléguée et les deux modèles interconnectés sont gérés chacun, par le partenaire qui le fournit (avec une certaine hiérarchie). Par conséquent, le modèle de processus invoqué est manipulé comme une « boite noire » avec seulement des entrées et des sorties visibles, le modèle de processus principal est perçu comme une « boite grise » puisqu'il faut exhiber la ou les activités du processus à déléguer, les autres parties du modèle peuvent rester invisibles et assimilables à des boites noires. La boite noire représentant le second processus vient remplacer (au niveau exécution) la partie ayant été déléguée du processus principal, c'est le cas de la « Sous-traitance ». Par contre, dans le cas d'une imbrication explicite, il s'agit de construire un modèle composite en remplaçant l'activité à déléguer dans le modèle du processus principal par le modèle du deuxième processus et le processus global sera géré par un coordinateur central.

(d) *Interconnexion pour transfert d'instances :* dans cette approche, les partenaires implémentent le même modèle de processus, l'objectif de la coopération est le partage de charge dans l'exécution des instances de processus. Ainsi, l'interconnexion des modèles consiste à les relier en injectant des règles de transfert sous forme d'activités alternatives afin de spécifier les conditions sous lesquelles il faudra transférer les instances de processus. Dans ce cas, les modèles

de WF sont perçus comme des « boites grises » où il s'agit d'exhiber les points de transfert dans le processus. Entre deux points de transfert consécutifs, la partie du modèle de processus reste abstraite (assimilée à une boite noire). Cette approche est typiquement celle du « Transfert de cas (étendu) ». Le contrôle d'exécution étant décentralisé ou mixte.

(e) *Interconnexion par coordination et échange de messages :* cette approche envisage d'interconnecter les modèles de processus en utilisant des activités d'interaction du type envoi/réception de messages ; ce modèle est typiquement celui du « WF faiblement couplé ». En effet, les partenaires maintiennent leurs modèles de processus, en les augmentant d'une structure d'interaction permettant l'échange de données au moment de l'exécution. Il s'agit réellement, d'une interconnexion de modèles qui sont perçus comme des « boites grises » où il faut montrer les points nécessaires à l'interconnexion. Le contrôle d'exécution étant décentralisé.

**Remarque :** Dans la suite, nous présentons d'autres aspects liés à la composition (ou l'interconnexion) de WF mais nous utilisons le terme *composition* qui regroupe les deux tout en gardant à l'esprit la différence existante entre les deux termes.

### III.5.4 Types de composition de WF

En considérant le degré d'automatisation de la composition, certains auteurs comme (Foster et al., 2003) classent la composition en deux types : la composition manuelle et la composition automatique.

- **La composition manuelle**

La composition manuelle suppose que l'utilisateur génère la composition à la main via un éditeur de texte et sans l'aide d'outils dédiés. Ainsi, l'utilisateur se charge de définir la composition des processus en se basant essentiellement sur sa connaissance des processus et leurs activités. La composition manuelle est considérée par certains auteurs comme l'un des verrous les plus importants au développement des architectures orientées services (Milanovic et al., 2004).

- **La composition automatique**

La composition automatique de WF est un champ d'intense activité (Rao et al., 2005), (Zeshan et al., 2011), (Pessini, 2014), avec des applications possibles à au moins deux domaines majeurs : la modélisation des processus métiers et le Web sémantique.

D'autres auteurs comme (Casati et al., 2001) considèrent que la composition des processus peut être qualifiée de composition statique ou composition dynamique.

- **La composition statique**

La composition statique passe par une phase de spécification durant laquelle les processus sont identifiés, interconnectés, compilés et déployés pour être utilisés.

- **La composition dynamique**

La composition dynamique implique la capacité de sélectionner à la volée, de composer et de faire inter- opérer des processus existants.

### III.5.5 Cycle de vie d'une composition de WF

Le cycle de vie d'une composition de WF est initialisé avec un nouveau besoin métier. Selon (Ramadour et al., 2009), ce cycle de vie comporte quatre phases :

- La phase d'analyse de la composition,
- La phase de conception de la composition,

- La phase d'implantation de la composition,
- La phase d'exécution de la composition.

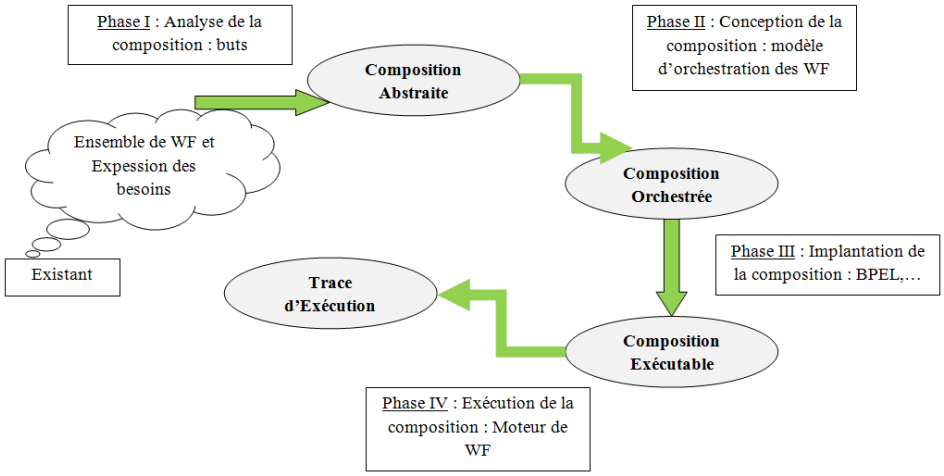



**Figure III.10.** *Cycle de vie d'une composition de WF*

### III.5.5.1 La phase d'analyse de la composition

Cette phase est centrée sur l'analyse du besoin exprimé sous la forme d'un but. Les besoins et donc les buts peuvent être complexes ; ils doivent alors être décomposés pour être réalisés. A ce niveau, la composition de WF est exprimée par un ensemble de sous-buts à réaliser ; elle est appelée composition **abstraite**. Elle ne fait référence à aucun WF, ni à aucune logique métier. Elle est décrite par un graphe de composition qui contient pour un but (le besoin), un ensemble de sous-buts permettant de le réaliser.

### III.5.5.2 La phase de conception de la composition

Cette phase définit une orchestration ou une chorégraphie des WF pour réaliser le besoin de coopération souhaité (partage de capacités, sous-traitance, enchainement, …). A ce niveau, la composition est exprimée sous la forme d'un «*WF global*» et elle est appelée une composition **orchestrée ou chorégraphiée**. Ce WF global fait référence aux WF qui le composent.

### III.5.5.3 La phase d'implantation de la composition

Cette phase consiste à implémenter le WF préalablement conçu, sous forme d'un processus exécutable au moyen d'un langage d'exécution tels que YAWL ou BPEL4WS qui référencie sous forme d'entrées/sorties des spécifications de ce processus. À ce niveau, la composition est dite **exécutable**.

### III.5.5.4 La phase d'exécution de la composition

Cette phase exécute une composition de workflows décrits dans un ou plusieurs fichiers de description de composition (fichiers BPEL) à l'aide d'un SGWF (système de gestion de WF). L'exécution produit un résultat en réponse au besoin. A ce niveau, la composition est appelée **une trace de composition**.

Il est important de souligner qu'au-delà d'organiser le processus d'ingénierie de la composition, ces quatre phases structurent une application à base de WF existants avec quatre niveaux d'abstraction gérés explicitement. Aussi, tous les niveaux de composition (abstraite,

orchestrée, exécutable, trace de composition) sont **réutilisables**. L'intérêt des quatre niveaux d'abstraction est double :

- Ils permettent de réduire la distance entre le besoin du demandeur exprimé sous la forme d'un but métier et les WF mis à disposition par les partenaires.
- Ils permettent d'augmenter le degré d'indépendance d'un besoin avec les WF disponibles et donc augmenter la flexibilité. Par exemple, une composition abstraite décrit un ensemble de buts à réaliser en faisant abstraction des WF existants.

A ce processus de composition, on peut ajouter une phase de validation de la composition qui se concentre sur trois principes directeurs :

- *Complétude* : toutes les activités nécessaires à l'exécution sont-elles opérationnelles.
- *Sûreté* : il ne se produit pas d'erreur inattendue.
- *Correction* : composition sûre et conforme aux exigences fonctionnelles et non-fonctionnelles telles que la performance et la fiabilité.

Cette validation peut se faire de deux manières différentes et à deux niveaux différents :

- Une validation *formelle*, avant la phase d'exécution, en utilisant des formalismes de simulation dédiés tels que les RdP ou les algèbres de processus, avec des outils de vérification formelle adéquats.
- Une validation *expérimentale*, après la phase d'exécution, basée sur l'observation et le recueil des traces d'exécution.

### III.5.6 Les problèmes liés à la composition des Workflows

Les problèmes liés à la composition sont déclinés en deux parties :

- Les problèmes traditionnels de la coopération,
- Les problèmes techniques liés à la composition.

Les problèmes traditionnels de la coopération ont été présentés dans le chapitre I de cette thèse et se résument dans la réticence à l'ouverture, le manque de flexibilité, la protection du savoir-faire, la préservation de l'autonomie et la confidentialité. Ces facteurs s'avèrent contraignants puisque la composition des WF qui vise à atteindre un objectif commun, nécessite une bonne compréhension des processus (ou des sous-processus) à composer et éventuellement, des adaptations au niveau des processus locaux afin de pouvoir les composer avec d'autres.

Les problèmes techniques liés à la composition peuvent se résumer dans les points suivants :

(a) L'hétérogénéité que l'on perçoit à différents niveaux des systèmes WF :
   - L'hétérogénéité des plates-formes.
   - L'hétérogénéité syntaxique des langages de description de WF (ou des modèles de WF).
   - L'hétérogénéité sémantique des données, des processus et des modèles organisationnels.

(b) Difficulté d'intégration et de composition des technologies distribuées, hétérogènes et en évolution rapide.

(c) La validation du WF composite est une tâche très difficile.

(d) La composition peut induire des problèmes tels que la redondance ou les inter-blocages (deadlock).

(e) Difficulté de partage équitable des compétences et des ressources entre les partenaires.

## III.5.7 Approches Existantes de composition de WF

Les travaux sur la composition de WF proposent des approches qui appartiennent à deux communautés différentes. Les premières sont issues du domaine de la composition de services (notamment les services Web), alors que les secondes s'inspirent du domaine de l'intelligence artificielle pour proposer des compositions dynamiques.

### III.5.7.1 Approches de composition orientées services

Les approches de composition orientées services se basent sur le fait qu'une composition de services est définie comme un processus métier notamment un WF à base de services. Dans cette catégorie, on peut trouver les techniques suivantes :

- ***Technique basée sur les graphes*** : dans la plateforme e-Flow proposé par Casati et al. (Casati et al., 2001) pour la spécification, la création de modèles de WF statiques et la gestion de services composites, un modèle de WF est représenté en interne sous forme d'un graphe pouvant être modifié dynamiquement en cours d'exécution. Ce graphe contient, en plus des nœuds représentant les services, des noeuds de décision et d'événements qui permettent de remplacer (au niveau Runtime) des services non disponibles par d'autres services ayant les mêmes fonctionnalités. Cette technique est aussi utilisée dans beaucoup d'autres travaux comme (Hashemian et al., 2005), (Lecué et al., 2008) et (Tripathy et al., 2012) pour une composition dynamique de services.

- ***Technique à base de patrons*** : les auteurs de (Khalaf et al. 2003) proposent une approche de composition à base de patrons qui consiste à créer des patrons de composition pouvant être composés, pour former de nouveaux patrons de composition. Cette solution facilite la composition par une intégration plus simple de patrons d'assemblage prédéfinis dans un nouveau patron mais ne règle pas le problème de la dynamicité de la composition de services. Les auteurs de (Li et al., 2010) proposent un protocole de médiation à base de patrons dans le cadre d'une composition dynamique de services.

- ***Techniques basées sur le Web sémantique*** : ces techniques sont bien adaptées à la composition dynamique de services (Charif et al., 2006). Par exemple, les auteurs de (Laukkanen et al., 2003) proposent d'utiliser les descriptions sémantiques de services Web pour pouvoir comparer les fonctionnalités en utilisant les notions de pré-conditions et d'effets de OWL-S. Les auteurs proposent deux alternatives pour composer dynamiquement des processus métiers : remplacer un service Web dans un processus métier existant par un autre service ayant des fonctionnalités similaires, ou définir un nouveau WF à partir des services Web disponibles.

Dans le même esprit, (Osman et al., 2005) proposent d'utiliser les technologies du Web sémantique, plus particulièrement OWL. Leur objectif est d'améliorer l'opération de recherche de services disponibles, en limitant la recherche de services à un domaine spécifique, en vue de construire dynamiquement des WF traitant du domaine en question tels que l'hôtelerie, le transport ou autre.

Les auteurs de (Rao et al., 2003) ont introduit une méthode de composition automatique de services Web sémantiques en utilisant des preuves de théorèmes logiques linéaires (Linear Logic Theorem Proving). Plus récemment et toujours pour une composition dynamique de services, (Yan et al., 2010) proposent une approche de composition basée sur une ontologie de qualité de service (QoS) et (Rostami et al., 2014) décrivent une approche utilisant le clusturing et les colonies de fourmis.

### III.5.7.2 Approches issues de l'intelligence artificielle (IA)

Parmi les approches de l'IA qui ont été adaptées à la composition de WF, nous retrouvons : la *configuration*, le *"AI Planning"* et les *" Markov Decision Process "*. Notons que ces approches sont valables dans le cas d'une composition dynamique de WF, dans un environnement métier incertain et dépendant des contraintes de disponibilité et d'interopérabilité entre systèmes hétérogènes. A notre sens, ces techniques s'avèrent utiles et intéressantes dans le cas de WF scientifiques (Barker et al., 2008), (Davidson et al., 2008) liés essentiellement à des domaines qui manipulent des volumes importants de données et se basent sur des calculs automatiques tels que la biologie (Romano et al., 2007), la chimie et la météorologie.

Dans le cadre des processus métiers, à notre sens, ces méthodes restent utilisables uniquement dans le cas d'une coopération occasionnelle avec un schéma de coopération du type « sous-traitance », par exemple.

- **La composition de WF à l'aide de la Configuration**

La configuration est une technique émergente de l'IA avec des applications à de nombreux domaines, dans lequel le problème peut être formulé comme la production d'une instance finie d'un modèle objet, soumis à des contraintes. Dans la composition de services, on peut citer (Patrick et al., 2005) et (Mesmoudi et al., 2011). La configuration consiste à construire (simuler la construction) d'un produit complexe à partir de composants choisis dans un catalogue de types décrits de manière hiérarchique. Le nombre et le type des composants requis n'est pas connu à l'avance. La description d'un WF est une instance d'un méta- modèle donné, la composition de WF peut être assimilée à un problème de configuration dans le sens où il est nécessaire d'introduire un certain nombre de transitions et de nœuds (fork, join, split, merge, transformations, séquences prédéfinies d'interaction avec l'utilisateur n'existant pas dans les WF donnés en arguments), et d'interconnecter les messages en entrée et en sortie des actions, pourvu qu'ils aient un type compatible (Patrick et al., 2005).
L'utilisateur d'un système de composition de WF fournit :

- une liste des WF candidats, sous la forme d'instances partiellement définies du méta-modèle utilisé (ex. un producteur de certains objets, un transporteur, un service de paiement électronique, etc.),
- les ontologies pour les types de données des messages entrants/sortants de ces WF (ex. type de voyage, les méthodes de paiement, etc.),
- le but à atteindre pour le résultat de la composition (ex. la réservation d'un ticket de train),
- la liste des informations qui pourront être fournies par l'utilisateur du WF composite (ex. un numéro de carte de crédit, le nom d'un produit, accepter/refuser une offre, etc.).

L'utilisateur du système de composition attend en retour un *WF composite* qui combine les WF candidats en garantissant la validité de toutes les contraintes d'intégrité. Parmi ces contraintes, certaines proviennent du méta-modèle: par exemple des contraintes garantissent que deux (ou plusieurs) WF ne peuvent pas se bloquer mutuellement, chacun attendant que l'autre envoie un message. D'autres contraintes sont plus spécifiques au problème : par exemple celles qui assurent que l'objet produit est effectivement celui qui est transporté.

- **La composition de WF à l'aide de AI Planning**

Par certains aspects, la composition de WF peut être vue comme un problème de planification (i.e un problème orienté but). Ce point de vue est celui de (Martinez et al., 2004) et (Carman et al., 2003), où les descriptions des états sont ambiguës et les définitions des opérateurs sont incomplètes. La même approche est choisie par la bibliothèque de composition interactive de services Web-SWORD qui génère les plans au moyen d'un langage de règles logiques exploitées en chaînage avant. Cette technique permet la prise en compte de contraintes et de préférences des utilisateurs pour former une composition cohérente de services, et est utilisée notamment dans le cas de WF scientifiques, mettant en œuvre des applications orientées calcul. Ces techniques nécessitent en général une interaction avec l'utilisateur qui propose une composition de WF et le système se charge de la vérifier tout en proposant des suggestions pour les actions futures. Un panorama des travaux utilisant le AI planning pour la composition de services se trouve dans Digiampietri et al., 2007). La technique de AI planning est aussi utilisée dans des travaux récents comme (Kuzu et al., 2012), (Markou et al., 2012) et (Omid et al., 2014) pour la composition de services.

- **La composition de WF en utilisant les MDP (Markov Decision Process)**

La composition de WF basée sur les processus de décision de Markov (Doshi et al., 2005), (Chen et al., 2009b), propose de pallier à une insuffisance relevée dans les techniques de composition basée sur la planification. Ces dernières permettent une composition de services ayant un comportement déterministe et peuvent échouer dans des situations d'exceptions non prévues. Par conséquent, la composition de WF en utilisant les MDP permet de recouvrir les situations d'exceptions dans les compositions de services en considérant des services aux comportements non déterministes, pouvant exister dans les environnements dynamiques et incertains. Informellement, un MDP modélise un problème comme un processus stochastique, séquentiel et entièrement observable. Une solution à un MDP produit une politique (ou un plan universel), une politique assigne à chaque état du système, une action qui devrait être optimale pendant une période (période de considération). Si un agent (modélisant un service) détient une politique, il saura quelle action entreprendre ensuite. Selon (Doshi et al., 2005), les MDP permettent d'associer des mesures de qualité à chaque WF afin de sélectionner la solution optimale. En générant efficacement des politiques, les MDP garantissent que les WF produits sont tolérants aux échecs et aux incertitudes.

## Synthèse

Dans ce chapitre, nous nous sommes intéressés à la modélisation et la composition de WF. Nous estimons que la modélisation est une étape primordiale dans le développement de WF et possède différents objectifs dont la description d'un processus métier, la génération de code et l'exécution d'instances ainsi que la vérification de propriétés sur les modèles générés. Après avoir établi les liens existants entre les concepts de base liés à la notion de modélisation, à savoir les concepts de modèle, méta-modèle et langage de modélisation, nous avons parlé brièvement de la composition de modèles dans l'IDM ; ceci nous permettra dans les prochains chapitres, d'examiner dans quelle mesure les critères relevés dans la composition de modèles dans l'IDM peuvent être appliqués à notre approche de composition (ou interconnexion) des modèles de WF, que nous présentons au chapitre V de cette thèse.

Pour revenir à la modélisation de WF, nous avons mis en évidence les différentes perspectives de modélisation de WFIO et nous avons rappelé les différents types de modélisation existant dans

la littérature. Toutefois, dans le cadre de notre travail, nous adoptons une modélisation à base d'activités car nous mettons en avant l'aspect processus dans la modélisation du WFIO (i.e les vues fonctionnelle, comportementale et interactionnelle). Côté langages de modélisation de WF, nous nous sommes arrêtés sur la notation unifiée UML et le formalisme des RdP.

Concernant la question de composition de WF, nous avons jugé nécessaire de faire une distinction entre la *composition* et l'*interconnexion* de processus WF. En effet, la première repose sur la génération d'un seul WF composite, géré de manière centralisée, à partir des WF à composer alors que la deuxième consiste à établir des liens entre les différents processus tout en maintenant un contrôle décentralisé d'exécution. Dans notre cas, les modèles sont des processus BPEL et la composition selon les cas, revient à la construction d'un modèle de processus global ou le maintien des modèles sources en ajoutant des liens permettant leur interconnexion. Nous avons identifié les différents modèles de composition (ou d'interconnexion) inspirés des schémas génériques de coopération définis dans (Van der Aalst, 99), en mettant en évidence le niveau d'abstraction avec lequel, les WF à composer (ou interconnecter) doivent être appréhendés. Ceux-ci peuvent être, selon les cas, complètement abstraits et assimilés à des « boites noires », partiellement abstraits et perçus comme des « boites grises » ou complètement visibles telles des « boites blanches ».

Enfin, il faut distinguer les approches statique et dynamique de composition de WF ; les dernières nécessitent l'établissement de liens sémantiques entre les modèles à composer et se basent généralement sur des techniques issues de l'intelligence artificielle. Dans notre cas, nous nous intéressons à la composition statique ou planifiée qui pour la plupart des auteurs est assimilée à une composition de services (notamment une orchestration de services). A notre sens, nous pouvons dire qu'une composition de services est assimilée à un WFIO, nous pouvons aussi dire qu'un WFIO est assimilé à une composition de services si les WF des différents partenaires sont appréhendés comme des « boites noires », avec uniquement des interfaces visibles ; dans ce cas les techniques de composition de services peuvent être appliquées à la composition de WF pour former un WFIO. Or, dans les modèles de coopération que nous considérons, un WF d'une organisation ne peut pas être toujours perçu telle une « boite noire ». Dans la plupart des cas, certains détails nécessaires à l'interconnexion doivent rester visibles et les autres doivent être dissimulés. Dans ce cas, la composition de WF ne peut pas être assimilée à une composition de services de différents partenaires mais plutôt à la composition d'*orchestrations* de services appréhendées comme des « boites grises ».

Par ailleurs, comme nous l'avons déjà précisé dans le chapitre I de ce document, une fois le modèle de WFIO mis en place, rien ne garantit sa stabilité et sa durabilité dans le temps. Dans un environnement métier instable, soumis à des évènements et des contraintes, les modèles de processus métier doivent être suffisamment flexibles pour supporter les changements et les systèmes qui les implémentent doivent fournir des mécanismes support de la flexibilité de ces modèles. Dans le prochain chapitre, nous abordons la question de flexibilité des modèles de WF, nous rappelons les causes de la flexibilité, ses différents aspects et les différents mécanismes qui ont été proposés dans la littérature pour supporter la flexibilité du WF.

# CHAPITRE IV

# La Flexibilité du Workflow

## Introduction

Dans tous les aspects de la vie, la flexibilité s'est toujours révélée indispensable. Organismes vivants ou entreprises, il s'agit bien souvent d'une question de survie. Sur le marché actuel en pleine mutation, la flexibilité n'est plus une possibilité, c'est une obligation. Avec Internet et un accès quasi-planétaire aux informations, les entreprises doivent faire preuve d'une souplesse sans précédent pour changer sans cesse et s'adapter à leur environnement (Chen et al., 2009a).

Les processus métiers sont d'une importance vitale pour les entreprises, ils constituent leur raison d'être. Une perturbation dans l'exécution de ces processus pourrait être intolérable et mener à des situations embarrassantes pour l'entreprise, notamment dans un environnement de globalisation et de concurrence accrue. Ces perturbations sont généralement dues à des évènements internes ou externes à l'entreprise, de nouvelles contraintes imposées par les clients potentiels ou par un changement des lois, par un changement d'organisation ou simplement suite à un besoin d'amélioration du processus métier. Afin de supporter aisément ces changements, les processus métiers et les systèmes qui les implémentent, particulièrement les systèmes de WF, doivent être suffisamment flexibles et disposant de mécanismes supportant leur adaptation et leur évolution, via un ensemble d'outils adéquats.

La possibilité d'apporter des changements à des processus métiers permet d'ajouter de la flexibilité à ces processus, en fournissant les outils nécessaires pour adapter leur exécution à de nouvelles conditions. D'une manière générale, les adaptations peuvent être i) statiques ou dynamiques ii) manuelles ou automatiques et iii) proactives ou rétroactifs (Courbis et Finkelstein, 2005). Les adaptations statiques sont réalisées grâce à des modifications dans le code source, au niveau « Buildtime », alors que les adaptations dynamiques modifient les caractéristiques logicielles au niveau « Runtime ». Les adaptations manuelles nécessitent une intervention humaine directe dans le système, tandis que les adaptations automatiques sont exécutées par le système lui-même (on parle de système auto-adaptatif) lorsqu'une certaine condition est vérifiée. Enfin, des adaptations proactives se produisent avant un événement spécifique, tandis que les adaptations réactives se produisent en réponse à l'occurrence d'un événement.

La flexibilité est donc une caractéristique importante et largement recherchée par les entreprises (Afflerbach et al., 2014). Dans le contexte des systèmes d'information, (Chelli, 2003) définit la notion de flexibilité comme étant la déclinaison informatique de la notion d'agilité des entreprises, et dans le cas d'entreprise utilisant des outils de WF, cela revient principalement à la flexibilité de ces outils et à leurs capacités de supporter les changements. Cette problématique a poussé un bon nombre de chercheurs à s'intéresser aux aspects de flexibilité des WF, en proposant différentes approches basées sur les caractéristiques des workflows, mais aussi sur les besoins des utilisateurs en termes d'adaptabilité, d'évolutivité et de réutilisabilité. Parmi la panoplie des travaux de recherche ayant traité de la question de flexibilité, nous pouvons citer (Saikali, 2001), (Shonenberg et al., 2007), (Regev et al., 2006), (Bastida et al., 2008), (Weber et al., 2004), (Sadiq et al., 2005), (Minor et al., 2010), (Xiao et al., 2011) et bien d'autres qui seront en partie évoqués dans la suite de ce chapitre.

Nous commençons ce chapitre par rappeler quelques définitions de la flexibilité, les motivations derrière la notion de flexibilité des WF et les taxonomies établies dans la littérature pour la flexibilité des processus métiers, cette dernière est souvent confondue avec la notion d'adaptabilité, englobant l'adaptabilité évolutive (ou évolutivité). Dans notre cas, nous percevons la flexibilité des modèles de WFIO sous trois aspects complémentaires à savoir l'adaptabilité, l'évolutivité et la réutilisabilité de ces modèles. Aussi, nous rappelons les définitions que nous

proposons pour ces trois aspects. Enfin, nous présentons un panorama des approches de flexibilité existantes que nous avons classées en différentes catégories, selon les mécanismes utilisées pour réaliser les changements des WF. Sur la base d'un certain nombre de critères que nous avons dégagés, nous établissons une comparaison de ces approches accompagnée d'une discussion.

## IV.1 Définitions de la flexibilité

Dans la littérature, plusieurs définitions de la flexibilité ont été proposées, nous en retenons les suivantes :

- **Définition IV.1** : Notion de flexibilité

*La flexibilité permet à l'entité qui la présente de résister à des forces et des contraintes sans se briser, se rompre ou se déformer définitivement.*

- **Définition IV.2** : Flexibilité du WF

*La flexibilité du Workflow est définie comme étant la capacité de pouvoir changer la définition d'un WF ainsi que celle de réutiliser les modèles pour la conception d'autres workflows* (Weske et al. 96).

- **Définition IV.3** : Flexibilité d'un système

*La flexibilité d'un système peut être définie par deux propriétés : celle de s'adapter à des situations ou des configurations nouvelles, et la capacité d'absorber les fautes ou les problèmes.* (Saikali, 2001),

Aussi, il distingue deux niveaux complémentaires de flexibilité du Workflow : la flexibilité du *modèle* WF et la flexibilité du *système* WF.

- ***Flexibilité du modèle WF*** : un modèle de WF flexible doit présenter un ensemble de propriétés dont l'adaptabilité et l'évolutivité du modèle sans remettre en cause son intégrité et sa cohérence globale, la prise en compte d'exécutions alternatives et la réutilisabilité en vue de construire d'autres modèles.
- ***Flexibilité du système WF*** : un système WF flexible est un système qui propose des mécanismes permettant de détecter des situations erronées (exceptions) et d'y faire face. Il doit fournir des mécanismes pouvant être mis en œuvre en cours d'exécution des instances de WF.

- **Définition IV.4** : Flexibilité de la modélisation

Selon (Regev et al., 2006), l*a notion « flexibilité de la modélisation » est utilisée pour désigner la capacité de mettre en œuvre les changements dans un processus métier en ne modifiant que les parties qui ont besoin d'être changées tout en gardant la stabilité des autres parties du processus. Une modélisation de processus est flexible si elle est capable d'être modifiée sans la remplacer complètement.*

Dans la suite, nous rappelons notre définition de la flexibilité d'un modèle de WF.

- **Définition IV.5** : Flexibilité d'un modèle de WF

*Un modèle de WF est dit flexible s'il peut subir des opérations d'**adaptation** ou d'**évolution** et s'il peut être **utilisé** pour la construction d'autres modèles, sans remettre en cause la cohérence du modèle initial.*

Nous reviendrons plus loin sur les aspects d'adaptation, évolution et réutilisation de modèles

## IV.2 Motivations de la flexibilité dans les WF

La flexibilité est une caractéristique nécessaire dans les WF qui sont souvent sujets à des changements imposés par des situations récurrentes et des contraintes qui proviennent de l'environnement des entreprises ou qui sont liées à des événements internes à l'entreprise elle-même (Singh et al., 2013). Ces situations imposent aux processus de WF d'être aisément adaptables afin de supporter efficacement les changements. Dans la suite, nous énumérons les principales situations auxquelles les WF doivent impérativement s'adapter:

### IV.2.1 Situations relatives à l'évolution des méthodes de travail

- *L'instabilité de l'environnement de travail*

Les méthodes de travail, les environnements, ainsi que toutes les branches d'ingénierie sont extrêmement dynamiques et en constante évolution. Afin de suivre cette évolution, deux solutions s'offrent à l'entreprise : la refonte radicale du processus métier ou un ajustement continuel mais très coûteux de son WF. Dans un contexte inter-organisationnel, l'instabilité de l'environnement de travail peut être due à la disparition d'un partenaire, la rupture d'un contrat,…etc.

- *Les avancées technologiques*

Les systèmes logiciels sont confrontés à des exigences d'évolution causés par les progrès techniques qui conduisent souvent à la reconfiguration des systèmes, par exemple, le remplacement et la mise à jour de composants logiciels, l'ajout de nouveaux composants, et le changement dans les interfaces des composants.

### IV.2.2 Les changements ad-hoc au niveau Run-time

- *Le besoin de modélisation tardive*

Dans certains cas, il est impossible de concevoir une spécification complète d'un modèle de processus WF. En raison de l'indisponibilité d'une description complète, le raffinement dynamique peut être nécessaire à l'exécution. C'est seulement lors de l'exécution, que certaines tâches peuvent être entièrement précisées. La même chose s'applique pour les interdépendances entre tâches ainsi que les ressources requises pour l'exécution des tâches.

- *Prise en compte de la participation des « décideurs » dans le processus*

Certains utilisateurs dans un système de WF doivent être traités comme des propriétaires du processus. En réalité, un utilisateur doit souvent ajuster et situer ses activités en fonction de circonstances spécifiques sous l'influence d'événements extérieurs et de son style de travail personnel. Les utilisateurs qui sont « décideurs » doivent être pris en compte dans l'exécution du processus WF.

- *Les situations d'erreur et les événements imprévus*

Un crash système, un conflit entre ressources, des opérations erronées, un enchainement incohérent entre activités ou des interactions incorrectes entre fragments de processus sont des situations qui peuvent causer des erreurs et des difficultés dans l'exécution des processus WF. De plus, les événements imprévus comme les routines externes, l'intervention des utilisateurs ou autre doivent être correctement capturés et traités, pour résoudre des problèmes liés au contexte

d'exécution du processus. Par conséquent, des mécanismes de gestion des erreurs et des événements imprévus sont très importants pour assurer un progrès sain des processus WF.

### IV.2.3 Le besoin de support pour l'évolution et la réutilisation du modèle

La définition et la gestion des procédés est une tâche critique et longue dans chaque organisation qui veut implanter un système de WF. En effet, il est difficile de définir un « bon » procédé en une seule fois. Pour faciliter l'acceptation des systèmes de WF, leur intégration doit être mise en œuvre de manière graduelle (Grigori, 2001). Un système de WF peut aider les utilisateurs à faire cette transition en plusieurs étapes :

- Une définition incomplète avec des modifications dynamiques au fur et à mesure des besoins exprimés par les utilisateurs, les opérations de modification dynamique doivent être simples et ne doivent pas conduire à des erreurs d'exécution ou à des états incohérents.
- L'analyse et l'amélioration du procédé, afin de prendre en charge et traiter les différentes exceptions pouvant survenir.
- L'amélioration du processus liée au besoin d'augmenter le parallélisme entre procédés
- Extension du processus pour couvrir de nouvelles fonctionnalités
- Dans un contexte inter-organisationnel, il peut s'agir d'une évolution de la coopération nécessitant d'introduire un nouveau fragment de processus fourni par un nouveau partenaire.
- *La réutilisation du modèle ou d'une partie du modèle :* pour économiser un effort de conception et de développement lors de la définition de nouveaux WF (partageant des points communs avec des WF existants) et pour adapter un schéma général à des cas plus spécifiques.

### IV.2.4 Les besoins de flexibilité des flots de données

Dans les modèles de WF traditionnels, l'échange de données entre activités est soit non considéré et la cohérence des données reste entièrement à la charge des programmeurs de l'application, soit trop restrictif (Grigori, 2001). Une autre contrainte est qu'une activité peut lire une donnée d'une autre activité, seulement si les deux activités sont connectées par le flux de contrôle. Ces restrictions sont trop fortes pour les types d'interactions existantes dans les applications coopératives. Il est donc nécessaire de trouver un nouveau modèle de gestion de flot de données aussi simple à utiliser et aussi sûr qu'un modèle de transaction traditionnel, mais suffisamment flexible pour :

- Permettre aux activités de s'échanger des données en cours d'exécution.
- Permettre à des activités situées dans des branches parallèles de modifier les mêmes données.
- Ne pas imposer que tous les flots de données soient complètement définis à l'avance.

## IV.3 Taxonomies de la flexibilité dans les processus métiers

Plusieurs travaux ont été menés pour tenter de proposer une taxonomie générique de la flexibilité des processus métiers. Dans ce contexte, deux grandes taxonomies ont été proposées : la première est celle de (Regev et al., 2006) et la deuxième est celle de (Schonenberg et al., 2007).

### IV.3.1 La taxonomie de Regev et al. (2006)

La taxonomie proposée dans (Regev et al. 2006) s'intéresse aux changements qui peuvent survenir durant le cycle de vie d'un processus métier. Cette taxonomie tente de répondre aux deux questions suivantes : (i) A quel niveau du processus réaliser le changement ? (ii) Quand réaliser le changement ? Elle repose sur trois dimensions principales :

(a) *Le niveau d'abstraction du changement* qui concerne le niveau d'application du changement dans un processus métier. Le changement peut concerner soit la spécification du processus (i.e le modèle de processus) ou l'instance du processus.

(b) *L'objet du changement* qui concerne les différents aspects du processus qui sont sujets au changement. Le changement peut concerner les activités du processus (dimension *fonctionnelle*), le contrôle de flux (dimension *comportementale*), les données du processus (dimension *informationnelle*) ou les différents protocoles utilisés dans le processus (dimension *opérationnelle*) ; dans un contexte inter-organisationnel, cette dernière dimension englobe la dimension *interactionnelle*.

(c) *Les propriétés du changement* comme : le *degré* du changement qui peut être partiel pour changer une partie du processus ou total (radical) pour créer un nouveau processus ; la *durée* du changement qui peut être temporaire ou permanent ; la *rapidité* d'application du changement qui peut être soit immédiate ou différée et l'*anticipation* du changement qui peut être soit planifié ou ad-hoc.

### IV.3.2 La taxonomie de Schonenberg et al. (2007)

Schonenberg et al. proposent une deuxième taxonomie qui s'intéresse à la flexibilité du point de vue opérationnel (Schonenberg et al., 2007). Pour cela, cette taxonomie tente de répondre à la question: Comment réaliser le changement dans le processus ? Elle distingue quatre manières de réaliser ce changement :

(a) *La flexibilité par conception* concerne la possibilité d'exprimer des chemins alternatifs d'exécution dans une modélisation du processus et cela en offrant la possibilité d'exprimer le parallélisme entre les activités, le choix de l'exécution d'une activité parmi plusieurs, l'instanciation multiple d'une activité dans la même exécution ou encore l'annulation de l'exécution d'une activité.

(b) *La flexibilité par déviation* concerne la possibilité de dévier l'exécution d'une séquence d'une instance de processus de sa définition initiale en permettant par exemple de défaire, de refaire ou de court-circuiter certaines activités.

(c) *La flexibilité par spécification partielle* concerne la possibilité d'exécuter un processus partiellement spécifié c.-à-d. la possibilité de définir certaines structures du processus dans la phase d'exécution en permettant la sélection tardive d'un fragment de processus parmi plusieurs alternatives ou la modélisation tardive pour spécifier des fragments à la volée durant l'exécution du processus.

(d) *La flexibilité par changement* concerne la possibilité de modifier la définition du processus dans la phase d'exécution en permettant à toutes les instances du processus existantes de migrer vers la nouvelle définition du processus.

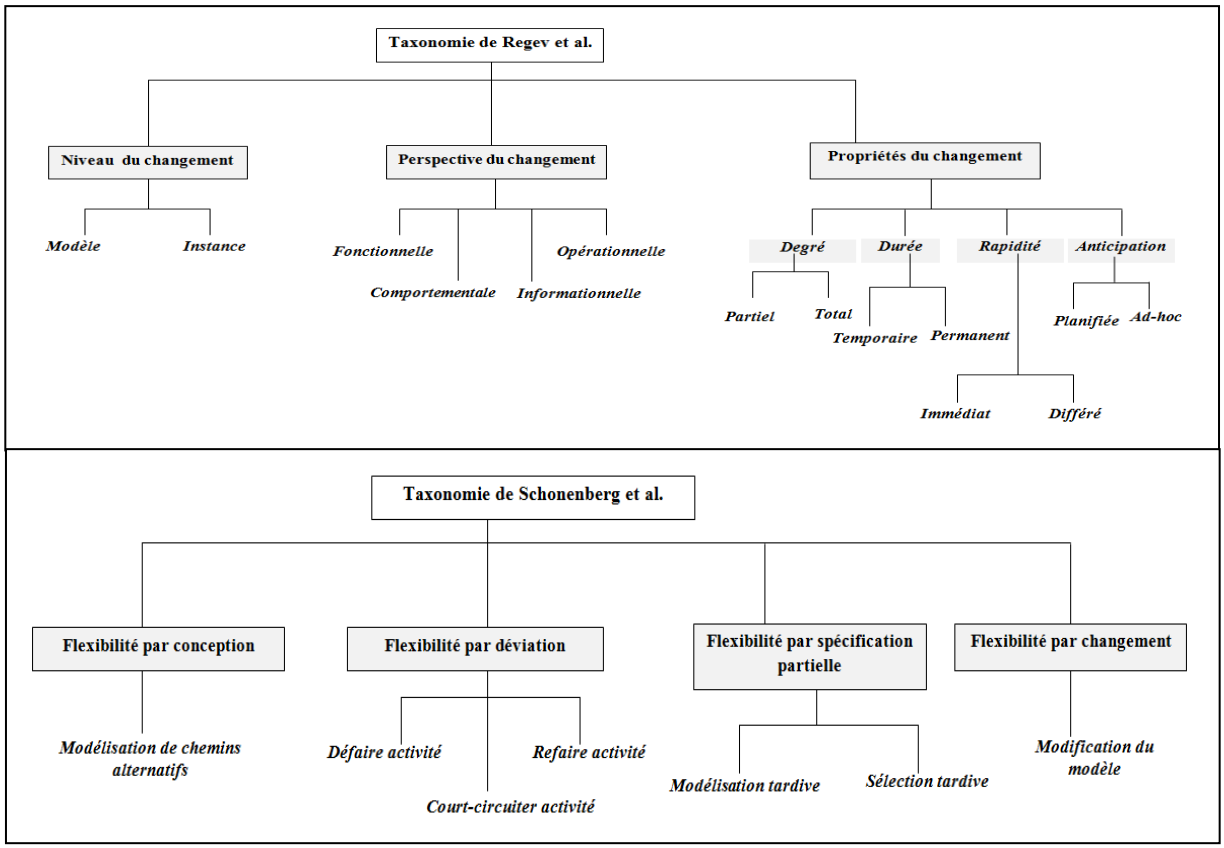



**Figure IV.1.** *Taxonomies de la flexibilité des processus métiers*

Par ailleurs, d'autres travaux comme (Han et al., 98), (Saikali, 2001), (Papazoglou et al., 2008) parlent plutôt d'adaptabilité (ou d'adaptation) d'un WF au lieu de flexibilité. Par exemple, Saikali dans sa thèse (Saikali, 2001), considère l'adaptabilité d'un WF comme une forme particulière de flexibilité du WF qui définit sa capacité à modifier la définition du modèle de processus sur lequel il est basé. Il distingue deux aspects de l'adaptabilité, dans le cas où l'adaptation est détectée et est réalisée par le système ou par un de ses agents, le système et le modèle sont dits *adaptatifs*. Alors que si les adaptations sont réalisées par une personne, via les moyens fournis par le système, le WF est dit *adaptable*.

Aussi, selon les besoins liés aux adaptations et les entités sur lesquelles s'effectuent les changements, les adaptations peuvent être distinguées conformément à une taxonomie établie. La taxonomie de (Han et al., 98) propose de distinguer les niveaux de l'adaptation selon le contexte, l'infrastructure, le processus et les ressources. Des années plus tard, les auteurs de (Papazoglou et al., 2008) proposent une autre taxonomie, appliquée notamment à la composition de services vue comme un processus métier ; cette deuxième taxonomie se focalise sur l'aspect processus et distingue deux types d'adaptation : une adaptation horizontale et une adaptation verticale.

### IV.3.3 La taxonomie de Han et al. (1998)

Les auteurs de (Han et al., 98) identifient quatre catégories d'adaptation dans un WF, considérées à différents degrés d'abstraction pour répondre à la question : A quel niveau réaliser le changement ou l'adaptation ? Les quatre catégories se résument dans:

(a) *L'adaptation au niveau du contexte (ou domaine) :* le rôle joué par le système WF est défini par un certain contexte organisationnel et métier qui définit la configuration des systèmes

d'information de l'entreprise. La modification du contexte peut avoir un impact sur la façon dont le système WF s'intègre dans son environnement.

(b) *L'adaptation au niveau des processus :* concerne principalement les modèles de WF et leurs composants. L'adaptation des modèles de WF s'effectue de manière dynamique lorsque les processus sont en cours d'exécution. Dans ce type d'adaptation, il est souvent nécessaire d'intégrer les dérivations et les extensions ad-hoc en vue de contrôler les processus de WF de manière flexible.

(c) *L'adaptation au niveau des ressources* : concerne l'adaptation de l'organisation, par exemple les changements de personnel ou de structure, qui peuvent avoir un impact direct sur la cohérence du WF et l'adaptation liée aux données en cas de modification de ces dernières. Dans cette catégorie, on retrouve :

- *L'adaptation des ressources liée à l'organisation* : concerne les changements dans l'organisation des structures de l'entreprise et des ressources qui y sont rattachées, par exemple les changements de personnel peuvent avoir un impact direct sur l'exécution des instances de processus WF. Ainsi, tout changement dans la structure de l'organisation doit se refléter sur les ressources des WF associés. La manière dont ces changements peuvent être réfléchis dépend des mécanismes mis en place pour faire face à ce type d'adaptation.

- *L'adaptation des ressources liée aux données* : les données et les structures de données peuvent changer au cours de l'exécution d'un processus WF. Habituellement, les données qui ne sont pas utilisées par un WF peuvent être changées de façon indépendante et sont accessibles par plusieurs autres applications simultanément. Mais si l'exécution d'un processus WF dépend de l'existence d'un élément de donnée ou d'une propriété spécifique de celui-ci, alors le processus WF doit être adapté aux changements de données. Bien sûr, cela exige que les changements liés aux données soient portées à la connaissance du SGWF afin qu'il puisse réagir aux changements.

(d) *L'adaptation au niveau de l'infrastructure :* elle concerne tous les changements dus aux avancées techniques et technologiques qui nécessitent d'adapter les configurations des systèmes d'information et des applications sous-jacentes.

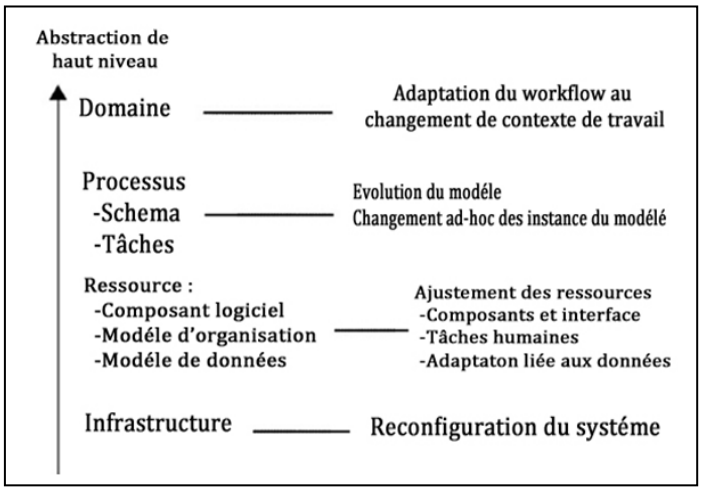



**Figure IV.2.** *Catégories des adaptations de WF selon (Han et al., 98)*

Notons que l'adaptation au niveau processus et l'adaptation au niveau des ressources rejoignent en grande partie, la taxonomie de flexibilité des processus métiers établie dans (Regev et al., 2006).

### IV.3.4 La taxonomie de Papazoglou et al. (2008)

La classification de (Papazoglou et al., 2008) a été établie dans un contexte de composition de services considérée comme un processus métier (un workflow). Dans une composition dynamique de services, notamment des services Web, l'environnement métier est souvent instable, des services peuvent apparaître et disparaitre. En général, le but recherché est une meilleure qualité de service, un service peut présenter plusieurs interfaces, chacune utilisée dans un contexte bien défini ; on parle souvent d'une adaptation pour une meilleure qualité de service, une adaptation dépendant du contexte et/ou une adaptation interne du service lui-même afin de répondre à certaines contraintes imposées.

Les auteurs dans (Papazoglou et al., 2008) considèrent les adaptations au niveau processus et distinguent deux catégories d'adaptation : les adaptations *verticales* et les adaptations *horizontales*, selon que les changements apportés affectent la structure du processus (contrôle de flux entre les services) ou non.

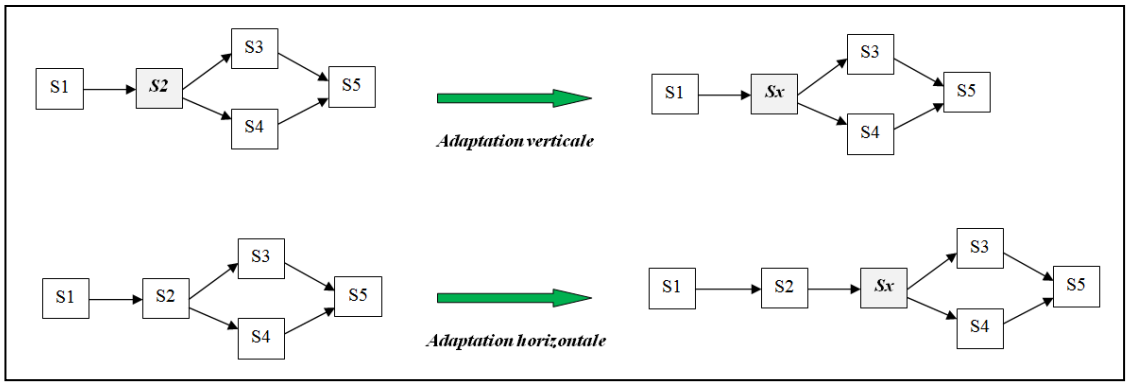



**Figure IV.3.** *Catégories des adaptations de WF selon (Papazoglou et al., 2008)*

(a) *Une adaptation verticale* des processus métiers se réfère aux modifications effectuées au sein d'un service (c'est à dire, la reconsolidation du service), qui n'affectent pas la structure ou la séquence d'exécution des services composant le processus. Jusqu'à présent, c'est là que la plupart des travaux concernant l'adaptation ont été développés. En effet, l'une des principales préoccupations de l'adoption des services Web comme canal pour exécuter des processus métiers est la qualité de service (QoS) et la nécessité de respecter les accords au niveau des services (Service Level Aggreement - SLA), à l'égard de certains fournisseurs de services. La plupart de ces travaux comme (Canfora et al., 2008), (Bastida et al., 2008), (Strunk et al., 2009) ont pour but d'améliorer la qualité de service et prévenir les violations au niveau SLA. Pour ce faire, ils s'appuient sur l'analyse des services de plusieurs fournisseurs et la sélection de l'offre qui satisfait au mieux les métriques définies.

(b) *Une adaptation horizontale* des processus métiers fait référence aux modifications apportées à la structure du modèle de processus telles que l'ajout ou la suppression d'activités (services) qui de suite modifient le comportement du processus (Charfi et al., 2009), (Leitner et al., 2010). Les adaptations horizontales permettent une manipulation plus large du processus de l'entreprise, par rapport à des approches verticales, qui sont limitées à des modifications au niveau du service. Le but de ce genre d'adaptation est de permettre au processus métier de répondre aux nouveaux besoins en modifiant son comportement sans avoir besoin de le redéployer.

En résumé, à partir des définitions et taxonomies décrites plus haut, on peut dire que la flexibilité du WF ne recouvre pas un seul mais plusieurs domaines et aspects, que sont : les processus, les données et l'organisation. Par conséquent, les contraintes de flexibilité du WF ne s'appliquent pas à un seul domaine mais à la combinaison des trois. Dans la pratique, on constate cependant que la flexibilité du WF se reflète principalement au niveau des processus, bien qu'elle implique et s'appuie également sur les deux autres niveaux : données et organisation. Par ailleurs, comme nous l'avons déjà souligné, la flexibilité du WF se définit à deux niveaux complémentaires : au niveau des systèmes et au niveau des modèles de processus (Saikali, 2001).

Ainsi, la nécessité de prise en compte de la dynamique des différents éléments d'un processus nous pousse à faire plus attention à la flexibilité et la souplesse de la *modélisation* du processus métier. Dans le cadre de cette thèse, nous nous intéressons à la *flexibilité des modèles de processus*, particulièrement des processus de WFIO. Dans la suite, nous revenons aux trois aspects que nous avons identifiés pour définir la notion de flexibilité d'un modèle de WFIO.

## IV.4 Aspects de flexibilité d'un modèle de WFIO

Rappelons qu'un modèle de WFIO se définit selon quatre perspectives complémentaires : la perspective fonctionnelle (sous-processus et activités), la perspective interactionnelle (structure d'interaction entre deux fragments de WF), la perspective organisationnelle (l'ensemble des partenaires impliqués dans le processus) et la perspective informationnelle (données manipulées par le processus).

Globalement, un modèle de WFIO est flexible s'il supporte aisément des modifications (ajout, suppression ou mise à jour) sur une ou plusieurs de ses perspectives de modélisation sans perturber la cohérence du modèle initial. Pour être plus précis, nous distinguons trois aspects complémentaires de la flexibilité d'un modèle de WFIO: adaptabilité, évolutivité et réutilisabilité.

### IV.4.1 Adaptabilité d'un modèle de WFIO

- **Définition IV.6** : Adaptabilité d'un modèle de WFIO

L'*adaptabilité* d'un modèle de WFIO désigne sa capacité à supporter des modifications (ajout, suppression et mise à jour) dans les entités qui le composent sans altérer la fonctionnalité globale du processus ni l'ensemble des partenaires impliqués dans le processus.

Nous nous inspirons des différents types d'adaptation (correctionnelle, adaptative, évolutive et perfective) identifiées dans (Ketfi et al, 2002) pour les applications logicielles afin de les translater au domaine du Workflow.

- *Adaptation correctionnelle ou corrective :* consiste à examiner le WF en cours d'exécution et à l'adapter s'il ne se comporte pas correctement ou comme le prévoyait son cahier des charges. Ce type d'adaptation concerne la correction des erreurs enregistrées dans le modèle de processus, il peut s'agir par exemple, d'une activité mal implémentée, un flux de contrôle incorrect, une donnée d'entrée non conforme à la donnée attendue, ou une mauvaise affectation de ressources,…etc.

- *Adaptation perfective* : ce type d'adaptation peut être appliqué dans un but d'amélioration du processus métier afin d'apporter une meilleure efficience et efficacité dans un contexte de CPI (Continous Process Improvement); par exemple il peut s'agir d'une parallélisation de deux activités initialement séquentielles, une réaffectation de ressources ou autre. L'objectif de telles modifications est d'améliorer les performances de l'application WF.

- *Adaptation adaptative* : considère que même si le processus WF s'exécute correctement, parfois son environnement d'exécution peut changer et nécessite son adaptation. Ce type d'adaptation est donc requis dans le cas de changements dans l'environnement du processus, nouvelles lois impliquant de nouvelles règles métier. Dans un contexte de BPR (Business Process Reengineering), il s'agit de s'aligner à ces nouvelles lois ou exigences métier, un exemple de ce type d'adaptation est la re-conception d'une partie du processus (ajout, suppression, modification d'activités et des conditions de leur application).

- *Adaptation évolutive* : s'intéresse à l'ajout de certaines fonctionnalités qui n'ont pas été prises en compte au moment du développement de l'application WF. Dans notre cas, nous classons cette dernière catégorie dans l'*évolutivité* des modèles.

Rappelons que nous nous intéressons aux adaptations sur les perspectives fonctionnelle, comportementale et interactionnelle d'un modèle de processus WFIO. Nous parlerons d'une adaptation *interne* (ou locale) si le changement s'effectue sur un fragment local d'un WF, et d'une adaptation *externe* (ou globale) si le changement touche à la structure d'interaction entre deux processus WF impliqués dans la coopération.

### IV.4.2 Evolutivité d'un modèle de WFIO

L'évolutivité d'un modèle de WFIO est une forme d'adaptabilité dite « adaptabilité évolutive » qui peut être classée dans l'adaptation évolutive. Dans le cas d'un WFIO, nous la distinguons par deux aspects d'évolution qui sont *l'évolution des fonctionnalités* et *l'évolution de la coopération*. L'évolution des fonctionnalités consiste à étendre le WF existant avec de nouvelles activités qui remplissent des fonctionnalités additionnelles. L'évolution de la coopération quant à elle, est liée au fait qu'un nouveau partenaire puisse être impliqué dans le WFIO dans un but d'extension du processus et d'ouverture vers l'environnement extérieur. Il faut noter que l'évolution de la coopération induit souvent une évolution des fonctionnalités.

- **Définition IV.7** : Evolutivité d'un modèle de WFIO

L'*évolutivité* d'un modèle de WFIO désigne sa capacité à supporter une expansion de sa fonctionnalité globale ou une expansion de la coopération (i.e ouverture à de nouveaux partenaires) sans altérer la cohérence du processus initial.

### IV.4.3 Réutilisabilité d'un modèle de WFIO

La réutilisabilité d'un modèle de WFIO est un aspect de la flexibilité qui permet la réutilisation d'un modèle de WFIO existant afin de construire d'autres modèles plus complexes. L'intérêt est de pouvoir manipuler le WFIO réutilisé comme une entité à part entière telle une « boite noire » qui peut facilement s'intégrer dans un modèle de WFIO global. Cette « boite noire » doit pouvoir subir aussi des adaptations internes sans altérer la cohérence du processus global dont elle fait partie.

- **Définition IV.8** : Réutilisabilité d'un modèle de WFIO

La *réutilisabilité* d'un modèle de WFIO définit sa capacité à s'intégrer aisément à d'autres modèles, sans altérer la cohérence du modèle global, dans le but de construire des modèles de WFIO plus complexes, par réutilisation de modèles existants.

Notons que dans la littérature, la notion de flexibilité d'un modèle de WF est souvent confondue avec l'adaptabilité du modèle (incluant parfois l'évolutivité avec des définitions différentes d'une référence à l'autre).

## IV.5 Profondeur du changement

Saikali dans sa thèse (Saikali, 2001), définit la profondeur d'un changement (ou d'une adaptation) comme étant la portée du changement sur les instances de processus et/ou le modèle de processus. En d'autres termes, selon les instances de processus concernées par l'adaptation, celle-ci peut se situer à différents niveaux de profondeur qui dépendent de la portée des modifications par rapport aux instances et au modèle de processus WF lui-même. Ces niveaux sont : le niveau "local", le niveau "local généralisé" et le niveau "global" comme le montre la figure IV.4.

### IV.5.1 Niveau local

Le niveau local représente le niveau le plus superficiel et correspond à une adaptation appliquée sur une instance particulière. Ce type d'adaptation est applicable pour l'exécution d'une instance particulière dont le comportement ne répond pas aux conditions et aux alternatives prévues dans le modèle. Par exemple, dans un processus de fabrication de composants électroniques, la phase de validation des paramètres liés à un composant particulier ne fournit pas les résultats attendus, le processus de fabrication ne peut donc pas se poursuivre normalement ; deux cas sont possibles pour répondre à cette situation exceptionnelle : soit abandonner le composant (donc l'instance du processus) ou alors refaire la phase de test pour une éventuelle validation.

### IV.5.2 Niveau local généralisé

Ce niveau représente un niveau plus profond que le niveau local et correspond proportionnellement à un besoin d'adaptation plus sérieux. Ce niveau englobe toutes les instances d'un WF en cours d'exécution. L'adaptation au niveau local généralisé ne remet en cause que la validité des instances en cours d'exécution. Par suite, elle ne s'applique pas aux nouvelles instances du processus. Il s'agit de nouvelles contraintes mais qui restent applicables temporairement car elles dépendent d'une exécution déjà effectuée en partie. Par exemple, dans un processus de surveillance de patients hospitalisés, tous les patients ayant déjà subi une analyse de type X doivent impérativement refaire cette analyse (duplication d'une activité du processus) car il s'avère que le réactif utilisé n'était pas de bonne qualité. Par contre, les nouvelles instances ne seront pas concernées par ce changement.

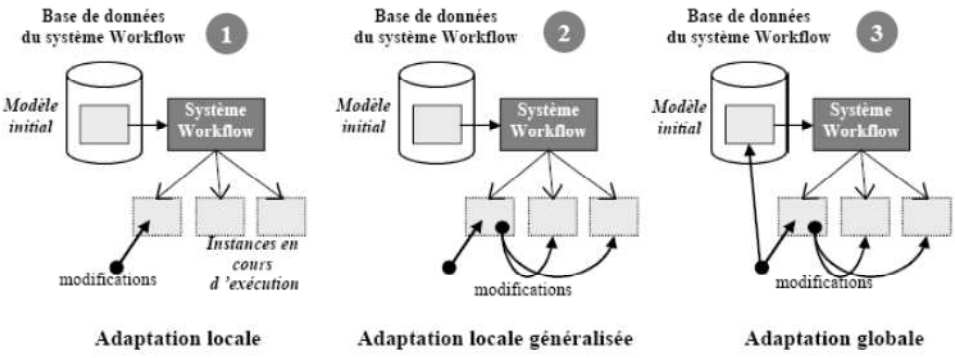



**Figure IV.4.** *Profondeur de l'adaptation* (Saikali, 2001).

### IV.5.3 Niveau global

Le niveau global est le plus profond de l'adaptation car les adaptations sont directement répercutées sur toutes les instances en cours d'exécution mais aussi et surtout sur le modèle de processus. Notons que les nouvelles instances sont elles aussi concernées par les modifications. Il s'agit de situations où une nouvelle loi est établie et doit impérativement être appliquée, induisant une modification dans la logique métier du processus. Par exemple, on peut considérer une re-conception dans le processus de fabrication d'un produit donné (permutation de deux activités dans le processus, ajout d'une activité, ajout d'une condition, … etc.)

## IV. 6 Approches et méthodes existantes de flexibilité des WF

D'après (Han et al., 98), il existe deux catégories principales d'approches pour la mise en œuvre de la flexibilité du WF : les approches *ponctuelles* et les approches par *méta-modèles*.

### IV.6.1 Approches ponctuelles

Les approches ponctuelles ("*open point approaches*") consistent à définir certains points dans un modèle de WF à partir desquels des adaptations peuvent être apportées : on les appelle également points "d'entrée". Ces approches englobent différentes formes de réalisation pouvant être classées en trois techniques : (i) les *choix multiples*, (ii) *l'allocation dynamique des ressources* et (iii) la *modélisation tardive* et la *modélisation ad-hoc*.

#### IV.6.1.1 La technique des choix multiples

Cette technique représente l'expression la plus simple des approches ponctuelles. Elle consiste à placer des "bornes" au sein du modèle de processus. Chaque borne, quand elle est atteinte, permet de stopper l'exécution du processus pour offrir à l'utilisateur plusieurs alternatives à la poursuite de l'exécution. Une borne est en général associée à une activité ou à un événement ayant lieu dans le processus. La technique des choix multiples est utilisée lorsque l'on ne peut pas fixer des choix lors de la conception du WF mais qu'on a une évaluation globale de la situation, qui permet de concevoir différentes alternatives. Deux modes sont possibles : le premier mode est de type "actif", ce sont les utilisateurs qui interviennent sur l'exécution du processus et indiquent leur choix au moteur de WF. Le deuxième mode est de type "passif", c'est le moteur de WF qui relance les utilisateurs afin de connaître l'évolution du processus, en leur proposant éventuellement des sous-processus possibles.

Une évolution hybride de cette technique permet à l'utilisateur d'introduire des "morceaux" de processus ou des sous-processus, stockés dans une librairie (réutilisation de modèles). Cette dernière technique est en fait à l'intersection entre celle des choix multiples et la modélisation tardive. Elle est notamment proposée par le système WASA (Weske, 96) pour les "WF Scientifiques" qui donne aux utilisateurs, le moyen de choisir un sous-processus dans une librairie, afin de compléter le WF en cours d'exécution. Cette technique est aussi mise en oeuvre dans plusieurs travaux comme (Colombo et al., 2006), (Bastida et al., 2008), (Xio et al., 2011).

#### IV.6.1.2 L'allocation dynamique de ressources

La technique d'allocation dynamique de ressources permet essentiellement de résoudre les problèmes dus à des conflits d'attribution de ressources (qu'elles soient matérielles ou humaines) lors de l'exécution du WF. En effet, la plupart des langages de modélisation de WF nécessitent de fixer l'affectation des ressources aux tâches lors de la phase de modélisation. Une première approche est de confier la résolution du problème à des applications d'ordonnancement, qui se

chargeront de trouver la meilleure solution. Une autre solution plus intéressante est proposée par l'allocation dynamique de ressource, qui s'intéresse à éviter le problème plutôt qu'à le résoudre. Cette approche consiste à associer aux tâches des types de ressources (en termes de rôles), plutôt que les ressources elles-mêmes.

L'allocation dynamique de ressources est entre autres, mise en oeuvre dans le système "Endeavor" (Bolcer et al. 96), (Kammer et al. 98) sous une forme assez évoluée. En effet, l'allocation dynamique y est utilisée pour l'attribution des ressources aux tâches mais également, pour choisir dynamiquement le comportement adéquat d'un objet du système, parmi l'ensemble de ses comportements. La possibilité d'allouer dynamiquement un comportement est une évolution de l'allocation dynamique de ressources, appelée "*late binding*" (Joeris, 2000) ou "liaison tardive de comportement". Elle se situe à mi-chemin entre l'allocation dynamique et la modélisation tardive. L'allocation dynamique des ressources est aussi utilisée dans bon nombre de travaux comme (Momotko et al., 2002), (Governatori et al., 2004), (Tsai et al., 2010), (Huang et al., 2011), (Hoenisch et al., 2013).

Par ailleurs, dans ce contexte et dans un autre cadre de recherche, nous avons travaillé sur l'allocation dynamique de ressources et nous avons proposé une approche d'allocation de ressources supportée par un algorithme d'affectation de tâches au niveau Runtime du processus, l'algorithme prend en compte la charge de travail effective et estimée des acteurs intervenant dans le processus WF, pour l'attribution des tâches, dans un but d'équilibrage de charge dans le système (Boukhedouma et al., 2008).

### IV.6.1.3 La modélisation tardive et la modélisation ad-hoc

L'expression "modélisation tardive" regroupe en fait deux déclinaisons sous une même appellation : la "modélisation tardive" proprement dite ("*late modelling*") et "l'affinage dynamique" des modèles ("*dynamic refinement*"). Dans les deux cas, il s'agit d'ajouter des éléments au modèle de processus. La modélisation ad-hoc quant à elle, est une étape plus évoluée qui consiste à modéliser en cours d'exécution (donc à la volée), certaines parties manquantes du processus. Celles-ci sont représentées au préalable, de manière abstraite, par des éléments génériques (des "boîtes noires") qui sont remplacés par la suite par une modélisation "on line", adéquate et détaillée. Un exemple de modélisation tardive est donné par le moteur Workflow du système "Baan" (Meijler et al. 98). Le système prévoit le cas où les concepteurs d'un modèle de WF ne peuvent connaître à l'avance l'ensemble des activités d'un processus, mais qu'ils sont capables d'en définir une importante partie.

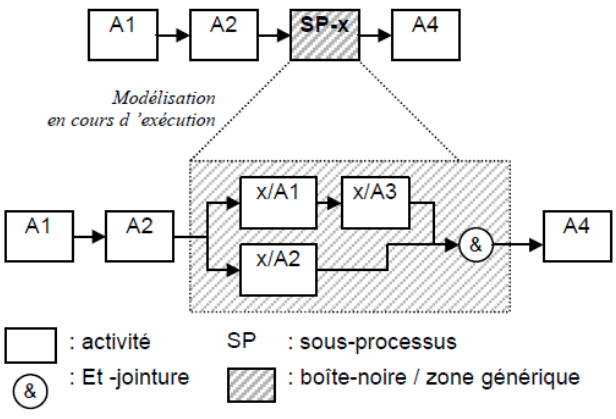



**Figure IV.5.** *La Modélisation tardive*

La modélisation dynamique est mise en œuvre notamment dans (Madhusudan et al., 2004), (Weber et al., 2004) et la modélisation ad-hoc se retrouve par exemple dans (Sadiq et al., 2005) et (Pesic et al., 2007).

### IV.6.2 Approches par méta-modèle

Cette catégorie d'approches utilisent des méta-modèles ou des méta-composants Workflows pour définir (et redéfinir) la structure, le type et les comportements des éléments constituant un modèle de WF.

(Saikali, 2001) définit un méta-modèle comme étant un modèle permettant de décrire les comportements et les relations existants entre des formalismes d'un même domaine, afin de permettre grâce à leur utilisation, la conception correcte de modèles d'objets du monde réel.

Au niveau du WF, l'approche par méta-modèle a donc pour but de permettre d'apporter des modifications sur les modèles de WF. Ainsi une approche par méta-modèles recouvre deux objectifs principaux : définir un méta-modèle WF qui sert de fondement à la construction et à la réutilisation de modèles de processus WF, et proposer un ensemble de primitives définissant des règles et des opérations permettant de manipuler les WF issus du méta-modèle.

L'approche d'adaptation par méta-modèle est par exemple, proposée par le modèle ROK ("*Reflexive Object Model*") (Edmond, 98), (Edmond, 2000). Ce modèle a initialement été conçu pour introduire de la flexibilité dans les SGBD et les systèmes coopératifs répartis et a évolué vers les systèmes de WF. ROK propose d'associer à tout objet d'une application informatique (un WF en particulier), cinq méta-objets qui correspondent chacun à une facette de l'objet. L'approche par méta-modèle est aussi utilisée dans (Saikali, 2001) et (Zhai et al., 2008) par exemple.

## IV.7 Techniques de mise en œuvre de la flexibilité des WF

En se basant sur les deux grandes classes d'approches de flexibilité de WF décrites dans la section précédente, un grand nombre de travaux de recherche se sont concentrés sur la mise en œuvre de la flexibilité des WF en utilisant différentes techniques permettant d'apporter une souplesse et une agilité aux modèles de processus WF. En fonction de la technique utilisée, nous avons identifié cinq familles d'approches support de la flexibilité des WF: les approches à base de règles, les approches à base de contraintes, les approches à base de cas, les approches à base d'aspects et les approches à base de patrons. Dans la suite, nous passons en revue un certain nombre de travaux proposés dans chacune des familles identifiées. En plus des travaux que nous avons réussi à classer dans l'une des cinq familles, nous présentons d'autres approches qui utilisent différentes autres techniques pour réaliser l'adaptation d'un WF. Nous terminons par présenter un récapitulatif des approches décrites avec une comparaison et une discussion selon certains critères liés aux aspects de flexibilité, que nous avons dégagés.

### IV.7.1 Les approches à base de règles

La modélisation à base de règles propose de modéliser la logique du processus par un ensemble de règles en utilisant des langages déclaratifs, dont l'exécution est assurée par des moteurs d'inférence de règles. L'ordre d'exécution des activités est déterminé en évaluant des conditions ou selon l'occurrence d'évènements spécifiques à surveiller ou à prédire.

Dans (Muller et al., 2004), les auteurs ont développé un système de WF baptisé AGENTWORK en combinant le paradigme Event-Condition-Action (ECA) qui est un simple formalisme utilisé pour capturer le comportement dynamique des systèmes d'information, et la

technologie des agents. Un agent représente une entité autonome qui permet de réaliser certaines actions du processus. Il s'agit d'un système qui peut adapter les parties non encore exécutées du processus WF de manière automatisée suivant deux stratégies d'adaptation appelées *adaptation réactive* et *adaptation prédictive.* Les adaptations consistent en des substitutions, des ajouts ou des annulations d'activités. L'adaptation réactive se déroule dès qu'une défaillance d'exécution est détectée pour adapter les parties de WF affectées par cette défaillance. L'adaptation réactive est réalisée lorsque l'adaptation prédictive n'est pas possible. L'adaptation prédictive permet de prédire l'occurrence d'un évènement qui affectera l'instance de processus d'une défaillance logique. Dans ce cas, l'adaptation est effectuée lorsque la partie de WF concernée doit être exécutée, notamment avant qu'une activité ne soit lancée. Cette approche repose sur le principe de la modélisation ad-hoc (ajout ou suppression d'activités en cours d'exécution), opère des adaptations au niveau instance et propose une adaptation du contrôle de flux et des données dans le processus.

Dans (Colombo et al., 2006), les auteurs présentent une plate-forme appelée SCENE, qui intègre un moteur d'exécution BPEL et des règles ECA. Ils proposent un langage de composition à travers lequel ils décrivent des compositions de services en termes de deux parties distinctes: une partie de processus, décrite en utilisant BPEL, qui définit le principal de la logique métier de la composition, et une partie déclarative décrite à l'aide des règles ECA. Ces règles sont vérifiées au niveau Runtime et sont utilisées pour réaliser des liaisons correctes entre le moteur BPEL et les services. Les règles sont spécifiées en termes d'évènements générés par les activités spécifiées dans la définition BPEL. Les événements sont liés à la logique métier, ils sont préalablement définis à l'aide de « Drools[11] » et sont contrôlés via les variables des activités. Les activités qui peuvent être adaptées sont d'abord liées à un proxy qui, en fonction de l'état du processus, fait suivre la requête au fournisseur de service correspondant. Cette approche effectue une adaptation verticale, opère au niveau instance et se base sur la technique des choix multiples.

Dans (Bastida et al., 2008), les auteurs présentent une approche pour la composition de services sensibles au contexte en utilisant une méthode en six étapes, pour définir un modèle exécutable composé de plusieurs services. Le processus final dispose d'un ensemble de variantes choisies pour plusieurs points de variation qui seront connectés à de nouveaux services, conformément à l'interprétation de certaines règles ECA (Evénement-Condition-Action). Les auteurs expriment les conditions d'adaptation en utilisant un langage propriétaire, qui associe une action de reconfiguration programmée à une propriété de l'information du contexte. Pendant l'exécution, lorsqu'un événement indiquant une condition de reconfiguration arrive, le point de variation est relié à un nouveau service. Cette approche tient compte du contexte d'exécution du processus, elle effectue des adaptations au niveau instance et repose sur la technique des choix multiples. Toutefois, elle est de type verticale selon la classification de (Papazoglou et al., 2008) puisqu'elle n'affecte pas la structure du processus.

(Kumar et al., 2009) proposent une approche à base de règles métier combinées avec des modèles génériques de processus pour la modélisation de processus WF flexibles. Au niveau « Runtime », l'instanciation du processus est effectuée en appliquant des règles à des cas spécifiques et en utilisant un algorithme de matérialisation. Les règles métier sont écrites dans un langage basé sur la logique tel que Prolog. L'objectif de cette approche est de capturer une connaissance plus approfondie des processus et parvenir à une approche holistique de la conception de processus robuste qui englobe flux de contrôle, ressources et données, et de faciliter la prise en charge des changements conformément aux changements de la politique métier. Cette

[11] http://docs.jboss.org/drools/release/5.2.0. Dernière visite le 25/08/2015

approche utilise le principe de la modélisation ad-hoc et effectue une adaptation au niveau instance, en affectant les aspects processus, ressources et données.

(Dohring et al., 2010) proposent une approche à base de règles liées au contexte (context-rules) pour appliquer dynamiquement, des opérations d'adaptation de WF au changement de contexte. Le travail a été motivé par les besoins des « Process-aware information systems » (PAIS) qui sont hautement sensibles aux changements de contexte. En effet, les approches classiques d'adaptation à base de règles s'avèrent parfois insuffisantes lorsque la modélisation de WF est basée sur un paradigme à base d'événements et lorsque la dépendance du contexte est importante. De ce fait, les WF doivent rester réactifs au contexte. Aussi, les auteurs montrent comment les règles de contexte hiérarchiques peuvent être intégrés pour adapter le WF à l'évolution des contextes de données et proposent un catalogue de patrons d'adaptation (pour des processus BPMN2) dirigée par les événements (event-driven). Un exemple de patron pourrait être la suppression dépendant du contexte, d'un segment de WF si un événement précis se produit. Cette approche est basée sur la modélisation dynamique, elle tient compte du changement de contexte et s'applique au niveau instance de processus, les adaptations s'opèrent sur des aspects processus.

Les auteurs de (Leitner et al ., 2010) proposent un framework d'adaptation appelé PREvent, un système qui intègre (i) la surveillance basée sur les événements, (ii) la prédiction des violations SLA (Service-level agreement) en utilisant des techniques d'apprentissage machine (machine-learning), et (iii) la prévention automatisée d'exécution de ces violations, par le déclenchement des mesures d'adaptation dans les compositions de services. Le framework est principalement composé de trois parties : un analyseur de composition, un SLA Predictor et un adaptateur de composition. L'analyseur de composition est responsable du suivi des données d'exécution, tandis que la prévision des violations est traitée par le SLA Predictor. Il utilise des techniques d'apprentissage pour identifier les services qui peuvent provoquer des violations de SLA à l'avenir. Enfin, l'adaptateur de composition est responsable d'identifier et d'appliquer des mesures d'adaptation. Dans cette approche, les événements à surveiller sont décrits à l'aide de Esper Query Language (EQL) et utilisent les résultats de ces définitions pour déclencher une adaptation dans le processus métier. Les auteurs déclarent que les capacités d'adaptation de leur solution sont limitées, concernant l'ajout et la suppression d'activités, car ils ne peuvent pas modifier les activités structurées et ne peuvent ajouter ou supprimer qu'un nombre limité d'activités simples. L'approche proposée repose sur la prédiction d'évènements, elle effectue une adaptation horizontale qui opère au niveau instance et est basée sur la modélisation ad-hoc.

Dans (Hu et al., 2014), une approche à base de règles est proposée dans un but de génération automatique de modèles de processus métiers flexibles, elle consiste à réaliser un assemblage de services sélectionnées dynamiquement, en tenant compte du contexte et en particulier des préférences de l'utilisateur. De plus, elle permet l'adaptation du modèle de processus généré selon les préférences de l'utilisateur. L'approche proposée est basée sur le principe de la modélisation ad-hoc, elle opère des adaptations horizontales, au niveau instance.

Notons que toutes les approches su-décrites n'abordent pas les aspects d'évolution et/ou de réutilisation des modèles. Toutefois, nous considérons que les approches orientées « composition de services » et qui supportent une adaptation horizontale (notamment l'ajout de nouveaux services) comme (Leitner et al., 2010), (Hu et al., 2014) supportent l'évolution des modèles, puisque forcément, l'ajout d'un nouveau service induit l'expansion de la coopération et des fonctionnalités du processus.

### IV.7.2 Les approches à base de contraintes

Les modèles de contraintes sont adaptés à la modélisation des processus métiers flexibles car ils permettent d'exprimer un grand nombre d'exécutions différentes en définissant un nombre limité de contraintes dans le modèle. La plupart des langages de modélisation des processus théoriques, tels que les réseaux de Petri, les algèbres de processus et les langages dédiés aux processus métiers comme BPMN et UML définissent des relations de causalité directe entre les activités du modèle de processus ; contrairement aux langages à base de contraintes qui sont de nature moins procédurale et utilisent un style plus déclaratif.

En utilisant les contraintes, la définition du comportement est limitée. Les contraintes peuvent être non locales par exemple « éventuellement, l'exécution de A est suivie par l'exécution de B » et négatives comme par exemple, « soit A soit B peuvent se produire mais pas les deux ». Les activités et les contraintes sur les activités sont les éléments clés de la modélisation à base de contraintes. Dans un modèle de WF, les contraintes définissent les limites dans lesquelles les activités peuvent être exécutées. Outre les activités et les contraintes sur ces activités, le modèle de contraintes peut inclure aussi un « mapping » qui définit si les contraintes sont optionnelles (peuvent être violées) ou obligatoires. Dans la famille des approches à base de contraintes, plusieurs travaux ont été développés, dont les suivants :

(Lu et al., 2009) proposent une approche à base de contraintes pour la prise en compte de la modélisation ad-hoc des processus métiers. Leur approche se base sur la conceptualisation d'un ensemble de contraintes pouvant être appliquées à un grand nombre d'instances au niveau Runtime. Dans le framework de modélisation développé, les besoins métiers sont exprimés sous forme d'un ensemble minimal et cohérent de contraintes. Cette approche permet une adaptation horizontale, elle est basée sur la modélisation ad-hoc et s'applique au niveau instance conformément à la vérification des contraintes ou non au cours de l'exécution des instances.

(Sadiq et al., 2005) proposent un ensemble de contraintes pour la spécification de WF flexibles. Ces contraintes ont pour but de fournir un équilibre approprié entre la flexibilité et le contrôle. Le framework de spécification des contraintes baptisé « Cameleon » est basé sur le concept de « pockets » de flexibilité qui permettent un changement ad-hoc et la construction de processus WF hautement flexibles. Il s'agit de spécifier partiellement le modèle de processus et de compléter la modélisation au niveau Runtime qui peut être unique pour chaque instance. L'approche se base sur une modélisation ad-hoc, met en œuvre une adaptation horizontale et s'applique au niveau instance.

(Pesic et al., 2007) décrivent un framework général pour la modélisation et l'implémentation de processus à base de contraintes. L'approche supporte les changements ad-hoc au niveau instance et au niveau modèle de processus. L'approche utilise le langage de modélisation déclaratif graphique *ConDec* et est basée sur un « mapping » flexible des contraintes graphiques vers des contraintes LTL (Linear Temporal Logic). L'approche permet entre autres l'ajout, la suppression ou la modification de contraintes ou d'activités, elle opère donc des adaptations au niveau contrôle de flux du processus. L'approche se base sur une modélisation ad-hoc, elle supporte les adaptations au niveau modèle de processus et au niveau instance.

Les auteurs de (Xiao et al., 2011) présentent un framework à base de contraintes pour l'adaptation dynamique des processus métiers. Leur approche utilise des fragments de processus, qui sont des compositions isolées d'activités visant à accomplir une tâche spécifique. Ces fragments de processus peuvent contenir tout type d'activités (par exemple, des invocations, des boucles, …etc.) et permettent même d'introduire des variables. Ils utilisent ces fragments pour

compléter le processus métier dans les régions prédéfinies appelées *points de variation*. L'utilisation des contraintes permet de déterminer lequel des fragments, pourrait mieux accomplir la tâche, puis l'utiliser pour composer le processus métier final. Si les contraintes changent, leur système peut alors envisager de nouveaux fragments et remplacer les anciens, mais ces modifications ne seront visibles que pour les nouvelles instances du processus. Cette approche se situe entre la modélisation tardive et les choix multiples, elle opère une adaptation horizontale qui s'applique au niveau instance. L'approche est considérée évolutive dans le sens où elle permet de définir de nouveaux fragments pour remplacer les anciens. Quant à la réutilisation, celle-ci se limite à la réutilisation de fragments au niveau des points de variation.

(Kolar et al., 2013) proposent une solution hybride regroupant les principes de gestion adaptative de cas et la modélisation à base de contraintes pour supporter la modélisation flexible de processus ad-hoc. En effet, ils proposent un patron de modélisation qui est utilisé dans des environnements de gestion de processus (systèmes de gestion de processus métiers) pour définir la structure d'un processus de manière déclarative en tenant compte des contraintes spécifiées. L'approche se base sur la modélisation ad-hoc et permet une adaptation au niveau instance en affectant les aspects fonctionnel et comportemental du processus.

Par rapport aux aspects d'évolution et de réutilisation de modèles, remarquons aussi que les approches décrites dans cette catégorie (à base de contraintes) n'abordent pas les deux aspects. Bien que dans un contexte de composition de services comme dans (Lu et al., 2009) et (Xiao et al., 2011), nous considérons que ces approches prennent en compte l'aspect évolution car elle opèrent une adaptation horizontale. Dans (Xiao et al., 2011), nous considérons un autre aspect d'évolution et un aspect de réutilisation liés respectivement, à la possibilité de définir de nouveaux fragments de processus et à la réutilisation des fragments au niveau des points de variation dans le processus.

### IV.7.3 Approches à base de cas « Case-based »

Dans le but de supporter une modélisation efficace des Workflows, certains systèmes proposent des modèles (templates) pour des processus métiers communs (assez répandus). Ces modèles sont appelés « cas (ou cases) » et peuvent être modifiés individuellement ou collectivement afin de former un nouveau WF qui doit satisfaire une certaine spécification. La principale caractéristique de ces approches est la réutilisation des modèles existants moyennant une adaptation. Dans cette perspective, un certain nombre d'approches ont été développées, nous en citons quelques unes.

(Madhusudan et al., 2004) proposent un framework appelé CODAW (Case Oriented Design Assistant for Workflow Modeling) pour supporter la modélisation de WF en adaptant des modèles de WF existants dans un répertoire de « templates ». L'approche proposée pour la gestion des modèles de WF est basée sur un cycle de vie structuré et des techniques de raisonnement à base de cas (*Case-Based Reasonning* ). Les auteurs proposent un modèle conceptuel de cas de WF (*WF cases*), un algorithme de similarité pour retrouver les modèles de WF adéquats et une approche de AI-Planning indépendante du domaine pour la composition des modèles (*WF cases*). L'approche s'oriente donc vers la modélisation par réutilisation de modèles existants et les adaptations se font au niveau modèle de processus.

(Weber et al., 2004) proposent une architecture d'un SGWF adaptatif. Le prototype qu'ils ont développé est appelé CBRFlow (*Case-Based Reasonning Flow*), il étend l'exécution d'un WF avec un raisonnement à base de cas conversationnel (*Conversational case-based reasoning* - CCBR) dans le but d'adapter le modèle de WF prédéfini et fournir un SGWF avec des capacités d'apprentissage. Les règles métier du modèle de WF prédéfini sont annotées avec des informations spécifiques liées au contexte au niveau Runtime, sous forme de cas en utilisant le

sous-système CCBR. Lorsque la réutilisation du cas devient fréquente, les cas sont manuellement remaniés en règles métier pour favoriser leur exécution automatique. Ce feedback supporte l'amélioration continue du processus (CPI) donnant lieu avec le temps à des processus métiers plus efficaces et faciles à gérer. L'approche supporte l'adaptation au niveau modèle de processus, elle utilise une modélisation tardive, en utilisant la technique de raisonnement à base de cas puisqu'elle repose sur les exécutions fréquentes de cas pour déduire des règles métier à incorporer dans le modèle.

(Minor et al., 2010) décrivent une nouvelle approche pour l'adaptation automatique de WF basée sur la réutilisation de cas d'adaptations antérieures. Ils proposent un framework général d'adaptation à base de cas (*case-based adaptation*) et une méthode pour la réutilisation de cas d'adaptations sur le WF. L'adaptation du WF lui-même est effectuée selon une approche « *case-based* » afin de favoriser la réutilisation d'expériences passées. Un cas d'adaptation stocke, dans une base dédiée, l'adaptation effectuée sous forme d'un but à réaliser (description de la requête de changement), le WF initial, le WF adapté et les éléments ayant subi un changement entre les deux versions de WF. Lorsqu'une nouvelle requête d'adaptation survient pour le WF (aux niveaux Buildtime ou Runtime), l'adaptation est traitée en se basant sur la réutilisation de cas d'adaptation précédents. Cette approche opère des adaptations au niveau modèle de processus et au niveau instance, elle favorise la réutilisation de scénarios d'adaptation et supporte une modélisation qui se situe entre les choix multiples (si le scénario est déjà passé) et la modélisation ad-hoc dans le cas contraire.

Dans le domaine du WF, les travaux de la communauté « cyber-infrastructure » s'orientent vers la collecte des traces d'exécution de WF dans le but de fournir une nouvelle source de connaissance basée sur l'expérience pour la génération de modèles de WF, il s'agit de bases de données hyper-volumineuses de traces d'exécution de WF. Le travail présenté dans (Leake et al., 2008) propose d'exploiter ces bases de données dans une perspective de réutilisation en appliquant des méthodes de CBR (« Case-Based Reasoning ») à ces traces d'exécution pour supporter la génération de modèles de WF scientifiques. Ils introduisent e-science WF comme un domaine de CBR, discutent les aspects techniques et proposent un support, pour la réutilisation de portions de WF existants afin de générer de nouveaux WF.

Concernant les approches décrites dans cette catégorie, une remarque globale est qu'elles sont toutes orientées vers la *réutilisation de modèles* de WF (ou fragments de modèles) existants pour la construction de nouveaux modèles, moyennant des adaptations à effectuer afin de satisfaire la spécification du modèle de WF à construire. Certaines autres approches comme celle décrite dans (Minor et al., 2010) s'oriente vers la réutilisation de scénarios d'adaptations enregistrées pour appliquer des adaptations futures sur le modèle de processus.

### IV.7.4 Approches orientées aspects

La programmation par aspects permet la séparation des mécanismes fonctionnels et non-fonctionnels d'un système. Dans le domaine du WF, ceci permet de faciliter la modélisation des processus et leur adaptation.

Dans (Sánchez et Villalobos, 2008), les auteurs utilisent une approche orientée aspect pour la modélisation et la mise en oeuvre de processus WF, en mettant l'accent sur la séparation entre les préoccupations et l'instrumentation. Ils introduisent les modèles exécutables, qui sont utilisés pour représenter les préoccupations transversales. Ils utilisent des objets ouverts, qui sont des représentations de l'état des éléments dans le modèle, pour surveiller l'invocation de services et adapter le processus de tissage d'interaction avec d'autres modèles avant (activation) et après

(désactivation) l'appel du service. Leur objectif est de créer des applications de WF comme des modèles exécutables qui peuvent ensuite être synchronisés à l'aide des appels de méthodes et l'occurrence d'événements. Cette approche met en œuvre une adaptation horizontale car elle touche à la structure d'interaction entre les services de la composition et se base sur la modélisation ad-hoc, elle opère au niveau instance.

Une extension de BPEL, en utilisant les aspects, est introduite dans (Charfi et al., 2009). Les auteurs présentent une architecture basée plug-in pour les processus auto-adaptatifs qui utilisent AO4BPEL (″Aspect-oriented extension for BPEL″). Leur proposition est d'avoir différents plug-in avec un objectif bien défini. Chaque plug-in dispose de deux types d'aspects: les aspects de surveillance qui observent les besoins d'adaptation du système et les aspects d'adaptation qui gèrent les situations détectées. A chaque fois que les conditions d'un aspect de surveillance sont remplies, le système utilise AO4BPEL pour tisser les aspects d'adaptation dans le processus au moment de l'exécution. Les aspects de surveillance peuvent être déployés à chaud dans le moteur BPEL, leur permettant d'ajouter ou de modifier les conditions d'adaptation au niveau Runtime. Cette approche opère au niveau instance, elle met en œuvre une adaptation horizontale car elle touche à la structure d'interaction entre les services de la composition et se situe entre l'approche des choix multiples et la modélisation tardive.

Dans (Geebelen et al., 2010), les auteurs présentent un framework d'adaptation basé sur le patron MVC (Model-View-Controller) pour l'application réelle des politiques d'adaptation dynamique des processus métiers. Pour améliorer la réutilisabilité, l'approche proposée se base d'une part, sur la séparation entre la logique d'adaptation et l'aspect fonctionnel du processus et d'autre part, sur la modularisation des tâches du WF dans des aspects réutilisables. Dans leur approche, l'idée de base est de concevoir un processus WF comme un modèle où les tâches sont spécifiées de manière abstraite. Ces tâches sont modélisées comme des *aspects* et leurs implémentations sont stockées dans une bibliothèque où elles sont choisies, selon les politiques de la logique d'adaptation. Cette logique est implémentée à l'aide d'un langage général orienté but applicable à tout type de politique. La bibliothèque contient des aspects de différentes activités qui peuvent être modularisées pour remplir une tâche spécifique. Les politiques d'adaptation dépendent des propriétés ou des valeurs de paramètres du processus en exécution. Une adaptation peut être annulée en restaurant l'état précédent du processus. L'approche utilise la notion d'activité abstraite, elle se situe entre la modélisation tardive et les choix multiples, elle supporte une adaptation horizontale, s'applique au niveau instance et permet la réutilisation d'aspects pour la conception d'autres activités.

Une solution orientée aspect utilisant le framework Spring .NET est présenté dans (Rahman et al., 2008). Les auteurs utilisent une approche à base de contrats pour relier un service Web à chaque invocation au niveau exécution. En utilisant les règles ECA, ils créent un contrat entre les partenaires participant à la composition. Pour réaliser l'adaptation du processus, ils peuvent modifier le contrat des règles ECA au niveau Runtime et attribuer un nouveau service Web pour l'appel. Ils peuvent également adapter une implémentation existante d'un service Web en utilisant les aspects pour tisser le nouveau comportement. Cette technique se base sur les règles ECA et la programmation par aspects, elle s'applique à une adaptation verticale et utilise la technique des choix multiples avec la possibilité de modifier les choix, qui selon les auteurs, constitue un aspect d'évolution de leur approche.

(Boukadi, 2009) propose un framework pour la coopération dynamique dans un contexte de composition de services, en incluant des aspects d'adaptation du processus métier et d'évolution des fonctionnalités des services. L'auteur utilise une approche à *base d'aspects* et tient compte du

changement du contexte d'exécution dépendant d'événements. L'auteur introduit le concept de « service domaine » qui doit s'adapter au changement de contexte à l'aide de mécanismes appropriés. Le framework proposé supporte l'adaptation adaptative et l'adaptation évolutive liée à l'expansion des fonctionnalités de services. Il permet la sélection dynamique des aspects en fonction du changement de contexte et l'activation dynamique des aspects appropriés au moment de l'exécution. L'approche supporte une adaptation horizontale car elle touche à la logique d'orchestration du processus métier, opère au niveau instance et est basée sur une modélisation ad-hoc dépendant du contexte.

Comme pour les approches précédentes, la catégorie « à base d'aspects » englobe des techniques supportant les adaptations horizontales (Sánchez et Villalobos, 2008), (Charfi et al., 2009), (Geebelen et al., 2010), (Boukadi, 2009) dans un contexte de composition de services qui est considéré comme un aspect d'évolution dans les modèles de processus. De plus, dans (Boukadi, 2009), on peut envisager une expansion des fonctionnalités du service en fonction du contexte. Quant à la réutilisabilité, certaines approches comme (Geebelen et al., 2010) supportent la réutilisation des aspects existants pour la conception de nouvelles activités.

### IV.7.5 Approches à base de patrons

Les patrons ont été initialement utilisés par Alexander et al. (Alexander et al., 77) pour décrire des solutions à des problèmes récurrents et les meilleures pratiques dans la conception d'architectures logicielles. Gamma et al. (Gamma et al., 95) ont appliqué les mêmes concepts pour le génie logiciel et ont décrit 23 patrons de conception.

Dans le domaine du WF, les patrons ont été introduits pour analyser l'expressivité des méta-modèles de processus (Van der Aalst, 2003), (Russel et al., 2006a). Dans ce contexte, les patrons de contrôle de flux décrivent différents constructeurs pour spécifier les activités et leur ordre d'exécution. Par ailleurs, des patrons pour décrire les interactions de services dans une chorégraphie ont été introduits dans (Barros et al., 2005). L'introduction des patrons de WF a eu un impact considérable sur la modélisation des processus métiers (Ait-Cheikh-Bihi et al., 2012) ainsi que sur l'évaluation de l'expressivité des langages de processus tels que BPEL, BPMN et UML. Beaucoup de travaux se sont penchés sur le développement d'approches support de la flexibilité du WF, en se basant sur les patrons de WF.

(Kim et al., 2007) décrivent une approche basée sur les patrons de changement (change patterns) pour le support du changement dans un environnment dynamique. En se basant sur les patrons de WF, ils identifient et catégorisent les patrons de changements qui peuvent survenir fréquemment sur les processus métiers. Leur approche tient compte du versionnement de processus. Afin de maximiser la flexibilté dans l'exécution de différentes versions du processus, les auteurs proposent un mécanisme pour l'exécution d'un processus abstrait avec une encapsulation du processus métier en cours d'exécution. De plus, ils proposent un mécanisme pour une sélection dynamique des versions de processus. Cette approche opère une adaptation horizontale, peut s'appliquer au niveau modèle et instance (gestion des exceptions). Elle repose sur le mécanisme de versionning des processus qui constitue un aspect d'évolution du modèle de processus.

(He et al., 2008) traitent le problème de l'adaptatbilité pour une composition de services dans un environnement dynamique et instable où des changements dans le processus doivent être opérés en cas de dégradation de la qualité de service. Les auteurs proposent une approche d'adaptation basée sur les patrons de WF pour une composition dynamique de services, l'approche prend en compte la cohérence de la logique métier du processus après une adaptation (c'est-à-dire cohérence

du contrôle de flux, des interactions entre les services et des données dans le processus). Cette approche opère une adaptation horizontale, au niveau instance, elle est basée sur une technique de choix multiples à partir d'un ensemble de services disponibles sur Internet.

Dans (Weber et al., 2008), les auteurs décrivent 18 patrons d'adaptation et 7 fonctionnalités support au changement dans les PAIS (Process-aware information systems). Ces patrons sont proposés pour supporter les changements dans des régions prédéfinies du processus sur les aspects fonctionnel et comportemental. Ils englobent des patrons de sélection tardive de fragments et des patrons de modélisation tardive. Dans ce travail, les processus sont modélisés à l'aide de BPMN et une étude comparative concernant le support des patrons et des fonctionnalités est fournie pour un ensemble d'outils académiques et commerciaux. L'approche proposée supporte une adaptation horizontale, elle se base sur plusieurs techniques selon les cas : choix multiples, modélisation ad-hoc ou modélisation tardive. Elle permet la réutilisation des scénarios de changement grâce à une fonctionnalité appropriée (« Change reuse feature »).

(Kapuruge et al., 2013) proposent une approche de modélisation orientée événements pour le support des patrons d'adaptation proposés dans (Weber et al., 2008) en vue de garantir une flexibilité des modèles de processus. L'approche proposée est basée sur la modélisation ad-hoc dépendant d'évènements pouvant survenir en cours d'exécution des instances de processus, elle supporte une adaptation horizontale et permet la réutilisation de scénarios de changement.

(Dohring et al., 2011) proposent une extension du méta-modèle BPMN2 en introduisant un marquage pour l'adaptation de segments de workflow dépendant du contexte. La principale contribution de ce travail est l'introduction de patrons de changement, de temps et de gestion d'exception dans un catalogue de modèles d'adaptation BPMN2 pouvant être directement utilisés dans l'approche. Celle-ci est personnalisable pour limiter le degré de flexibilité au changement requis, par exemple, en termes de l'ensemble des patrons d'adaptation qui sont autorisés selon les règles spécifiées dans le workflow. Un avantage de l'approche est qu'elle est intuitive pour un concepteur qui est déjà familiarisé avec la notation BPMN, le deuxième avantage c'est la facilité de modification et d'extension des patrons proposés, en plus de la garantie de cohérence du modèle après adaptation. Cette approche opère une adaptation horizontale, peut s'appliquer au niveau modèle et instance. Elle prend en compte les informations relatives au contexte et combine les patrons et les règles pour effectuer l'adaptation. Un aspect d'évolution de l'approche est la possibilité d'extension des patrons proposés.

Le travail de (Russel et al., 2006b) présente un framework de classification à base de patrons pour caractériser la gestion des exceptions dans les systèmes de WF, notamment les PAIS (Process- aware information systems). Le framework a été utilisé pour examiner les capacités de support des exceptions d'un certain nombre de systèmes de WF, de langages de modélisation et d'exécution de processus métier. Sur la base des résultats recueillis, les auteurs proposent leur propre langage indépendant de toute technologie pour définir différentes stratégies de gestion d'exceptions dans les workflows. Ce langage offre la possibilité d'assister graphiquement un concepteur dans la définition et la gestion des situations d'exception grâce à des patrons d'exceptions.

Notons que les approches d'adaptation à base de patrons se sont d'abord penchées sur l'identification de situations récurrentes de changement dans les workflows et ont proposé des catégories de patrons liés notamment, aux aspects contrôle de flux et gestion d'exceptions dans le processus. La plupart d'entre elles ont été proposées dans un contexte de composition de services et supportent une adaptation horizontale, ce qui leur donne une dimension d'évolution du modèle. Un aspect d'évolution spécifique a été considéré dans (Kim et al., 2007) lié au versionnement du

processus et un autre aspect relevé dans (Dohring et al., 2011) lié à la possibilité d'extension des patrons. Quant à la réutilisation, les approches proposées se limitent à la réutilisation de services (notamment des services web) disponibles sur Internet et ne vont pas jusqu'à la réutilisation de modèles de processus eux-mêmes. Par ailleurs, l'approche présentée dans (Weber et al., 2008) permet la réutilisation de scénarios d'adaptation grâce à une fonctionnalité fournie à cet effet.

### IV.7.6 Autres approches

(Saikali, 2001) propose un framework pour WF flexibles, baptisé 2FLOW, qui est basé sur un méta-modèle objet et met en oeuvre des mécanismes objets Workflows qui, combinés avec des mécanismes réflexifs, permettent d'adapter les workflows hors et en cours d'exécution. Il propose des solutions pour la construction de WF flexibles, ainsi qu'une méthode pour la conception de WF à l'aide des composants du framework. Son approche se distingue par la mise en oeuvre des mécanismes Workflows Objets (spécialisation) et aussi par l'introduction d'une indépendance entre les acteurs et les activités grâce à une approche innovante du concept de rôle. 2FLOW permet la prise en charge des trois niveaux d'adaptation : local à l'instance en cours, local généralisé qui affecte l'ensemble des instances en cours d'exécution, et global qui se répercute en plus sur la définition du modèle de WF.

(Casati et al., 2001) décrivent leur système e-Flow permettant la spécification de processus métiers comme une composition de services web auto-configurables au niveau Runtime, conformément à la disponibilité des services au moment de l'exécution du processus et aux besoins des clients. Les auteurs proposent des fonctionnalités d'adaptation dynamique permettant d'agir au niveau instance et au niveau modèle de processus sur différents aspects du processus. E-Flow propose des règles de vérification de la consistance des modèles après chaque adaptation ainsi que des règles d'autorisation pour garantir que les changements soient opérés uniquement par les utilisateurs autorisés. Un processus dans e-Flow est représenté par un graphe avec différents types de nœuds dont les nœuds de type « event » qui permettent de capturer les événements en cours d'exécution du processus. Dans un processus e-Flow, on peut concevoir des nœuds génériques devant être adaptés en cours d'exécution. L'approche supporte une adaptation horizontale, au niveau instance et modèle, elle se base sur la modélisation ad-hoc et les choix multiples.

(Zhai et al., 2008) présentent un framework réflexif pour supporter l'adaptabilité d'une composition de services Web spécifiée à l'aide du langage BPEL. Les auteurs définissent un méta-modèle pour construire une self-représentation d'une composition de services Web. Le méta-modèle peut être modifié pour s'adapter au changement de l'environnement. Ainsi, un mécanisme de réflexion implémenté dans le framework est utilisé pour ajuster automatiquement la composition de services Web. Le framework permet d'ajouter ou supprimer des activités à partir du processus original dans le méta-modèle et de translater les changements sur le processus en cours d'exécution. Cette approche repose sur une technique d'adaptation par méta-modèle, elle opère des changements au niveau instance et modèle de processus.

(Chaari, 2008) propose de construire un processus collaboratif comme une composition dynamique de services. D'abord, il spécifie manuellement son processus collaboratif tel un processus *abstrait* ensuite en tenant compte des paramètres non fonctionnels des services disponibles, il effectue une sélection des services adéquats afin de construire le processus. Cette approche opère une adaptation verticale qui consiste en une substitution d'une activité abstraite par un service sélectionné, elle se base sur la technique des choix multiples, ne présente aucun aspect d'évolution ou de réutilisation de modèles de processus.

Dans (Canfora et al., 2008), les auteurs présentent un framework pour la liaison et la reconsolidation des services Web composites. Ils intègrent la surveillance de la qualité de service des processus métiers, et le temps d'utilisation, basés essentiellement sur les facteurs de prix, de disponibilité et de fiabilité (ainsi que des attributs personnalisés) comme un moyen pour mesurer la qualité de service. Le processus métier est d'abord créé comme un WF abstrait, sans aucune liaison aux services. Une fois le WF défini, les services sont liés à un premier ensemble de fournisseurs pris dans un pool. Si l'accord de niveau service (Service Level Agreement -SLA) est violé lors de l'exécution du processus, ou la qualité de service se dégrade indiquant une forte probabilité qu'une rupture de service se produise, le système adapte le processus et relie les services aux nouveaux fournisseurs du pool qui respectent la qualité de service. Cette approche propose une adaptation verticale, s'applique au niveau instance et repose sur la technique des choix multiples (sélection de services dans un pool).

Une approche pour adapter les processus BPEL est présentée dans (Strunk et al., 2009). Les auteurs de ce travail offrent à l'utilisateur une interface pour définir des services alternatifs au moment du design, qui seront utilisés pour adapter le processus. Leur solution comporte trois composants : une couche proxy, un composant de surveillance et un composant de reliaison (reconsolidation). Le proxy reçoit les requêtes à partir du moteur BPEL et lui répond, ce qui permet une souplesse pour le changement du fournisseur de service si nécessaire. Le composant de surveillance surveille les événements et déclenche le composant de reconsolidation si une violation SLA est détectée. Le composant de reconsolidation remplace le service défaillant par un service équivalent pris du pool de services prédéfinis au niveau Build-time. Cette approche propose une adaptation verticale, s'applique au niveau instance et se base sur la technique des choix multiples.

(Esper, 2010) propose une approche de collaboration et d'interconnexion de processus interentreprises, qui s'appuie sur une collaboration à base de services multi-contextuels. Les services métier qui sont partagés par les partenaires sont modélisés selon trois vues différentes et complémentaires à savoir la vue de gestion, la vue de médiation et la vue de sécurité. L'approche se base sur un hypergraphe de collaboration comportant des services abstraits qui doivent être instanciés au niveau Runtime. Elle permet la réalisation de plusieurs scénarios de collaboration interentreprises en se basant sur les concepts clés tels que le service, l'hypergraphe, les relations d'interdépendance entre les différentes vues. Ces concepts visent à enrichir la composition de services et assurer une exécution contextuelle. L'approche supporte une adaptation verticale, au niveau instance et est basée sur une modélisation ad-hoc dépendant du contexte d'exécution. Les aspects d'évolution et de réutilisation de modèles n'ont pas été abordés dans ce travail.

Une autre approche qui repose sur la combinaison des règles et des contraintes est décrite dans (Lam, 2015). L'auteur présente un framework pour la représentation de règles métier et la détection de violation de règles durant l'exécution de processus métiers spécifiés par des modèles déclaratifs. Une règle métier est exprimée en langage naturel anglais restreint rattaché à un modèle de contraintes proposé qui est exprimé en utilisant la logique linéaire (LTL). Une violation de règle métier est détectée de manière automatique en utilisant l'outil LWB de preuve de théorèmes (Logic WorkBench).

Enfin, il ne s'agit pas d'énumérer toutes les approches de manière exhaustive, il existe encore d'autres approches telles que (Lins et al., 2007), (Ardissono, 2007), (Modafferi et al, 2005), (Lee et al., 2007), basés sur le concept de *WF abstrait* ou d'*activité abstraite* permettant une adaptation par modélisation tardive (ou ad-hoc) du WF ou encore en utilisant un formalisme de la logique (Mosser et al., 2013).

Dans la suite, nous présentons un récapitulatif des approches que nous avons décrites en les comparant selon un certain nombre de critères que nous avons établis.

### IV.7.7 Récapitulatif des approches de flexibilité des WF

Les tableaux IV.1, IV.2, IV.3 et IV.4 présentent un résumé des approches de flexibilité du WF, précédemment décrites. Les critères que nous avons pu dégager et que nous avons jugé intéressants pour notre synthèse sont :

(a) *La technique* sur laquelle repose l'approche de flexibilité proposée. En effet, nous remarquons que tous les travaux décrits reposent sur l'une des techniques présentées dans la section IV.6 à savoir les approches ponctuelles (choix multiples, modélisation tardive et modélisation ad-hoc) et les approches par méta-modèles. Nous remarquons aussi qu'un certain nombre d'approches combinent deux techniques à la fois, notamment la technique des choix multiples et la modélisation tardive ou ad-hoc.

(b) *L'objet de l'adaptation* que nous avons spécifié en deux critères : adaptation du contrôle de flux (représentant l'aspect comportemental du processus), adaptation des activités (représentant l'aspect fonctionnel). Nous remarquons que la grande majorité des approches proposées se sont focalisées sur des adaptations au niveau processus (contrôle de flux et activités) bien que dans tous les cas une vérification de la cohérence des autres aspects (notamment les données manipulées par les activités) est nécessaire. Peu de travaux comme (Muller, 2004), (Kumar et al., 2009) et (Barba et al., 2013) se sont explicitement intéressés aux aspects données et ressources en plus de l'aspect processus. Toutefois, il faut noter que des travaux comme (Jin et al., 2013), (Wang et al., 2009) se sont focalisés spécifiquement sur les adaptations aux niveaux données et ressources, notamment dans le cas de WF scientifiques. Enfin, notre objectif est d'examiner les travaux qui se sont penchés sur l'aspect processus auquel nous nous intéressons dans le cadre de cette thèse.

Notons que l'*adaptation des interactions* est un aspect auquel nous nous intéressons dans le cadre de cette thèse, néanmoins il n'a pas été mis en évidence dans les tableaux récapitulatifs puisque nous considérons que dans un contexte de composition de services, tout ajout, suppression ou modification d'un service ou une restructuration du contrôle de flux induit de suite, une mise à jour des interactions entre l'orchestrateur de services et les services orchestrés ou entre les services eux-mêmes dans le cas d'une chorégraphie de services. On peut dire qu'une adaptation des interactions est induite par une adaptation au niveau fonctionnel ou comportemental. Dans notre cas, nous avons considéré qu'il est nécessaire de développer l'aspect « adaptation des interactions » car les interactions dans le processus, se distinguent d'un patron de coopération à l'autre et donnent lieu à des patrons d'adaptation spécifiques conformément au patron de coopération auquel obéit le modèle de processus WFIO (voir Chapitre VI).

(c) *Evolution :* ce critère permet de décrire les aspects d'évolution des modèles de processus. Nous remarquons que toutes les approches orientées composition de services et qui supportent une adaptation horizontale comme (Casati et al., 2001), (Pesic et al., 2007), (Boukadi, 2009), (Charfi et al., 2009), (Geebelen et al., 2010) présentent un aspect d'évolution qui est l'ajout de services (d'autres partenaires), ce qui induit une expansion de la coopération (et donc des fonctionnalités) dans le sens de notre définition de l'évolution des modèles. La principale différence avec notre perception de l'évolution des modèles c'est que dans notre cas, nous nous intéressons à l'évolution de modèles de processus en ajoutant des briques qui ne sont pas toujours appréhendées comme des « boites noires » mais parfois comme des « boites grises » où il faut opérer des adaptations sur les parties visibles du modèle de processus, contrairement aux approches proposées où le service à ajouter est une brique perçue comme une « boite noire » à intégrer dans une composition de

services. Par ailleurs, certaines approches telles que (Rahman et al., 2008), (Xiao et al., 2011), (Kim et al., 2007) et (Dohring et al., 2011) présentent des aspects d'évolution spécifiques, respectivement, la possibilité de définir de nouveaux choix, la possibilité de définir de nouveaux fragments de modèles, le versionnement de modèles de processus et la possibilité d'extension des patrons d'adaptation.

(d) *Réutilisation :* ce critère permet de décrire les aspects de réutilisation des modèles de processus. Nous remarquons que toutes les approches d'adaptation à base de cas « Case-based » favorisent la réutilisation de modèles ou de fragments de modèles existants pour la génération d'autres modèles de WF, moyennant des adaptations. En effet, ce sont des approches orientées modélisation par réutilisation. D'autres aspects de la réutilisation sont relevés dans (Geebelen et al., 2010), (Saikali, 2001) et (Weber et al., 2008) qui consistent respectivement, en la réutilisation d' « aspects » pour la conception de nouvelles activités, la réutilisation d'« objets » dans le génération de nouveaux modèles de processus et la réutilisation des scénarios de changement pour la prise en charge d'adaptations semblables.

(e) *Prise en compte du contexte* : ce critère décrit le fait que l'adaptation se fasse en fonction de certaines informations liées au contexte d'exécution. Ainsi, un changement de contexte déclenche automatiquement l'adaptation appropriée selon les valeurs de certaines variables décrivant le contexte d'exécution. Notons qu'un certain nombre d'approches relativement récentes telles que (Boukadi, 2009), (Esper , 2010), (Pesic et al., 2007), (Bastida et al., 2008) et (Dohring et al., 2010) se sont penchées sur la modélisation du contexte afin de l'introduire comme un paramètre agissant directement sur l'application des adaptations sur un modèle de processus.

(f) *Profondeur du changement :* ce critère permet de distinguer les approches qui opèrent des changements locaux (au niveau instance) de celles qui opèrent des changements globaux (au niveau modèle de processus). Nous remarquons que toutes les approches décrites opèrent des adaptations au niveau instance et peu d'entre elles comme (Minor et al., 2010), (Saikali, 2001), (Casati et al., 2001) supportent des adaptations au niveau modèle de processus. Le fait que toutes les approches opèrent des adaptations au niveau instance est du principalement, à la nature dynamique et instable de l'environnement d'exécution. Ce dernier subit des changements fréquents d'une exécution à l'autre du processus, notamment dans un contexte de composition de services où l'on cherche à maximiser les paramètres de QoS qui sont eux très sensibles à l'environnement d'exécution.

**TAB IV.1.** *Récapitulatif des approches de flexibilité à « base de règles » et à « base de contraintes »*

| **Référence de l'approche** | **Approche globale utilisée** | **Adaptation** | | **Evolution** | **Réutilisation** | **Prise en compte du contexte** | **Profondeur du changement** |
|---|---|---|---|---|---|---|---|
| | | **Adaptation du flux de contrôle** | **Adaptation d'activités** | | | | |
| **Approches à base de Règles** | | | | | | | |
| *(Muller, 2004)* | Modélisation ad-hoc | + | + | - | - | - | Instance |
| *(Colombo et al., 2006)* | Choix multiples | - | + | - | - | - | Instance |
| *(Bastida et al., 2008)* | Choix multiples | - | + | | - | + | Instance |
| *(Kumar et al., 2009)* | Modélisation ad-hoc | + | + | | - | - | Instance |
| *(Dohring et al., 2010)* | Modélisation dynamique | + | + | - | - | + | Instance |
| *(Leitner et al., 2010)* | Modélisation ad-hoc | + | + | Ajout de services | - | - | Instance |
| *(Hu et al., 2014)* | Modélisation ad-hoc | + | + | Ajout de services | - | + | Instance |
| **Approches à base de Contraintes** | | | | | | | |
| *(Sadiq et al., 2005)* | Modélisation ad-hoc | + | + | - | - | - | Instance |
| *(Lu et al., 2009)* | Modélisation ad-hoc | + | + | Ajout de services | - | - | Instance |
| *(Pesic et al., 2007)* | Modélisation ad-hoc | + | + | Ajout de services | - | + | Modèle et Instance |
| *(Xio et al., 2011)* | Modélisation tardive et choix multiples | + | + | Définition de nouveaux fragments de modèles. | Réutilisation de fragments de modèles existants | - | Instance |
| (Kolar et al., 2013) | Modélisation ad-hoc | + | + | - | Réutilisation de patron de modélisation | - | Instance |

**TAB IV.2.** *Récapitulatif des approches de flexibilité à « base de cas» et à « base d'aspects »*

| **Référence de l'approche** | **Technique utilisée** | **Adaptation** | | **Evolution** | **Réutilisation** | **Prise en compte du contexte** | **Profondeur du changement** |
|---|---|---|---|---|---|---|---|
| | | **Adaptation du flux de contrôle** | **Adaptation d'activités** | | | | |
| **Approches à base de Cas** | | | | | | | |
| *(Madhusudan et al., 2004)* | Modélisation tardive | + | + | - | Réutilisation de modèles existants | - | Modèle |
| *(Weber et al., 2004)* | Modélisation tardive | + | + | - | Réutilisation de modèles existants | - | Modèle |
| *(Minor et al., 2010)* | Modélisation ad-hoc et choix multiples | + | + | - | Réutilisation de scénarios d'adaptation | - | Modèle et Instance |
| *(Leake et al., 2008)* | | + | + | - | Réutilisation de fragments de WF | - | Modèle |
| **Approches Orientées Aspects** | | | | | | | |
| *(Sánchez et Villalobos, 2008)* | Modélisation ad-hoc | + | + | Ajout de services | - | - | Instance |
| *(Rahaman et al., 2008)* | Choix multiples | - | + | Définition de nouveaux choix | - | - | Instance |
| *(Charfi et al., 2009)* | Modélisation tardive et choix multiples | + | + | Ajout de services | - | - | Instance |
| *(Boukadi, 2009)* | Modélisation ad-hoc | + | + | Ajout de services | | + | Instance |
| *(Geebelen et al., 2010)* | Modélisation tardive et choix multiples | + | + | Ajout de services | Réutilisation d'aspects existants | - | |

**TAB IV.3.** *Récapitulatif des approches de flexibilité à « base de patrons»*

| **Référence de l'approche** | **Technique utilisée** | **Adaptation** | | **Evolution** | **Réutilisation** | **Prise en compte du contexte** | **Profondeur du changement** |
|---|---|---|---|---|---|---|---|
| | | **Adaptation du flux de contrôle** | **Adaptation d'activités** | | | | |
| **Approches à base de Patrons** | | | | | | | |
| *(Kim et al., 2007)* | Choix multiples | + | + | Versionnement des modèles | - | - | Instance et Modèle |
| *(He et al., 2008)* | Choix multiples | + | + | Ajout de services | - | - | Instance et Modèle |
| *(Weber et al., 2008)* | Choix multiples<br>Modélisation tardive<br>Modélisation ad-hoc | + | + | - | Réutilisation des scénarios de changement | - | Instance et Modèle |
| *(Dohring et al., 2011)* | Modélisation ad-hoc | + | + | Extension des patrons | - | + | Instance et Modèle |
| (Kapuruge et al., 2013) | Modélisation ad-hoc | + | + | - | Réutilisation de scénarios de changement | - | Instance et Modèle |

**TAB IV.4.** *Récapitulatif des autres approches de flexibilité*

| **Référence de l'approche** | **Technique utilisée** | **Adaptation** | | **Evolution** | **Réutilisation** | **Prise en compte du contexte** | **Profondeur du changement** |
|---|---|---|---|---|---|---|---|
| | | **Adaptation du flux de contrôle** | **Adaptation d'activités** | | | | |
| *(Saikali, 2001)* | Par Méta-modèle | + | + | - | Réutilisation d'objets | - | Instance et Modèle |
| *(Casati et al., 2001)* | Choix multiples et Modélisation ad-hoc | + | + | Ajout de services | - | - | Instance et Modèle |
| *(Zhai et al., 2008)* | Par Méta-modèle | + | + | Ajout de services | - | - | Instance et Modèle |
| *(Chaari, 2008)* | Choix multiples | - | + | - | - | - | Instance |
| *(Canfora et al., 2008)* | Choix multiples | - | + | - | - | - | Instance |
| *(Strunk et al., 2009)* | Choix multiples | - | + | - | - | - | Instance |
| *(Esper, 2010)* | Modélisation ad-hoc | - | + | - | - | + | Instance |

## Synthèse

Dans ce chapitre, nous nous sommes penchés sur la question de flexibilité des workflows, un problème qui a suscité l'intérêt d'un bon nombre de chercheurs dans le domaine du WF, depuis déjà presque deux décennies. Ce problème a pris encore plus d'ampleur avec l'avènement des SOA et la publication des services sur Internet, en témoignent la panoplie de travaux qui ont été proposés dans le domaine pour répondre notamment à la question d'adaptabilité des processus métiers.

Nous avons rappelé que la flexibilité d'un WF se distingue selon deux axes principaux : la flexibilité des systèmes WF et la flexibilité des modèles de processus WF qui constitue notre principale problématique dans le cadre de cette thèse. La flexibilité d'un modèle de WF se caractérise par le niveau d'application du changement et la manière d'appliquer le changement, ceci étant repris dans les différentes taxonomies de la flexibilité identifiées respectivement dans (Regev et al., 2006), (Schonenberg et al., 2007), (Han et al., 98) et (Papazoglou et al., 2008). La taxonomie présentée dans (Han et al., 89) propose de distinguer les adaptations au niveau du contexte, au niveau de l'infrastructure, au niveau du processus et au niveau des ressources. Par ailleurs, avec l'avènement des SOA, un processus métier est souvent considéré comme une composition de services. Dans cette perspective, la classification proposée par (Papazoglou et al., 2008) se concentre sur le niveau processus et distingue deux types d'adaptation : les adaptations horizontales affectant la structure du processus métier et les adaptations verticales sans impact sur la structure du processus.

Concernant le niveau d'application des changements dans un processus WF, la majeure partie des travaux se sont concentrés sur les aspects fonctionnel et comportemental (i.e l'aspect processus), ce qui est le cas pour nous, en plus de l'aspect interactionnel que nous considérons puisque nous nous plaçons dans un contexte de WF inter-organisationnel. Il faut noter que des travaux existent aussi pour l'adaptation au niveau des ressources et des données comme (Jin et al., 2013), (Wang et al., 2009).

Pour définir la flexibilité des modèles de WFIO, nous avons dégagé trois aspects complémentaires de la flexibilité qui sont l'adaptabilité, l'évolutivité et la réutilisabilité et nous avons remarqué que dans la littérature les travaux confondent souvent l'adaptabilité avec l'évolutivité (dans le sens où un modèle qui subit des modifications est un modèle qui évolue). Dans notre cas, nous proposons une autre définition de l'évolution d'un modèle que se reflète sur deux perspectives principales: l'expansion des fonctionnalités du processus et l'expansion de la coopération.

Nous avons rappelé les deux grandes classes d'approches de flexibilité existantes dans le domaine du WF qui sont les approches ponctuelles et les approches par méta-modèle et nous avons passé en revue un bon nombre de travaux portant sur la flexibilité des WF. Selon les mécanismes utilisés pour la mise en œuvre de la flexibilité, nous avons catégorisé ces travaux dans cinq familles : les « approches à base de règles », les « approches à base de contraintes », les «approches à base de cas », les « approches à base d'aspects » et « les approches à base de patrons », d'autres approches qui utilisent des techniques spécifiques ont été aussi présentées.

Nous avons effectué une analyse de ces approches selon un certain nombre de critères concernant notamment la technique utilisée, l'objet de l'adaptation, la prise en compte du contexte d'exécution, la profondeur du changement et les aspects d'évolution et de réutilisation des modèles. Nous avons déduit que chacune des approches proposées pouvait s'inscrire dans une des catégories : approches ponctuelles ou approches par méta-modèle, souvent dans les approches

ponctuelles en combinant parfois deux techniques différentes. Quant à la profondeur de l'adaptation, toutes les approches opèrent une adaptation au niveau instance que nous avons justifié par la nature dynamique et instable de l'environnement d'exécution, notamment dans un contexte de composition de services où l'on cherche à maximiser les paramètres de QoS. Toutefois, certains travaux tels que (Minor et al., 2010), (Saikali, 2001), (Casati et al., 2001) se sont penchés sur les adaptations au niveau du modèle de processus.

Par rapport aux aspects d'évolution, nous avons remarqué que toutes les approches orientées composition de services et qui supportent une adaptation horizontale comme (Casati et al., 2001), (Boukadi, 2009), (Charfi et al., 2009) présentent un aspect d'évolution qui est l'ajout de services (fournis par d'autres partenaires), ce qui induit une expansion de la coopération (et donc des fonctionnalités) dans le sens de notre définition de l'évolution des modèles. D'autres approches supportent des aspects d'évolution spécifiques comme le versionnement des modèles (Kim et al., 2007), la possibilité de définir de nouveaux fragments (Xiao et al., 2011), la possibilité d'extension des patrons (Dohring et al., 2011),…etc. Dans notre cas, nous réalisons l'évolution des modèles de processus en proposant des patrons d'évolution permettant une expansion des fonctionnalités et/ou une expansion de la coopération, et ce en apportant des modifications (notamment des ajouts) à des modèles de processus qui sont perçus comme des « boites grises » où certains points du processus doivent rester visibles. Par exemple, dans une architecture de type « transfert de cas », une expansion de la coopération nécessite une mise à jour de la politique de transfert et donc une modification au niveau des points d'interaction (points de transfert) entre les processus pris deux à deux.

Quant aux aspects de réutilisation de modèles, nous avons remarqué que les approches d'adaptation à base de cas « Case-based » favorisent la réutilisation de modèles ou de fragments de modèles existants pour la génération d'autres modèles. Dans notre cas, nous proposons des patrons de réutilisation de modèles qui permettent d'intégrer un modèle de WFIO existant dans un autre modèle de WFIO obéissant soit au même patron de coopération ou à des patrons de coopération différents, afin de construire des modèles complexes par réutilisation. L'intérêt est de pouvoir créer une coopération entre *groupes de partenaires* (au lieu d'une coopération entre partenaires) et de manipuler le WFIO intégré comme une entité à part entière, dans le sens où si une adaptation est nécessaire, elle ne devrait pas inquiéter la cohérence du modèle global.

Dans le prochain chapitre, nous allons développer la première partie de notre contribution qui concerne les patrons de coopération à base de services.

# Partie II

# Contributions et Bilan

# CHAPITRE V

# Notre Approche pour l'Interconnexion des Workflows : Patrons de Coopération à Base de Services

## Introduction

Avant de rentrer dans le vif de ce chapitre qui décrit la première partie de notre contribution, rappelons que notre travail est lié à la coopération planifiée, dans laquelle les partenaires sont connus à priori et le modèle de processus inter-organisationnel est clairement défini, dès la phase de conception. Ce type de coopération est très répandu dans les relations de partenariat de longue durée (alliances interentreprises, consortium d'entreprises,…etc.) qui sont plus réalistes et plus adéquates pour la réalisation de grands projets dans divers domaines tels que l'industrie automotive, la production pharmaceutique/parapharmaceutique, l'industrie électronique (production de circuits intégrés),…etc.

Dans ce type de coopération, l'interconnexion entre processus métiers est réalisée selon un schéma de coopération bien défini, en fonction des offres de chaque partenaire dépendant de leurs compétences respectives et de l'objectif métier global à atteindre. Les schémas de coopération proposés dans (Van der Aalst, 99 ), (Van der Aalst, 2000 ) constituent le point de départ de notre travail de thèse, ils supportent différentes architectures de WFIO et définissent les différentes manières avec lesquelles les partenaires métiers peuvent communiquer, coopérer et coordonner leurs processus.

Puisque dans leurs formes initiales, ces architectures s'avèrent rigides et les modèles de processus métiers supportés s'avèrent difficilement adaptables, nous avons jugé nécessaire de nous pencher sur la question de redéfinition de ces schémas de coopération de manière à obtenir des processus de WFIO suffisamment flexibles, dans le but de faciliter leur adaptation, évolution et/ou réutilisation. Pour cela, comme nous l'avons déjà souligné dans le chapitre I, nous utilisons une approche basée SOA pour la restructuration des modèles de processus (intra-organisationnels) et pour l'interconnexion de ces modèles en vue de construire un modèle de processus inter-organisationnel (WFIO) aisément adaptable pour supporter les changements pouvant l'affecter.
Dans notre cas, un modèle de WFIO obtenu par interconnexion (ou composition) de modèles de WF fournis par des partenaires métiers, obéit à un schéma de coopération clairement défini que nous exprimons à l'aide d'un nouveau concept appelé « Patron de Coopération à Base de Services » (PCBS).

Les patrons de coopération que nous définissons s'appliquent à des processus WF à base de *services*, en vue de les composer (ou les interconnecter). Toutefois dans la réalité, les entreprises peuvent avoir déjà implémenté leurs processus comme des WF à base d'activités. Aussi, nous nous focalisons dans un premier temps sur le rapprochement que l'on peut effectuer entre un WF à base d'activités et un WF à base de services, en vue de déterminer les correspondances de concepts existant entre les deux types de WF et dans quelle mesure, on pourrait restructurer un WF à base d'activités en un WF à base de services. Pour cela, nous dégageons les grandes lignes d'une démarche de restructuration et d'interconnexion dont les deux premières phases appelées respectivement, phase de *mapping* et phase d'*abstraction* serviront de support pour la restructuration des modèles de WF à interconnecter.

Concernant la phase de mapping, nous reprenons le méta-modèle de définition de WF à base d'activités en nous basant sur le noyau proposé par la WFMC et en raffinant certains concepts. Nous examinons particulièrement, les concepts clés de *processus*, *sous-processus* et *activité*. D'un autre côté, en nous basant sur un noyau du méta-modèle de définition de WF à base de services, nous examinons de plus près les concepts clés de *service* (*orchestrateur*, *composite*, *composant* et *élémentaire*). Enfin, nous établissons une correspondance de concepts entre le WF à base d'activités et le WF à base de services. L'objectif est de montrer comment effectuer le mapping pour passer du premier au second.

La phase d'abstraction nous amène à définir trois niveaux de visibilité avec lesquelles les processus WF peuvent être présentés en vue de les interconnecter. Ces niveaux d'abstraction vont de la plus fine granularité fonctionnelle (« activité » ou « service élémentaire ») à la moins fine (« processus » ou « service orchestrateur »), en passant par un niveau intermédiaire (« sous-processus » ou « service composant »). Dans une coopération B2B, cette phase d'abstraction est nécessaire dans la mesure où les partenaires doivent exposer leurs processus métiers avec un degré d'abstraction maximum pour préserver leur autonomie et leur savoir-faire et avec le minimum de visibilité nécessaire à l'interconnexion des processus.

La phase d'interconnexion représente l'essentiel de ce chapitre et consiste en une description des différents patrons de coopération à base de services (PCBS) que nous proposons pour réaliser l'interconnexion (ou la composition) de modèles de WF, selon des architectures de coopération bien identifiées. Nous restons conformes avec les définitions de composition et d'interconnexion de modèles que nous avons énoncées au chapitre III, aussi nous parlerons d'une *composition* de modèles si l'opération génère un seul modèle de WFIO (composite) et nous parlerons d'une *interconnexion* de modèles si l'opération génère deux (ou plus) modèles de WF avec des liens entre eux pour former le modèle de WFIO global. Globalement, notre approche d'interconnexion de WF selon un schéma de coopération donné repose sur trois questions essentielles : (1) Comment restructurer les processus WF en services ? (2) Comment réaliser le contrôle d'exécution des instances ? (3) Comment définir les interactions entre les services fournis par les différents partenaires ?

Pour chaque PCBS proposé, nous présenterons une conceptualisation qui comporte une description générale (référence, nom, structure, utilisation…), un schéma générique du patron et un méta-modèle qui définit les principaux concepts du patron et les liens entre eux ; une dernière partie comportera les règles de mapping et d'abstraction requises pour le patron, les contraintes imposées sur le flux de données des WF à interconnecter ainsi que les caractéristiques de l'opération d'interconnexion (ou composition) de modèles, spécifiques au patron. Nous proposons aussi une formalisation des différents patrons en définissant un opérateur de coopération pour chacun d'eux.

La suite de ce chapitre est organisée comme suit : nous commençons par présenter un bref aperçu des patrons de WF, suivi d'un méta-modèle de WF à base d'activités et les principaux concepts qui y sont rattachés. Un méta-modèle de définition de WF selon une approche SOA est ensuite décrit pour ressortir aussi les concepts clés. Nous présentons ensuite les phases de notre démarche de restructuration et d'interconnexion de WF et nous enchainons par la description des différents PCBS proposés.

## V.1 Aperçu sur les patrons de Workflow

Depuis plusieurs années, l'approche à base de patrons connait un réel succès dans divers domaines, commençant par l'ingénierie des logiciels (Gamma, 1995 ), **(**Font-Conte et al., 1999 ). Un pattern (ou patron) est une solution récurrente qui décrit et résout un problème général dans un contexte particulier. Les patterns informatiques fournissent des solutions réutilisables et adaptables pour la résolution des problèmes informatiques, indépendamment de tout langage de programmation. L'avantage de l'approche à base de patrons réside dans le fait que le patron fournit une unité de raisonnement très modulaire, en effet chaque patron existe pour répondre à un problème type. Cette modularité facilite la maintenabilité d'une solution à base de patrons et favorise la réutilisation de patrons élémentaires existants pour l'élaboration de patrons plus complexes.

Comme les design patterns (patrons de conception), les workflow patterns (Van der Aalst et al., 2003 ), (Russel et al., 2006a) représentent des solutions approuvées et adoptées pour des problèmes récurrents du workflow, ces patterns sont classés selon leurs utilisations (ou domaine de la solution), en différents types dont les principaux sont les suivants:

- Les « *Control flow patterns* » : expriment les solutions liées au contrôle de flux dans les workflows (séquence, parallélisme, synchronisation…)
- Les « *Data patterns* » : expriment les solutions aux problèmes liés aux données dans les workflows (formatage de requête, les instances de données…)
- Les « *Resource patterns* » : expriment les solutions liées à la gestion des ressources dans les workflows (allocation, indexation…)
- Les « *Exception Handling patterns* » : comme leur nom l'indique, représentent les patterns liés à la gestion des erreurs et des exceptions, durant l'exécution des instances de processus.

Puisque dans le cadre de cette thèse, nous nous concentrons sur les aspects fonctionnel, comportemental et interactionnel des processus de WFIO, nous donnons un bref aperçu des patrons de contrôle de flux. Ces derniers sont classés en quatre catégories comme le montre la figure V.1.

- les patrons de base de contrôle de flux (basic control flow patterns) qui englobent la séquence (sequence), le parallélisme (parallel), la synchronisation (synchronization), le choix exclusif (exclusive choice) et la fusion simple (simple merge).
- les patrons de branchement avancés (advanced branching patterns) où on trouve le choix multiple (multiple choice), la fusion multiple (multiple merge), la fusion synchronisée (synchronized merge), la terminaison implicite (implicite termination) et le discriminateur structuré (structured discriminator).
- les patrons d'instances multiples (multiple instance patterns) comportant le patron d'instances multiples sans synchronisation (multiple instance without synchronization).
- les patrons à base d'état (state based patterns) comme le choix différé (deffered choice), le milestone, l'annulation (canstellation) et l'entrelacement (parallel interleaving).

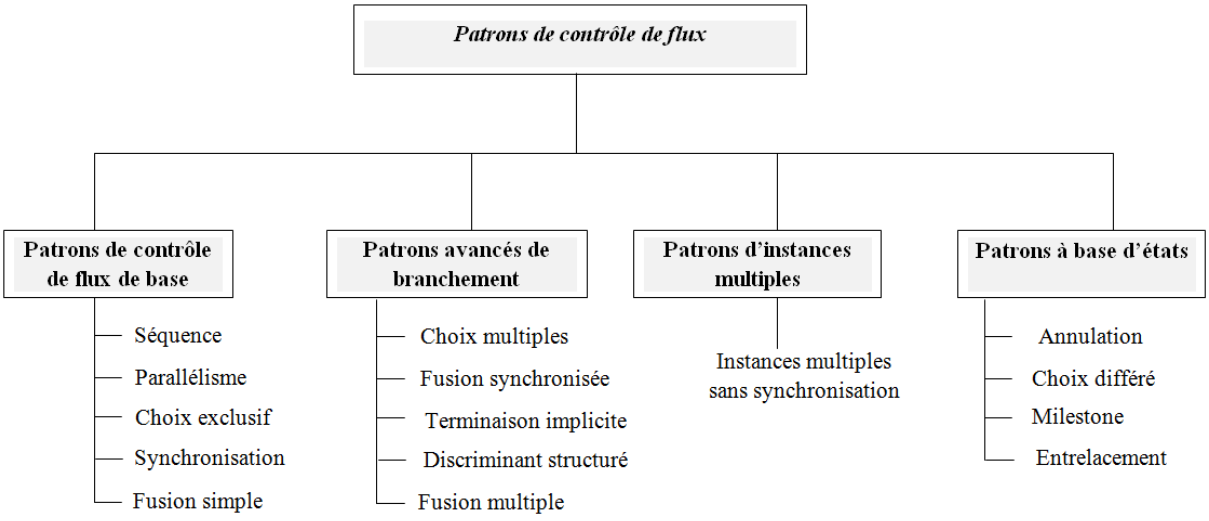



**Figure V.1.** *Les patrons de contrôle de flux*

## V.2 Concepts du Workflow à base d'activités

Rappelons que notre objectif est d'abord, de restructurer un WF à base d'activités en WF à base de services, pour cela il est nécessaire d'examiner les principaux concepts sur lesquels repose le WF à base d'activités afin de pouvoir établir une correspondance entre les concepts du modèle source (WF à base d'activités) et ceux du modèle cible (WF à base de services).

### V.2.1 Méta-modèle de définition de processus WF à base d'activités

Dans cette section, nous reprenons le méta-modèle de WF pour la définition de processus, dans lequel nous essayons de raffiner les principaux concepts.

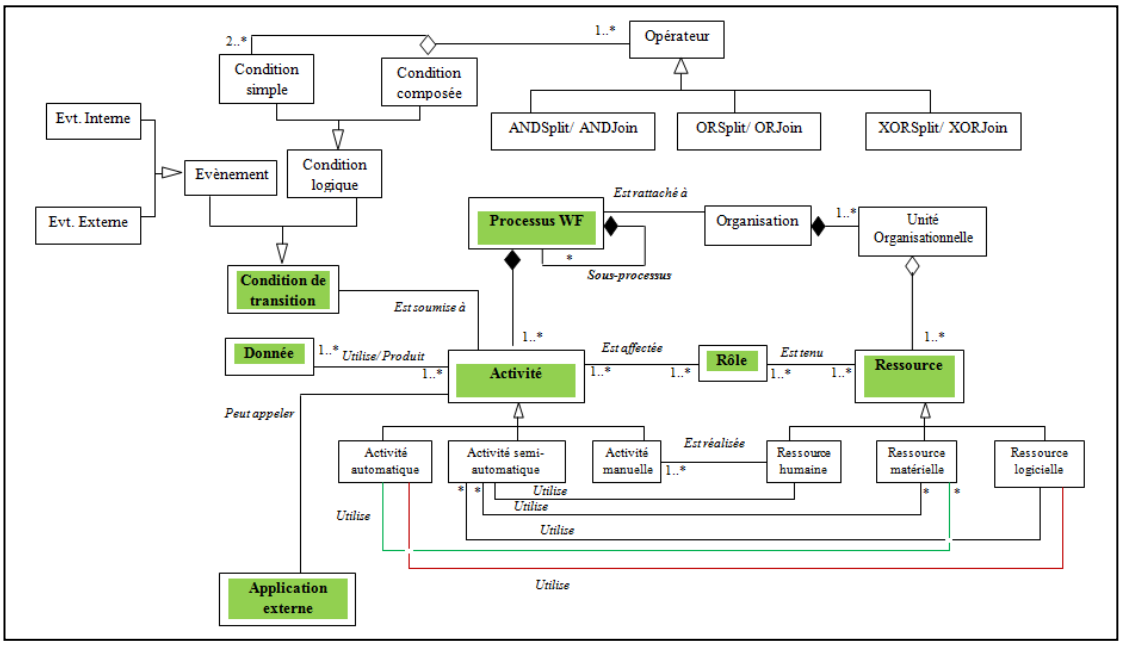



**Figure V.2.** *Méta-modèle de Workflow à base d'activités*

Au niveau intra-organisationnel, un processus de WF est rattaché à une organisation qui est structurée en un ensemble d'unités organisationnelles comportant des ressources humaines, matérielles et logicielles. Le processus WF peut être décomposé en un ensemble de sous-processus qui ont la même structure que le processus global. Au niveau le plus bas de la décomposition, on trouve les activités. Une activité utilise et produit des données, son exécution est soumise à une condition de transition pouvant être un événement interne ou externe à l'organisation, elle peut aussi être une condition logique simple ou composée permettant de contrôler les exécutions parallèles, alternatives et les synchronisations dans le processus. L'activité peut être *manuelle* réalisée par une ressource humaine, *semi-automatique* faisant intervenir une ressource humaine et une ressource de type matériel ou logiciel ou *automatique* réalisée par une ressource matérielle ou logicielle. Une activité peut appeler une application externe (qui n'est pas développée dans le cadre du WF). Le concept de rôle est très important dans le WF car il permet au niveau Runtime, une affectation flexible des ressources selon leurs compétences dans le processus et leur charge de travail. Ainsi, une activité affectée à un rôle donnée peut être réalisée par n'importe quelle ressource qui détient ce rôle.

Dans la suite, nous essayons d'examiner de plus près les concepts d'activité, de sous-processus et de processus WF.

### V.2.2 Le concept de « processus »

En examinant le concept de processus WF à part, on peut dire qu'un processus WF est rattaché à une organisation, il est peut être composé de sous-processus, il est soumis à un ou plusieurs événements déclencheurs qui sont externes, il utilise des ressources de l'organisation, il consomme des données d'entrée afin de fournir une valeur ajoutée et produire des données de sortie. Cette valeur ajoutée représente la fonctionnalité métier remplie par le processus (ou en d'autres termes les objectifs du processus).

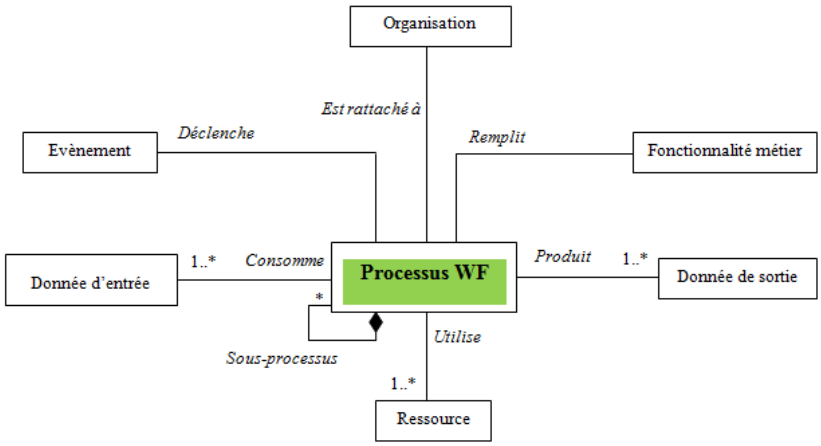



**Figure V.3.** *Le concept de processus WF*

### V.2.3 Le concept de « sous-processus »

Concernant le concept de sous-processus, il faut noter que dans le cas où le processus WF est complexe, il est souvent décomposé en sous-processus. Chaque sous-processus à son tour, peut être décomposé en sous-processus moins complexes.

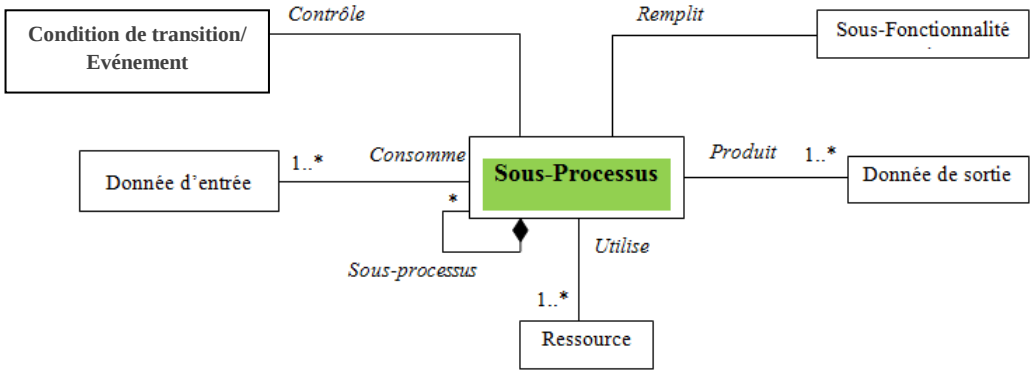



**Figure V.4.** *Le concept de sous-processus*

Comme le processus, un sous-processus consomme des données d'entrée et produit des données en sortie, ces données restent internes au processus WF et ne sont pas livrables à l'environnement extérieur. Un sous-processus est soumis à des conditions qui contrôlent son exécution et éventuellement à des événements internes ou externes. En utilisant les ressources de l'organisation, un sous-processus fournit une valeur ajoutée par la réalisation d'une sous-fonctionnalité qui fait partie de la fonctionnalité métier globale fournie par le processus WF. On peut dire que la fonctionnalité métier remplie par le processus WF est une agrégation des sous-fonctionnalités remplies par les différents sous-processus composants.

### V.2.4 Le concept « d'activité »

La définition et l'exécution d'un processus WF repose sur le concept clé de l'activité. Celle-ci représente une unité de travail non décomposable (i.e élémentaire). Une activité réalise une fonction élémentaire dans le processus, en transformant des données d'entrée en données de sortie. Comme le sous-processus, une activité est soumise à une condition de transition et éventuellement à des événements internes ou externes.

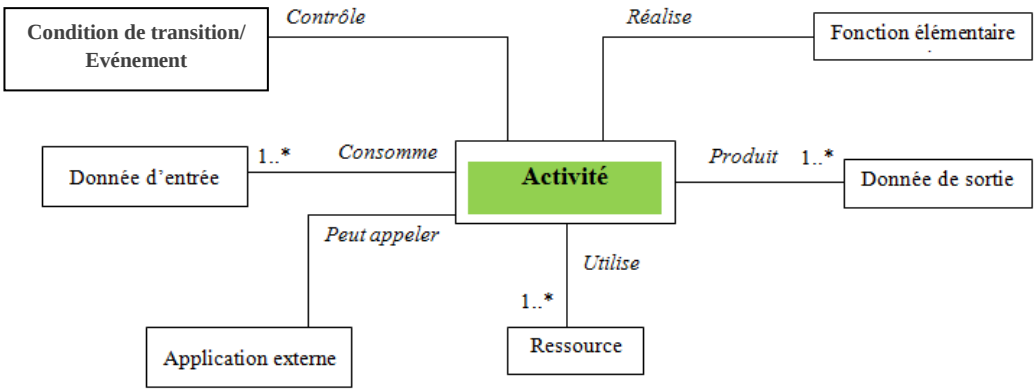



**Figure V.5.** *Le concept d'activité*

L'activité peut être manuelle, semi-automatique ou automatique ; une activité manuelle n'est pas sous le contrôle du SGWF. Dans ce cas, afin de permettre la progression du processus, on peut envisager des signaux de validation pour signaler que l'activité manuelle a été réalisée et injecter éventuellement les résultats de l'activité qui seront consommées par les activités suivantes dans le processus. Une activité semi-automatique fait intervenir une ressource humaine et une ressource matérielle ou logicielle et une activité automatique est réalisée forcément en utilisant une ressource logicielle et éventuellement une ressource matérielle. Une activité peut faire appel à des applications externes pour soutenir sa réalisation.

### V.2.5 Exemple de définition d'un processus à base d'activités

Le diagramme d'activités de la figure V.6 représente un processus de WF de gestion des commandes de clients, faisant intervenir différents acteurs internes à une organisation. A l'arrivée d'une commande, les étapes de ce processus sont : (1) L'agent de vente approuve le BP (Bon de Préparation) et l'envoie au service préparation. (2) Le préparateur prépare la marchandise qui sera contrôlée au service contrôle (cohérence de la marchandise avec le BP), génère un bon de livraison (BL) et l'envoie au service expédition. (3) Le validateur du service contrôle valide le BL, prépare une facture et l'envoie au service comptabilité. (4) Le validateur envoie un email de notification au client pour l'informer que sa commande est prête. (5) L'agent d'expédition livre la marchandise au client.

Le processus est ensuite spécifié à l'aide de jPDL, un langage de définition de processus métier (voir Listing V.1) et les principaux concepts sont exhibés dans le tableau qui suit (TAB V.1).

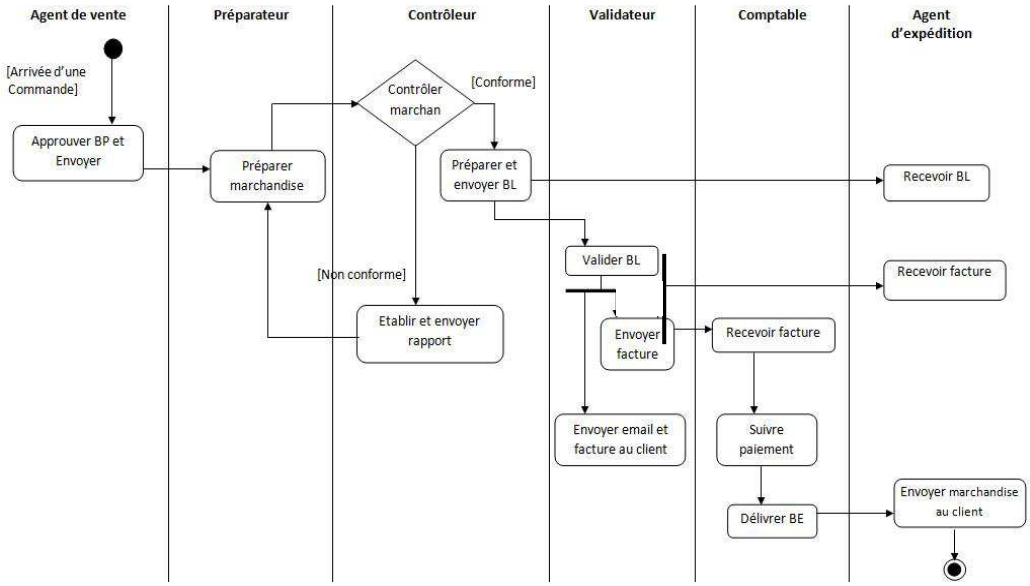



**Figure V.6.** *Diagramme d'activité d'un processus WF « traiter commande »* (Boukhedouma et al., 2011a )

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2"
name="SaidalGenericLab">
<swimlane name="preparerbp">
<assignment actor-id="preparateurbp"></assignment>
</swimlane>
<swimlane name="preparermarchandise">
<assignment actor-
id="preparateurmarchandise"></assignment>
</swimlane>
<swimlane name="controlermarchandise">
<assignment actor-
id="controleurmarchandise"></assignment>
</swimlane>
<swimlane name="validerbp">
<assignment actor-id="validateurbp"></assignment>
</swimlane>
<swimlane name="validerbp">
<assignment actor-id="validateurbp"></assignment>
</swimlane>
<swimlane name="validateurpayement">
<assignment actor-id="comptable"></assignment>
</swimlane>
<swimlane name="delivrerbe">
<assignment actor-id="comptable"></assignment>
</swimlane>
<swimlane name="livrermarchandise">
<assignment actor-
id="livreurmarchandise"></assignment>
</swimlane>

<swimlane name="verifierbe">
<assignment actor-
id="livreurmarchandise"></assignment>
</swimlane>
<start-state name="ReceptionCommande">
<task name="start" swimlane=""></task>
<transition to="Verifier
Commande"></transition>
</start-state>
<task-node name="PreparerMarchandise">
<task name="PreparerMarchandise"
swimlane="preparermarchandise"></task>
<transition to="NotifierControleur"></transition>
</task-node>
<task-node name="ControlerPreparation">
<task name="ControlerMarchandise"
swimlane="controlermarchandise"></task>
<transition to="Preparation"
name="toPreparation"></transition>
<transition to="Preparer_Rapport"
name="toPreparer_Rapport"></transition>
</task-node>
<fork name="Preparation">
<transition to="Notifier_Validateur"
name="toNotifier_Validateur"></transition>
<transition to="PreparerBL"
name="toPreparerBL"></transition>
</fork>
…
</process-state>
<end-state name="end-with-succes"></end-state>
<end-state name="end-GL"></end-state>
</process-definition>
```

**Listing.V.1.** *Spécification du processus « traiter commande » avec le langage jPDL*

Le concept de « *process-state* » dans jPDL correspond à *l'invocation d'un sous-processus* à partir d'un processus parent (processus invocateur). L'instance du processus « parent » s'arrête pendant tout le temps d'exécution du sous-processus invoqué et reprend l'exécution lorsque ce dernier est terminé.

**TAB V.1.** *Correspondance de quelques concepts entre le méta-modèle de WF et le langage jPDL*

| **Concept du méta-modèle de processus** | **Concept jPDL** |
|---|---|
| Processus/ sous-processus | process/sub-process |
| Activité manuelle / semi-automatique | task-node |
| Activité automatique | node |
| Rôle | swimlane |
| Ressource humaine | actor |
| Artefact | variable |
| Enchaînement séquentiel | transition |
| Opérateur de parallélisme | fork |
| Opérateur de synchronisation | join |
| Opérateur de disjonction | decision |

Dans le but d'établir une correspondance des principaux concepts entre WF à base d'activités et WF à base de services, nous commençons par établir un méta-modèle de définition de processus WF conformément à une approche SOA.

## V.3 Concepts de WF à base de services

Dans cette section, nous décrivons un méta-modèle de définition d'un WF selon une approche SOA, à partir duquel nous faisons ressortir trois principaux concepts : *service orchestrateur (composite)*, *service composant* et *service élémentaire* afin de montrer leurs correspondances respectives avec les concepts de *processus WF*, *sous-processus* et *activité* que nous avons explicités dans la section V.2.

### V.3.1 Méta-modèle de définition de processus WF selon une approche SOA

Si l'on considère un partenaire métier représentant une organisation donnée qui implémente son processus métier (ou son WF) selon une approche SOA, on peut dire qu'un processus métier (ou un WF) est défini par une composition de services. Un service peut être composé d'autres services chacun remplissant une sous-fonctionnalité du processus global, on parle de *service composant.* Particulièrement un service composite peut être le *service orchestrateur* qui définit le contrôle de flux d'un processus WF composé de services, via une fonction d'orchestration. Celle-ci utilise des opérateurs de contrôle de flux pour exprimer la séquence, le parallélisme, l'exécution alternative et la synchronisation entre services.

Au niveau le plus bas de la décomposition, on retrouve les *services élémentaires* qui remplissent des fonctions élémentaires à travers les opérations qui les implémentent. Une opération est soumise à une pré-condition, produit un effet et génère une post-condition. Chaque opération utilise des ressources pour sa réalisation. Un service (élémentaire ou autre) peut être *interactif* (semi-automatique) ou *automatique*. Un service interactif utilise des ressources matérielles et/ou logicielle et nécessite l'intervention d'une ressource humaine contrairement au service automatique qui s'exécute sans une quelconque interaction avec un utilisateur humain.

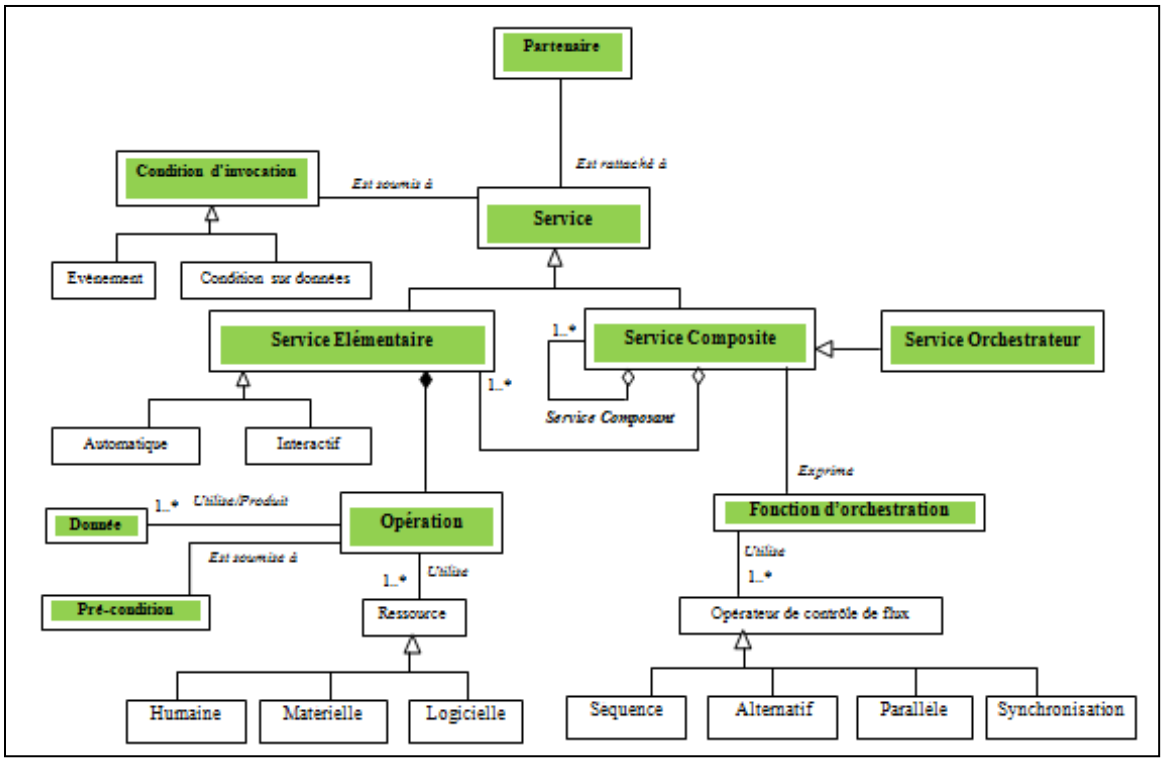



**Figure V.7.** *Méta-modèle de définition de processus WF selon une approche SOA*

A partir du méta-modèle présenté dans la figure V.7, nous reprenons les concepts de service orchestrateur, service composite, service composant et service élémentaire, pour ressortir les principales entités qui y sont rattachées et pouvoir effectuer ensuite un rapprochement avec les concepts de processus, sous-processus et activité (dans le WF à base d'activités).

### V.3.2 Les concepts de « service composite » et « service orchestrateur »

Un service orchestrateur (au niveau local) est rattaché à un partenaire métier, il hérite d'un service composite dont le comportement est défini via une fonction d'orchestration.

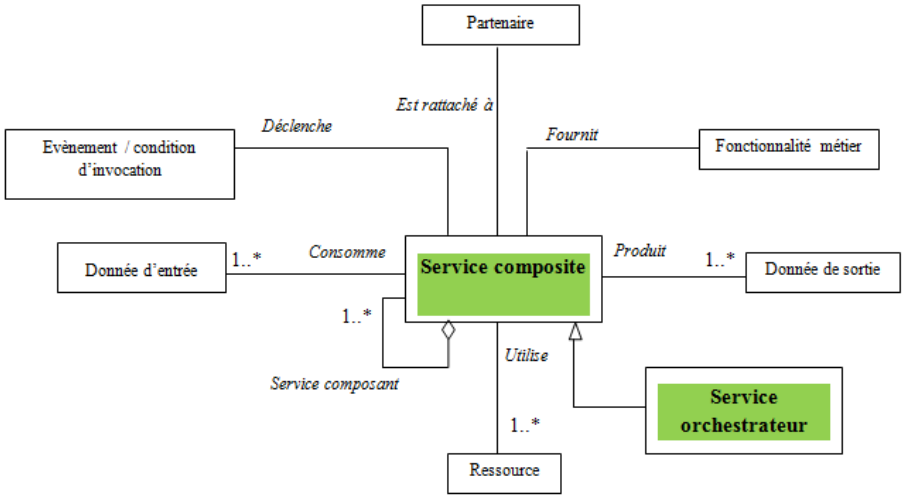



**Figure V.8.** *Concepts de service composite et service orchestrateur*

Le service orchestrateur remplit une fonctionnalité métier globale obtenue par agrégation des sous-fonctionnalités remplies par les services composants (qui peuvent être élémentaires ou composites). Il est soumis comme tout service, à une condition d'invocation qui peut être un

évènement de déclenchement de processus, il consomme des données et produit des résultats et il a besoin de ressources pour son exécution. De ce fait, un *service orchestrateur* local à un partenaire métier peut être assimilé à un *processus WF* intra-organisationnel composé d'un ensemble de services (les services composants) qui peuvent être à leur tour composés d'autres services.

### V.3.3 Le concept de « service composant »

Le service composant peut être lui-même composé d'autres services, on voit bien qu'il est entouré des mêmes concepts que le service composite, sauf qu'il remplit une sous-fonctionnalité. L'agrégation des sous-fonctionnalités remplies par les services composants permet d'obtenir la fonctionnalité globale du processus. On peut donc dire que le *service composant* est l'analogue de *sous-processus* dans le WF à base d'activités.

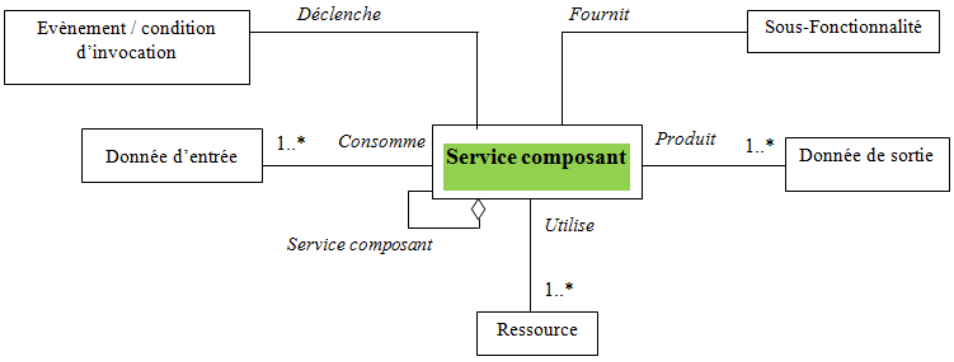



**Figure V.9.** *Le concept de service composant*

### V.3.4 Le concept de « service élémentaire »

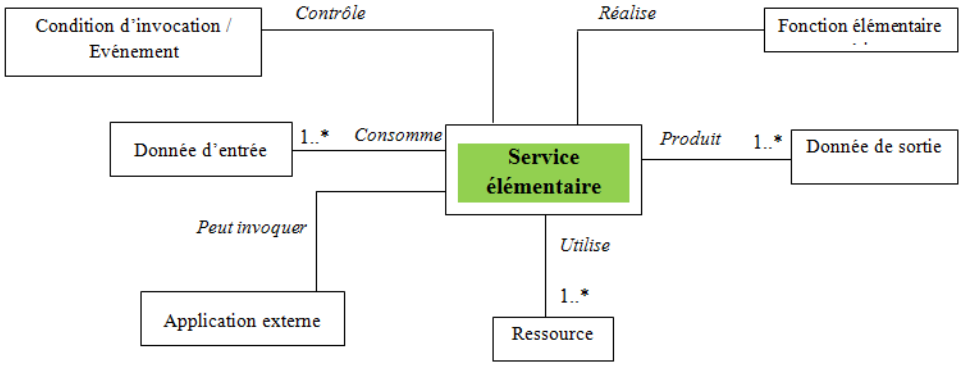



**Figure V.10.** *Le concept de service élémentaire*

Le service élémentaire représente le plus bas niveau de la décomposition, c'est un service non décomposable. Il réalise une fonction élémentaire dans le processus, en transformant des données d'entrée en données de sortie (suite à l'effet des opérations qui le composent), sous le contrôle de conditions et/ou d'événements, en utilisant des ressources. On peut donc dire que le *service élémentaire* est l'analogue d'*activité* dans le WF à base d'activités. Des exemples de processus WF à base de services spécifiés avec le langage BPEL sont fournis dans les annexes C et D de ce document.

En résumé, à travers la correspondance de concepts établie, nous pouvons dire qu'une *activité* peut être encapsulée dans un *service élémentaire*, un *sous-processus* peut être encapsulé dans un *service composant* qui peut lui-même être composite (puisque le sous-processus peut être composé d'autres sous-processus) et un *processus WF* peut être encapsulé dans le *service orchestrateur* qui est le service composite global.

## V.4 Niveaux d'abstraction dans un processus WF

Dans un scénario de coopération, le souci principal de chaque partenaire est la préservation de son autonomie et de son savoir-faire sur son WF local. A cet effet, ce dernier doit être fourni avec un degré d'abstraction maximum mais qui en même temps doit laisser un minimum de visibilité permettant de réaliser son interconnexion avec des WF fournis par d'autres partenaires. Selon le schéma de coopération à réaliser, ceci peut se faire en envisageant différents niveaux d'abstraction du processus: le niveau « activité », le niveau « sous-processus » et le niveau « processus », comme le montre les schémas de la figure V.11.

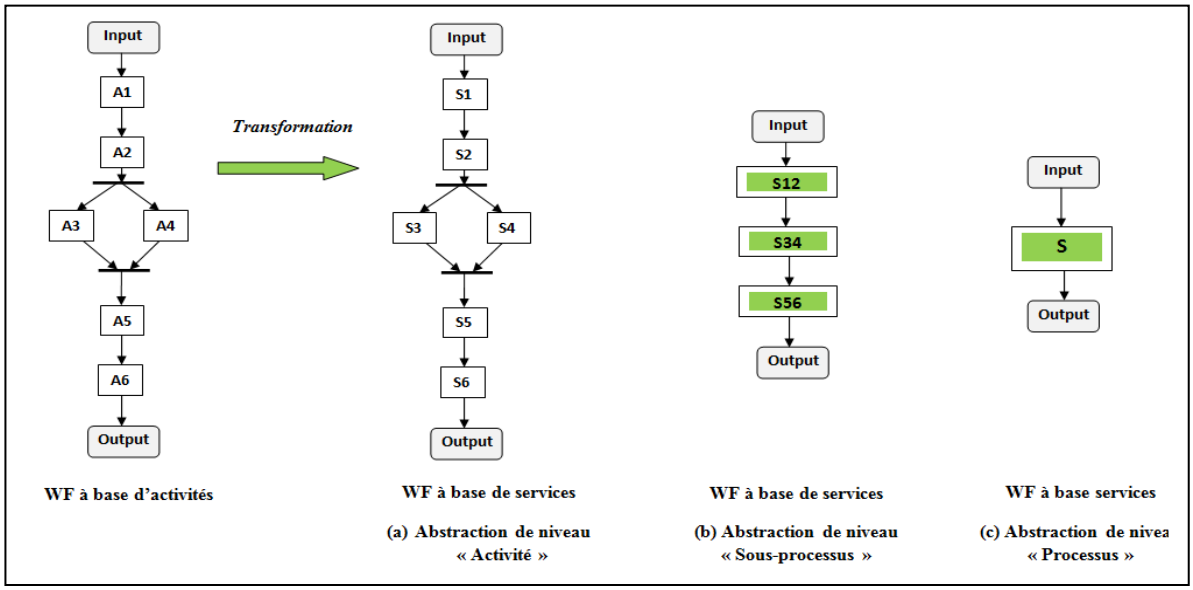



**Figure V.11.** *Niveaux d'abstraction dans un processus WF*

### V.4.1 Niveau « activité »

Représente le niveau le plus bas niveau d'abstraction (granularité la plus fine), où chaque activité du processus sera encapsulée dans un service (élémentaire). A ce niveau la granularité d'encapsulation est l'activité. Ainsi, le processus sera perçu comme une « boite blanche ».

### V.4.2 Niveau « sous-processus »

Représente un niveau intermédiaire d'abstraction où le processus sera découpé en sous-processus et chaque sous-processus sera encapsulé dans un service (composant). Ce niveau permet de montrer certains détails du processus et cacher d'autres détails qui ne sont pas nécessaires à l'interconnexion entre les processus. La granularité d'encapsulation étant le sous-processus (ensemble d'activités et de sous-processus éventuellement), le processus sera donc, assimilé à une « boite grise ».

### V.4.3 Niveau « processus »

Représente le plus haut niveau d'abstraction (forte granularité), le processus WF sera entièrement encapsulé dans un service (orchestrateur), aucun détail du processus ne sera exhibé à l'environnement extérieur, seules les entrées et les sorties seront visibles. La granularité d'encapsulation étant le processus entier qui sera perçu comme une « boite noire ».

Après avoir établi les correspondances entre les principaux concepts du WF à base d'activités et ceux du WF à base de services, nous nous focalisons sur la description de la démarche que nous proposons pour restructurer un WF à base d'activités en WF à base de services et pour interconnecter ces WF restructurés. Intuitivement, il s'agit de répondre aux questions suivantes : comment les partenaires doivent présenter leurs processus WF pour les mettre en coopération avec les processus WF d'autres partenaires? Comment effectuer l'interconnexion de ces processus ?

## V.5 Démarche de restructuration et d'interconnexion de processus

Dans cette section, nous décrivons de manière générique, les principales phases de notre démarche de restructuration de WF à base d'activités en WF à base de services et d'interconnexion de WF à base de services. La démarche s'articule autour de trois phases principales (voir Figure V.12 ci-dessous): une phase de *mapping*, une phase d'*abstraction* et une phase d'*interconnexion.*

- **Phase I : Phase de mapping "Activité – Service"**

Entrée : WF à base d'activités
Sortie : WF à base de services

Cette phase considère que les WF implémentés sont des WF à base d'activités que nous voulons restructurer en WF à base de services. A cet effet, conformément au rapprochement entre les concepts d'activité et de service élémentaire que nous avons présenté dans la section V.3, il s'agit de :

- Identifier les activités du processus
- Encapsuler chaque activité Ai dans un service élémentaire Si tel que :

  Input (Si) = Input(Ai)
  Output(Si) = Output(Ai)
  Condition_invocation (Si) = Condition_transition(Ai)

- **Phase II : Phase d'abstraction des processus**

Entrée : WF à base de services avec fine granularité fonctionnelle
Sortie : WF à base de services avec granularité fonctionnelle moins fine

Cette phase consiste à prendre en entrée le WF à base de services résultat de la première phase dont la granularité est de type « activité » ou « service élémentaire » et de construire un WF à base de services avec une granularité moins fine, de manière à abstraire les détails fonctionnels du processus qui ne sont pas nécessaires à l'opération d'interconnexion. Nous verrons que cette phase d'abstraction dépend de l'architecture de coopération à réaliser entre deux WF et diffère donc d'un schéma de coopération à l'autre. Il s'agit dans cette phase de :

1. Identifier les points d'interaction entre les processus à interconnecter
2. Découper le processus en sous-processus au niveau des points d'interaction. Un sous-processus doit répondre aux conditions suivantes :

- Il doit être délimité: (i) par le point de début du processus et le premier point d'interaction ou (ii) par deux points d'interaction consécutifs ou (iii) par le dernier point d'interaction et le point de fin du processus.
- Si un bloc de branchement parallèle ou alternatif (SPLIT) appartient au sous-processus, le bloc de jonction (JOIN) doit forcément appartenir au même sous-processus.

3. Encapsuler chaque sous-processus SPi dans un service composant SCi (composé de plusieurs services élémentaires et/ou composites), tel que :
   - Input (SCi) = Input (SPi) = Input (première activité de SPi)
   - Output (SCi) = Output (SPi) = Output (dernière activité de SPi )
   - Condition-invocation(SCi) = Condition-transition (SPi) = Condition-transition (première activité de SPi)

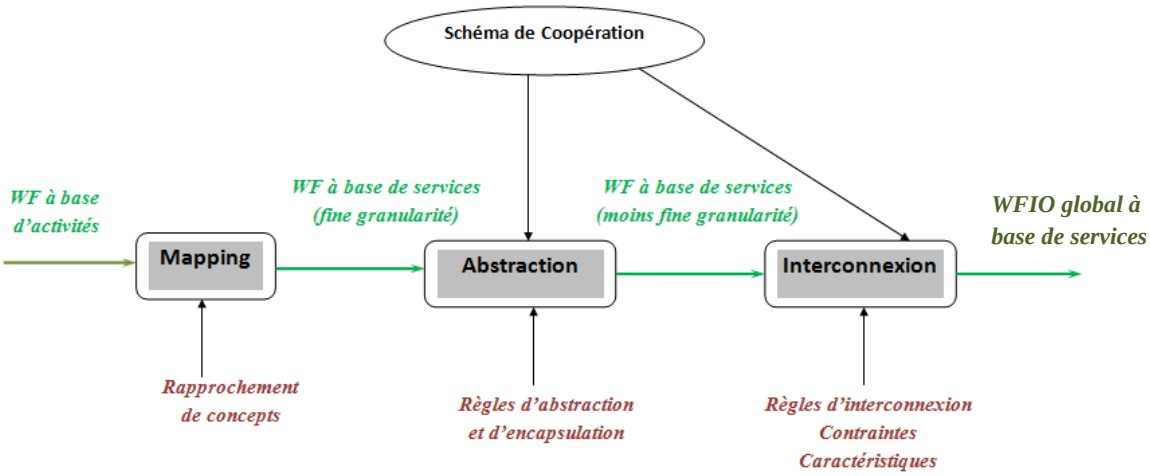



**Figure V.12.** *Phases de la démarche de restructuration et d'interconnexion*

- **Phase III : Phase d'interconnexion**

Entrée : Les WF (deux ou plus) abstraits à base de services + schéma de coopération à réaliser
Sortie : Un WFIO à base de services

Dans cette phase, il s'agit de réaliser l'interconnexion (ou la composition) selon les règles spécifiques à chaque schéma de coopération. Les règles de spécification pour réaliser l'interconnexion (ou la composition) seront détaillées pour chaque patron de coopération que nous proposons. Globalement, un schéma d'interconnexion est proposé pour chaque patron afin de prendre en compte les caractéristiques et les contraintes de chaque architecture de WFIO quant au découpage du WF en services, le contrôle d'exécution des instances et la structure d'interaction (sens des invocations/réponses, les données transmises, les points de synchronisation,… etc.).

A ce niveau, l'interconnexion entre les différents processus revient à des invocations de services de partenaires différents, ces services peuvent encapsuler des activités, des sous-processus ou alors le processus WF entier. Selon le schéma de coopération à réaliser, ces invocations peuvent être de type synchrone (invoke/receive/reply) ou asynchrone (invoke/receive), elles peuvent être directes (entre les processus des partenaires) ou indirectes (en passant par un orchestrateur central).

Conceptuellement, l'interconnexion entre deux processus WF est décrite par un méta-modèle de définition de processus de WFIO. Dans ce méta-modèle, nous définissons un concept central appelé « Patron de Coopération à Base de Services (PCBS)» que nous décrivons dans la suite.

## V.6 Patron de coopération à base de services

Notre approche d'interconnexion de WF selon un schéma de coopération donné repose sur trois questions essentielles que nous avons déjà énoncées.

1. Comment restructurer les processus WF en services ?
2. Comment réaliser le contrôle d'exécution des instances ?
3. Comment définir les interactions entre les services fournis par les différents partenaires ?

Ces trois questions définissent les trois dimensions principales sur lesquelles repose notre définition du concept de patron de coopération à base de services (PCBS). Par suite, un PCBS est défini par: la distribution des services sur les sites des partenaires, le contrôle d'exécution et la structure d'interaction entre les différents fragments de processus, comme le montre la figure V.13. A ce niveau, nous définissons ce concept de manière générique afin de couvrir les six architectures de WFIO considérées et de ressortir les principaux concepts de définition d'un WFIO à base de services. Plus loin dans la description des différents patrons de coopération, nous procéderons à un raffinement de ce méta-modèle pour l'adapter aux différentes spécificités requises pour chaque patron.

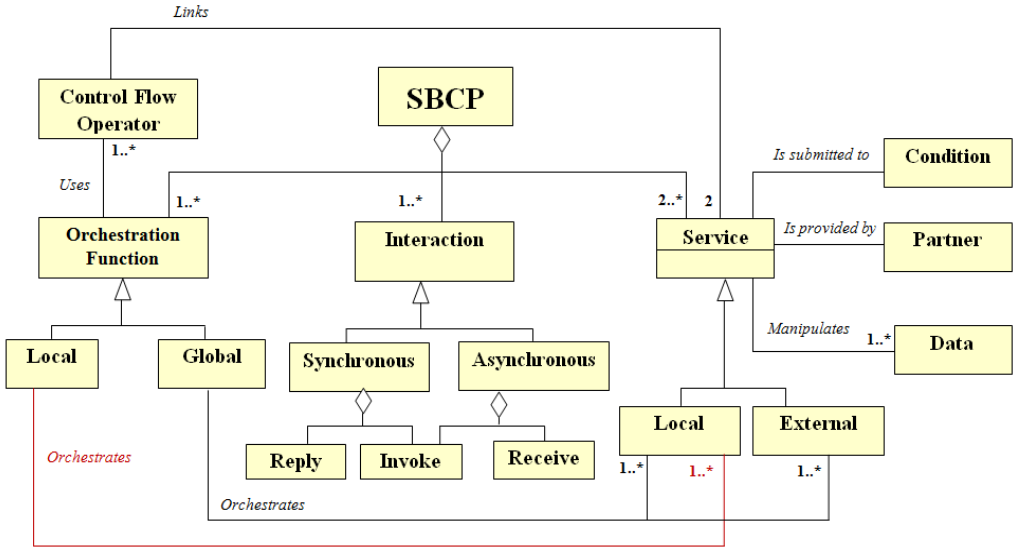



**Figure V.13.** *Méta-modèle d'un Patron de Coopération à Base de Services (PCBS)* (Boukhedouma et al., 2014b )

### V.6.1 La dimension « distribution de services »

Concernant la première dimension qui est la distribution des services sur les sites des partenaires, nous considérons que chaque service encapsule une partie ou la totalité d'un WF local (intra-organisationnel), selon le niveau d'abstraction requis et est implémenté au niveau du partenaire qui le fournit. Cette dimension correspond à la dimension « partitionnement du processus » déjà définie pour un WFIO à base d'activités (cf. chapitre II). Du point de vue d'un partenaire donné, un service peut être soit implémenté localement (service local) soit fourni par un partenaire externe (service externe).

### V.6.2 La dimension « contrôle d'exécution »

La seconde dimension qui correspond au contrôle d'exécution (pouvant être centralisé, décentralisé, hiérarchisé ou mixte) est exprimée à travers le concept de *fonction d'orchestration* qui abstrait la structure du processus en termes de contrôle de flux et d'interactions entre les services composant le processus. Ainsi, le contrôle centralisé est exprimé par une fonction d'orchestration globale implémentée sur le site d'un seul partenaire qui contrôle l'exécution de tout le processus de WFIO. Par contre, le contrôle décentralisé est exprimé à travers un ensemble de fonctions d'orchestration ; chacune implémentée au niveau d'un partenaire afin de contrôler l'exécution du fragment de processus implémenté localement. Pour supporter le contrôle hiérarchisé, il existe une fonction d'orchestration globale qui contrôle les invocations de services locaux et externes et un ensemble de fonctions d'orchestration locales qui contrôlent l'exécution de WF secondaires impliqués dans une coopération.

### V.6.3 La dimension « interactions »

La troisième dimension définit les interactions entre les services fournis par les différents partenaires pour former le processus de WFIO global. Cette dimension est exprimée via les activités d'interaction du type invoquer/recevoir (invoke/receive) pour la communication asynchrone et invoquer/recevoir/répondre (invoke/receive/reply) pour la communication synchrone.

### V.6.4 Contrôle de flux et fonction d'orchestration

Une fonction d'orchestration exprime les liens de séquence, de parallélisme et d'exécution alternative entre les services fournis par différents partenaires impliqués dans une coopération. De ce fait, deux services consécutifs dans la définition du processus, sont reliés entre eux par un opérateur de contrôle de flux. Sur la figure V.14, nous introduisons les opérateurs de base de contrôle de flux utilisés pour exprimer la *fonction d'orchestration*. Ces opérateurs sont décrits à l'aide d'une notation générale indépendamment de tout langage de spécification ou plateforme.

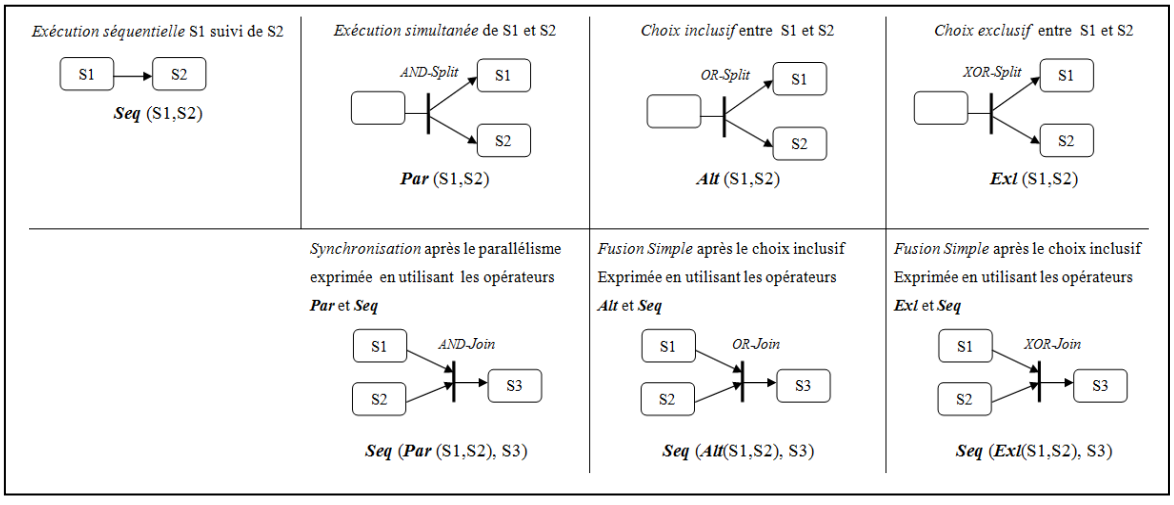



**Figure V.14.** *Opérateurs de base de contrôle de flux*

La figure V.15 illustre le concept de *fonction d'orchestration* sur un exemple de WF comportant cinq services S1, S2,… S5 en plus des services *Sin* et *Sout* correspondant à la réception des données et le renvoi de résultats du processus. Pour des raisons de clarté, la fonction

d'orchestration peut être structurée en blocs ***Bi*** pouvant être séquentiels, parallèles ou alternatifs comme le montre la figure V.15.

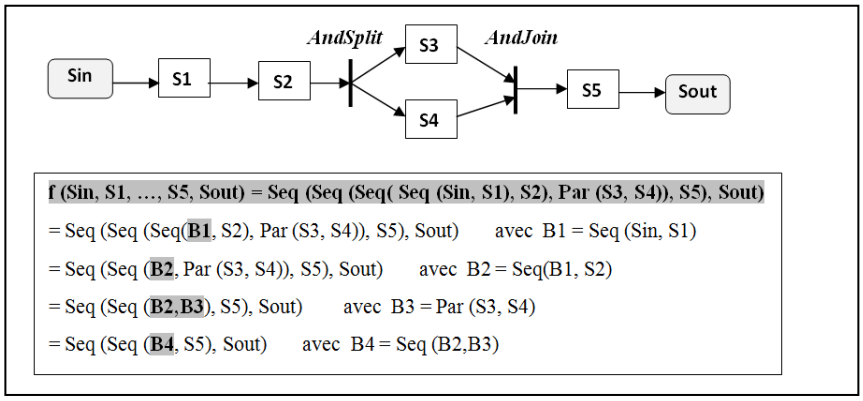



**Figure V.15.** *Exemple de fonction d'orchestration*

La fonction d'orchestration peut être représentée par un arbre binaire avec deux types de nœuds : les *services* et les *opérateurs* ; un parcours *préfixé* de l'arbre permet de générer la structure du processus.
En utilisant ce concept de fonction d'orchestration, nous introduisons les définitions suivantes pour WF à base de services et WFIO à base de services:

### V.6.5 Définitions de WF et WFIO à base de services

- **Définition V.1:** WF à base de services

Un WF (intra-organisationnel) à base de services, rattaché à un partenaire i est défini par une paire **< S, f >** où :

**S** = {***Sin***, Si1, Si2, …, ***Sout***} est un ensemble de services Sij locaux (implémentés au niveau du partenaire i) avec deux services particuliers *Sin* pour la réception des données d'entrée et *Sout* pour le renvoi des résultats (données de sortie)

**f** est une fonction d'orchestration locale (implémentée au niveau du même partenaire) telle que :

**f** (Sin, Si1, Si2, … , Sout) = Sin ***op1*** *Si1* ***op2*** *Si2*… ***opn*** *Sout* = ***Si*** et op1, … opn sont des opérateurs de contrôle de flux.

***Si*** étant le service orchestrateur représentant le WF fourni par le partenaire ***i***

- **Définition V.2:** WFIO à base de services

Soient WF1 = **< SL1, f1 >** et WF2 = **< SL2, f2 >** deux WF intra-organisationnels rattachés respectivement à deux partenaires i et j
**SL1** = {***Sin1***, S11, S12, …, S1n1, ***Sout1***} et **SL2** = {***Sin2***, S21, S22, …, S2n2, ***Sout2***} représentent deux ensembles de services locaux à WF1 et WF2, respectivement.

Un WFIO à base de services obtenu par la mise en coopération de WF1 et WF2 est défini par une paire **<S, F>** où S = SL1 **U** SL2 et F est un ensemble de fonctions d'orchestration f, où

**f** (Sin, Si1, Si2, … , Sim, …,Sout) = Sin ***op1*** *Si1* ***op2****Si2*… ***opk*** *Sout*= ***Si***
avec Sij appartenant à SL1USL2.

Intuitivement, un WFIO à base de services reliant deux workflows WF1 et WF2, est défini par l'union des services composant WF1 et WF2 et une ou plusieurs fonctions d'orchestration qui décrivent le contrôle de flux entre les services des deux WF. Selon le patron de coopération implémenté, une fonction d'orchestration **f** associe à un ensemble de services ***Sij*** de S, un service composite ***Si*** correspondant au WFIO global ou à un fragment du WFIO implémenté au niveau d'un partenaire. Les interactions internes sont définies par les interactions entre les services d'un même partenaire, alors que les interactions externes sont définies par les interactions entre les services de partenaires distincts.

**Remarque**

Pour revenir aux notions de composition et d'interconnexion de modèles de WF expliquées dans le chapitre III, en se basant sur le concept de fonction d'orchestration, on peut dire que:

- Dans le cas où le contrôle de flux dans le WFIO est exprimé par une seule fonction d'orchestration globale, l'interconnexion entre les WF des différents partenaires repose sur une opération de ***composition*** des modèles de WF mis en coopération.
- Dans le cas où le contrôle de flux dans le WFIO est exprimé par plusieurs fonctions d'orchestration locales avec des liens entre les services des différents partenaires, l'interconnexion entre les WF des différents partenaires repose sur une opération d'***interconnexion*** des modèles de WF mis en coopération.

Dans la section suivante, nous décrivons les patrons de coopération que nous proposons pour l'interconnexion de WF à base de services.

## V.7 Description des patrons de coopération à base de services

Pour décrire les différents patrons de coopération que nous proposons, nous revenons toujours aux trois questions précédemment énoncées (restructuration en services et distribution, contrôle d'exécution et interaction). Pour y répondre, il faudra préciser pour chaque patron, le niveau d'abstraction requis pour la mise en coopération des WF, il faudra aussi préciser la manière de définir le contrôle de flux entre les services et la manière d'assurer les interactions entre eux.

Pour chaque patron proposé, nous présentons une conceptualisation qui comporte une description générale, un schéma générique du patron et un méta-modèle qui définit les principaux concepts du patron et les liens entre eux. Une dernière partie comportera les règles de mapping et d'abstraction requises pour le patron, les contraintes imposées sur le flux de données des WF à interconnecter ainsi que les caractéristiques de l'opération d'interconnexion (ou de composition) de modèles, spécifiques au patron. Une formalisation de chaque patron est proposée en définissant pour chacun d'eux, un opérateur appelé « *opérateur de coopération* ». Dans un but d'illustration, des exemples de WFIO obéissant aux différents patrons de coopération sont présentés.

### V.7.1 Le patron « Partage de Charge » - PCBS1

#### V.7.1.1 Conceptualisation du patron « Partage de Charge »

Le patron de coopération «Partage de Charge» (ou "Capacity Sharing" ) référencé par PCBS1 (ou SBCP1) obéit à une architecture de type « Partage de Charge ». Ce patron se base sur l'orchestration d'un ensemble de WF fournis par des partenaires métiers, à l'aide d'une fonction d'orchestration *globale* implémentée au niveau d'un site de partenaire ou d'un tiers externe jouant

le rôle d'orchestrateur central, pour exprimer l'ordre l'invocation des services impliqués dans le WFIO global.

Dans ce type d'architecture, le contrôle est centralisé, les interactions sont de type synchrone entre le client (l'orchestrateur) et les services fournis par les différents partenaires. La figure V.16 décrit le patron de coopération « Partage de Charge » en spécifiant le schéma générique du patron, le méta-modèle décrivant les concepts de définition d'un WFIO obéissant au patron, ce méta-modèle correspond en grande partie au méta-modèle générique de PCBS de la figure V.13, avec des spécificités relatives à l'architecture « Partage de Charge ». La dernière partie de la description spécifie les règles de mapping, le niveau d'abstraction requis ainsi que les règles, les caractéristiques et les contraintes de l'interconnexion de modèles (qui est une composition de modèles dans ce cas puisqu'elle donne lieu à un seul modèle composite). Le niveau d'abstraction requis diffère d'un cas à l'autre, il peut varier de la plus grande granularité (processus) jusqu'à la plus fine (activité) en passant par le niveau intermédiaire (le sous-processus).

Reference-Patron: **PCBS1 ou SBCP1**
Nom: Patron "Partage de charge" ou "Capacity Sharing" Pattern
Coopération: Partager l'exécution d'un processus WF global implémenté à une seule localisation.
Contrôle: Centralisé
Structure: Orchestration globale de services fournis par un ensemble de partenaires
Type d'interaction: Interactions synchrones entre le client (l'orchestrateur) et les services métier.
Utilisation dans la pratique: Coopération dynamique basée sur l'orchestration de services

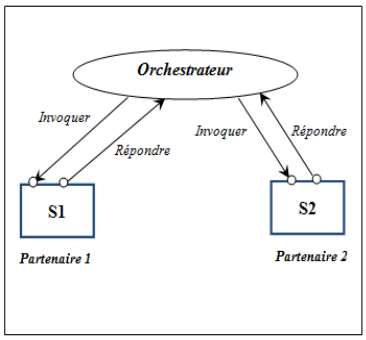
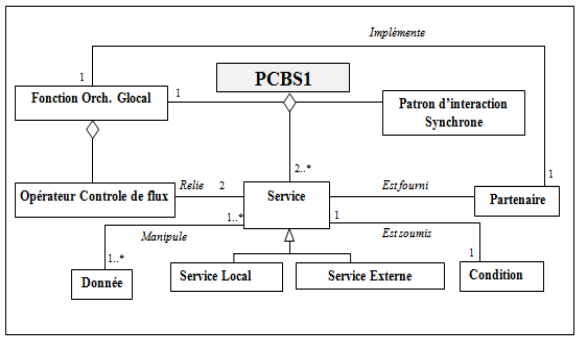



Schéma du Patron "Partage de charge"




Meta-modèle du Patron "Partage de charge"

**Règle de mapping**
**R1.1:** Encapsuler chaque activité d'un WF dans un service.
**Règle d'abstraction**
**R1.2:** Abstraction de niveau « activité », « sous-processus » ou « processus », selon les cas dans la pratique.
**Règles , caractéristiques et contraintes de l'interconnexion**
**R1.3:** Spécifier une fonction d'orchestration globale (contrôle de flux entre les services).
**Caractéristiques de l'interconnexion**
Type d'opération : Composition de modèles
Nombre de modèles en entrée/sorties : plusieurs/un
**Contrainte**
**C1.1** : Les données d'entrée d'un service sont les données de sortie du (ou des) service(s) qui le précèdent directement, dans le processus global.

**Figure V.16.** *Description du patron « Partage de charge »- PCBS1*

Par exemple, dans un processus de prise en charge d'un voyage pour un client qui voudrait un billet d'avion, une réservation d'hôtel et une location de voiture, si les contraintes (dates, lieu, prix) sont fixées au départ, les trois processus peuvent s'exécuter en parallèle et n'ont pas besoin d'interagir ensemble pour des échanges d'information, dans ce cas on a besoin d'une granularité forte (de niveau « processus »). Dans le cas où la réservation de vol doit dépendre du prix du billet d'avion (chercher le moins cher, par exemple), on aura besoin d'une granularité intermédiaire (de niveau « sous-processus ») pour la compagnie aérienne et d'une granularité de niveau « processus » pour les deux autres WF (processus de réservation de chambre d'hôtel et processus de location de voiture) qui peuvent s'exécuter simultanément, une fois que la réservation du billet d'avion est faite. La granularité de niveau « activité » est nécessaire lorsque les processus mis en coopération nécessitent des interactions entre eux (indirectement via l'orchestrateur central), à chaque étape de leur progression.

Pour réaliser un WFIO obéissant au patron « Partage de charge », l'opération à effectuer est une *composition* de modèles (deux modèles – ou plus- en entrée et un modèle composite en sortie). Une contrainte sur les données est imposée entre chaque service et le service qui le suit dans l'orchestration globale, de telle sorte que les données de sortie d'un service soient égales aux données d'entrée du service suivant, afin de garantir la cohérence du flux de données dans le processus global.

### V.7.1.2 Exemple de processus WFIO selon le patron « Partage de Charge »

Le diagramme d'activité de la figure V.17 décrit un WFIO obéissant au patron « Partage de charge ». Il s'agit d'une requête client pour la réservation d'un billet d'avion et d'une chambre d'hôtel, en passant par une agence de voyage.

Dans notre exemple, l'agence de voyage joue le rôle *d'orchestrateur central* qui invoque les WF de la compagnie aérienne et de l'hôtel respectivement, selon un ordre bien déterminé. L'orchestrateur central se charge de transmette les données d'entrée aux différents WF invoqués et de récupérer les résultats de sortie pour finalement les retourner au client. Donc, il n'y a pas d'interactions directes entre les WF des partenaires « compagnie aérienne » et « hôtel ». Dans cet exemple, les WF des deux partenaires sont fournis avec un niveau d'abstraction élevé (niveau « processus ») encapsulés entièrement dans un service composite, puisque l'orchestrateur central a seulement besoin d'envoyer les données d'entrée et de recevoir les résultats de sortie des deux WF.

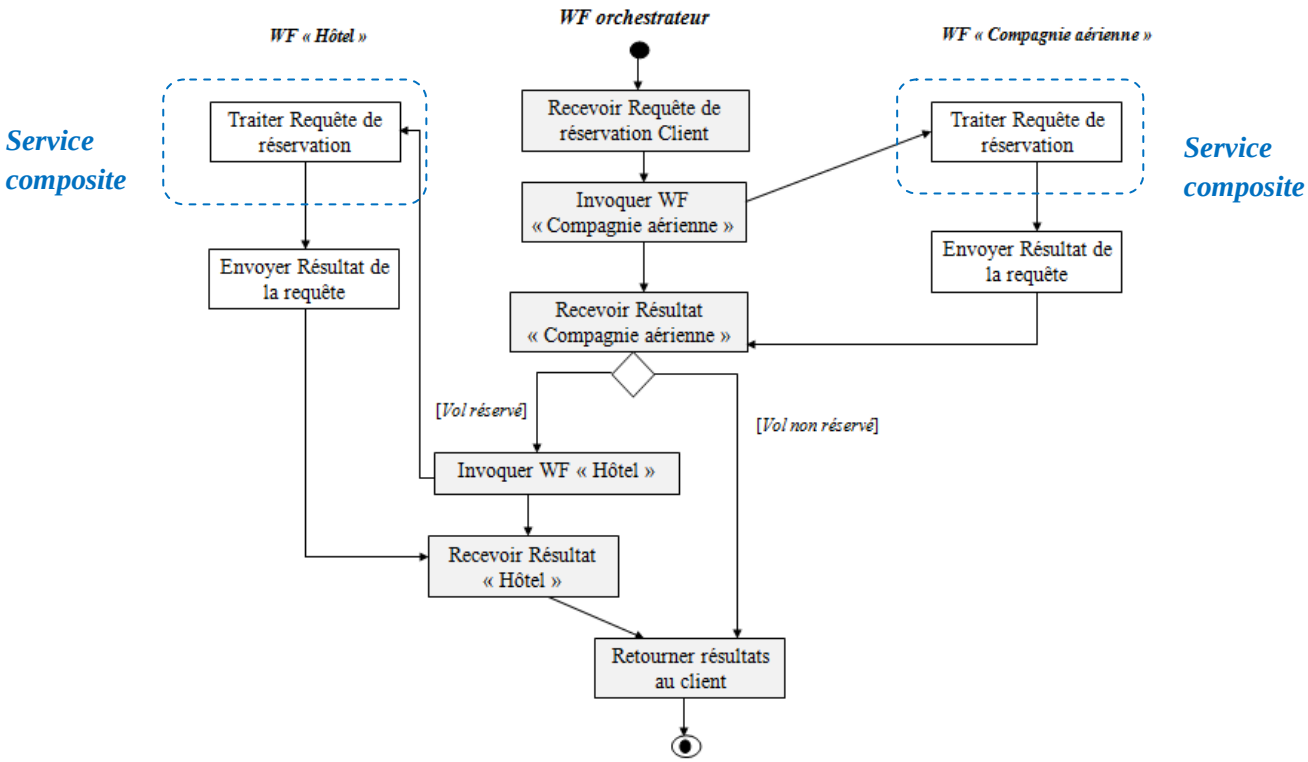



**Figure V.17.** *Exemple d'un WFIO obéissant au patron « Partage de charge »*

### V.7.1.3 Formalisation du patron « Partage de Charge »

Pour la formalisation du patron « Partage de Charge », nous introduisons la définition d'un opérateur de coopération désigné par [CS].

- **Définition V.3 :** l'opérateur [CS]

Soient WF1 = < SL1, f1 > et WF2 = < SL2, f2 > où **SL1** = {***Sin1***, S11, S12, …, S1n1, ***Sout1***} et **SL2** = {***Sin2***, S21, S22, …, S2n2, ***Sout2***} représentent deux ensembles de services locaux à WF1 et WF2, respectivement.
On définit un opérateur **[CS]** pour « Capacity Sharing », qui construit un WFIO obéissant au patron de coopération « Partage de Charge » par composition de WF1 et WF2, comme suit :

**[CS]** (WF1, WF2) = < **S, f** > où:

S = SL1 **U** SL2 et
**f** (Sin, Si1, Si2, … , Sim, …,Sout) = Sin ***op1*** *Si1* ***op2****Si2*… ***opk*** *Sout* = ***Si***
avec Sij appartenant à SL1 **U** SL2 et op1, op2, … opk sont des opérateurs de contrôle de flux.

### V.7.2 Le patron « Exécution Chainée » - PCBS2

Dans une architecture de type *« Exécution Chainée »*, le WFIO global est partitionné en fragments disjoints s'exécutant en *séquence*, chaque fragment est implémenté localement au niveau du partenaire concerné et est géré par un SGWF local.

### V.7.2.1 Conceptualisation du patron « Exécution Chainée »

Pour le patron « Exécution Chainée » à base de services, nous proposons d'*encapsuler entièrement* le WF de chaque partenaire dans un service ; sur la figure V.18, *S1* représente le WF (service orchestrateur) du partenaire 1 et *S2* représente le WF du partenaire 2, l'instance de processus est exécutée selon la succession de services (*S1, S2*). D'une manière générale, le premier service (*S1*) de la séquence est déclenché par un évènement externe (arrivée d'une nouvelle instance), les autres services de la séquence sont invoqués chacun, par le service précédent. En d'autres termes, un service *Si+1* est invoqué par le service *Si* qui le précède dès que ce dernier termine son exécution (Boukhedouma et al., 2014b).

Naturellement, cette architecture est implémentée comme une chorégraphie de services avec un contrôle *décentralisé*. Le patron « Exécution Chainée » est décrit par le méta-modèle présenté dans la figure V.18. Rappelons qu'au niveau interne, les services *S1* et *S2* sont implémentés comme des services composites. En effet, un service interne ***Sij*** peut encapsuler un sous-processus (ou activité) du partenaire ***i***. Ainsi, un service ***Si*** fourni par le partenaire ***i*** est implémenté comme un ensemble de services orchestrés localement à l'aide d'une *fonction d'orchestration locale* avec deux services particuliers *Sini* et *Souti* pour la réception des données d'entrée et le renvoi des résultats. Chaque orchestrateur local ***i*** reçoit les données d'entrée à travers le service *Sini,* invoque son service métier local *Si* (composée des services *Sij*) et invoque ensuite le service *Si+1* avec les résultats d'exécution du service *Si*. Particulièrement, le service *Sin* du premier fragment de la séquence reçoit les données d'une nouvelle instance et le service *Sout* du dernier fragment de la séquence renvoie les résultats d'exécution complète de l'instance.

Remarquons que le niveau d'abstraction requis dans ce cas est de type « processus » (forte granularité). L'opération à effectuer est une interconnexion de modèles ( deux modèles en entrée et deux modèles en sortie avec des liens entre eux) en respectant la contrainte sur les données d'entrée/sortie entre chaque paire de services consécutifs dans la séquence. Ces éléments sont spécifiés dans la dernière partie de la description du patron (voir figure V.18).

Patron-référence: **PCBS2 ou PCBS2**
Nom: Patron "Exécution chainée" ou "Chained Execution" Pattern
Coopération: Exécution séquentielle de fragments de processus (encapsulés dans des services composites) fournis par un ensemble de partenaires.
Contrôle: Décentralisé
Structure: Un ensemble de services orchestrés par un ensemble de fonctions d'orchestration locales.
Type d'interaction: Synchrone ou One-way
Utilisation dans la pratique: Assez répandu dans les processus de la « supply-chain management »
Exemple: Un processus de WFIO impliquant trois partenaires: un fournisseur de matières premières, un producteur de produits semi-finis et un producteur de produits finis

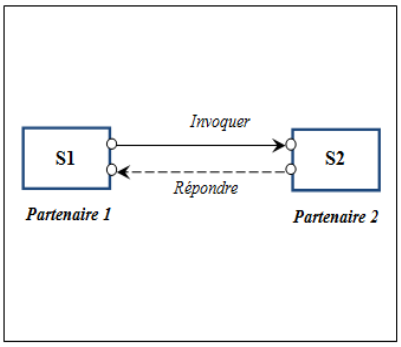
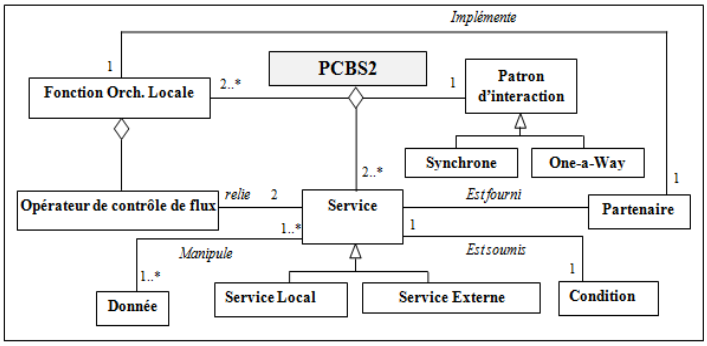



Schéma du Patron "Exécution chainée"




Meta-modèle du Patron "Exécution chainée"

**Règle de mapping**
**R2.1:** Encapsuler chaque activité d'un WF dans un service.
**Règle d'abstraction**
**R2.2:** Encapsuler Chaque WF de partenaire dans un service composite.
⟹ Abstraction de niveau « processus »
**Règles , caractéristiques et contraintes de l'interconnexion**
**R2.3:** Spécifier une fonction d'orchestration locale sur chaque site de partenaire (contrôle de flux entre les services locaux).

**Caractéristiques de l'interconnexion**
Type d'opération : Interconnexion de modèles
Nombre de modèles en entrée/sorties : plusieurs/plusieurs
Liens entre les modèles : Insérer une activité « invoke » à la fin de chaque WF (sauf le dernier de la séquence) pour l'invocation d'un service externe et la transmission de données d'un WF au suivant. Une activité « receive » est automatiquement insérée au début de chaque WF pour récupérer les données transmises du WF précédent.
**Contrainte**
**C2.1** : Les données d'entrée d'un service sont les données de sortie du service qui le précède directement, dans la séquence du processus global.

**Figure V.18.** *Description du patron « Exécution chainée »- PCBS2*

### V.7.2.2 Exemple de processus WFIO selon le patron « Exécution chainée »

La figure V.19 ci-dessous décrit un processus de WFIO obéissant au patron « Exécution chainée ». Le processus est lié au domaine de la production de circuits intégrés et fait intervenir deux partenaires : *partenaire 1* qui implémente tout le WF de la conception du circuit intégré et *partenaire 2* qui implémente le WF de fabrication du circuit. Les deux WF s'exécutent dans un ordre séquentiel et sont supposés implémentés selon une architecture SOA. Ainsi, toutes les phases de chaque processus sont considérées comme des services (composites). Le processus de chaque

partenaire est entièrement encapsulé dans un service composite encadré par les services d'interaction *Sin* et *Sout*.

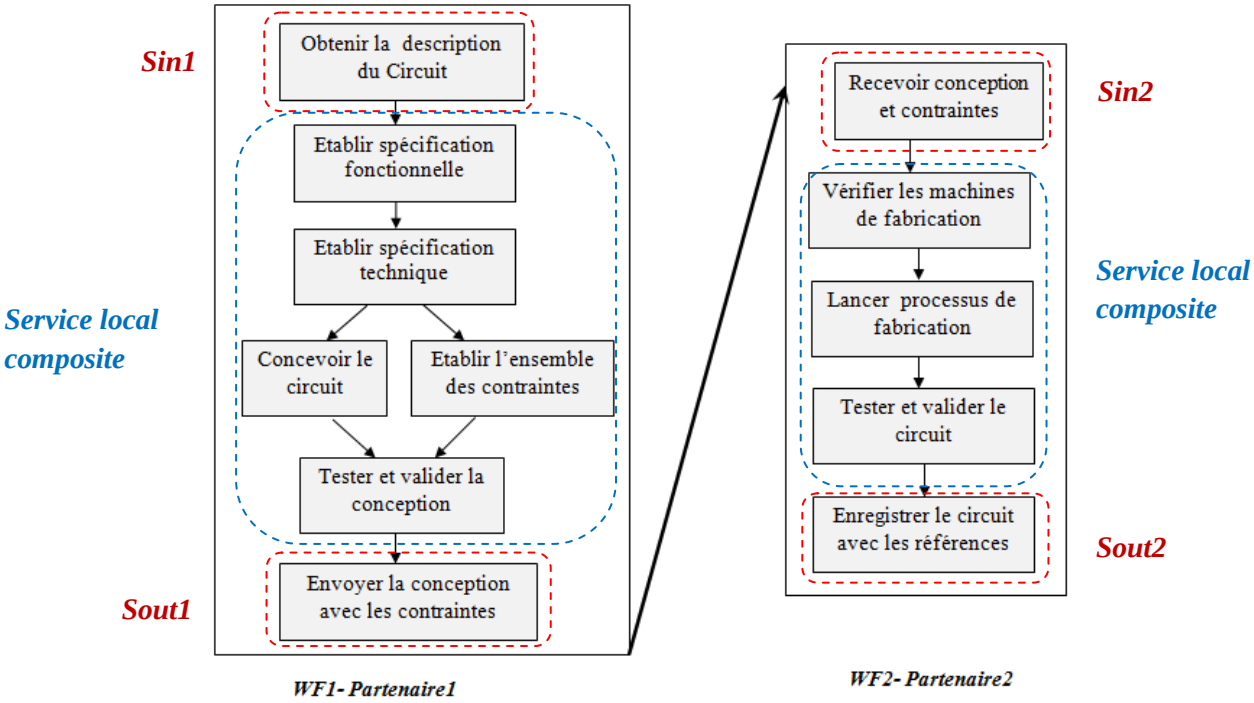



**Figure V.19.** *Exemple de WFIO selon le patron « Exécution chainée »*

### V.7.2.3 Formalisation du patron « Exécution Chainée »

Pour la formalisation du patron « Exécution chainée », nous introduisons la définition d'un opérateur de coopération désigné par [CE].

- **Définition V.4 :** L'opérateur [CE]

Soient WF1 = < SL1, f1 > et WF2 = < SL2, f2 > où SL1 et SL2 représentent deux ensembles de services locaux à WF1 et WF2 respectivement et f1, f2 sont les fonctions d'orchestration respectives de WF1 et WF2 avec f1(Sin1, …, Sout1) = *Sin1* ***op1*** *S11* ***op2*** *S12* … ***opn*** *Sout1* = **S1**

et

f2 (Sin2,…, Sout2)= *Sin2* ***op1*** *S21* ***op2*** *S22* … ***opm*** *Sout2*= ***S2***

On définit un opérateur **[CE]** pour « Chained Execution » qui construit un WFIO obéissant au patron de coopération « Exécution chainée » par interconnexion de WF1 et WF2, comme suit :

**[CE]** (WF1, WF2) = < S, F > où:
S = SL1 **U** SL2 et F = {$\mathcal{f}$1, $\mathcal{f}$2} avec
$\mathcal{f}$1 = **Sub** (f1, Sout1, **S2**) et $\mathcal{f}$2 = **Sub** (f2, Sin2, **S'in2)**

S'in2 représente le service d'entrée du WF2 avec un flux de données mis à jour tel que :
Input (S'in2) = Output (S1)

Dans la définition précédente, nous remarquons l'utilisation d'un opérateur « Sub » que nous définissons pour la substitution d'un service par un autre dans une fonction d'orchestration.

- **Définition V.5:** L'opérateur « **Sub** »

Soit une fonction d'orchestration f et soient deux services Sx référencé dans f et Sy (non référencé dans f). L'opérateur **Sub** s'applique à une fonction d'orchestration f et permet de substituer le premier service Sx par le second Sy, pour obtenir une nouvelle fonction d'orchestration **f'**. On écrira : **f' = Sub (f, Sx, Sy)**

En d'autres termes, **f'** n'est autre que **f** dans laquelle, **Sy** est substitué à **Sx**.

La figure suivante illustre le concept de fonction d'orchestration pour une « Exécution chainée ».

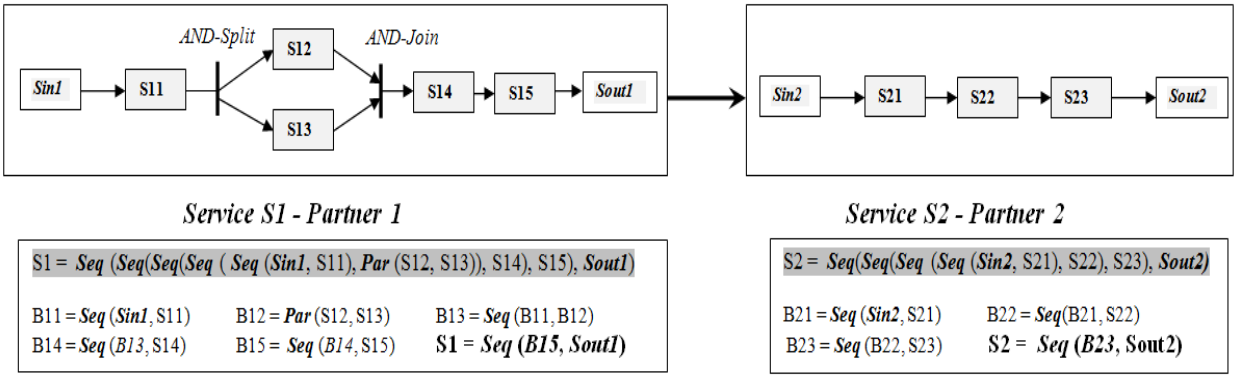



**Figure V.20.** *Exemple de fonctions d'orchestration pour une « Exécution chainée »*

Les fonctions d'orchestration f1 et f2 liées respectivement, aux WF des partenaires 1 et 2 sont exprimées comme suit :

**f1** *(Sin1,S11, …, Sout1) = Seq (Seq(Seq(Seq(Seq(Sin1, S11), Par(S12, S13)), S14), S15), Sout1)*
= Seq (B15, Sout1)

**f2** *(Sin2,S21, …, Sout2) = Seq(Seq(Seq(Seq(Sin2, S21), S22), S23), Sout2)*

Le WFIO obtenu selon un patron « Exécution chainée » sera défini via deux fonctions d'orchestration $f1$ et $f2$ telles que :

$f1$ (Sin1, S11, …, S15, Sout1) = Seq (B15, **S2**) et

$f2$ (Sin2', S21, …, Sout2) = Seq(Seq(Seq (Seq (**Sin'2**, S21), S22), S23), Sout2)

En d'autes termes, $f1$ = **Sub** (f1, Sout1, S2) et $f2$ = **Sub** (f2, Sin2, S'in2)

### V.7.3 Le patron « Sous-traitance » - PCBS3

L'architecture de « Sous-traitance » suppose l'existence d'un partenaire principal implémentant localement, son propre modèle de WF et son propre SGWF devant sous-traiter certaines activités auprès d'autres partenaires (secondaires). Ces derniers hébergent localement leurs WF et leurs SGWF respectifs.

#### V.7.3.1 Conceptualisation du patron « Sous-traitance »

Pour le patron « Sous-traitance », nous proposons *d'encapsuler entièrement* chaque WF secondaire impliqué dans la coopération dans un service (Boukhedouma et al., 2012c), (Boukhedouma et al., 2014b ).

Référence-Patron: **PCBS3 ou SBCP3**
Nom: Patron "Sous-traitance" ou "Subcontracting" Pattern
Coopération: Externalisation de services à d'autres partenaires
Structure: Un ensemble de services internes et externes orchestrés par une fonction d'orchestration globale implémentée au niveau d'un partenaire principal et un ensemble de fonctions d'orchestration locales implémentées au niveau des partenaires secondaires correspondants.
Contrôle : Hiérarchisé
Type d'interaction : Synchrone
Utilisation dans la pratique : Très répandu entre partenaires métier ayant des compétences complémentaires, dans un domaine d'activité donné.
Exemples : Processus de production pharmaceutique, industrie auto-motive, fabrication et assemblage de circuits intégrés, etc.

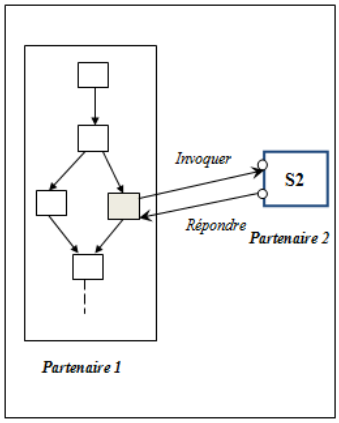
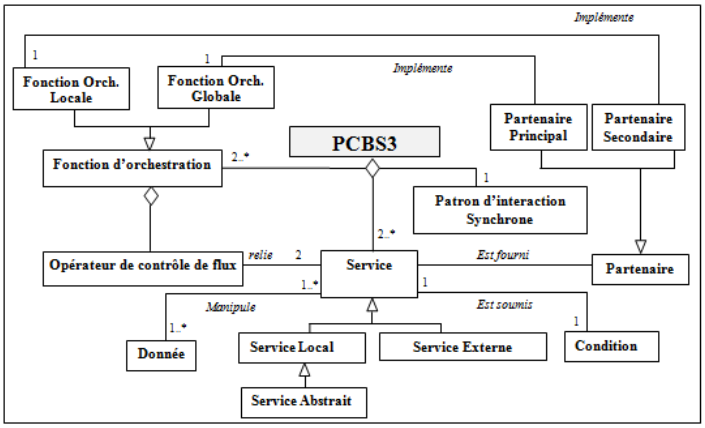



Schéma du Patron "Sous-traitance"




Meta-model du Patron "Sous-traitance"

**Règle de mapping**

**R3.1:** Encapsuler chaque activité d'un WF dans un service.

**Règles d'abstraction**

**R3.2** : Découper le WF principal en sous-processus au niveau des activités à sous-traiter et encapsuler chaque sous-processus dans un service

⇨ Abstraction de niveau « sous-processus »

**R3.3:** Encapsuler Chaque WF secondaire dans un service composite.

⇨ Abstraction de niveau « processus »

**Règles , caractéristiques et contraintes de l'interconnexion**

**R3.4:** Spécifier une fonction d'orchestration globale au niveau du partenaire principal et une fonction d'orchestration locale sur chaque site de partenaire secondaire (contrôle de flux entre les services locaux).

**Caractéristiques de l'interconnexion**

Opération : Interconnexion de modèles
Nombre de modèles en entrée/sortie : plusieurs/plusieurs
Liens entre les modèles : Substituer le service *abstrait* dans le WF principal par le service correspondant fourni par le partenaire secondaire.
Une activité « receive » est automatiquement insérée au début du WF secondaire pour récupérer les données transmises du WF principal.
Une activité « reply » est automatiquement insérée à la fin du WF secondaire pour retourner les résultats au WF principal.
Insérer une activité « receive » dans le WF principal après l'activité « invoke » correspondante pour recevoir les résultats transmis par le WF secondaire et permettre la progression du WF principal.

**Contraintes**

**R3.5** : Les données d'entrée d'un service secondaire invoqué sont les données d'entrée du service abstrait correspondant dans le processus principal.
**R3.6** : Les données de sortie d'un service secondaire invoqué sont les données de sortie du service abstrait correspondant dans le processus principal.

**Figure V.21.** *Description du patron « Sous-traitance »- PCBS3*

Sur la figure V.21, le partenaire 1 implémente le WF principal et le partenaire 2 fournit son WF secondaire comme un service *S2* global, *S2* peut être un service composite complexe mais du point de vue du partenaire principal, il est abstrait dans une entité élémentaire. Donc, le partenaire 1 invoque le service *S2* fourni par le partenaire 2. L'ensemble des services est *orchestré* au niveau du partenaire principal à l'aide d'une fonction d'orchestration *globale*. En local, les partenaires secondaires implémentent chacun, une fonction d'orchestration locale.

Dans cette architecture, le contrôle d'exécution est *hiérarchisé* car le WF principal gère l'exécution du WFIO global et contrôle l'invocation du (ou des) service(s) externe(s). Le WF principal comporte des invocations à des services « *abstraits* ». Ces derniers sont des services non implémentés localement et doivent être sous-traités auprès de partenaires secondaires. Le niveau d'abstraction requis est de type « sous-processus » pour le WF principal qui doit découper son processus au niveau des services abstraits en sous-processus encapsulés dans des services ; il est de type « processus » pour le WF secondaire qui sera entièrement encapsulé dans un service.

La mise en coopération d'un WF principal et d'un WF secondaire se fait via une opération de type interconnexion de modèles (deux modèles en entrée et deux modèles en sortie avec des liens entre eux), un lien entre les modèles correspond à une *substitution* de l'invocation du service *abstrait* par l'invocation d'un service concret encapsulant le WF fourni par le partenaire secondaire. Pour maintenir un flux de données cohérent dans le processus, une contrainte d'égalité est imposée entre les données d'entrée (resp. de sortie) du service abstrait et les données d'entrée (resp. de sortie) du service concret invoqué

### V.7.3.2 Exemple de processus WFIO selon le patron « Sous-traitance »

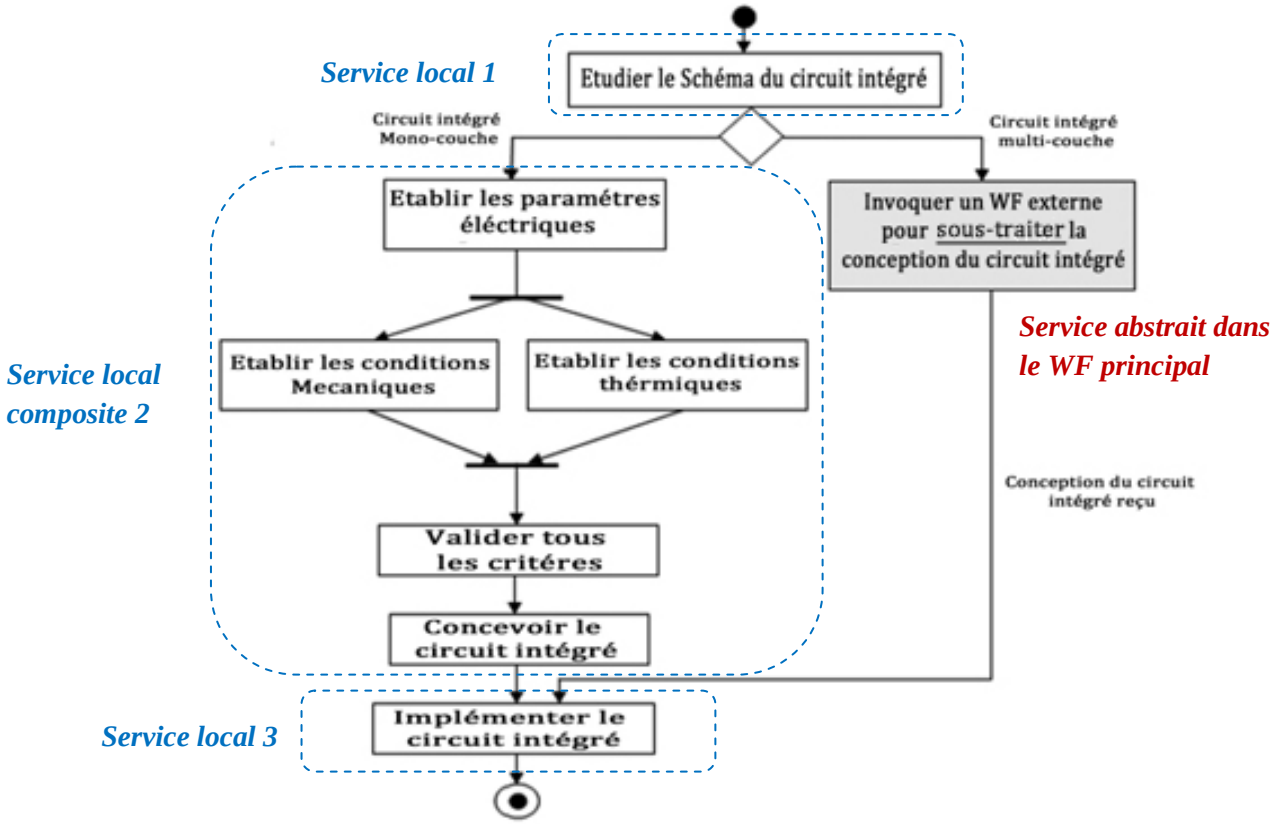



**Figure V.22.** *Exemple d'un WFIO selon le patron « Sous-traitance »*
*Vue du processus Principal*

Le diagramme d'activités UML de la figure V.22 décrit un processus de WFIO destiné à la conception et la réalisation de circuits intégrés (CI) à des clients potentiels. Le processus implique deux partenaires (un partenaire principal et un partenaire secondaire). A la réception d'une commande client, le partenaire principal étudie le schéma du circuit, s'il s'agit d'un circuit monocouche, il est entièrement conçu et mis en œuvre localement. Dans le cas contraire (circuit multicouches), sa conception est sous-traitée à un partenaire externe (secondaire) car le partenaire principal ne dispose pas des ressources et des compétences nécessaires à la conception de ce type de circuits. Le résultat du traitement du partenaire externe est retourné au partenaire principal.

Sur la figure V.22, sont exposées les phases les plus importantes du processus. Les deux WF sont supposés être implémentés selon une architecture SOA. Ainsi, toutes les phases de chaque processus sont considérées comme des services (composites). Le processus du partenaire principal peut être abstrait selon un niveau « sous-processus » et le processus secondaire selon un niveau « processus » (entièrement encapsulé dans un service composite).

### V.7.3.3 Formalisation du patron « Sous-traitance»

Pour la formalisation du patron « Exécution chainée », nous introduisons la définition d'un opérateur de coopération désigné par [CE].

- **Définition V.6 :** Opérateur [SC]

Soient WF1 = < SL1, f1 > et WF2 = < SL2, f2 > où SL1 et SL2 représentent deux ensembles de services locaux à WF1 et WF2 respectivement et f1, f2 sont les fonctions d'orchestration respectives de WF1 et WF2 avec f1 *(Sin1, … Sout1) = Sin1* ***op1*** *S11* ***op2*** *S12 …**opn** Sout1* = **S1**

et f2 *(Sin2, …, Sout2) = Sin2* ***op1*** *S21* ***op2*** *S22 …* ***opm*** *Sout2*= ***S2***

Soit ***Sx*** appartenant à SL1, un service *abstrait* (non implémenté localement) de WF1.

On définit un opérateur **[SC]** pour « **S**ub**C**ontracting » qui construit un WFIO obéissant au patron de coopération « Sous-traitance» par interconnexion de WF1 et WF2, de la manière suivante:

**[SC]** (WF1, (Sx, WF2)) = < S, F > où:

S = SL1 **U** SL2 et F = {$f1$, $f2$} avec

$f1$ = **Sub** (f1, Sx, **S2**) et $f2$ = **Sub** (f2, Sin2, **S'in2)**

S'in2 représente le service d'entrée du WF2 avec un flux de données mis à jour tel que :

Input (S'in2) = Iutput (Sx)

La figure suivante illustre le concept de fonction d'orchestration sur un exemple de « sous-traitance ». Ce WFIO implique deux partenaires, le partenaire 1 implémente un WF composé de services internes S11, S12, S13, S14, S15 et d'un service abstrait Sx, et le partenaire 2 implémente un WF composé de services internes S21, S22, S23.
Le WF du partenaire1 invoque le service externe S2 (substituant le service Sx) du partenaire2 qui devra s'exécuter du point Sin2 à Sout2 ensuite redonnera la main au processus principal pour continuer son exécution.

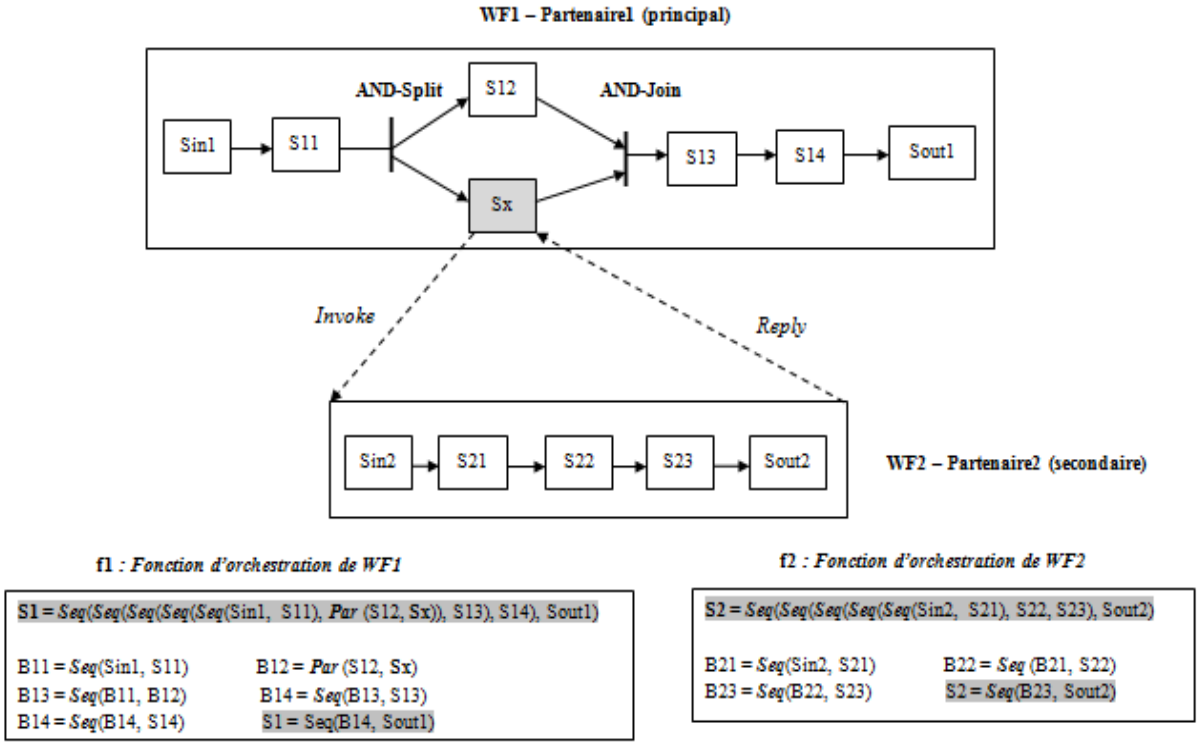



**Figure V.23.** *Illustration de la fonction d'orchestration pour une « Sous-traitance »*

### V.7.4 Le patron « Transfert de cas (resp. Transfert de cas étendu)» - PCBS4 (resp. PCBS5)

Le « *Transfert de cas* » définit une forme de coopération assez répandue dans les échanges B2B, notamment entre partenaires métiers exerçant la même activité et devant satisfaire un grand nombre de clients potentiels, dans les meilleurs délais, d'où l'intérêt de leur alliance. Dans l'architecture *« Transfert de cas »*, les partenaires métiers partagent tous le *même modèle de processus* implémenté au niveau de chacun d'eux. Leur coopération consiste à transférer l'exécution d'une instance de processus d'une localisation à une autre, c'est-à-dire d'un partenaire vers un autre afin de prendre en charge une partie ou la totalité de son exécution. Les règles de transfert (conditions de transfert, source, destination) sont décrites dans la *politique de transfert* définie conjointement par les différents partenaires, au niveau Buildtime.

Une variante du « Transfert de cas » appelée « Transfert de cas étendu » diffère de la première par la possibilité d'implémentions différentes de certaines activités internes, au niveau des partenaires. Cette variante permet aux partenaires de garder un certain degré d'autonomie sur les activités implémentées localement, sans pour autant changer la logique métier du processus global.

A tout moment, une instance de processus se trouve au niveau d'un seul partenaire, le transfert de l'instance doit se faire à partir d'un point stable du processus appelé *point de transfert* et doit prendre en compte les *données d'état* de l'instance afin de maintenir la cohérence du système. Le transfert d'instances se fait pour des raisons d'équilibrage de charge entre les partenaires ou à cause d'une indisponibilité de ressources adéquates ou autre.

Avant de donner la description du patron « Transfert de cas (étendu) », quelques définitions de notions de base (données d'état, point de transfert, politique de transfert) liées à ce patron, s'imposent.

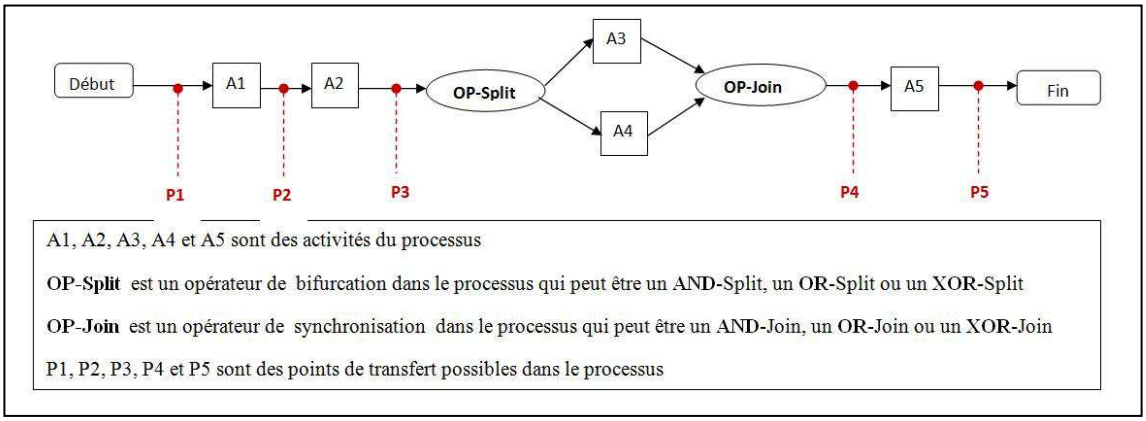
### V.7.4.1 Définitions

- **Définition V.7 :** Données d'état d'une instance

Les données d'état d'une instance de processus sont principalement: (i) les informations générales sur l'instance (identifiant, description, date de déclenchement, …), (ii) l'historique d'exécution (activités correctement terminées, état de l'instance), (iii) l'activité à exécuter directement après le transfert de l'instance et (iv) les données d'entrée pour la suite d'exécution de l'instance.

- **Définition V.8:** Point de transfert

Un point de transfert est un état du processus qui garantit la *cohérence* d'exécution de l'instance si un transfert est effectué d'une localisation à une autre.
En effet, un point de transfert remplit les conditions suivantes : (i) doit se situer avant le début d'une activité ou après la fin d'une activité. (ii) ne doit pas interrompre l'exécution d'une activité. (iii) ne doit pas se situer entre un opérateur de routage (SPLIT) et l'opérateur de synchronisation (JOIN) correspondant. En d'autres termes, si une branche parallèle ou alternative est engagée, le transfert d'instance ne peut s'effectuer qu'après synchronisation.
Un point de transfert particulier peut être le *début du processus*.




- **Définition V.9 :** Politique de transfert

Une politique de transfert est un ensemble de règles définies conjointement au niveau Buildtime, par les partenaires impliqués dans la coopération pour contrôler les transferts d'instances.
Une règle de transfert est rattachée à un point de transfert et est définie par une paire (condition, action) qui signifie : si la condition est vraie alors transférer l'instance vers le partenaire spécifié dans l'action, sinon continuer l'exécution de l'instance au niveau de son site actuel.

Dans la suite, nous énonçons les règles de restructuration d'un processus de WF devant être implémenté selon les patrons « Transfert de cas » ou « Transfert de cas étendu ».

### V.7.4.2 Restructuration du processus WF en services pour le patron « Transfert de cas »

Notre approche pour réaliser le patron « Transfert de cas (étendu) » à base de services repose sur le partitionnement du processus en sous-processus au niveau des points de transfert spécifiés, en vue de les encapsuler dans des services (Boukhedouma et al., 2011a ). Le partitionnement du processus en sous-processus dépend donc des points de transfert choisis par les concepteurs du WFIO, selon la politique de transfert adoptée par les partenaires. Un sous-processus est une partie du processus, délimitée par deux points de transfert ou par le point de début du processus et le premier point de

transfert ou le dernier point de transfert et le point de fin du processus. Ainsi, un même service peut encapsuler une seule activité ou un ensemble d'activités. La figure V.25 montre un schéma de partitionnement du processus et d'encapsulation des sous-processus dans des services. Le processus sera donc découpé en une séquence de services pouvant être invoqués par le processus.

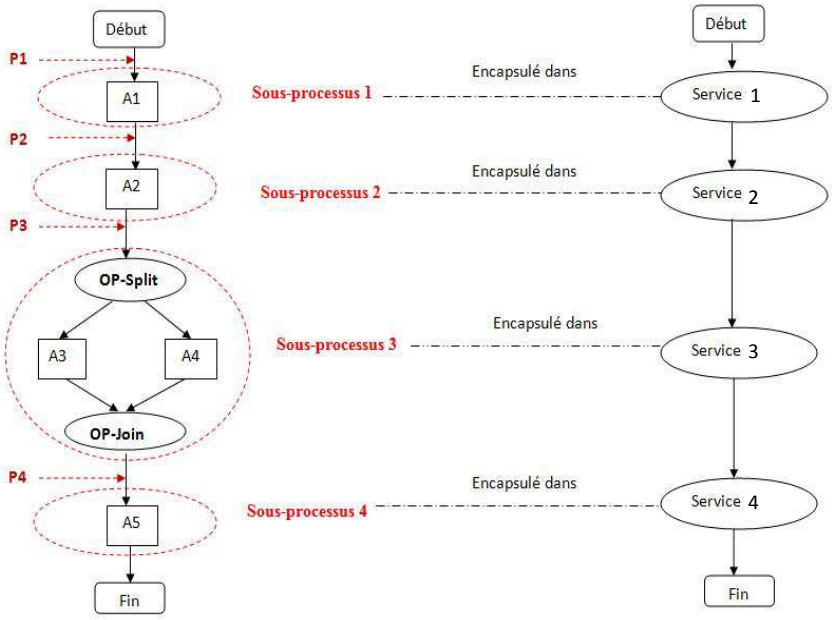



**Figure V.25.** *Partitionnement d'un processus en sous-processus* (Boukhedouma et al., 2011a)

A ce niveau les règles de transfert ne sont pas encore injectées dans le processus, pour ce faire nous proposons d'introduire des blocs alternatifs dans le modèle de processus, au niveau des points de transfert identifiés. Ces blocs correspondent aux actions à entreprendre selon la valeur de la condition de transfert et auront la forme suivante :

```
If (condition) invoke external service
      Else invoke local service
```

Les conditions de transfert peuvent être injectées dans les modèles de processus des deux partenaires et les transferts d'instance peuvent s'effectuer dans les deux sens selon les actions spécifiées dans les règles de transfert (voir figure V.26).

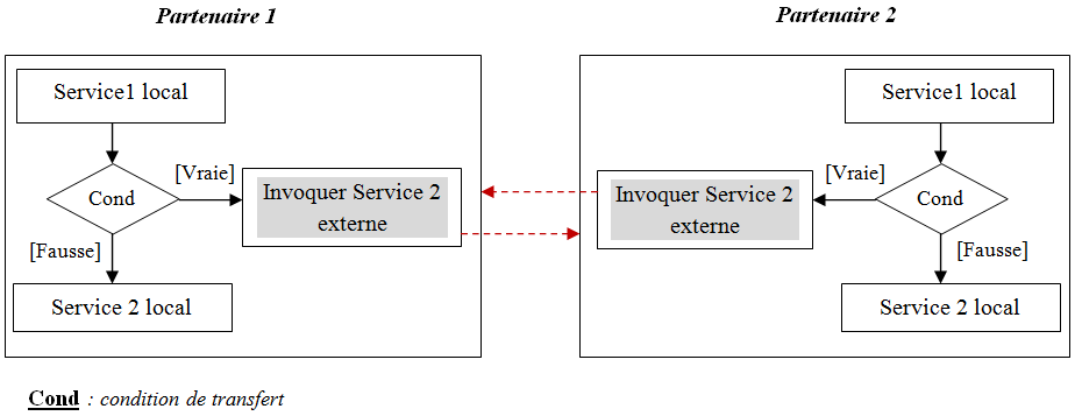



**Figure V.26.** *Partie d'un schéma de WFIO obéissant au « Transfert de cas »*

Notons que le processus de WFIO global et les services qui le composent sont *dupliqués* au niveau de chaque site de partenaire impliqué dans la coopération. De plus, un service externe à un partenaire est un service implémenté au niveau d'un autre partenaire.

La figure V.27 montre une partie du schéma de WF transformé comportant deux activités A1 et A2 avec un point de transfert entre les deux. Les invocations de services se font sous les *conditions* définies dans la *politique de transfert*. Si la condition (cond1 ou cond2) de transfert est vraie, l'instance est transférée à un autre partenaire via l'invocation du service (externe) à exécuter. Dans le cas contraire, le service est exécuté localement.

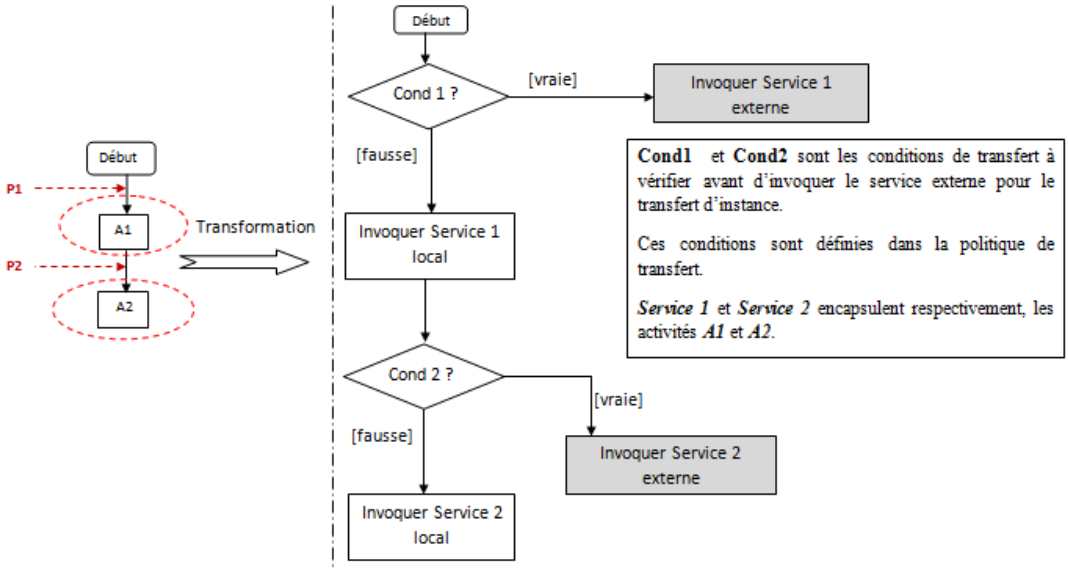



**Figure V.27.** *Restructuration de WF à base d'activités en WF à base de services pour le patron « Transfert de cas »*

### V.7.4.3 Conceptualisation du patron « Transfert de cas (étendu) »

Dans le patron « Transfert de cas (étendu) », le niveau d'abstraction requis est de granularité intermédiaire « sous-processus ». Ce patron est réalisé par la définition d'une fonction d'orchestration locale au niveau de chaque partenaire décrivant le flux de contrôle entre les services locaux et les services externes invoqués. Dans ce cas, la fonction d'orchestration peut être décrite par une séquence d'invocations directes ou alternatives entre des services locaux et externes.

Le contrôle d'exécution peut être complètement décentralisé ou mixte ; il est décentralisé si chaque système de partenaire gère directement ses interactions avec les autres partenaires. Par contre, le contrôle est mixte si les interactions entre les processus des partenaires se font indirectement, via un *coordinateur central*. Le premier mode s'avère approprié pour une politique de transfert simple avec des règles de transfert déterministes alors que le second mode est applicable dans le cas d'une politique de transfert complexe avec des règles de transfert non déterministes. Ainsi, lorsque le coordinateur central reçoit une requête de transfert d'instance, il doit appliquer un algorithme de sélection pour déterminer le partenaire qui peut recevoir l'instance, ceci peut être utile pour des besoins d'équilibrage de charges dans le système ; cette architecture est explicitée dans l'un de nos travaux (Boukhedouma et al., 2012a ).

Référence-Patron: **PCBS4 (resp. PCBS5) ou SBCP4 (resp. SBCP5)**
Nom: Patron "Transfert de cas (étendu)" ou "(Extended) Case Transfer" Pattern
Coopération: Partager l'exécution des instances de processus, conformément au même modèle de WF, en les transférant d'un partenaire à l'autre, selon un ensemble de règles de transfert.
Structure : Un ensemble de services internes et externes orchestrés par une fonction d'orchestration globale implémentée au niveau de chaque partenaire.
Contrôle: Décentralisé/ Mixte
Type d'interaction: Synchrone ou One-Way
Utilisation dans la pratique: Assez répandu entre partenaires métiers exerçant la même activité avec des compétences et des ressources complémentaires.
Exemple : Processus de la « supply chain management » impliquant différents partenaires de même profil.

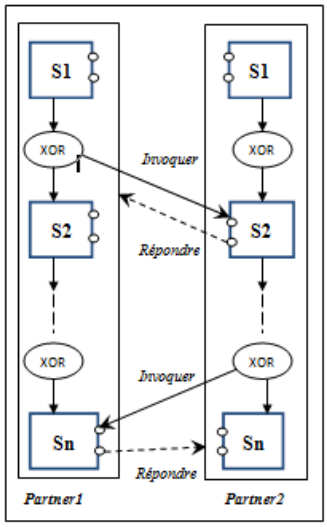
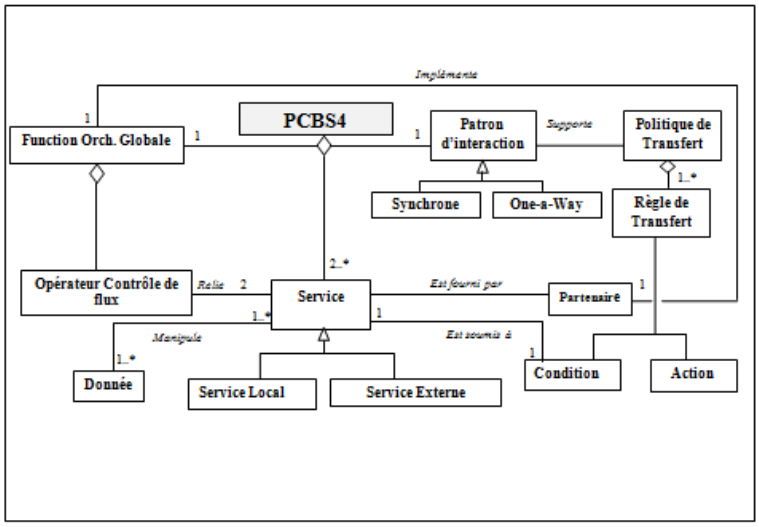



Schéma du Patron "Transfert de cas"




Meta-modèle du Patron "Transfert de cas"

**Règle de mapping**

**R4.1:** Encapsuler chaque activité d'un WF dans un service.

**Règles d'abstraction**

**R4.2:** Structurer le processus WF en sous-processus, selon la définition suivante :

Un sous-processus dans un processus WF est délimité par : (i) deux points de transfert ou (ii) le point de début du processus et le premier point de transfert ou (iii) le dernier point de transfert et le point de fin du processus.

**R4.3 :** Encapsuler chaque sous-processus dans un service

⇨ Abstraction de niveau « sous-processus »

**Règles , caractéristiques et contraintes de l'interconnexion**

**R4.4:** Spécifier une fonction d'orchestration globale au niveau de chaque partenaire (contrôle de flux entre les services locaux et externes).

**Caractéristiques de l'interconnexion**

Opération : Interconnexion de modèles

Nombre de modèles en entrée/sortie : plusieurs/plusieurs

Liens entre les modèles : Transformer le modèle de processus en des invocations de services locaux et externes selon les conditions de transfert

Les règles de transfert sont injectées dans les différents processus WF en insérant des activités alternatives d'invocation de services locaux et externes, conformément au schéma suivant:

```
If (condition) invoke external service
        Else invoke local service
```

**Contrainte**

**C4.1** : Les données d'entrée d'un service externe invoqué sont les données de sortie du service qui précède l'action alternative correspondante dans le processus invocateur.

**Figure V.28.** *Description du patron « Transfert de cas (étendu)»- PCBS4(PCBS5)*

Le patron « Transfert de cas » repose donc, sur une opération d'interconnexion de modèles qui se fait via des opérations alternatives d'invocation de services locaux ou de services externes (fournis par un autre partenaire). Une contrainte est imposée sur les données de sortie du service qui précède l'action alternative et les données d'entrée du service externe invoqué, afin de maintenir un flux de données cohérent dans le processus global.

### V.7.4.4 Exemple d'un processus de WFIO selon le patron « Transfert de cas»

La figure V.29 ci-dessous décrit un processus de WFIO obéissant au patron « Transfert de cas». Le processus est lié au domaine de la production pharmaceutique et fait intervenir deux partenaires : *partenaire 1* et *partenaire 2* qui implémentent le même modèle de processus. Le WF comporte un seul point de transfert correspondant à la condition de transfert « *absence de licence de production* ». En effet, dès la réception d'une demande de production de médicaments, si le partenaire ne dispose pas de la licence de production, l'instance est transférée au deuxième partenaire qui implémente le même modèle (sur la figure, on représente seulement un partenaire).

**Remarque :** Toutes les phases du processus qui se trouvent après le point de transfert peuvent être encapsulées dans un seul service composite.

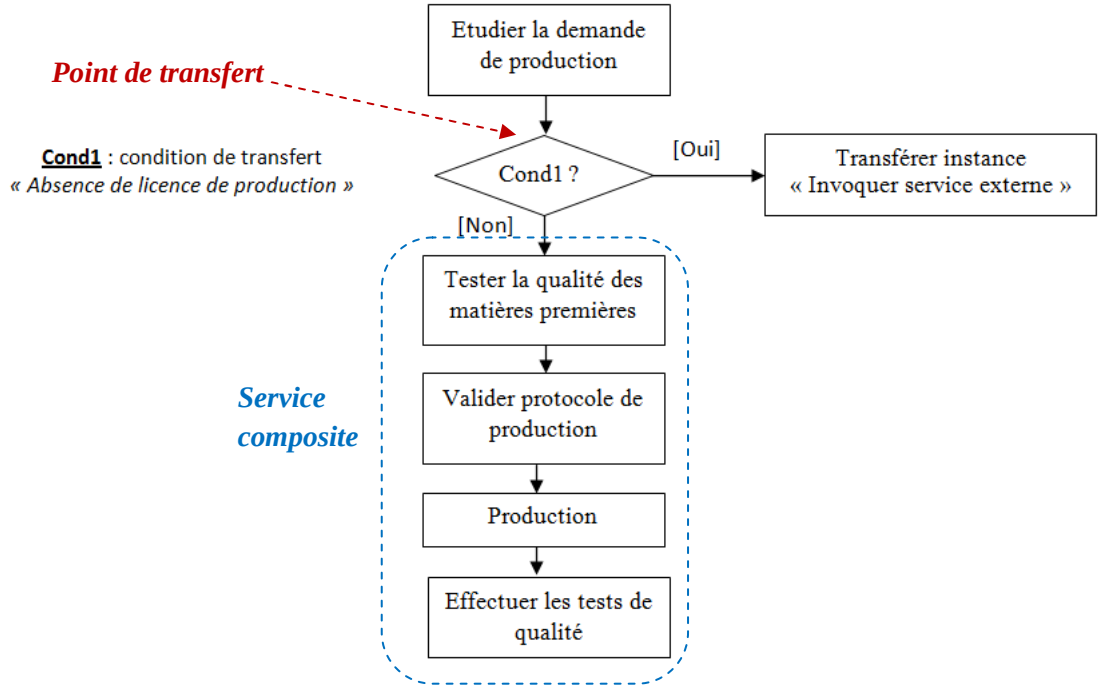



**Figure V.29.** *Exemple d'un WFIO obéissant au patron « Transfert de cas »*

### V.7.4.5 Formalisation du patron « Transfert de cas»

Avant d'énoncer la définition de l'opérateur **[CT]** pour la formalisation du patron « Transfert de cas (étendu) », nous énonçons la définition d'une règle de transfert.

- **Définition V.10 :** Règle de transfert

Soit WF = < SL, f > un WF lié à un partenaire métier.

Une règle de transfert R dans WF, est définie par la paire (Si, (Ci, Ai)) où Si ∈ SL est un service de référence correspondant à la localisation du point de transfert dans le processus (Si est le service avant ou après le point de transfert), Ci est la condition de transfert et Ai l'action de transfert.

- **Définition V.11 :** Opérateur **[CT]**

Soient WF1 = < SL1, f1 > et WF2 = < SL2, f2 > où SL1 et SL2 représentent deux ensembles de services locaux à WF1 et WF2 respectivement et f1, f2 sont les fonctions d'orchestration respectives de WF1 et WF2.

avec f1 *(Sin1,…, Sout1) = Sin1* ***op1*** *S11* ***op2*** *S12 …****opn*** *Sout1* = **S1**

et f2 (*Sin2,… Sout2*) = *Sin2* ***op1*** *S21* ***op2*** *S22 …****opm*** *Sout2*= ***S2.***

Et soit un ensemble de règles de transfert ℝ = {(Si, Ri) où Ri = (Ci, Ai) , 1<= i<=n}.

On définit un opérateur **[CT]** pour « Case Transfer » qui construit un WFIO obéissant au patron de coopération « Transfert de cas» par interconnexion de WF1 et WF2, en utilisant l'ensemble ℝ des règles de transfert, de la manière suivante:

**[CT ]** (WF1, WF2, ℝ) = < S, F > où:

S = SL1 **U** SL2 et F = {$\mathcal{f}$1, $\mathcal{f}$2} avec

$\mathcal{f}$1 est obtenue à partir de f1 par insertion de blocs alternatifs de la manière suivante :

Pour tout (Si, Ri) ∈ ℝ et Si ∈ SL1 faire

**Ins (f1, Bj, Si, mode)**

$\mathcal{f}$2 est obtenue à partir de f2 par insertion de blocs alternatifs de la manière suivante :

Pour tout (Si, Ri) ∈ ℝ et Si ∈ SL2 faire

**Ins (f2, Bj, Si, mode)**

**Bj =**
```
If (Ci)invoke external service Sj
                    Else invoke local service Sj
```

Si représente le service de référence pour la localisation du point de transfert.

**mode** peut être "Avant " ou " Après " et définit la localisation du point de transfert par rapport au service *Si* de référence.

Remarquons l'utilisation de l'opérateur **Ins** que nous définissons pour l'insertion d'un service dans une fonction d'orchestration « Avant » ou « Après » un autre service. Les modes « Alt » et « Par » pour l'insertion en parallèle ou en alternatif, bien que figurant dans la définition, ne sont pas utilisés dans ce cas.

- **Définition V.12 :** L'opérateur « **Ins »**

Soit une fonction d'orchestration f et soient un service Si référencé dans f et un bloc Bj (ou un service Sj). L'opérateur **Ins** s'applique à une fonction d'orchestration f et permet d'ajouter le bloc Bj (ou le service Sj) avant ou après le service Si pour obtenir une nouvelle fonction d'orchestration f', en spécifiant le point d'insertion (« mode ») par rapport au service de référence Si.

On écrira : **f'** = **Ins (f, Bj, Si, mode)** mode ∈ {"Avant", "Après", "Alt", "Par"}

En d'autres termes f' est obtenue à partir de f en insérant le bloc Bj avant (ou après) le service de référence Si.

### V.7.5 Le patron « Faiblement couplé » - PCBS6

Un WFIO faiblement couplé repose sur un protocole d'interaction asynchrone entre les WF impliqués dans la coopération. Par conséquent, la structuration d'un WFIO faiblement couplé en WFIO à base de services repose sur les points d'interaction entre les fragments de WF. Par suite, nous proposons d'abord d'isoler les activités d'interaction dans le WF de chaque partenaire afin de les encapsuler dans des *services d'interaction*. Après cela, nous proposons de structurer le processus WF de chaque partenaire en sous-processus qui seront encapsulés dans des services locaux. Dans la suite, nous énonçons les règles qui permettent de supporter cette restructuration.

#### V.7.5.1 Règles de restructuration des WF en services pour le patron « Faiblement couplé »

Le découpage du processus en activités d'interaction et en sous-processus se fait en utilisant des règles que nous décrivons dans la suite. Ces règles diffèrent selon que le bloc à restructurer soit séquentiel ou contient des activités alternatives ou parallèles. Pour rester conforme avec la description du patron (Figure V.32), nous commençons la numérotation des règles à partir de R6.2 puisque la règle R6.1 est réservée à l'opération de mapping (activité - service élémentaire).

- **Restructuration d'un bloc séquentiel en services**

Pour restructurer un bloc séquentiel en services, nous proposons les règles suivantes (Boukhedouma et al., 2013b ):

- **Règle R6.2** : Isoler les activités d'interaction et encapsuler chacune d'elles dans un service d'interaction « Service-send » ou « Service-receive ».

- **Règle R6.3** : Découper le bloc séquentiel en sous-processus, au niveau des points d'interaction. Un sous-processus sera composé d'une ou de plusieurs activités et délimité par : (i) soit deux points d'interaction consécutifs, (ii) soit le point début du processus et le premier point d'interaction, (iii) soit le dernier point d'interaction et le point de fin du processus.

- **Règle 6.4** : Encapsuler chaque sous-processus dans un service

La figure V.30 montre une transformation d'un schéma de processus en utilisant les règles susdécrites.

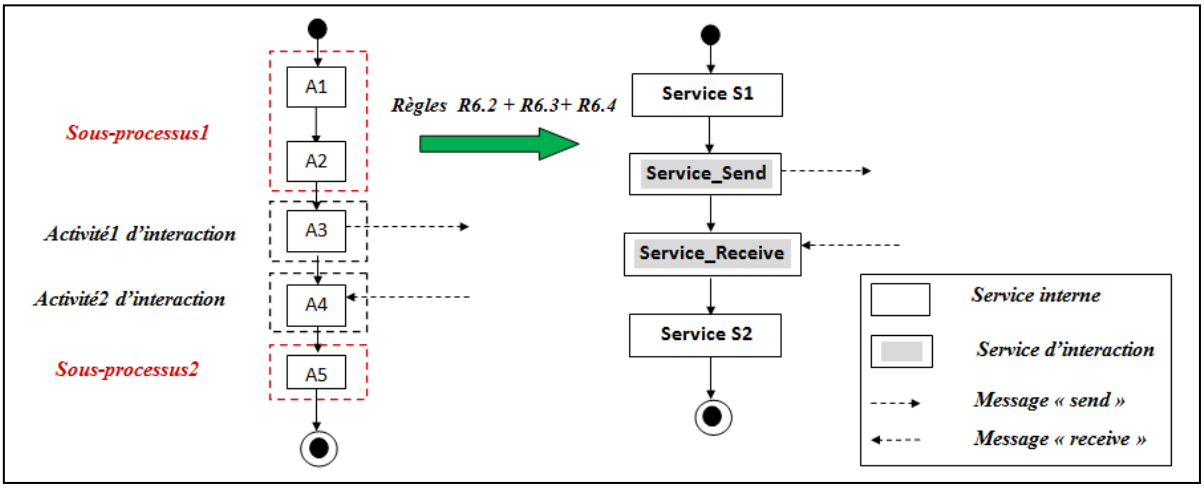



**Figure V.30.** *Restructuration d'un bloc séquentiel en services* (Boukhedouma et al., 2013b)

- **Restructuration d'un bloc parallèle ou alternatif en services**

Pour la restructuration d'un bloc parallèle ou alternatif en services, deux possibilités sont envisageables:

**Règle R6.5**

1. Si le bloc ne contient aucune activité d'interaction, il est considéré comme une activité unique.
2. Si le bloc contient au moins une activité d'interaction:
   - Insérer un point d'interaction fictif au niveau du point OP-Split et au niveau du point OP-Join correspondant et découper le processus au niveau de ces deux points.
   - Appliquer la règle R6.3 sur la branche contenant les activités d'interaction.

**Règle R6.4** : Encapsuler chaque sous-processus dans un service interne.

La figure V.31 montre une transformation d'un schéma de processus en utilisant les règles su-décrites

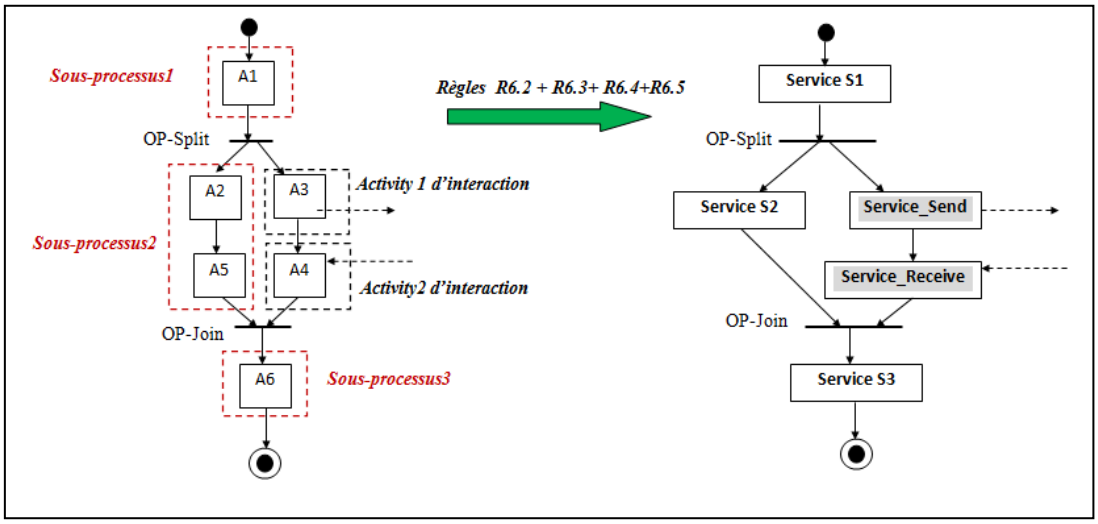



**Figure V.31.** *Restructuration d'un bloc alternatif ou parallèle en services*
(Boukhedouma et al., 2013b )

### V.7.5.2 Conceptualisation du patron « Faiblement couplé »

L'interconnexion des WF conformément au patron « Faiblement couplé » repose d'abord sur la transformation de chaque WF en une orchestration de services internes et services d'interaction, en utilisant les règles précédemment décrites.

Le niveau d'abstraction requis dans ce cas est de type « sous-processus », le patron faiblement couplé est réalisé par la définition d'une fonction d'orchestration locale au niveau de chaque partenaire décrivant le flux de contrôle entre les services locaux et les services d'interaction, comme le montre le méta-modèle de la figure V.32. Le contrôle d'exécution étant décentralisé et les interactions sont asynchrones, de type « One-Way » ou « Two-way »

Référence-Patron: **PCBS6** ou **SBCP6**
Nom: Patron "Faiblement couplé" ou "Loosely coupled" Pattern
Coopération: Echange de données entre fragments de WF conformément à un protocole de communication
Structure : un ensemble de fonctions d'orchestration locales orchestrant des services locaux et des services d'interaction.
Contrôle: Décentralisé
Type d'interaction: Asynchrone
Utilisation dans la pratique: Répandu entre partenaires métiers qui s'échangent des données entre eux pour réaliser un WFIO global.
Exemples: Processus de production, processus de e-commerce impliquant des clients, des fournisseurs, des producteurs et des organismes financiers.

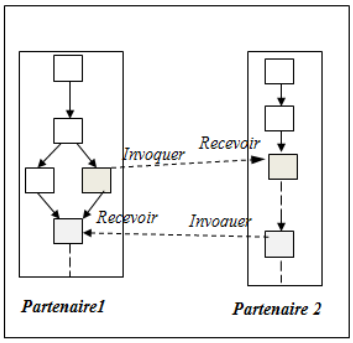
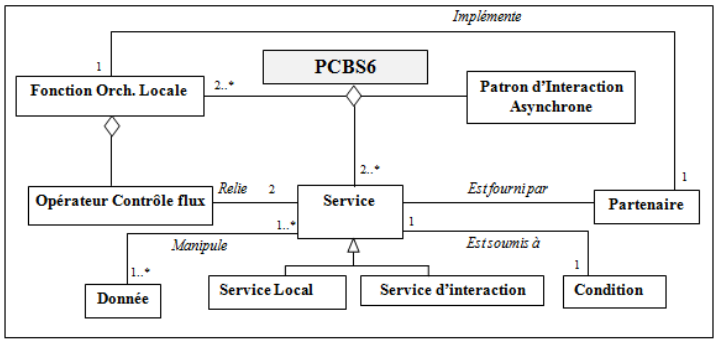



Schéma du Patron "Faiblement couplé"




Meta-modèle du Patron "Faiblement couplé"

**Règles de mapping**

**R6.1:** Encapsuler chaque activité interne d'un WF dans un service élémentaire.
**R6.2**: Isoler chaque activité d'interaction et l'encapsuler dans un service d'interaction "Service-send" ou "Service-receive".

**Règles d'abstraction**

Pour le découpage du processus en sous-processus, on définit les règles R6.3 et R6.4

**R6.3**: sur une branche séquentielle (voir figure V.30)
Un sous-processus dans un processus WF est délimité par: (i) deux activités d'interaction ou (ii) le point de début et la première activité d'interaction ou (iii) par le point de fin et le dernière activité d'interaction.
**R6.4:** Encapsuler chaque sous-processus dans un service interne.
**R6.5**: dans un bloc alternatif ou parallèle (voir figure V.31)
Deux possibilités sont envisageables:
(1) Si le bloc ne contient aucune activité d'interaction, il est considéré comme une activité unique.
(2) Si le bloc contient au moins une activité d'interaction:
- Insérer un point d'interaction fictif au niveau du point OP-Split et au niveau du point OP-Join correspondant et découper le processus au niveau de ces deux points.
- Appliquer la règle R6.3 sur la branche contenant les activités d'interaction.

⇒**Abstraction de niveau « sous-processus »**

**Règles , caractéristiques et contraintes de l'interconnexion**

**R6.5:** Spécifier une fonction d'orchestration locale au niveau de chaque partenaire (contrôle de flux entre les services locaux et les services d'interaction).

**Caractéristiques de l'interconnexion**

Opération : Interconnexion de modèles
Nombre de modèles en entrée/sorties : plusieurs/plusieurs
Liens entre les modèles : Les services d'interaction « Service-send » et « Service-receive »

**Contrainte**

**C6.1** : Les données d'entrée d'un service « Service-receive » sont les données de sortie du service « Service-send » correspondant.

**Figure V.32.** *Description du patron de coopération « Faiblement couplé » - PCBS6*

L'opération qui construit le WFIO est de type interconnexion de modèles qui se fait via des opérations d'invocation de services d'interaction « Service-send » et « Service-receive ». Une contrainte est imposée sur les données de sortie du service « Service-send » et les données d'entrée du service « Service-receive » correspondant, afin de maintenir un flux de données cohérent dans le processus global.

### V.7.5.3 Exemple d'un processus de WFIO selon le patron « Faiblement couplé»

La figure V.33 montre le diagramme d'activité décrivant un exemple de processus WFIO obéissant au patron « Faiblement couplé ». Le processus consiste à gérer des commandes de clients pour un certain type de produits et implique deux partenaires : un client et un fournisseur. Le client envoie sa commande au fournisseur qui vérifie la disponibilité des produits. Si la quantité du produit est suffisante pour satisfaire la commande, le client est notifié par un message « Préparer Commande », sinon il est notifié par un message « Non disponibilité ». Dans le cas où la quantité de produit est disponible, le fournisseur prépare la livraison et la facture correspondante et les envoie au client. Ce dernier réceptionne la livraison et la facture, effectue le paiement et envoie le reçu de paiement au fournisseur. Les services internes sont mis en évidence sur le schéma du processus, les autres services sont des services d'interaction du type « envoyer », « notifier », « recevoir » supportant les échanges de messages qui sont schématisés par des flèches en pointillés.

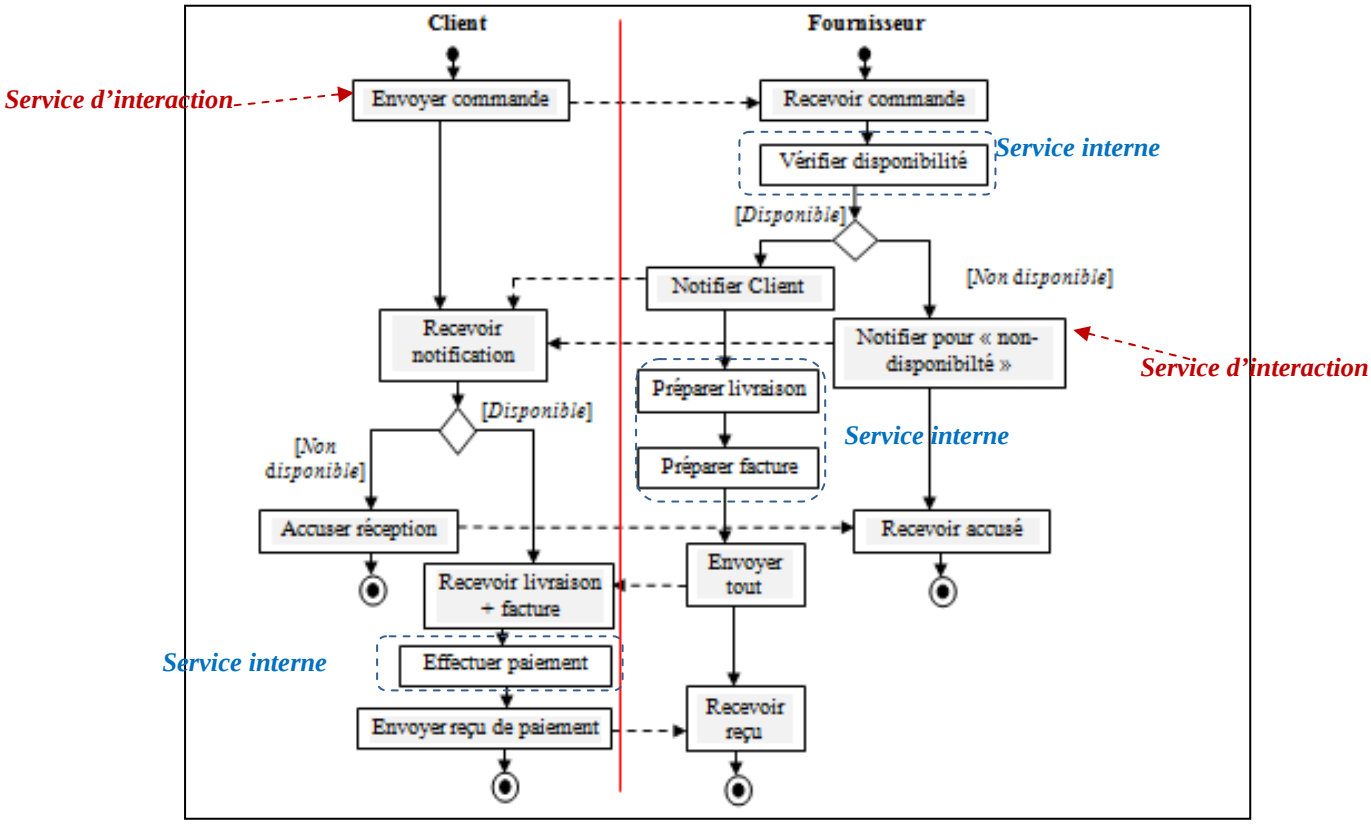



**Figure V.33.** *Exemple d'un WFIO « Gestion de Commandes Clients » selon le patron « Faiblement couplé »*

### V.7.5.4 Formalisation du patron « Faiblement couplé»

Avant d'énoncer la définition de l'opérateur **[LC]** pour la formalisation du patron « Faiblement couplé », nous énonçons la définition du concept de point d'interaction.

- **Définition V.13 -** Point d'interaction

Soit WF = < SL, f > un WF lié à un partenaire métier.

Un point d'interaction Pi dans WF est défini par la paire (Si, Sj), où Si ∈ SL est un service de référence correspondant à la localisation du point d'interaction dans le processus. Sj est un service d'interaction de type « Service-send » ou « Service-receive» devant être inséré « avant » ou « après » le service Si.

- **Définition V.14 -** L'opérateur **[LC]**

Soient WF1 = < SL1, f1 > et WF2 = < SL2, f2 > où SL1 et SL2 représentent deux ensembles de services locaux à WF1 et WF2 respectivement et f1, f2 sont les fonctions d'orchestrations respectives de WF1 et WF2
avec f1 *(Sin1, …, Sout1) = Sin1* ***op1*** *S11* ***op2*** *S12 …* ***opn*** *Sout1* = **S1**
et f2 *(Sin2, …, Sout2) = Sin2* ***op1*** *S21* ***op2*** *S22 …* ***opm*** *Sout2=* ***S2***

Et soient deux ensembles de points d'interaction **I1** et **I2** rattachés respectivement à WF1 et WF2 avec **I1** = {(S1i, S1j), 1<= i <= n} et **I2** = {(S2i, S2j), 1<= i <= n}.

On définit un opérateur **[LC]** pour « Loosely Coupled » qui construit un WFIO obéissant au patron de coopération « Faiblement couplé» par interconnexion de WF1 et WF2, de la manière suivante:

**(LC )** ((WF1, **I1**), (WF2, **I2**)) = **< S, F >** où:
S = SL1 **U** SL2 et F = {$\mathcal{f}$1, $\mathcal{f}$2} avec

$\mathcal{f}$1 est obtenue à partir de f1 par insertion de services d'interaction de la manière suivante :

Pour tout (S1i, S1j) appartenant à **I1** faire

**Ins (f1, S1i, S1j, mode)**

$\mathcal{f}$2 est obtenue à partir de f2 par insertion de services d'interaction de la manière suivante :

Pour tout (S2i, S2j) appartenant à **I2** faire

**Ins (f2, S2i, S2j, mode)**

**mode ∈** {"Avant", "Après", "Par" , "Alt"}

La figure V.34 illustre le concept de fonction d'orchestration sur un schéma de WFIO obéissant au patron « Faiblement couplé ». Le processus implique deux partenaires, le partenaire 1 implémente un WF composé de services internes S11 et S13 et des services d'interaction S12, S14 et S15 et le partenaire 2 implémente un WF composé de services internes S23 et S25 et de services d'interaction S21, S22 et S24. Les activités d'interaction correspondent à l'activité « invoke » d'un partenaire et l'activité « receive » de l'autre partenaire.

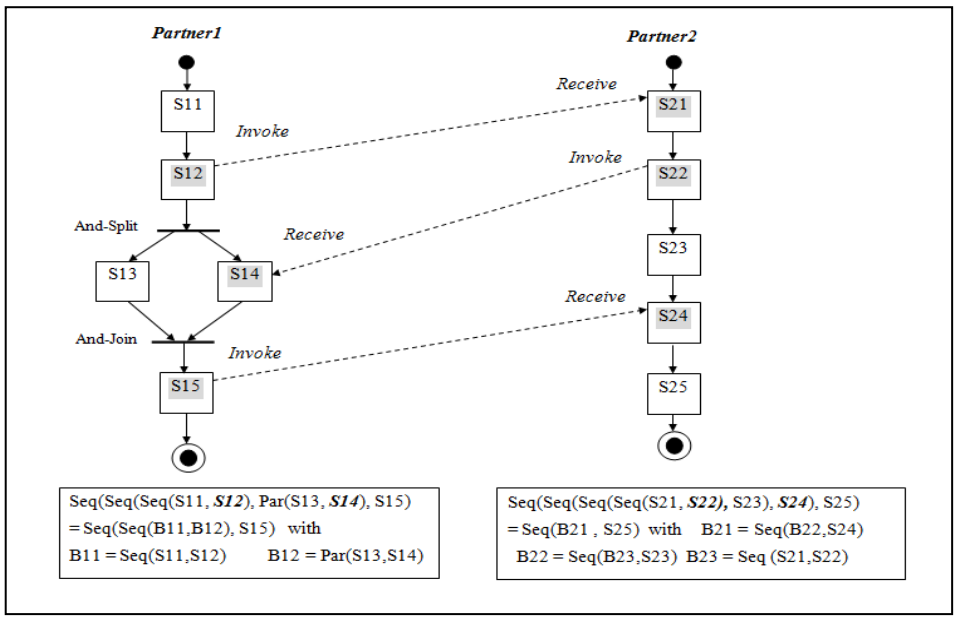



**Figure V.34.** *Illustration de la fonction d'orchestration pour le patron "Faiblement couple"*

### V.7.6 Récapitulatif des patrons de coopération à base de services

Le tableau V.2 suivant résume les différents aspects de définition de processus WFIO obéissant aux patrons de coopération précédemment décrits. L'aspect fonctionnel exprime le découpage du processus en services (internes, externes ou d'interaction), l'aspect comportemental décrit le contrôle de flux entre les services des partenaires qui peut être exprimé via une fonction d'orchestration globale et/ou un ensemble de fonctions d'orchestration locales. L'aspect interactionnel spécifie le patron d'interaction utilisé pour réaliser la communication entre les services des différents partenaires. L'aspect organisationnel définit le rôle de chaque partenaire dans le WF global, un partenaire peut jouer le rôle d'orchestrateur central ou le rôle d'un simple participant dans le processus. L'aspect architectural identifie les sites de partenaires qui jouent le rôle de serveur (fournisseur de service) et ceux qui jouent le rôle de client (demandeur de service) ou les deux à la fois.

**TAB V.2.** *Aspects de définition de WFIO selon les patrons de coopération*

| Patron de coopération à base de services | Aspect fonctionnel | Aspect comportemental | Aspect interactionnel | Aspect organisationnel | Aspect architectural |
|---|---|---|---|---|---|
| **PCBS1- SBCP1**<br>**« Partage de charge »**<br>"Capacity Sharing" | Ensemble de services locaux (éventuellement) et externes | Fonction d'orchestration globale | Patron d'interaction synchrone "invoke"-> "receive"-> "reply" | Un partenaire (ou un tiers externe) joue le rôle d'orchestrateur | Un site client (ou client/serveur éventuellement) et un ensemble de sites serveurs |
| **PCBS2- SBCP2**<br>**« Exécution chainée »**<br>"Chained Execution" | Ensemble de services locaux et externes | Un ensemble de fonctions d'orchestration locales | Patron asynchrone one-way "invoke"-> "receive"<br>ou<br>patron synchrone "invoke"-> "receive"-> "reply" | Tous les partenaires jouent le rôle de participants | Tous les sites de partenaires sont client/serveur excepté le dernier qui est seulement serveur |
| **PCBS3- SBCP3**<br>**« Sous-traitance »**<br>"Subcontracting" | Ensemble de services locaux et externes | Une fonction d'orchestration globale et un ensemble de fonctions d'orchestration locales | Patron synchrone "invoke"-> "receive"-> "reply" | Le partenaire principal joue le rôle d'orchestrateur et les partenaires secondaires sont des participants | Le site du partenaire principal est client/serveur et les sites des partenaires secondaires sont seulement serveurs |
| **PCBS4- PCBS5**<br>**SBCP4 - SBCP5**<br>**« Transfert de cas (étendu) »**<br>"(Extended) Case Transfer" | Ensemble de services locaux et externes | Une fonction d'orchestration globale sur chaque site de partenaire | Patron asynchrone one way "invoke"-> "receive"<br>ou<br>Patron synchrone "invoke"-> "receive"-> "reply" | Tous les partenaires jouent le rôle de participants | Tous les sites de partenaires sont client/serveur |
| **PCBS6- SBCP6**<br>**« Faiblement couplé »**<br>"Loosely Coupled" | Ensemble de services locaux et d'interaction | Un ensemble de fonctions d'orchestration locales | Patron asynchrone "invoke"-> "receive"-> "invoke" | Tous les partenaires jouent le rôle de participants | Tous les sites de partenaires sont client/serveur |

## Synthèse

Dans ce chapitre, nous nous sommes focalisés sur la première partie de notre contribution qui consiste à décrire les patrons de coopération à base de services, que nous proposons en vue de supporter la coopération entre des processus WF, selon les schémas de coopération définis dans (Van der Aalst, 99 ), (Van der Aalst, 2000 ). Nous avons commencé par établir une correspondance entre les concepts décrivant un WF à base d'activités et les concepts de définition d'un WF à base de services. Nous avons ensuite, présenté les grandes lignes de notre démarche de restructuration et d'interconnexion de WF qui s'articule autour de trois phases principales : une phase de mapping, une phase d'abstraction et une phase d'interconnexion. La phase de mapping utilise la correspondance de concepts (activité, service élémentaire). Nous avons proposé pour chaque patron, d'effectuer un mapping de premier niveau (fine granularité) afin de pouvoir restructurer aisément le processus en services (composants ou orchestrateur) en cas de besoin.

La phase d'abstraction est nécessaire pour maintenir le maximum d'autonomie et de savoir-faire sur un processus local et pour exhiber uniquement les détails nécessaires à l'interconnexion, elle repose sur trois niveaux, le plus fin étant le niveau « activité » ou « service élémentaire », le niveau intermédiaire étant le « sous-processus » ou le « service composant » et enfin le plus haut niveau d'abstraction est défini par la granularité « processus » ou « service orchestrateur ». Dans un scénario de coopération, l'abstraction requise pour un processus WF dépend de l'architecture de WFIO à réaliser. En effet, pour chaque schéma de coopération proposé, nous avons spécifié le niveau d'abstraction requis et les règles qui permettent de l'obtenir, notamment dans les architectures « Transfert de cas » et « Faiblement couplée » pour lesquelles, nous avons établi des règles de restructuration des modèles de processus.

La phase d'interconnexion représente l'essentiel de ce chapitre et a consisté en une conceptualisation et une formalisation des différents patrons de coopération. La conceptualisation d'un patron repose sur un méta-modèle de définition de WFIO obéissant au patron en question que nous avons obtenu à chaque fois par adaptation du méta-modèle générique préalablement établi. La conceptualisation consiste aussi à spécifier les contraintes sur le flux de données dans le processus global, en plus des caractéristiques de l'opération d'interconnexion (composition proprement dite ou interconnexion de modèles) pour chaque patron de coopération. Le méta-modèle de définition d'un PCBS fait ressortir trois dimensions principales : la distribution des services, le contrôle d'exécution et la structure d'interaction. Le contrôle d'exécution repose sur le concept de fonction d'orchestration que nous avons aussi utilisé pour proposer des définitions formelles de WF et de WFIO à base de services et pour formaliser les patrons de coopération proposés, en définissant un opérateur de coopération spécifique à chaque patron. Pour les illustrations de concepts dans ce chapitre, nous avons présenté des exemples de WFIO spécifiques à chaque patron de coopération.

Côté implémentation, nous nous sommes focalisés sur la phase d'interconnexion de WF, selon les schémas de coopération considérés. Notre framework de coopération est décrit dans le chapitre VII de ce document. Les phases de mapping et d'abstraction n'ont pas été implémentées puisque notre objectif principal, en plus de l'interconnexion des modèles de WF, est de nous pencher sur les aspects de flexibilité (adaptabilité, évolutivité, réutilisabilité) des modèles de WFIO à base de services.

Dans le prochain chapitre, nous continuons sur les aspects conceptuels de notre contribution où nous nous penchons sur la question de flexibilité des modèles de WFIO.

# CHAPITRE VI

# Notre Approche Support de la Flexibilité des Modèles de WFIO

## Introduction

Dans le chapitre précédent, nous nous sommes focalisés sur la question d'interconnexion de processus WF fournis par différents partenaires métiers, pour former un processus de WFIO global, modélisant une coopération planifiée qui obéit à un patron de coopération clairement défini. Notre principal objectif était d'obtenir des modèles de WFIO suffisamment flexibles pour supporter différents changements pouvant survenir dans le processus global. Pour cela, nous avons opté pour une approche à base de services qui offre un certain degré de flexibilité aux modèles de processus en question. Dans le chapitre IV de ce document, nous avons rappelé les principales situations de changement auxquelles pourraient être confrontées les entreprises, les obligeant de suite à réajuster leurs processus métiers dans les meilleurs délais et à moindre coût. Ces situations de changement peuvent être classées dans un contexte de CPI (*Countinous Process Improvement*) ou de BPR (*Business Process Reingineering*). De plus, nous avons identifié plusieurs catégories d'approches support de la flexibilité des WF, entre autres les approches à base de *règles*, les approches à base de *contraintes*, les approches à base de *cas*, les approches à base *d'aspects* et les approches à base de *patrons* dont la nôtre.

Dans notre contexte de travail, nous distinguons trois aspects de la flexibilité des modèles de WFIO qui sont l'*adaptabilité*, l'*évolutivité* et la *réutilisabilité*. De manière informelle et intuitive, l'adaptabilité est la capacité d'un modèle à supporter des changements sans perturber l'aspect de coopération et sans affecter la fonctionnalité globale du WFIO ; l'évolutivité est la capacité du modèle à supporter une expansion de la fonctionnalité globale du processus et/ou une expansion de la coopération et la réutilisabilité est la possibilité d'intégrer (utiliser) un modèle de WFIO dans un autre afin de définir des modèles de WFIO plus complexes par réutilisation de modèles de WFIO existants. Nous verrons que les aspects d'évolution et de réutilisation sont étroitement liés l'un à l'autre dans le sens où une expansion de la coopération est parfois réalisée par réutilisation de modèles de WFIO existants. En effet, dans notre cas, la réutilisation est achevée pour une évolution des modèles de WFIO. En d'autres termes, on parle d'*évolution par réutilisation* ou de *réutilisation pour l'évolution* des modèles de processus.

Dans ce chapitre, nous décrivons notre approche support de la flexibilité des modèles WFIO qui est une approche à base de patrons. En effet, nous proposons un ensemble de patrons d'adaptation que nous classons en trois catégories selon les trois dimensions définissant un patron de coopération à base de services (PCBS) : les patrons d'adaptation de services, les patrons d'adaptation de contrôle de flux et les patrons d'adaptation des interactions. Les premiers affectent les aspects fonctionnel et comportemental du processus, les seconds affectent l'aspect comportemental et les troisièmes affectent l'aspect interactionnel du processus. Il faut noter que les deux premières catégories de patrons s'appliquent de la même manière à un modèle de WFIO obéissant indifféremment à l'un des PCBS décrits dans le chapitre V. La troisième catégorie (i.e les patrons d'adaptation des interactions) dépend du PCBS auquel obéit le modèle de WFIO soumis à l'opération d'adaptation. Concernant les aspects d'évolution et de réutilisation de modèles, nous introduisons deux catégories de patrons plus élaborés : les patrons de coopération *généralisés* et les patrons de coopération *composites*.

## VI.1 Motivations

Le besoin de flexibilité dans les processus métiers s'est toujours imposé par le manque de stabilité de l'environnement métier soumis à des évènements internes et/ou externes et les exigences des clients potentiels, dans un contexte de concurrence accrue.

Dans une optique d'amélioration continue (CPI) ou de re-conception de processus (BPR), chaque WF participant à une coopération peut subir des changements en vue de son amélioration et son alignement avec de nouvelles exigences métier. Ces changement peuvent être une redéfinition d'une activité, un changement de contrôle de flux entre deux activités en vue d'améliorer le temps de réponse du processus, prendre en compte de nouvelles contraintes, exprimer des dépendances entre activités ou encore avoir un contrôle supplémentaire sur l'exécution des instances du processus. Ces changements n'affectent qu'un fragment de WF et n'ont pas d'influence sur le WF global pourvu qu'ils n'altèrent pas la structure d'interaction avec d'autres fragments impliqués dans le WFIO.

Un autre aspect d'adaptation pourrait affecter la structure d'interaction et consiste essentiellement à ajouter, supprimer ou modifier un point d'interaction entre deux fragments de WF impliqués dans un WFIO. Ces situations sont envisageables dans le cas d'une rupture de contrat entre deux partenaires, une défaillance d'un fragment de WF qu'il faut remplacer ou encore une simple modification de la structure d'interaction entre les fragments de WF existants.

Nous pensons aussi à des situations d'adaptation *évolutive* dans un contexte inter-organisationnel et là nous nous penchons sur deux aspects de l'évolution qui sont l'expansion des fonctionnalités et l'expansion de la coopération. Principalement, cette dernière est motivée par le besoin d'ouverture d'un WFIO à de nouveaux partenaires en vue d'étendre le processus de WFIO à de nouvelles fonctionnalités ou encore la possibilité de tirer profit des ressources et des compétences offertes par un nouveau partenaire. Par exemple, on pourrait envisager l'implication d'un processus WF fourni par un organisme bancaire qui vient s'ajouter à un WFIO impliquant déjà un fournisseur et un producteur, en vue d'inclure le processus de paiement dans le WF global. L'externalisation de nouveaux services dans une « Sous-traitance » par exemple, est aussi une forme d'expansion de la coopération. Ce type d'évolution est supporté par le concept de patron de coopération *généralisé*, puisque le WFIO obtenu après expansion, obéit toujours au patron de coopération initial.

Un autre aspect sur lequel nous nous sommes penchés est la réutilisation des modèles de WFIO existants. Ce point est motivé par un besoin éventuel de construire un WFIO complexe mais qu'une partie de ce WF est déjà implémenté sous forme d'un WFIO existant. Les modèles de WFIO obtenus par réutilisation sont complexes et reposent sur la combinaison de deux patrons de coopération parmi ceux décrits au chapitre V. Cependant, cette complexité du modèle peut être atténuée en considérant le WFIO réutilisé comme une entité à part entière que l'on pourrait adapter ou remplacer, sans affecter le processus global obéissant au patron de coopération *composite*, un concept que nous introduisons pour supporter la réutilisation de modèles de WFIO, dans un but d'évolution de modèles existants.

Dans la suite, nous décrivons en détail chacun des trois aspects de flexibilité que nous considérons dans le cadre de cette thèse.

## VI.2 Adaptabilité et adaptation d'un modèle de WFIO

L'adaptabilité d'un modèle mesure sa capacité à supporter les changements sans altérer la fonctionnalité globale ni l'aspect de coopération dans le processus global.

- **Définition VI.1:** Adaptation d'un modèle de WFIO

*L'adaptation d'un modèle de WFIO est l'opération qui permet d'obtenir un modèle de WFIO adapté (modifié) à partir d'un modèle de WFIO initial et qui consiste à apporter un changement sur ce modèle sans étendre sa fonctionnalité globale ou l'ensemble des partenaires impliqués.*

### VI.2.1 Catégories d'adaptation dans un modèle de WFIO

Comme nous l'avons souligné, dans notre cas, nous classons les adaptations de modèles de WFIO en trois catégories, conformément aux trois dimensions sur lesquelles repose la définition d'un PCBS (cf. chapitre V) : (i) adaptations de *services*, (ii) adaptations de *contrôle de flux*, (iii) adaptations des *interactions* (Boukhedouma et al., 2013a).

Les adaptations de services concernent les opérations d'ajout, de suppression, de substitution, de fusion ou de décomposition de services, dans un fragment de WF local à un partenaire impliqué dans la coopération. Ces opérations induisent des changements dans la *fonction d'orchestration* locale et affectent forcément le contrôle de flux dans le fragment concerné par l'adaptation.

Les adaptations de contrôle de flux concernent un changement dans l'enchainement entre deux services, nous parlerons de séquentialisation de services lorsqu'il s'agit de mettre en séquence deux services qui initialement étaient en parallèle ou en alternatif. Nous parlerons de parallélisation de services lorsqu'il est nécessaire de mettre deux services en parallèle qui ne l'étaient pas initialement et enfin une alternation de services pour relier deux services par une condition inclusive ou exclusive.

Les adaptations des interactions se réfèrent à l'ajout, la modification ou la suppression de points d'interaction entre deux fragments de WF impliqués dans une coopération. Nous verrons que les adaptations de cette dernière catégorie, contrairement aux deux premières, diffèrent selon le patron de coopération qui relie les fragments de WF interconnectés. La figure suivante résume les différentes catégories d'adaptations d'un modèle de WFIO que nous considérons, dans cette thèse.

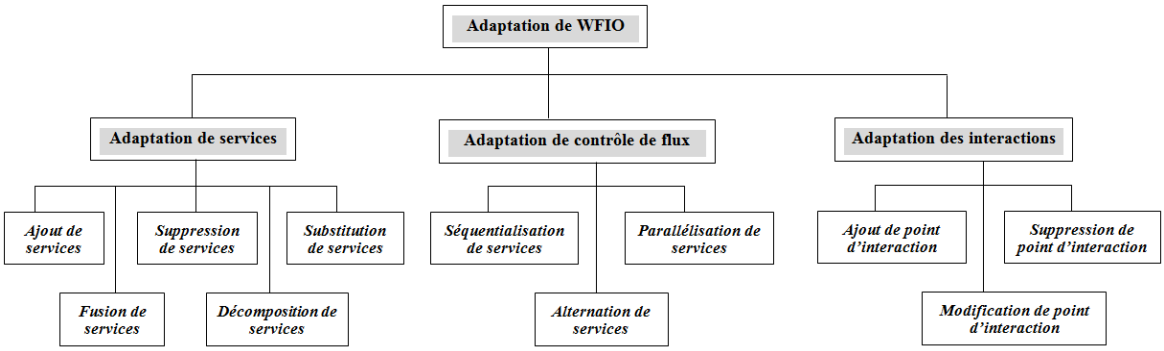



**Figure VI.1.** *Vue d'ensemble des catégories d'adaptation dans un WFIO*

Dans la suite, nous reprenons chacune des catégories d'adaptation et nous explicitons les différents patrons d'adaptation que nous avons conçus et implémentés, ainsi qu'une formalisation à travers un certain nombre d'opérateurs que nous définissons.

**Remarque** : Notons que le niveau d'abstraction requis sur les modèles de processus à adapter est de type « Activité » ou « Sous-processus » pour les deux catégories d'adaptation (adaptation de s ervices et adaptation de contrôle de flux), selon la granularité des services composant le

processus local à un partenaire, tandis que pour la troisième catégorie (i.e adaptation des interactions), le niveau d'abstraction est fonction du patron de coopération auquel obéit le modèle de WFIO. Ceci sera illustré dans la description des différents patrons.

### VI.2.2 Adaptations de services

#### VI.2.2.1 Ajout de services

Un ajout de service consiste à ajouter une étape dans le processus, on pourrait envisager une étape de vérification ou de validation d'une partie du processus qui n'a pas d'effet sur la fonctionnalité globale du WFIO. Dans une opération d'ajout de services dans un modèle de WF, plusieurs scénarios peuvent être envisagés : l'ajout dans un bloc séquentiel, l'ajout dans un bloc parallèle et l'ajout dans un bloc alternatif (inclusif ou exclusif). Dans ces deux derniers cas, deux configurations sont possibles : le bloc parallèle ou alternatif est nouvellement créé suite à l'ajout du nouveau service ou alors un ajout dans un bloc parallèle ou alternatif déjà existant. Ces différentes situations donnent lieu à différents patrons d'adaptation liés à l'ajout de service.

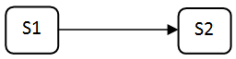
#### A- Description des patrons d'ajout de services

##### A.1 Les patrons élémentaires d'ajout dans un bloc séquentiel

Ces patrons concernent l'ajout d'un service dans un bloc séquentiel avant ou après un service de référence Sr. La figure suivante décrit les deux patrons élémentaires d'ajout référencés par AP1.1 et AP1.2, respectivement pour l'ajout *avant* et l'ajout *après* un service de référence donné.

Référence du patron : **PA1.1 (ou AP1.1)**
Nom du patron : **Ajout séquentiel « avant »**
Description : Ajout d'une étape dans le processus sur une branche séquentielle avant un service de référence
Service de référence : S1 Niveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : Néant

*Avant adaptation*




*Seq(S1, S2)*

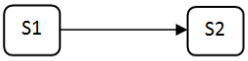
*Après adaptation*


*Seq(Seq (S',S1), S2)*

**Contraintes:** Input(S1) = Output(S')

Référence du patron : **PA1.2 (ou AP1.2)**
Nom du patron : **Ajout séquentiel « après »**
Description : Ajout d'une étape dans le processus sur une branche séquentielle après un service de référence.
Service de référence : S1
Patrons utilisés : Néant

*Avant adaptation*




*Seq(S1, S2)*

*Après adaptation*


*Seq(Seq (S1, S'), S2)*

**Contraintes:** Input(S') = Output(S1) et Input (S2) = Output (S')

**Figure VI.2.** *Description des patrons élémentaires d'ajout de service AP1.1 – AP1.2*

### A.2 Les patrons d'ajout dans un bloc parallèle ou alternatif

Ces patrons concernent l'ajout d'un service sur une branche d'un bloc alternatif (inclusif ou exclusif) ou parallèle, ces derniers nécessitent l'utilisation des patrons élémentaires AP1.1 ou AP1.2, selon les cas. La figure VI.3 fournit une description des patrons d'ajout AP1.3, AP1.4 et AP1.5, respectivement pour l'ajout d'un service sur une branche alternative inclusive, sur une branche alternative exclusive ou sur une branche parallèle. Un ajout dans un bloc alternatif nécessite éventuellement une mise à jour de la pré-condition (OP-Split) et/ou la post-condition (OP-Join) du bloc.

Référence du patron : **PA1.3 (ou AP1.3)**
Nom du patron : **Ajout inclusif**
Description : Ajout d'une étape dans le processus sur une branche alternative inclusive
Service de référence : S2 Nniveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : AP1.2 (ou AP1.1 dans d'autres cas)

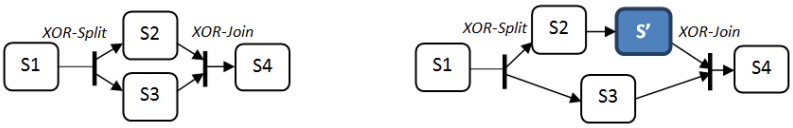
*Avant adaptation* *Après adaptation*


*Seq(Seq(S1, Alt(S2, S3)), S4)* *Seq(Seq(S1, Alt(Seq(S2, S'), S3)), S4)*

**Contraintes :** Input(S') = Output(S2) et Input (S4) ⊆ Output (S') ⋃ Output (S3)

Référence du patron : **PA1.4 (ou AP1.4)**
Nom du patron : **Ajout exclusif**
Description : Ajout d'une étape dans le processus sur une branche alternative exclusive
Service de référence : S2
Patrons utilisés : AP1.2 (ou AP1.1 dans d'autres cas)

*Avant adaptation* *Après adaptation*

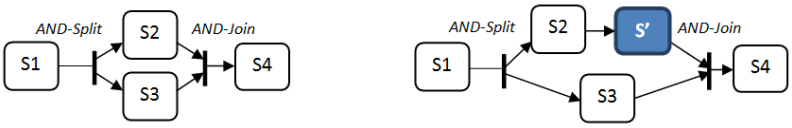



*Seq(Seq(S1, Exl(S2, S3)), S4)* *Seq(Seq(S1, Exl (Seq(S2, S'), S3)), S4)*

**Contraintes:** Input(S') = Output(S2) et Input (S4) = Output (S') **ou bien** Input (S4) = Output (S3)

Référence du patron : **PA1.5 (ou AP1.5)**
Nom du patron : **Ajout parallèle**
Description : Ajout d'une étape dans le processus sur une branche d'un bloc parallèle
Service de référence : S2
Patrons utilisés : AP1.2 (ou AP1.1 dans d'autres cas)

*Avant adaptation* *Après adaptation*




*Seq(Seq(S1, Par(S2, S3)), S4)* *Seq(Seq(S1, Par (Seq(S2, S'), S3)), S4)*

**Contraintes :** Input(S') = Output(S2) et Input (S4) = Output (S') ⋃ Output (S3)

**Figure VI.3.** *Description des patrons d'ajout dans un bloc alternatif ou parallèle AP1.3- AP1.4 – AP1.5*

### A.3 Les patrons de création de bloc alternatif ou parallèle

Certaines opérations d'ajout de services nécessitent la création d'un bloc alternatif ou parallèle initialement inexistant dans la structure du processus. Ces patrons sont appelés patrons de création de blocs et sont décrits dans la figure VI.4 ci-dessous. Remarquons que Ces patrons nécessitent l'introduction de nouveaux opérateurs *Alt*, *Exl* ou *Par* dans l'expression de la fonction d'orchestration qui ne figuraient pas avant l'adaptation.

Référence du patron : **PA1.6 (ou AP1.6)**
Nom du patron : **Création de bloc inclusif**
Description : Ajout d'un service dans le processus en alternatif avec un service de référence dans un bloc séquentiel
Service de référence : S2 Niveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : Néant

*Avant adaptation* *Après adaptation*


**Contraintes:** Input(S') ⊆ Output(S1)
**Spécification** : Création d'un bloc alternatif inclusif (Ajout d'une condition)

Référence du patron : **PA1.7 (ou AP1.7)**
Nom du patron : **Création bloc exclusif**
Description : Ajout d'un service dans le processus en alternatif exclusif avec un service de référence dans un bloc séquentiel
Service de référence : S2
Patrons utilisés : Néant

*Avant adaptation* *Après adaptation*


**Contraintes:** Input(S') ⊆ Output (S1)
**Spécification** : Création d'un bloc alternatif exclusif (Ajout d'une condition)

Référence du patron : **PA1.8 (ou AP1.8)**
Nom du patron : **Création bloc parallèle**
Description : Ajout d'un service dans le processus en parallèle avec un service de référence
Service de référence : S2
Patrons utilisés : Néant

*Avant adaptation* *Après adaptation*


**Contraintes:** Input(S') ⊆ Output (S1)
**Spécification** : Création d'un bloc parallèle

**Figure VI.4.** *Description des patrons d'ajout par création de bloc alternatif ou parallèle AP1.6- AP1.7 – AP1.8*

## B. Exemple d'utilisation d'un patron d'ajout de service

Dans la figure VI.5, nous reprenons l'exemple du processus de WFIO (déjà décrit dans le chapitre V) destiné à la conception et la réalisation de circuits intégrés (CI) à des clients potentiels. Le processus implique deux partenaires (un partenaire principal et un partenaire secondaire) et obéit au patron de coopération « Sous-traitance ». Les différentes phases du processus sont encapsulées dans des services pouvant être des services composites vu leur complexité. A droite, nous reprenons le même processus en spécifiant les noms de services par S11, S12, … S2 afin d'illustrer aisément l'expression de la fonction d'orchestration.

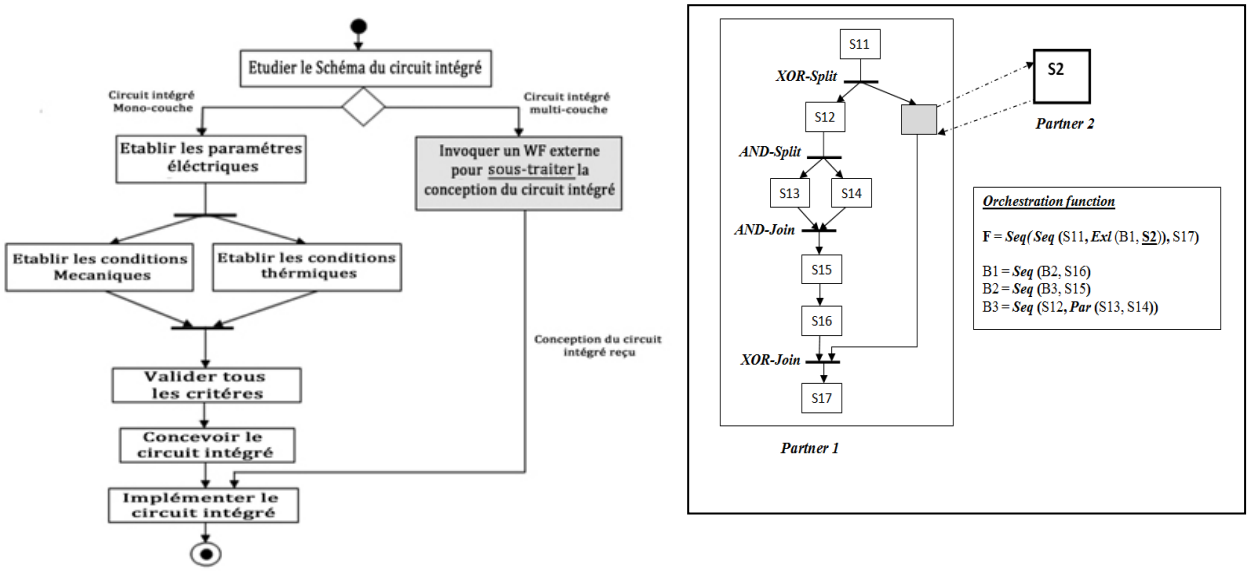



**Figure VI.5.** *Exemple de WFIO « Réalisation de circuits intégrés »*

Concernant l'aspect adaptation, le concepteur du processus WFIO peut décider d'ajouter un service « Valider les paramètres électriques » après le service « Etablir les paramètres électriques » afin d'introduire une étape de validation qui peut être nécessaire pour la conception de circuits complexes, avant d'établir les conditions mécaniques et thermiques. On obtient alors le schéma indiqué sur la figure VI.6 ci-dessous, avec la fonction d'orchestration correspondante obtenue en transformant le WFIO initial. Le service S' nouvellement inséré correspond à « Valider des paramètres électriques »

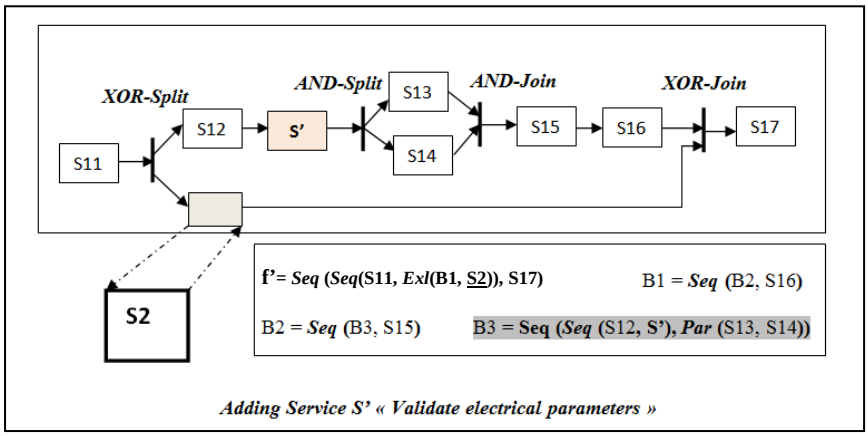



**Figure VI.6.** *Exemple d'adaptation par « ajout de service »*

## C. Formalisation du patron « Ajout de service »

Pour la formalisation du patron d'ajout de service, nous proposons la définition suivante de l'opérateur « **Ins »** (cette définition est déjà introduite au chapitre V)

- **Définition VI.2 :** Adaptation par ajout de service – Opérateur « **Ins »**

Soit WF = (SL, f) un fragment de WF participant à une coopération dans un WFIO et soient un service S à ajouter dans WF et un service de référence Sr appartenant à SL. L'ajout du service S dans le fragment de WF (*avant*, *après*, en *parallèle* ou en *alternatif* avec le service Sr de référence) donne un fragment de WF adapté désigné par WF' = (SL', f') avec

**SL' = SL U{S} et** f' = **Ins** (f, S, Sr, "mode")

**mode** ∈ {Avant, Après, Par, Alt}

L'opérateur **Ins** s'applique à une fonction d'orchestration f et permet d'ajouter un service S avant, après, en parallèle ou en alternatif avec un service de référence Sr pour obtenir une nouvelle fonction d'orchestration f'. On écrira : **f'** = **Ins (f, S, Sr,** "**mode**"**)**

### VI.2.2.2 Suppression de services

La *suppression* est l'opération inverse de l'ajout ; une adaptation basée sur la suppression de service pourrait être nécessaire pour éliminer une étape du processus qui n'a plus lieu d'exister, suite à un changement de la logique métier, par exemple. Comme pour l'ajout de services, deux formes sont distinguées, dans la suppression: la suppression de service dans un bloc séquentiel (i.e une seule branche) et la suppression de service dans un bloc parallèle ou alternatif (composé de deux branches).
Aussi, pour les blocs non séquentiels, deux configurations sont envisageables: (i) le service à supprimer fait partie d'une séquence de services (voir Figure VI.8), (ii) le service à supprimer est seule sur la branche (voir Figure VI.9).

### A- Description des patrons de suppression de services

#### A.1 Le patron élémentaire de suppression

Ce patron référencé par PA2.1 (ou AP2.1) concerne la suppression d'un service dans un bloc séquentiel et est décrit dans la figure suivante :

Référence du patron : **PA2.1 (ou AP2.1)**
Nom du patron : **Suppression séquentielle**
Description : Suppression d'une étape dans le processus sur une branche séquentielle
Patrons utilisés : Néant Niveau d'abstraction : « Activité » ou « Sous-processus »
Service à supprimer : S2


**Contrainte:** Input(S3) = Output(S1)

**Figure VI.7.** *Description du patron de suppression élémentaire AP2.1*

### A.2 Les patrons de suppression plus élaborés

Ces patrons concernent la suppression de services dans un bloc alternatif (inclusif ou exclusif) ou parallèle et nécessitent l'utilisation du patron élémentaire AP2.1. La figure VI.8 décrit les patrons de suppression AP2.2, AP2.3 et AP2.4 respectivement pour la suppression d'un service sur une branche alternative inclusive, sur une branche alternative exclusive ou sur une branche parallèle. Une suppression dans un bloc alternatif nécessite éventuellement une mise à jour de la pré-condition (OP-Split) et/ou la post-condition (OP-Join) du bloc.

Référence du patron : **PA2.2 (ou AP2.2)**
Nom du patron : **Suppression inclusive**
Description : Suppression d'une étape dans le processus sur une branche d'un bloc inclusif
Service à supprimer : S2 Niveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : AP2.1

*Avant adaptation* *Après adaptation*


*Seq(Seq(S1, Alt(Seq(S2, S3), S4)), S5)* *Seq(Seq(S1, Alt(S3, S4)), S5)*

**Contraintes:** Input(S3) = Output(S1) Mise à jour de la condition

Référence du patron : **PA2.3 (ou AP2.3)**
Nom du patron : **Suppression exclusive**
Description : Suppression d'une étape dans le processus sur une branche d'un bloc exclusif
Service à supprimer : S2
Patrons utilisés : AP2.1

*Avant adaptation* *Après adaptation*


*Seq(Seq(S1, Exl(Seq(S2, S3), S4)), S5)* *Seq(Seq(S1, Exl(S3, S4)), S5)*

**Contraintes :** Input(S3) = Output(S1) Mise à jour de la condition

Référence du patron : **PA2.4 (ou AP2.4)**
Nom du patron : **Suppression parallèle**
Description : Suppression d'une étape dans le processus sur une branche d'un bloc parallèle
Service à supprimer : S2
Patrons utilisés : AP2.1

*Avant adaptation* *Après adaptation*


*Seq(Seq(S1, Par(Seq(S2, S3), S4)), S5)* *Seq(Seq(S1, Par(S3, S4)), S5)*

**Contraintes :** Input(S3) = Output(S1)

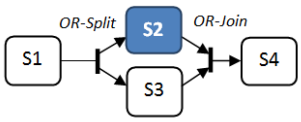
**Figure VI.8.** *Description des patrons de suppression dans un bloc alternatif ou parallèle AP2.2- AP2.3 – AP2.4*

### A.3 Les patrons de suppression de bloc alternatif ou parallèle

Certaines opérations de suppression de services induisent la suppression d'un bloc alternatif ou parallèle (selon les cas) ; il s'agit des configurations où le service à supprimer est seul sur la branche alternative ou parallèle, c'est-à-dire qu'il ne se trouve pas en séquence avec d'autres services sur la branche. Ce type d'adaptation induit une suppression de bloc qui n'a plus lieu d'exister puisque le seul service qui constituait une des branches du bloc a fait l'objet d'une suppression (voir Figure VI.9).

Référence du patron : **PA2.5 (ou AP2.5)**
Nom du patron : **Suppression sur une branche alternative « single »**
Description : Suppression d'une étape dans le processus sur une branche alternative inclusive comportant un seul service
Niveau d'abstraction : « Activité » ou « Sous-processus »
Service à supprimer : S2
Patrons utilisés : Néant

*Avant adaptation*




*Après adaptation*

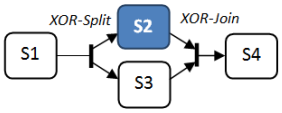

*Seq(Seq(S1, Alt(S2, S3)), S4)*

*Seq(Seq(S1, S3), S4)*

**Contraintes:** Input(S3) = Output(S1) Input(S4) = Output(S3) et Suppression de la condition

Référence du patron : **PA2.6 (ou AP2.6)**
Nom du patron : **Suppression sur une branche exclusive « single »**
Description : Suppression d'une étape dans le processus sur une branche alternative exclusive comportant un seul service
Service à supprimer : S2
Patrons utilisés : Néant

*Avant adaptation*




*Après adaptation*

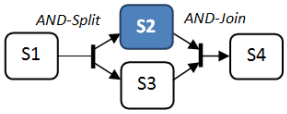

*Seq(Seq(S1, Exl(S2, S3)), S4)*

*Seq(Seq(S1, S3), S4)*

**Contraintes:** Input(S3) = Output(S1) Input(S4) = Output(S3) et Suppression de la condition

Référence du patron : **PA2.7 (ou AP2.7)**
Nom du patron : **Suppression sur une branche parallèle « single »**
Description : Suppression d'une étape dans le processus sur une branche parallèle comportant un seul service
Service à supprimer : S2
Patrons utilisés : Néant

*Avant adaptation*




*Après adaptation*


*Seq(Seq(S1, Par(S2, S3)), S4)*

*Seq(Seq(S1, S3), S4)*

**Contraintes:** Input(S3) = Output(S1) Input(S4) = Output(S3) et Suppression du parallélisme

**Figure VI.9.** *Description des patrons de suppression dans un bloc alternatif ou parallèle « single » AP2.5- AP2.6 – AP2.7*

### B. Formalisation du patron « Suppression de service »

Pour la formalisation du patron de suppression de service, nous proposons la définition suivante de l'opérateur « **Rem** »

- **Définition VI.3 :** Adaptation par suppression de service – L'opérateur « Rem »

Soit WF = (SL, f) un fragment de WF participant à une coopération dans un WFIO et soit un service S appartenant à SL. La suppression du service S dans le fragment de processus WF donne un fragment de WF adapté désigné par WF'= (SL', f') avec

**SL' = SL - {S} et** f' = **Rem** (f, S)

L'opérateur « **Rem** » s'applique à une fonction d'orchestration f et permet de supprimer le service S pour obtenir une nouvelle fonction d'orchestration f'. On écrira : **f'** = **Rem (f, S)**

### VI.2.2.3 Substitution de services

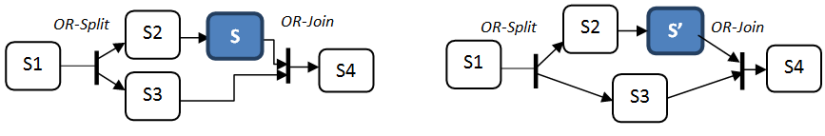
La substitution de services est l'opération qui consiste à remplacer un service par un autre. Cette opération pourrait être nécessaire dans un but correctionnel (remplacer un service défaillant ou mal conçu) ou dans un but perfectif (améliorer le temps de réponse du processus par exemple). Dans notre cas, la substitution d'un service S par un service S' se fait par insertion du service S' après S suivie d'une suppression du service S. Selon la structure du processus et le service à substituer, différents patrons de substitution référencés par PA3.1, PA3.2, PA3.3 et PA3.4 respectivement pour une substitution dans un bloc séquentiel, inclusif, exclusif ou parallèle, sont envisagés.

### A- Description des patrons de substitution de services

La figure VI. 10 décrit le patron de substitution dans un bloc alternatif inclusif désigné par PA3.2. Ce dernier utilise les patrons PA1.3 d'insertion et PA2.2 de suppression.

Référence du patron : **PA3.2(ou AP3.2)**
Nom du patron : **Substitution alternative**
Description : Remplacement d'un service dans le processus par un autre sur une branche alternative Niveau d'abstraction : « Activité » ou « Sous-processus »
Service à substituer : S Service substituant : S'
Patrons utilisés : PA1.3 et PA2.2

*Avant adaptation* *Après adaptation*




*Seq(Seq(S1, Alt(Seq(S2, S), S3), S4)* *Seq(Seq(S1, Alt (Seq(S2, S'), S3)), S4)*

**Contraintes:** Input(S') = Output (S2) et Input (S4) ⊆ Output(S') U Output (S3)

**Ou bien**


*Seq(Seq(S1, Alt(S, S3)), S4)* *Seq(Seq(S1, Alt(S', S3)), S4)*

**Contraintes:** Input(S') = Output (S1) et Input (S4) ⊆ Output(S') U Output (S3)

**Figure VI.10.** *Description du patron de substitution alternative - AP3.2*

Le patron de substitution dans un bloc séquentiel est référencé par PA3.1, les patrons utilisés dans ce cas sont PA1.1 (ou PA1.2) et PA2.1 respectivement, pour l'insertion «avant » (ou l'insertion « après ») et la suppression.

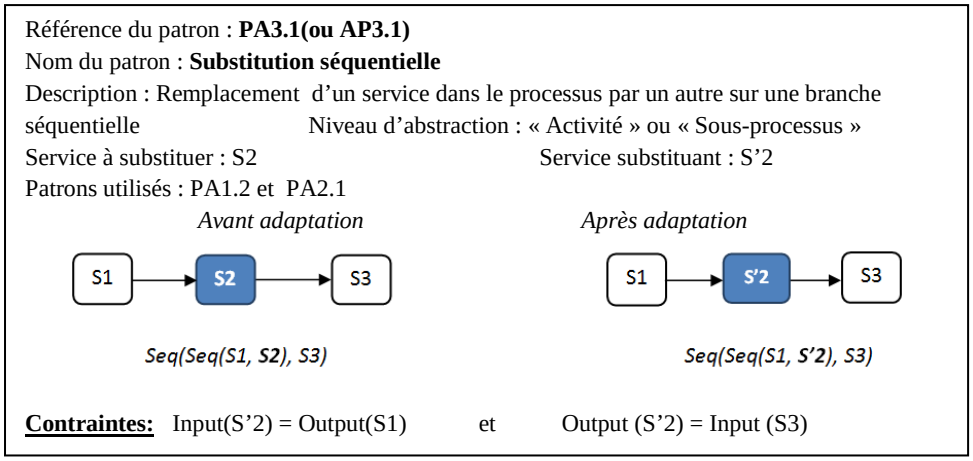



**Figure VI.11.** *Description du patron de substitution séquentielle de services AP3.1*

### B. Formalisation du patron « Substitution de service »

Pour la formalisation du patron de substitution de service, nous proposons la définition suivante de l'opérateur «**Sub**»

- **Définition VI.4 :** Adaptation par substitution de service – L'opérateur « Sub »

Soit WF = (SL, f) un fragment de WF participant à une coopération dans un WFIO et soit un service S appartenant à SL, à substituer par un service S' dans WF. La substitution du service S par S' dans le fragment WF donne un fragment de WF adapté désigné par WF' = (SL', f') avec

**SL' = SL - {S}U{S'} et** f' = **Sub** (f, S, S')

L'opérateur « **Sub »** s'applique à une fonction d'orchestration f et permet de substituer le service S par S' pour obtenir une nouvelle fonction d'orchestration f'.

On écrira : f' = **Sub** (f, S, S')

- Propriétés de l'opérateur **« Sub »** :

| |
|---|
| **Sub** (f, S, S') = **Rem** (**Ins** (f, S', S, "Après"), S) |
| **Sub** (f, S, S') = **Rem** (**Ins** (f, S', S, "Avant"), S) |

### VI.2.2.4 Fusion de services

La *fusion de services* peut concerner deux services reliés par un opérateur de séquence, de choix inclusif, de choix exclusif ou de parallélisme. Une fusion de services peut être appliquée pour simplifier le modèle de processus par abstraction d'un ensemble de services dans un seul. L'opération de fusion est réalisée en remplaçant les services concernés par la fusion, par un seul service. Logiquement la fonctionnalité couverte par le service fusion est *l'agrégation des fonctionnalités* fournies par les deux services objet de la fusion. De la même façon que les opérations d'ajout, de suppression et de substitution, on distingue la fusion d'un bloc séquentiel de la fusion d'un bloc alternatif ou parallèle. Remarquons que puisque les services à fusionner sont

dans le même bloc, l'opération de fusion consiste à supprimer un bloc et le remplacer par un service.

### A- Description des patrons de fusion de services

Concrètement, la fusion de deux services revient à des opérations d'ajout/suppression dans un bloc séquentiel, parallèle ou alternatif, selon les cas. Nous commençons d'abord par l'opération d'insertion afin de considérer un des services à fusionner comme service de référence (un repère) qui nous permettra d'effectuer une insertion « avant » ou « après » ce service de référence. Ceci étant repris dans la description des patrons de fusion PA4.1 et PA4.4, par exemple (voir Figure VI.12). Notons que ces patrons utilisent les patrons d'insertion PA1.x et les patrons de suppression PA2.x.

Référence du patron : **PA4.1(ou AP4.1)**
Nom du patron : **Fusion séquentielle**
Description : Remplacement de la succession de deux services dans le processus par un service représentant leur fusion Niveau d'abstraction : « Activité » ou « Sous-processus »
Services à fusionner : S2, S3 Service fusion: S'
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1

*Avant adaptation* *Après adaptation*


*Seq(Seq(Seq(S1, S2), S3), S4)* *Seq(Seq(S1, S'), S4)*

**Contraintes:** Input(S') = Output(S1) et Output (S') = Input (S4)

Référence du patron : **PA4.4(ou AP4.4)**
Nom du patron : **Fusion parallèle**
Description : Remplacement de deux services parallèles par un seul service
Services à fusionner : S2, S3 Service fusion: S'
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1

*Avant adaptation* *Après adaptation*


*Seq(Seq(S1, Par(S2, S3)), S4)* *Seq(Seq(S1, S'), S4)*

**Contraintes:** Input(S') = Output(S1) et Output (S') = Input (S4)

**Figure VI.12.** *Description des patrons de fusion de services AP4.1- AP4.4*

### B. Formalisation du patron « Fusion de services »

Pour la formalisation du patron de fusion de services, nous proposons la définition suivante de l'opérateur «**Merge**»

- **Définition VI.5 :** Adaptation par fusion de services - L'opérateur « Merge »

Soit WF = (SL, f) un fragment de WF participant à une coopération dans un WFIO, soient deux service S1 et S2 appartenant à SL et reliés par un opérateur de contrôle de flux et soit un service S'donné. La fusion de S1 et S2 en S' dans le fragment WF donne un fragment de WF adapté désigné par WF' = (SL', f') avec **SL' = SL - {S1, S2}U{S'} et** f' = **Merge** (f, S1, S2, S')

L'opérateur **Merge** s'applique à une fonction d'orchestration f et permet de fusionner les services S1 et S2 en S'pour obtenir une nouvelle fonction d'orchestration f'.

On écrira : **f'** = **Merge (f, S1, S2, S')**

En d'autres termes f' est obtenue à partir de f en substituant le bloc composé des services S1, S2 référencés dans f par le service S'.

Propriétés de l'opérateur « **Merge** » :

Dans un bloc séquentiel

**Merge** (f, S1, S2, S') = **Rem (Rem (Ins**(f, S', S2, "Après" ) ,S1), S2)

**Merge** (f, S1, S2, S') = **Rem (Rem (Ins**(f, S',S1, "Avant" ) ,S1), S2)

Dans un bloc parallèle ou alternatif

**Merge** (f, S1, S2, S') = **Rem (Rem (Ins**(f, S',Sx, "Après" ) ,Sx), Sy)

**Merge** (f, S1, S2, S') = **Rem (Rem (Ins**(f, S',Sx, "Avant" ) ,Sx), Sy)

Sx, Sy représentent indifféremment S1 ou S2

### VI.2.2.5 Décomposition de services

L'opération de décomposition est l'inverse de la fusion, il s'agit de décomposer un service pour obtenir un bloc composé de deux services séquentiels, alternatifs ou parallèles. Une décomposition peut être faite pour avoir un meilleur contrôle sur l'exécution du processus (bloc séquentiel), ou pour améliorer le temps de réponse du processus (bloc parallèle) ou encore pour prendre en charge de nouvelles contraintes (bloc alternatif). Afin de maintenir une cohérence de la logique métier dans le processus, la fonctionnalité couverte par le bloc de services composants à insérer est identique à la fonctionnalité du service objet de la décomposition.

### A- Patrons de décomposition de services

Comme pour les patrons de substitution et de fusion, la décomposition est réalisée en utilisant les patrons d'insertion et de suppression. Remarquons aussi que nous commençons par insérer le bloc B avant ou après le service de référence S, ensuite nous procédons à la suppression de S. Les figures VI.13 et VI.14 illustrent deux patrons de décomposition de services référencés par PA5.1 et PA5.2 pour une décomposition séquentielle et alternative, respectivement.

Référence du patron : **PA5.1(ou AP5.1)**
Nom du patron : **Décomposition séquentielle**
Description : Remplacement d'un service par une séquence de deux services
Service à décomposer: S2 Bloc de décomposition: Seq(S', S'')
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1

*Avant adaptation*


*Seq(Seq(S1, S2), S3)*

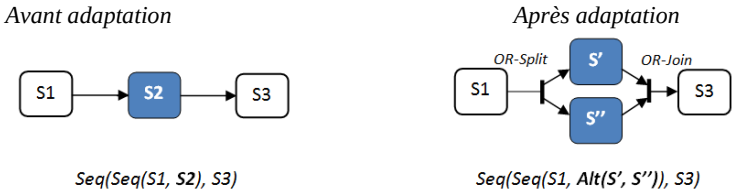
*Après adaptation*


*Seq(Seq(Seq(S1, S'), S''), S3)*

**Contraintes:** Input(S') = Output(S1) et Output (S") = Input (S3)

**Figure VI.13.** *Description du patron de décomposition séquentielle AP5.1*

Référence du patron : **PA5.2(ou AP5.2)**
Nom du patron : **Décomposition alternative**
Description : Remplacement d'un service par un bloc alternatif composé de deux services
Services à décomposer : S2 Bloc de décomposition: Alt (S', S'')
Patrons utilisés : PA1.6 et PA2.1 Niveau d'abstraction : « Activité » ou « Sous-processus »




**Contraintes:**

Input(S') = Output(S1) ou Input (S") = Output (S1)
Output (S') = Iutput(S3) ou Output (S") = Input (S3)
Ajout d'une condition

**Figure VI.14.** *Description du patron de décomposition alternative AP5.2*

## B. Exemple d'utilisation d'un patron de décomposition de services

Reprenons l'exemple de production de produits pharmaceutiques déjà vu au chapitre V.

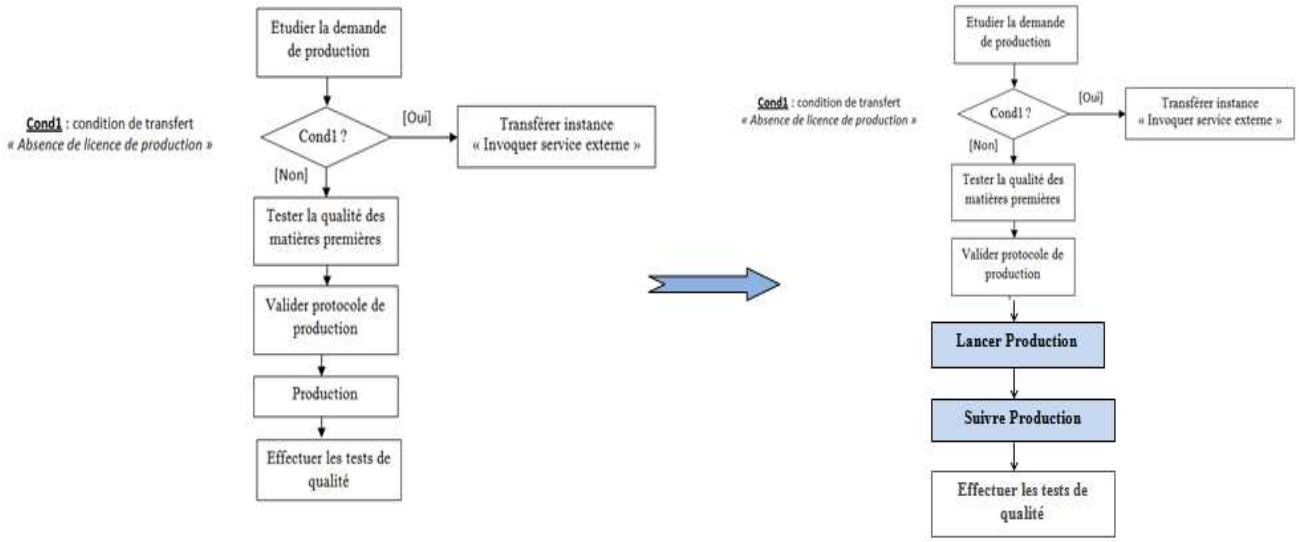



**Figure VI.15** *Exemple d'une adaptation – Décomposition de service*

Sur un fragment de WF local à un partenaire métier, il est possible d'apporter des modifications qui soient invisibles aux autres partenaires et sans altérer la logique métier du processus global. Sur l'exemple de la figure VI.15, il est possible de décomposer le service « Production » en deux services séquentiels « Lancer Production » et « Suivre Production » afin de mieux contrôler la phase de production dans le processus ; le schéma du processus adapté est décrit à droite de la figure VI.15.

### C. Formalisation du patron « Décomposition de services »

Pour la formalisation du patron de décomposition de service, nous proposons la définition suivante de l'opérateur « **Dec** »

- **Définition VI.6 :** Adaptation par décomposition de services - L'opérateur « Dec »

Soit WF = (SL, f) un fragment de WF participant à une coopération dans un WFIO et soit un service S appartenant à SL, à décomposer en un bloc B = S1***op***S2 dans WF (op est un opérateur de contrôle de flux). La décomposition de S en S1***op***S2 dans le fragment WF donne un fragment de WF adapté désigné par WF'= (SL', f') avec

**SL' = SL - {S}U{S1, S2}et** f' = **Dec** (f, S,B)

L'opérateur **Dec** s'applique à f et permet de remplacer le service S par un bloc B reliant deux services S1 et S2 pour obtenir une nouvelle fonction d'orchestration f'.
On écrira : **f'** = **Dec (f, S, B)**

En d'autres termes, f' est obtenue à partir de f en substituant le service S référencé dans f par le bloc B.

- Propriétés de l'opérateur « **Dec »** :

| |
|---|
| **Dec** (f, S, B) = **Rem (Ins** (f, B, S, "Après"), S) |
| **Dec** (f, S, B) = **Rem (Ins**(f, B, S, "Avant"), S) |

### VI.2.3 Adaptation de contrôle de flux

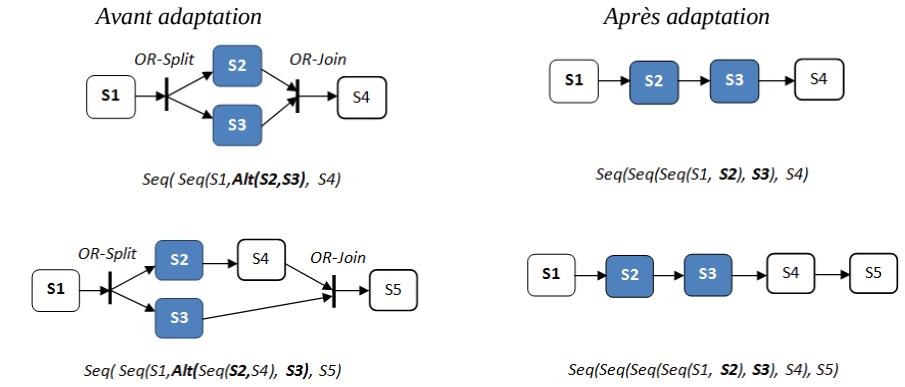
Une autre catégorie d'opérations d'adaptation concerne la modification du flux de contrôle (donc la fonction d'orchestration) sans altérer les services impliqués dans le processus. Il s'agit particulièrement de remplacer un opérateur de contrôle de flux par un autre; on pourrait par exemple remplacer un opérateur de parallélisme (par) par un opérateur de séquence (seq) si les services deviennent dépendants l'un de l'autre, ou alors remplacer un opérateur de séquence par un opérateur de parallélisme dans le cas contraire pour augmenter le degré de parallélisme dans le processus. Les adaptations de contrôle de flux concernent donc un changement dans la logique d'enchainement entre deux services existants sans pour autant modifier les services eux-mêmes, il s'agit d'une simple *restructuration* de services. Les opérations d'adaptation de contrôle de flux donnent lieu, selon les cas, soit à une séquentialisation, une alternation ou une parallélisation de services.

#### VI.2.3.1 Patrons de séquentialisation de services

Ces patrons sont appliqués dans le cas où l'on veut restructurer deux services initialement reliés par un opérateur de choix ou de parallélisme, pour les mettre en séquentiel. Dans la figure VI.16, nous décrivons les patrons de séquentialisation de deux services initialement reliés par un opérateur de choix inclusif. Les autres patrons se réalisent de manière quasiment identique.

D'autres configurations plus complexes peuvent se présenter par exemple si S2 est en séquence avec S4 et S3 est aussi en séquence avec un autre service S6, il s'agira de remplacer S2 et S3 (initialement en parallèle ou en alternatif) par un bloc séquentiel Seq (S2, S3) qui sera suivi d'un bloc alternatif ou parallèle composé des services S4 et S6.

Référence du patron : **PA6.1, PA6.2 (ou AP6.1, AP6.2)**
Nom du patron : **Séquentialisation de services alternatifs**
Description : Mise en séquence de deux services initialement en alternatif
Services à restructurer : S2, S3 Niveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : PA2.5 et PA1.1ou PA1.2




**Contraintes:**

Input(S3) = Output(S2)
Input (S4) = Output(S3) Suppression de la condition

**Figure VI.16.** *Description des patrons de restructuration de services AP6.1- AP6.2*

### VI.2.3.2 Patrons d'alternation ou de parallélisation de services

Ces patrons sont utilisés pour réaliser l'opération inverse de la séquentialisation. En effet, il s'agit de mettre en alternatif ou en parallèle deux services s'exécutant initialement en séquence. Ces patrons induisent forcément la création d'un bloc alternatif ou parallèle. Par exemple, sur la figure VI.17, le patron PA6.3 correspondant à l'opération d'alternation inclusive, est réalisé via la création d'un bloc alternatif Alt(S2, S3) ensuite la suppression du service dupliqué S2 ou S3.

- **Exemple d'utilisation d'un patron d'adaptation de contrôle de flux**

Si nous reprenons le modèle de WFIO de la figure VI.5, une autre adaptation de ce modèle pourrait consister en une réorganisation du flux de contrôle dans le processus en mettant les services S13 et S14 (correspondant aux phases « Etablir les conditions mécaniques » et « Etablir les conditions thermiques » en séquentiel au lieu de les mettre parallèle, car dans certains cas, il pourrait exister une dépendance entre les conditions mécaniques et les conditions thermiques du circuit.

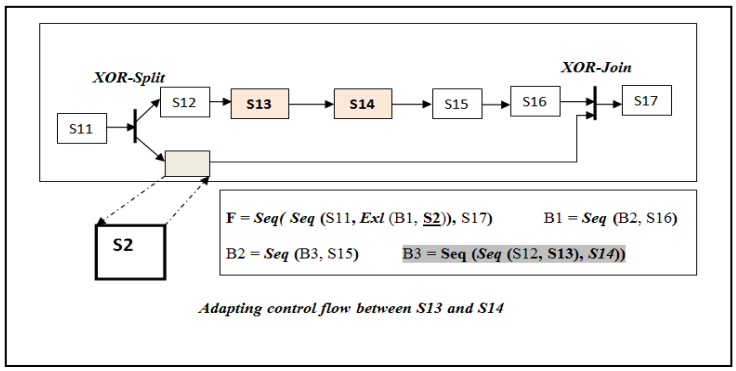
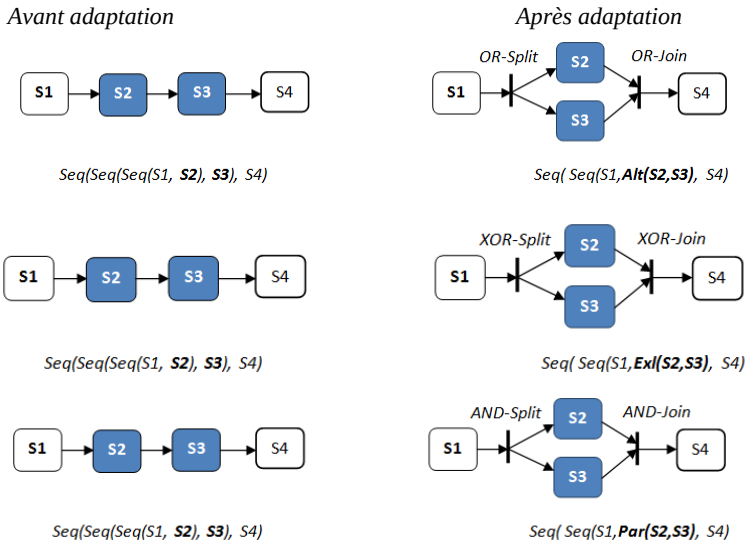



**Figure VI.17.** *Exemple d'adaptation du contrôle de flux - Séquentialisation de services*

La figure suivante décrit les patrons d'alternation et de parallélisation de services référencés par PA6.3, PA6.4 et PA6.5.

Référence du patron : **PA6.3, PA6.4 / PA6.5 (ou AP6.3, AP6.4/AP6.5)**
Nom du patron : **Alternation de services / Parallélisation de services**
Description : Mise en alternatif (inclusif ou exclusif) ou en parallèle de deux services initialement en séquence
Services à restructurer : S2, S3 Niveau d'abstraction : « Activité » ou « Sous-processus »
Patrons utilisés : PA1.6 et PA2.1 ou PA1.7 et PA2.1 ou PA1.8 et PA2.1




**Contraintes:**

Input(S3) = Output(S1)
Input (S4) = Output(S2) U Output (S3)
Input (S4) = Output(S2) ou bien Input (S4) = Output (S3) (pour le patron PA6.4)
Ajout d'une condition pour les patrons PA6.3 et PA6.4

**Figure VI.18.** *Description des patrons de restructuration de services AP6.3- AP6.4- AP6.5*

### VI.2.4 Adaptation des interactions

Les opérations d'adaptation décrites dans les deux catégories précédentes, s'appliquent indifféremment aux modèles de WFIO obéissant à l'un ou l'autre des patrons de coopération (PCBS) décrits dans le chapitre V. En effet, ces adaptations affectent les services ou leur enchainement dans des fragments de WF locaux sans impact sur la structure d'interaction entre les fragments formant le WFIO, pourvu que la logique métier ne soit pas affectée localement pour maintenir la cohérence du WFIO global. Par contre, les adaptations des interactions sont des adaptations spécifiques à chaque patron de coopération. Dans une « Exécution chainée », on pourrait avoir besoin de changer l'ordre d'exécution des fragments (ou supprimer ou encore remplacer un fragment), ce qui induit une modification dans la structure d'interaction sans pour autant affecter l'aspect coopération ou la fonctionnalité globale du processus. Dans une « Sous-traitance », il peut arriver de supprimer un point d'interaction si le service ayant été sous-traité auparavant, devient disponible dans le WF principal ou encore si on veut modifier le service à sous-traiter. Dans un « Transfert de cas (étendu) », une modification de la politique de transfert affecte la structure d'interaction en ajoutant, supprimant ou modifiant les points de transfert. Dans

un modèle de WFIO « Faiblement couplé », le protocole de communication peut être adapté, ainsi les points d'interaction peuvent subir des opérations d'ajout, de suppression ou de modification. Dans un WFIO obéissant au « Partage de charge », les interactions sont de type synchrone entre l'orchestrateur central et les services fournis par les partenaires impliqués dans la coopération. L'ajout d'une interaction est lié à l'ajout d'un service fourni par un des partenaires du groupe déjà impliqué dans la coopération, autrement il s'agira d'une évolution. Dans la suite, nous décrivons les différents patrons d'adaptation des interactions spécifiques à chaque patron de coopération.

Dans la suite, les patrons d'adaptation des interactions seront référencés par *AP7.x.y* où *x* représente le numéro du patron de coopération auquel obéit le modèle de WFIO à adapter (par exemple 2 pour le patron « Exécution chainée » - PCBS2) et *y* représente un numéro d'un patron d'adaptation dans la catégorie AP7.*x* lié à un PCBS*x* donné.

### VI.2.4.1 Adaptation des interactions dans le patron « Partage de charge »

L'adaptation des interactions dans un WFIO obéissant au patron « Partage de charge » consiste à apporter des changements aux points d'interaction entre l'orchestrateur central du WFIO et les différents services (fragments de WF) impliqués dans la coopération. Ceci peut être induit par la substitution d'un fragment de WF, fourni par un même partenaire existant déjà dans le groupe, en vue d'une adaptation corrective ou perfective. Une autre situation d'adaptation pourrait consister en une restructuration du processus en modifiant l'enchainement des services qui implique forcément des changements au niveau des points d'interaction, puisque toutes les invocations de services passent par l'orchestrateur central. Pour ce type d'adaptation, le niveau d'abstraction requis peut être de type « Sous-processus » ou « Processus » selon la localisation des points d'interaction entre l'orchestrateur de services et les différents fragments de WF.

### VI.2.4.2 Adaptation des interactions dans le patron « Exécution chainée »

L'adaptation des interactions dans un WFIO obéissant au patron « Exécution chainée », peut être réalisée selon trois configurations différentes : (i) permutation de fragments, (ii) substitution d'un fragment par un autre, (iii) suppression de fragments. La figure VI.19 ci-dessous décrit les patrons d'adaptation des interactions pour des modèles de WFIO obéissant au patron de coopération « Exécution chainée » (PCBS2). Remarquons que les patrons d'adaptation des interactions pour l'exécution chainée référencés par PA7.2.x, utilisent les patrons d'insertion et de suppression de services PA1.1, PA1.2 et PA2.1 et les fonctions d'orchestration adaptées sont exprimées à l'aide de l'opérateur **Sub** de substitution et l'opérateur **Rem** de suppression.

#### A- Permutation de fragments

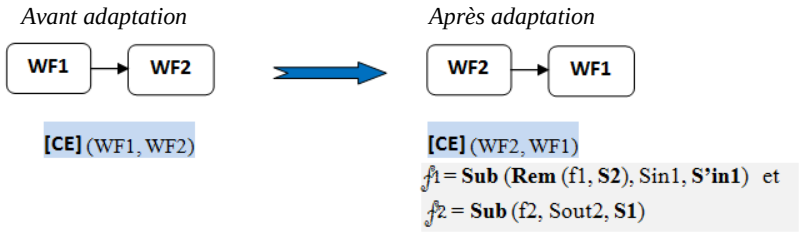
Une permutation de deux fragments de WF impliqués dans un WFIO global induit une adaptation des points d'interaction dans le processus. Prenons l'exemple d'un WFIO lié au domaine médical et impliquant trois partenaires, un centre d'accueil des patients, un laboratoire d'analyses médicales et un centre de radiologie. Supposons que le WFIO initial soit composé de la séquence (WF1, WF2, WF3) respectivement pour l'accueil, les examens biologiques et les examens radiologiques, on pourrait envisager le cas de certains patients qui subissent d'abord des examens radiologiques ensuite des examens biologiques. Ainsi, le modèle de WFIO serait adapté selon la séquence (WF1, WF3, WF2).

**Remarque**

Pour la description des patrons dans la suite, on considère que WFi est un fragment de WF encapsulé dans un service Si, fourni par un partenaire i. *fi* est la fonction d'orchestration

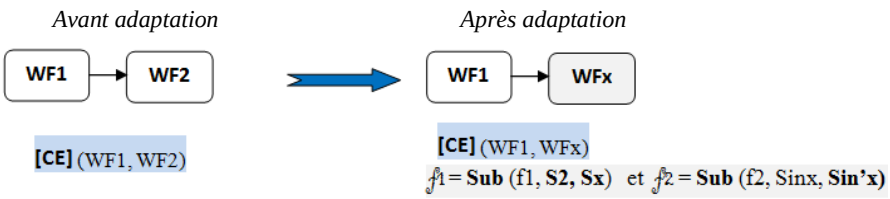
correspondante. *Sini* et *Souti* sont des services d'interaction servant à la réception des données (Input) et le renvoi de résultats (Output) par les services correspondants.

Référence du patron : **PA7.2.1(ou AP7.2.1)**
Nom du patron : **Permutation de fragments dans une « Exécution chainée »**
Description : Changer l'ordre d'exécution des services dans un WFIO obéissant au patron de coopération « Exécution chainée »
Fragments à permuter : WF1, WF2 Niveau d'abstraction : « Processus »
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1




**Contrainte:** Input(S1) = Output(S2)

Référence du patron : **PA7.2.2(ou AP7.2.2)**
Nom du patron : **Substitution de fragments dans une « Exécution chainée »**
Description : Remplacer un fragment de WF par un autre dans un WFIO obéissant au patron de coopération « Exécution chainée »
Fragments à remplacer : WF2 Fragment remplaçant : WFx Niveau d'abstraction : « Processus »
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1




**Contraintes:** Input(Sx) = Output(S1)

Référence du patron : **PA7.2.3 (ou AP7.2.4)**
Nom du patron : **Suppression d'un fragment dans une « Exécution chainée »**
Description : Supprimer un fragment de WF dans un WFIO obéissant au patron de coopération « Exécution chainée »
Fragments à supprimer : WF2 Niveau d'abstraction : « Processus »
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1

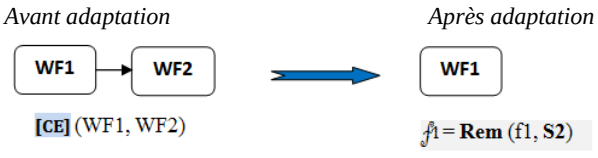



**Figure VI.19.** *Description des patrons d'adaptation des interactions dans une « Exécution chainée »*

### B- Substitution d'un fragment par un autre

La substitution d'un fragment de WF impliqué dans un WFIO global peut avoir lieu lorsque le fragment de WF devient défaillant ou dans le cas d'une rupture de contrat avec un partenaire donné. Ceci induit une adaptation des points d'interaction entre le nouveau fragment, celui qui le précède (éventuellement) et celui qui le suit (éventuellement). On pourrait envisager dans l'exemple du WFIO précédent, une rupture de contrat avec le centre de radiologie et donc la substitution de WF3 par WF4, un autre WF implémentant le processus d'examens radiologiques fourni par un nouveau partenaire.

### C- Suppression d'un fragment

La suppression d'un fragment de WF impliqué dans un WFIO peut survenir suite à une rupture de contrat avec un partenaire donné. Concrètement, le WFIO sera amputé d'une phase de traitement (logiquement au début ou à la fin mais pas au milieu dans le cas où il implique plus de deux partenaires).

## VI.2.4.3 Adaptation des interactions dans le patron « Sous-traitance »

L'adaptation des interactions dans une sous-traitance peut consister en deux opérations principales : (i) annulation d'une sous-traitance, (ii) changement de sous-traitance.

La figure suivante décrit les patrons d'annulation d'une sous-traitance et de changement de sous-traitance références dans l'ordre, par PA7.3.1 et PA7.3.2.

Référence du patron : **PA7.3.1(ou AP7.3.1)**
Nom du patron : **Annulation d'une sous-traitance**
Description : Remplacer un WF secondaire par un service interne dans un WFIO obéissant au patron de coopération « Sous-traitance »
Service à remplacer : S2 Service remplaçant : S1i
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1 Niveau d'abstraction : « Sous-processus »

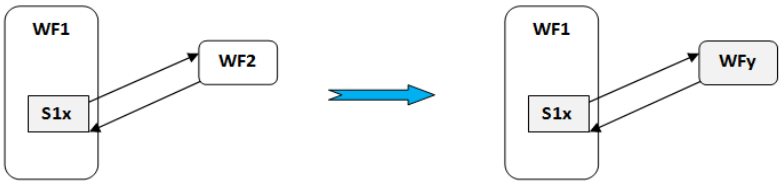
*Avant adaptation* *Après adaptation*


**[SC]** (WF1, (S1x, WF2))
$f1 = \mathbf{Sub}$ (f1, S1x, **S2**)
et $f2 = \mathbf{Sub}$ (f2, Sin2, **S'in2**)

$f1' = \mathbf{Sub}$ ($f1$, S2, **S1i**)

Référence du patron : **PA7.3.2 (ou AP7.3.2)**
Nom du patron : **Changement de sous-traitance**
Description : Remplacer un service externe par un autre service externe dans un WFIO obéissant au patron de coopération « Sous-traitance »
Service à remplacer : S2 Service remplaçant : Sy
Patrons utilisés : PA1.1 ou PA1.2 et PA2.1 Niveaux d'abstraction : « Sous-processus » / « Processus »

*Avant adaptation* *Après adaptation*




**[SC]** (WF1, (S1x, WF2))
$f1 = \mathbf{Sub}$ (f1, S1x, **S2**)
et $f2 = \mathbf{Sub}$ (f2, Sin2, **S'in2**)

**[SC]** (WF1, (S1x, WFy))
$f1' = \mathbf{Sub}$ ($f1$, S1x, **Sy**)
et $f2' = \mathbf{Sub}$ ($f2$, Siny, **S'iny**)

**Contraintes:** Input (S3) = Input (S1x) et Output (Sy) = Output (S1x)

**Figure VI.20.** *Description des patrons d'adaptation des interactions dans une « Sous-traitance »*

**A- Annulation d'une sous-traitance** (substitution d'un service externe par un service interne)

L'annulation d'une sous-traitance peut avoir lieu si le partenaire principal décide d'implémenter localement un service qui était initialement sous-traité chez un partenaire secondaire et donc de rompre le contrat de partenariat, une situation assez fréquente dans le domaine du B2B. Dans ce cas, on pourrait retomber sur un seul WF local (celui du partenaire principal) si la seule sous-traitance dans le processus est celle qui vient d'être annulée. L'annulation d'une sous-traitance induit la suppression du point d'interaction qui reliait le WF du partenaire principal et le WF du partenaire secondaire. Cette configuration est décrite par le patron PA7.3.1 dans la figure VI.20,

**B- Changement de la sous-traitance** (substitution d'un service externe par un autre service externe)

Une situation assez répandue aussi dans une coopération de type sous-traitance est le changement de partenariat. En effet, une rupture de contrat peut avoir lieu entre un partenaire principal et un partenaire secondaire pour être remplacée par un nouveau contrat de partenariat entre le partenaire principal et un nouveau partenaire secondaire. Dans ce cas, il s'agit d'une substitution d'un service externe par un autre service externe dans le processus principal.

### VI.2.4.4 Adaptation des interactions dans le patron « Transfert de cas (étendu) »

Dans le patron de coopération « Transfert de cas (étendu) » les adaptations des interactions sont liées au changement de la politique de transfert. Ainsi, trois types d'adaptations peuvent être envisagés : (i) ajout d'un point de transfert, (ii) suppression d'un point de transfert, (iii) modification d'un point de transfert comme le montre le tableau TAB VI.1.

**TAB VI.1.** *Description des patrons d'adaptation des interactions pour le « Transfert de cas »*

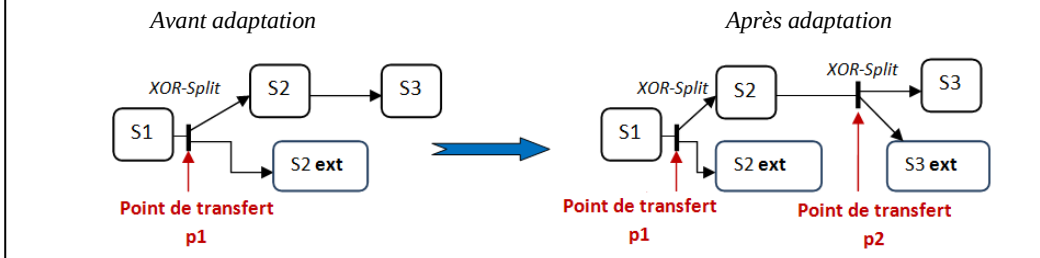
| **Patrons d'adaptation des interactions dans un « Transfert de cas » - PA7.4.y** | | | |
|---|---|---|---|
| **Référence du patron** | **Description du patron** | **Spécifications** | **Patrons utilisés** |
| **PA7.4.1** | Ajouter un point de transfert | - **Créer un sous-processus commençant au nouveau point de transfert**<br>- **Ajouter une activité alternative dans le processus**<br>Spécifier la condition de transfert<br>Ajouter une activité "invoke" dans le bloc alternatif<br>(appel du service externe encapsulant le sous-processus)<br>Ajouter une activité "exit" dans le processus invocateur<br>- **Mettre à jour le flux de données dans le processus** | PA1.7 |
| **PA7.4.2** | Supprimer un point de transfert | - **Supprimer l'invocation du service externe dans le bloc alternatif correspondant au point de transfert**<br>- **Mettre à jour le flux de données dans le processus** | PA2.5<br>PA1.1<br>PA1.2 |
| **PA7.4.3** | Modifier un point de transfert | - **Mettre à jour la condition de transfert associée au point de transfert**<br>et/ou<br>- **Mettre à jour l'action de transfert**<br>Remplacer l'invocation d'un service externe du bloc alternatif par l'invocation d'un autre service externe | PA3.2 |

Le tableau VI.1 fournit quelques éléments de spécification pour les patrons d'adaptation des interactions dans le « Transfert de cas ». Ceux-ci reviennent à la création de sous-processus et de blocs exclusifs dans le cas d'un ajout de point de transfert, la suppression du bloc exclusif dans le cas d'une suppression de point de transfert et enfin, la substitution d'un service externe par un autre service externe dans un bloc exclusif ou simplement la mise à jour de la condition de transfert dans le cas d'une modification de point de transfert à modifier.

### A- Ajout d'un point de transfert

L'ajout d'un point de transfert pourrait être nécessaire pour partager l'exécution d'une partie du processus entre l'invocation d'un service local et l'invocation du service externe correspondant afin d'équilibrer la charge entre les partenaires, par exemple. La figure VI.21 décrit le patron d'ajout d'un point de transfert et la figure VI.22 donne un exemple d'utilisation de ce patron sur un modèle de WFIO obéissant au patron « Transfert de cas ».

Référence du patron : **PA7.4.1 (ou AP7.4.1)**
Nom du patron : **Ajout d'un point de transfert**
Description : Ajouter un point de transfert dans un WFIO obéissant au patron de coopération « Transfert de cas»
Point de transfert: "Après" S2 Niveau d'abstraction : « Sous-processus »
Patrons utilisés : PA1.7 (patron de création de bloc exclusif)




**[CT]** (WF1, WF2, ℟) = < S, F >
avec F = {$f_1$, $f_2$}

**[CT]** (WF1, WF2, ℟')
$f_1$' = Ins ($f_2$, B3, S2, "Après")
$f_2$' = Ins ($f_2$, B3, S2, "Après")
**B3 =** If (C3) invoke **S3ext** else invoke **S3**

**Contrainte:** Input (S3) = Input (S3 ext) = Output (S2)

**Figure VI.21.** *Description du patron « Ajout d'un point de transfert »*

Le patron AP7.4.1 d'ajout de point de transfert décrit sur la figure VI.21, consiste à créer un nouveau sous-processus encapsulé dans un service (S3) et adapter la fonction d'orchestration par insertion d'un bloc exclusif en utilisant le patron PA1.7. L'ensemble des règles de transfert est aussi mis à jour.

### B- Suppression d'un point de transfert

La suppression d'un point de transfert peut avoir lieu dans le cas où un partenaire P qui disposait de ressources adéquates pour la prise en charge d'une partie donnée du processus, se retrouve dans une situation d'indisponibilité de ces ressources. On pourrait donc, envisager un changement dans une règle de transfert de telle manière que le partenaire P ne soit plus concerné par le transfert de l'instance au niveau du point de transfert en question qui devra par conséquent, être supprimé.

### C- Modification d'un point de transfert

Un autre cas d'adaptation pourrait consister juste en une modification de la règle de transfert, c'est-à-dire la condition de transfert ou l'action de transfert. Une modification de l'action de transfert implique une substitution d'un service externe par un autre service externe.

A titre d'exemple, la figure suivante illustre l'exemple d'un WFIO obéissant au patron transfert de cas qui subit une adaptation de type « Ajout d'un point de transfert ». En effet, dans un souci d'équilibrage de charge entre les partenaires, il est nécessaire d'adapter le processus en ajoutant un bloc exclusif pour modéliser le scénario « Si surcharge de production au niveau d'un partenaire, transférer l'instance à l'autre partenaire ».

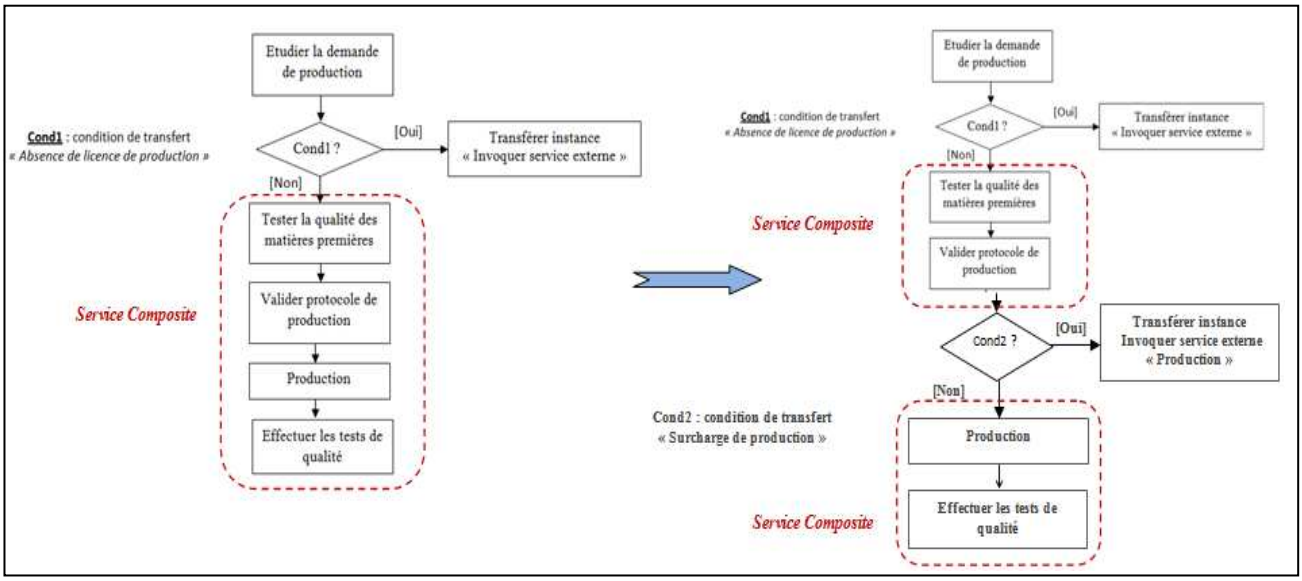



**Figure VI.22.** *Exemple d'adaptation des interactions « Ajout d'un point de transfert »*

### VI.2.4.5 Adaptation des interactions dans le patron « Faiblement couplé »

Les adaptations des interactions dans le patron « Faiblement couplé » concernent les modifications qui peuvent être apportées au protocole de communication. Celles-ci sont aussi de trois types : (i) Ajout d'un point d'interaction, (ii) Modification d'un point d'interaction, (iii) Suppression d'un point d'interaction. Le tableau VI.2 fournit quelques éléments de spécification pour les patrons d'adaptation des interactions dans le « Faiblement couplé ». Ceux-ci consistent globalement à ajouter ou supprimer des services d'interaction (Service_send et Service_receive) et utilisent donc les patrons d'ajout PA1.x et de suppression PA2.x de services.

**TAB VI.2.** *Description des patrons d'adaptation des interactions pour le patron « Faiblement couplé »*

| **Patrons d'adaptation des interactions dans le patron « Faiblement couplé » - PA7.6.y** | | | |
|---|---|---|---|
| **Référence du patron** | **Description du patron** | **Spécifications** | **Patrons utilisés** |
| **PA7.6.1** | Ajouter un point d'interaction | - Insérer un service d'interaction « Service_send » dans le processus invocateur<br>- Insérer un service d'interaction correspondant « Service_receive » dans le processus invoqué<br>- Restructurer le processus en utilisant les règles R6.3, R6.4 et R6.5 (voir chapitre V).<br>- Mettre à jour le flux de données dans le processus | PA1.X |
| **PA7.6.2** | Supprimer un point d'interaction | - Supprimer le service d'interaction « Service_send » dans le processus invocateur<br>- Supprimer le service d'interaction correspondant « Service_receive » dans le processus invoqué<br>Si l'interaction est de type « Two-way »<br>- Supprimer les services d'interaction « Service_send » et « Service-receive » correspondant à la réponse de la requête supprimée.<br>- Restructurer le processus en utilisant les règles R6.3, R6.4 et R6.5 (voir chapitre V).<br>- Mettre à jour le flux de données dans le processus | PA2.X |
| **PA7.6.3** | Modifier un point d'interaction | - Mettre à jour le flux de données entre le processus invocateur et le processus invoqué<br>- Modifier le mode d'interaction (One-way/Two-way) | PA1.X<br>PA2.X |

### A- Ajout d'un point d'interaction

L'ajout d'un point d'interaction peut avoir lieu si au cours du temps, un échange de données, à un point donné du processus, devient nécessaire entre deux fragments de WF en coopération. Ce type d'adaptation induira l'ajout d'un "Service-send" dans le fragment de WF invocateur et d'un "Service-receive" dans le fragment de WF invoqué, si l'interaction est de type "One-way". Dans le cas où l'interaction est de type "Two-way", l'ajout de deux autres services "Service-send" et "Service-receive" est nécessaire mais dans le sens inverse (invocation/réponse asynchrone). La figure VI.23 décrit le patron d'ajout d'un point d'interaction de type "One-way" référencé par AP7.6.1 dans un WFIO obéissant au PCBS6.

### B- Suppression d'un point d'interaction

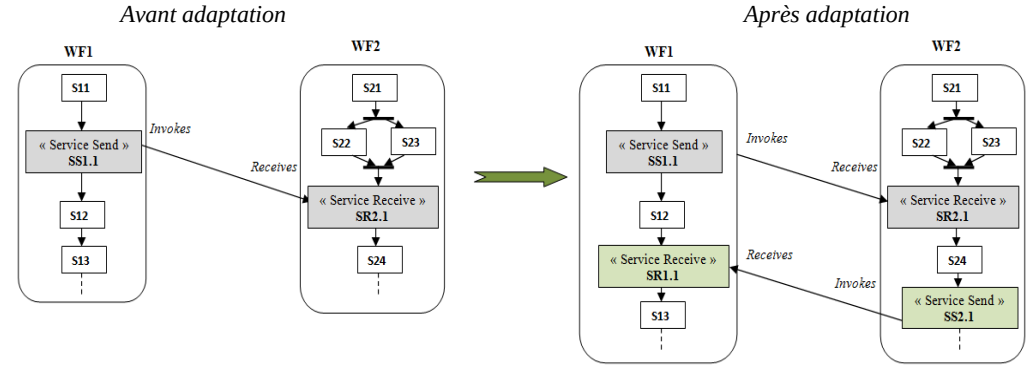
La suppression d'un point d'interaction peut avoir lieu pour éliminer une interaction entre deux fragments de WF si l'échange de données lié au point d'interaction en question, n'est plus nécessaire à la progression du processus (par exemple éliminer une notification). La suppression d'un point d'interaction de type "One-way" se traduit par la suppression du "Service-send" lié à ce point d'interaction ainsi que la suppression du "Service-receive" correspondant. Dans le cas d'une interaction de type "Two-way", il faudra en plus supprimer les "Service-send" et "Receive-receive" correspondant à la réponse dans le sens inverse.

### C- Modification d'un point d'interaction

La modification d'un point d'interaction P consiste à modifier les données échangées entre deux fragments de WF, au niveau de ce point. Il s'agira de remplacer les services d'interaction "Service-send" et "Service-receive" correspondant au point d'interaction P par d'autres services

d'interaction qui fournissent et récupèrent les nouvelles données attendues ou alors la modification du mode d'interaction qui induit soit une insertion ou une suppression de services d'interaction.

Réference du patron: **PA7.6.1** (ou **AP7.6.1)**
Nom du patron: **Ajout d'un point d'interaction de type "One-Way"**
Description: Insérer un service d'interaction "Service-Send" et le "Service-Receive" correspondant dans un modèle de WFIO obéissant au patron « Faiblement couplé »
Services à ajouter: SS2.1 et SR1.1 Niveau d'abstraction : « Sous-processus »
Patrons utilisés: AP1.1 ou AP1.2




$f1' = Ins(f1, SR1.1, S12, "Avant")$
et
$f2' = Ins(f2, SS2.1, S24, "Après")$

**Contraintes:** Input (SR1.1) = Output (S12)
Output (SR1.1) = Input (S13)
Output (S24) = Input (SS2.1)
Output (SS2.1) = Input (SR1.1)

**Figure VI.23.** *Description du patron d'adaptation « Ajout de point d'interaction »*

## VI.3 Evolutivité et évolution d'un modèle de WFIO

Selon notre vision, l'évolutivité d'un modèle de WFIO mesure sa capacité à supporter l'expansion sur deux aspects qui sont : l'expansion des fonctionnalités et/ou l'expansion de la coopération.

- **Définition VI.7:** Evolution d'un modèle de WFIO

*L'évolution d'un modèle de WFIO est l'opération qui consiste à apporter un changement pour étendre la fonctionnalité globale et/ou la coopération (ouverture à de nouveaux partenaires) du WFIO et qui permet d'obtenir un modèle de WFIO étendu (évolué) à partir d'un modèle de WFIO initial.*

Ainsi, un processus de WFIO évolue s'il peut être étendu à des fonctionnalités additionnelles ou s'il permet l'expansion du nombre de partenaires et des services externes fournis. Notons que les deux perspectives ne sont pas exclusives ; l'expansion de la coopération induit souvent une expansion des fonctionnalités et l'expansion des fonctionnalités peut nécessiter une expansion de la coopération.

- ***Expansion des fonctionnalités***

L'évolution d'un modèle de WFIO selon la perspective *fonctionnalité* peut se faire soit par l'ajout de services (resp. blocs) internes remplissant des fonctionnalités complémentaires, soit par l'ajout de services externes fournis par de nouveaux partenaires (suite à une expansion de la coopération), ou encore par substitution de services (resp. blocs) par d'autres services qui fournissent des

fonctionnalités additionnelles ; on dit que le processus évolue en fonctionnalité. Ainsi, si l'expansion des fonctionnalités est induite par une expansion de la coopération, on parle d'évolution *inter-organisationnelle*. Par contre, si l'expansion des fonctionnalités a lieu sans impact sur l'aspect coopération, on parle d'évolution intra-organisationnelle. Dans tous les cas, pour effectuer de telles opérations sur les modèles de processus WFIO, on peut se référer aux opérations d'adaptation explicitées dans la section VI.2 de ce chapitre.

- ***Expansion de la coopération***

L'évolution selon la perspective de *coopération* se fait à travers l'ouverture du WFIO à de nouveaux partenaires. L'ajout de partenaires et donc de services externes peut préserver le patron de coopération du WFIO initial ou donner lieu à la combinaison de deux patrons de coopération différents. L'expansion de la coopération avec préservation du patron de coopération est supportée par le concept de patron de coopération *généralisé*. La deuxième configuration, celle de l'expansion par combinaison de patrons de coopération différents est supportée par le concept de patron de coopération *composite* qui définit un aspect de la réutilisation de modèles de WFIO pour l'évolution d'autres modèles de WFIO.

### VI.3.1 Patrons de coopération généralisés

Les patrons de coopération que nous avons introduits au chapitre V ont été décrits pour une coopération entre deux partenaires métiers qui fournissent chacun, son fragment de WF. La coopération pourrait être étendue en impliquant d'autres partenaires, tout en respectant le même patron de coopération ; on parle alors de patron de coopération *généralisé*.

- **Définition VI.8 :** Patron de coopération généralisé

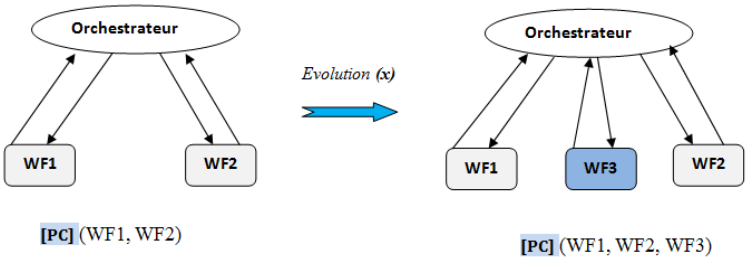
*Un patron de coopération généralisé est un patron qui définit une coopération entre trois partenaires ou plus. Cette catégorie de patrons supporte l'évolution d'un modèle de WFIO obéissant à un PCBS donné.*

Dans la suite, nous décrivons les patrons généralisés liés à chacun des patrons de coopération (PCBS) considérés dans le cadre de ce travail. Les patrons d'évolution sont référencés par **PEx.y** où PE désigne « Patron d'Evolution », x correspond au numéro du patron de coopération de base (2 pour « l'Exécution chainée », 3 pour la « Sous-traitance »,…) et y une lettre alphabétique qui désigne un type d'évolution du patron de coopération x.

#### VI.3.1.1 Expansion du « Partage de charge »

L'expansion du partage de charge peut se faire par l'introduction de nouveaux services (encapsulant des fragments de WF) fournis par de nouveaux partenaires dans le processus de WFIO global. Ces nouveaux services seront rattachés aux services déjà existants dans la fonction d'orchestration globale, via les opérateurs de contrôle de flux adéquats afin de définir leur enchainement avec les services initialement existants. L'ajout de nouveaux services peut se faire de manière séquentielle, alternative ou parallèle, conformément à la logique métier souhaitée dans le WFIO étendu. La figure VI.24 montre un patron d'expansion générique du « Partage de charge » référencé par EP1.x; le patron étant générique car nous n'avons pas la structure exacte du processus de WFIO initial. Il utilise selon les cas, les patrons d'ajout de services PA1.x puisqu'il s'agit d'insérer des services en séquence, en parallèle ou en alternatif dans une fonction d'orchestration globale.

Référence du patron : **PE1.x (ou EP1.x)**
Nom du patron : **Evolution (x) du partage de charge**
Description : Ajouter un fragment de WF fourni par un nouveau partenaire dans un WFIO obéissant au patron « Partage de charge – PCBS1 »
Fragment à ajouter: WF3 Niveau d'abstraction : « Processus » ou « Sous-processus »
Patrons utilisés : PA1.x (Ajout de service)




[PC] (WF1, WF2)

[PC] (WF1, WF2, WF3)

Contraintes: dépendent de la structure du WFIO global.

**Figure VI. 24.** *Description du patron d'évolution générique PE1.x - Expansion du Partage de charge*

Reprenons l'exemple de réservation d'un voyage déjà décrit au chapitre V, impliquant deux partenaires métiers : une compagnie aérienne et un hôtel. Un scénario d'expansion de la coopération dans ce cas, pourrait consister à étendre le WFIO à un troisième partenaire fournissant un WF « Location de voitures ». Ce dernier pourra être invoqué simultanément avec le WF « Réservation d'hôtel » selon le modèle de WFIO décrit dans la figure VI.25. Dans ce cas, il s'agit d'un ajout avec création de bloc parallèle en utilisant le patron d'adaptation PA1.5 (patron de création d'un bloc parallèle).

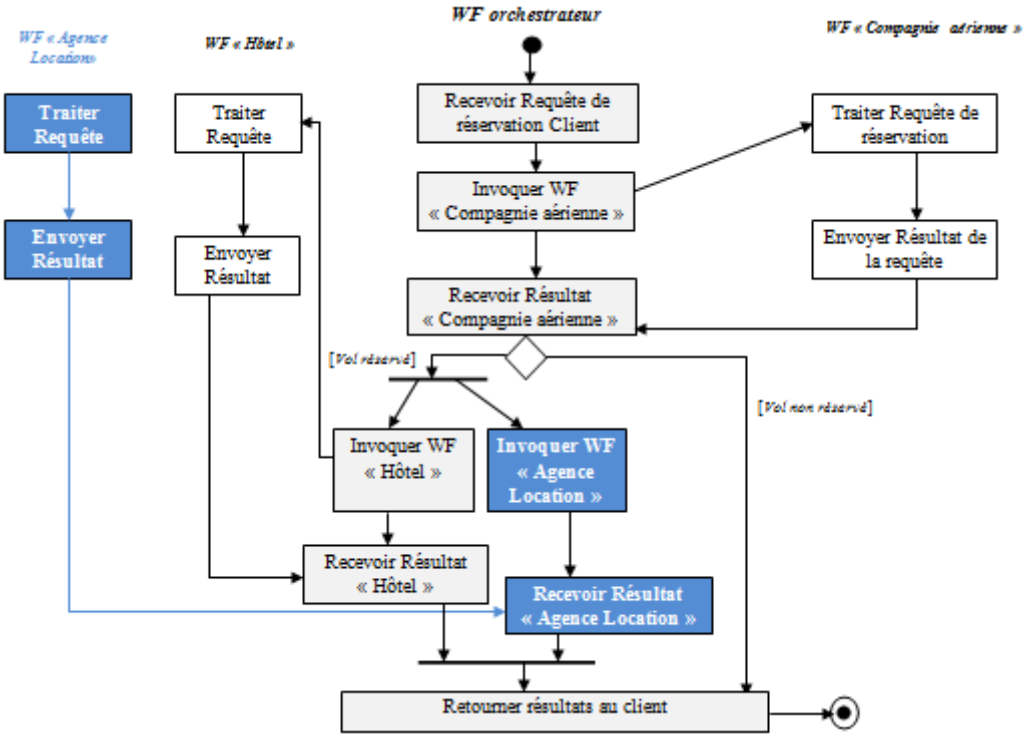



**Figure VI.25**. *Exemple d'expansion du patron « Partage de charge » : WFIO « Réservation de voyage »*

### VI.3.1.2 Expansion de « l'Exécution chainée »

L'exécution chainée peut être étendue dans l'un des cas suivants (Boukhedouma et al., 2012d):

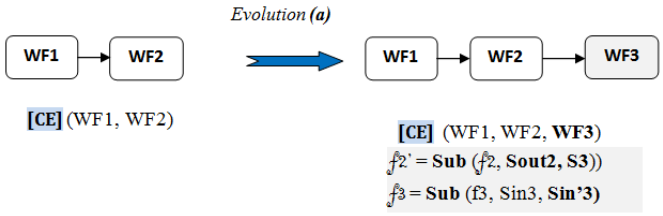
a) Ajouter à la séquence de services, un nouveau service externe (encapsulant un WF) fourni par un nouveau partenaire dans le but d'une expansion des fonctionnalités ou l'ajout d'une phase intermédiaire au processus.

b) Remplacer un service de la séquence par un bloc exclusif composé de deux nouveaux services fournis par deux partenaires. Dans tous les cas, les instances suivent exclusivement l'une des deux branches et on se trouve toujours dans un WFIO obéissant au patron « Exécution chainée ».

Partant d'un WFIO composé initialement d'une séquence de deux processus *WF1, WF2*. La figure VI.26 ci-dessous illustre les deux évolutions su-décrites. On suppose qu'un workflow (service composite) *WFx* du partenaire *x* est composé d'une séquence *Sinx, Sx* (le service métier composite) et *Soutx*. Idem pour le *WFy* du partenaire *y*.

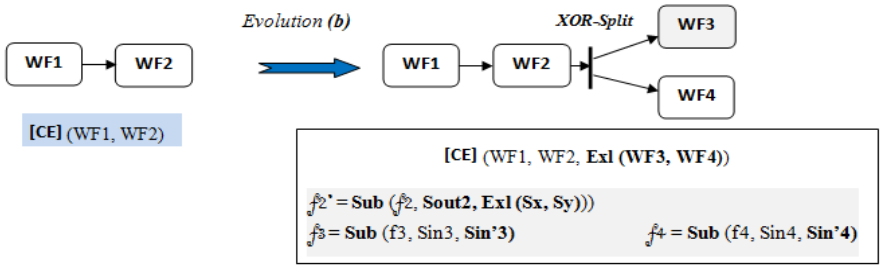
L'ajout d'un nouveau fragment de WF dans un modèle de WFIO obéissant au patron de coopération « Exécution chainée » peut être utile pour compléter le processus de WFIO et l'étendre à une phase de traitement supplémentaire.

Référence du patron : **PE2.a (ou EP2.a)**
Nom du patron : **Evolution (a) de l'exécution chainée**
Description : Ajouter un fragment de WF fourni par un nouveau partenaire dans un WFIO obéissant au patron « Exécution chainée – PCBS2 »
Point d'ajout : ″Après″ WF2 Fragment à ajouter: WF3 – niveau d'abstraction « Processus »
Patrons utilisés : PA1.1 ou PA1.2




Contrainte: Output (WF2) = Input (WF3)

Référence du patron : **PE2.b (ou EP2.b)**
Nom du patron : **Evolution (b) de l'exécution chainée**
Description : Ajouter de fragments exclusifs de WF fourni par de nouveaux partenaires dans un WFIO obéissant au patron « Exécution chainée – PCBS2 »
Point d'ajout : ″Après″ WF2 Fragments à ajouter: Exl (WF3, WF4)
Patrons utilisés : PA1.7 (création de bloc exclusif)




Contrainte: Output (WF2) = Input (WF3) ou bien Output (WF2) = Input (WF4)

**Figure VI. 26.** *Description des patrons d'évolution PE2.a et PE2.b Expansion de « l'Exécution chainée »*

Considérons l'exemple d'un WFIO lié à une chaine de production d'un certain type de produits. Si initialement le WFIO impliquait deux partenaires implémentant respectivement un WF1 « Production de matières premières » et un WF2 « Production de produits semi-finis », ce WFIO pourrait être étendu au WF d'un troisième partenaire, WF3 pour la « Production de produits finis » (voir Figure VI.27). On obtiendra un WFIO obéissant au patron « Exécution chainée » généralisé WFIO = **[CE]** (**[CE]** (WF1,WF2), WF3). La même configuration peut être envisagée dans un WFIO de prise en charge des patients. Si initialement, le WFIO impliquait deux partenaires : un centre d'analyses médicales (WF1 « Analyses médicales ») et un centre médical (WF2 « Diagnostic médical), il peut être étendu à un troisième partenaire pouvant être un hôpital ou une clinique pour l'hospitalisation et le suivi (WF3 « Hospitalisation et suivi »).

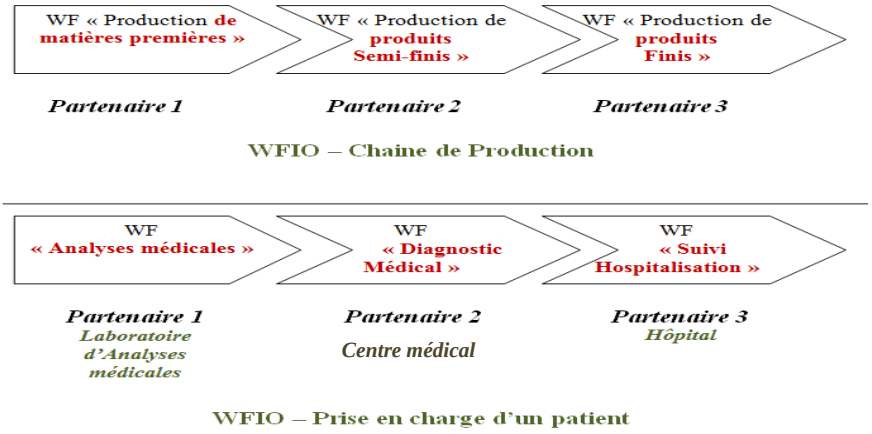



**Figure VI. 27.** *Exemples de WFIO obéissant au patron généralisé « Exécution chainée »*

- **Propriétés de l'opérateur [CE]** « Associativité »

**[CE]** (*WF1*, *WF2*, *WF3*) = **[CE]** (**[CE]** (*WF1*, *WF2*), *WF3*)

**[CE]** (*WF1*, *WF2*, *WF3*) = **[CE]** (*WF1*, **[CE]** (*WF2*, *WF3*))

### VI.3.1.3 Expansion de la « Sous-traitance »

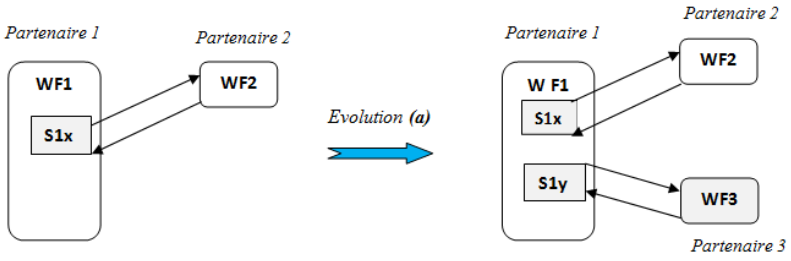
Dans le cas d'une sous-traitance, l'expansion de la coopération peut se faire selon deux scénarios différents : une sous-traitance de *même niveau* ou une sous-traitance *multi-niveaux* (Boukhedouma et al., 2012b). La sous-traitance de même niveau suppose l'existence d'un WFIO composé d'un WF principal WF1 implémenté par *Partenaire 1* et un WF secondaire WF2 implémenté par *Partenaire 2*; le WFIO peut être étendu pour sous-traiter une autre activité du WF principal (WF1) à un troisième partenaire *Partenaire 3* implémentant un autre WF secondaire (WF3). Les schémas d'expansion de la sous-traitance peuvent être résumés dans les trois configurations suivantes :

a) Externalisation de nouveaux services qui consiste à remplacer un service interne par un service externe si on a besoin d'étendre la sous-traitance.
b) Remplacer un service externe par un bloc exclusif composé de deux services *Sx* et *Sy* fournis par deux partenaires tels que le service *Sx* soit invoqué dans certains cas (selon une condition) et dans d'autres cas c'est le service *Sy* qui est invoqué ; ceci dans le but de répondre à de nouvelles contraintes.

c) Remplacer un service externe par un bloc parallèle composé de deux services *Sx* et *Sy* fournis par deux partenaires et devant être exécutés simultanément dans le but d'améliorer le temps de réponse du processus.

Les trois configurations donnent lieu à trois patrons d'évolution référencés par PE3.a, PE3.b et PE3.c. La figure suivante décrit le patron d'évolution PE3.a.

Référence du patron : **PE3.a (ou EP3.a)**
Nom du patron : **Evolution (a) de la sous-traitance**
Description : Sous-traiter un autre service à un nouveau partenaire dans un WFIO obéissant au patron « sous-traitance – PCBS3 »
Service à sous-traiter : S1y WF secondaire: WF3 - niveau d'abstraction « Processus »
Patrons utilisés : PA3.x WF principal : W1- niveau d'abstraction « Sous-processus »




**[SC]** (WF1, (S1x, WF2))
$f1$ = **Sub** (f1, S1x, **S2**)
et $f2$ = **Sub** (f2, Sin2, **S'in2)**

**[SC]** (WF1, (S1x, WF2), **(S1y, WF3)**)
$f1'$ = **Sub** ($f1$, S1y, **S3**)
$f3'$ = **Sub** ($f3$, Sin3, **S'in3)**

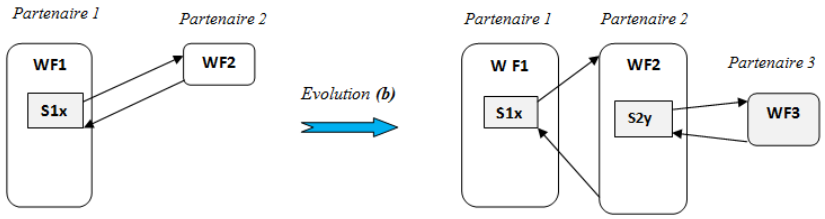
Contraintes: Input (S1y) = Input (WF3) et Output (S1y) = Output (WF3)

**Figure VI. 28.** *Description du patron d'évolution PE3.a – Evolution (a) de la sous-traitance*

- ***Sous-traitance multi-niveaux**:* Une autre configuration d'expansion de la *sous-traitance* est envisagée lorsqu'un partenaire *secondaire* opère à son tour une évolution de son modèle de processus pour sous-traiter une partie de son WF à un autre partenaire secondaire. Ceci implique de nouveaux partenaires et donc de nouveaux services externes et par suite, une expansion de la hiérarchie dans la sous-traitance, on parle de sous-traitance *multi-niveaux*. Dans ce cas, l'opération d'évolution concerne le WF secondaire et consiste à substituer un service interne par un service externe. Donc le WF2 est secondaire par rapport à WF1 et principal par rapport à WF3, la figure VI.29 décrit le patron d'évolution correspondant à une sous-traitance multi-niveaux.

Pour illustrer cette configuration, prenons l'exemple d'un *institut de formation* qui offre des formations à une catégorie de stagiaires dans une spécialité donnée, selon un planning bien déterminé et une procédure d'inscription bien définie. L'institut de formation sous-traite les examens finaux auprès d'un *centre d'examens* qui se charge d'organiser les sessions d'examens et accueillir les stagiaires de différents instituts. Le centre d'examens à son tour, sous-traite l'élaboration de sujets et les évaluations auprès d'un troisième partenaire qui est une *école supérieure* disposant de compétences dans le domaine.

Référence du patron : **PE3.d (ou EP3.d)**
Nom du patron : **Evolution (d) de la sous-traitance – sous-traitance multi-niveaux**
Description : Sous-traiter un service d'un WF secondaire dans un WFIO obéissant au patron « sous-traitance – PCBS3»
Service à sous-traiter : S2y — WF secondaire: WF3 – niveau d'abstraction « Processus»
Patrons utilisés : PA3.x — WF principal : WF2 – niveau d'abstraction « Sous-processus »




**[SC]** (WF1, (S1x, WF2))
$f1$ = **Sub** (f1, S1x, **S2**)
$f2$ = **Sub** (f2, Sin2, **S'in2)**

**[SC]** (WF1, (S1x, **[SC]** (WF2, **(S2y, WF3)**)
$f2'$ = **Sub** ($f2$, S2y, **S3**)
$f3'$ = **Sub** ($f3$, Sin3, **S'in3)**

Contraintes: Input (S2y) = Input (WF3) et Output (S2y) = Output (WF3)

**Figure VI. 29.** *Description du patron d'évolution PE2.d – Sous-traitance multi-niveaux*

Les scénarios d'évolution sus-décrits peuvent être exprimés à travers des opérations de substitution et de décomposition déjà présentées dans la section VI.2.2. La seule différence est que ces opérations concernent des services externes et nécessitent concrètement d'importer les services externes des partenaires.

- **Propriétés de l'opérateur [SC]**

**[SC]** (*WF1*, (S1x, *WF2*), (S1y, *WF3*)) = **[SC]** (*WF1*, (S1y, *WF3*), (S1x, *WF2*))

**[SC]** (*WF1*, (S1x, *WF2*), (S1y, *WF3*)) = **[SC]** (**[SC]** (*WF1*, (S1x, *WF2*)), (S1y, *WF3*))

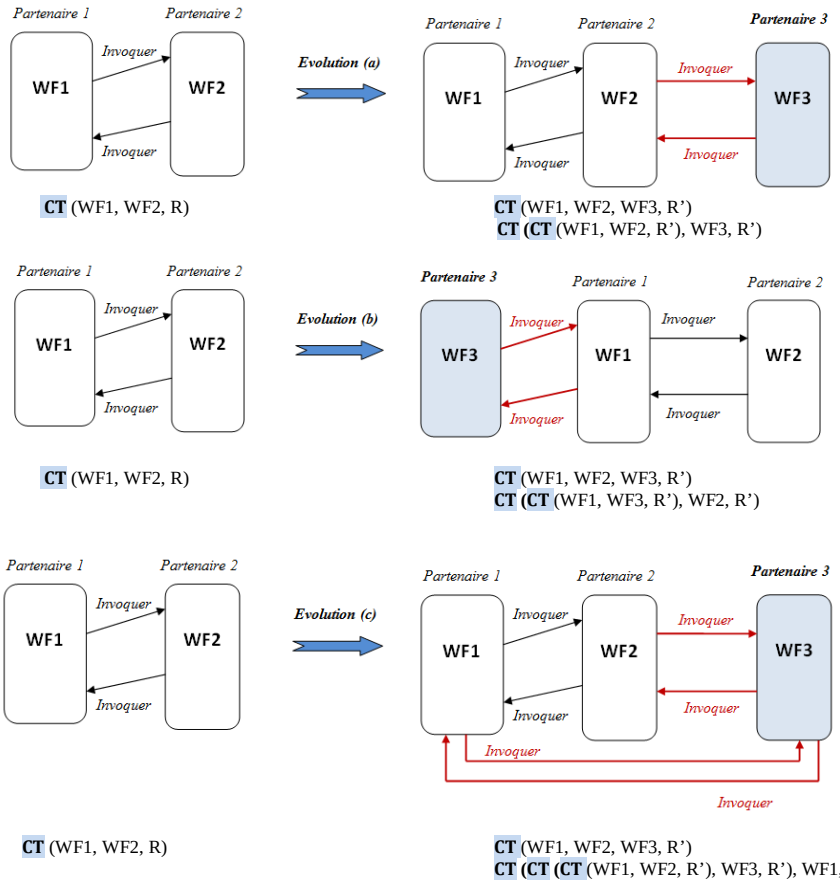
**[SC]** (*WF1*, (S1x, *WF2*), (S1y, *WF3*)) = **[SC]** (**[SC]** (*WF1*, (S1y, *WF3*)), (S1x, *WF2*))

### VI.3.1.4 Expansion du « Transfert de cas (étendu)»

L'expansion de la coopération dans le patron « Transfert de cas » se traduit par la duplication du modèle de WF au niveau d'un nouveau site de partenaire et l'extension (et la modification) de l'ensemble des règles de transfert pour inclure le nouveau partenaire dans la coopération. Ainsi, un transfert d'instance peut avoir lieu entre Partenaire 1 et Partenaire 2 ou entre Partenaire 2 et Partenaire 3 ou encore entre Partenaire 1 et Partenaire 3.

L'ouverture d'un WFIO obéissant au « Transfert de cas » à un nouveau partenaire est souvent due à la surcharge de travail au niveau des partenaires déjà impliqués dans la coopération, induite soit par l'augmentation du nombre de clients potentiels ou alors la non disponibilité de certaines ressources nécessaires à l'exécution des instances. Le nouveau partenaire doit forcément exercer la même activité que les partenaires initiaux : production pharmaceutique par exemple. La figure VI.30 décrit de manière générique les patrons d'expansion de la coopération relatifs au « Transfert de cas (étendu)» référencés par PE4.x; ces patrons consistent à mettre à jour les points de transfert dans le WFIO et utilisent les patrons d'adaptation des interactions PA7.4.x liés au « Transfert de cas (étendu) ».

Référence du patron : **PE4.x (ou EP4.x)**
Nom du patron : **Evolution (x) du transfert de cas (étendu)**
Description : Impliquer un nouveau partenaire dans un WFIO obéissant au patron « Transfert de cas (étendu) – PCBS4 – PCBS5»
Nouveau WF : WF3 Niveau d'abstraction : « Sous-processus »
Patrons utilisés : PA7.4.x (Patrons d'ajout/modification/suppression de points de transfert)




Contraintes: liées à l'ajout/suppression/modification des points de transfert

**Figure VI. 30.** *Description des patrons d'évolution PE4.x – Expansion du transfert de cas (étendu)*

- **Propriétés de l'opérateur [CT]**

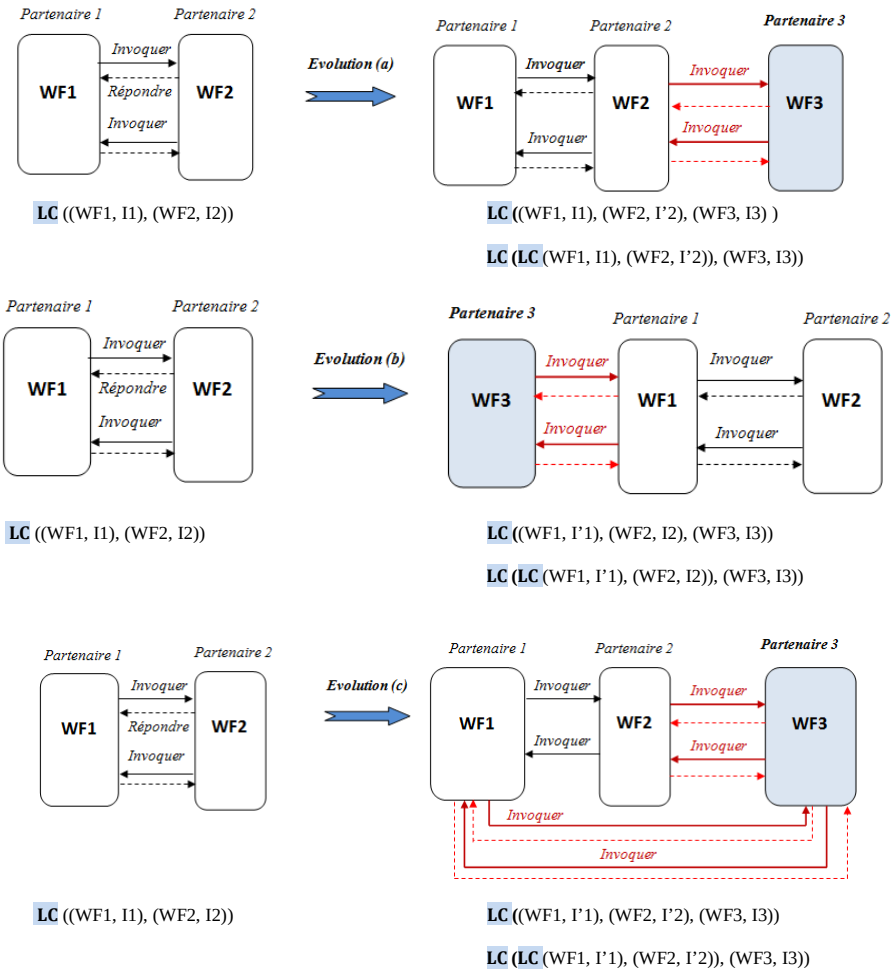
**[CT]** (*WF1*, *WF2*, *WF3*, R)) = **[CT]** (**[CT]** (*WF1*, *WF2*, *R*), WF3, R)

**[CT]** (*WF1*, *WF2*, *WF3*, R)) = **[CT]** (WF1, **[CT]** (*WF2*, *WF3*, *R*), *R*)

### VI.3.1.5 Expansion du patron « Faiblement couplé »

L'expansion de la coopération dans le patron « Faiblement couplé » induit l'extension du modèle de processus existant à un nouveau fragment fourni par un nouveau partenaire impliquant de suite l'ajout de points d'interaction entre les fragments initiaux et le nouveau fragment de WF, pris deux à deux (Boukhedouma et al., 2013b). La figure VI.31 décrit de manière générique les patrons d'expansion de la coopération relatifs au patron « Faiblement couplé» référencés par PE6.x; ces patrons consistent à mettre à jour les points d'interaction dans le WFIO et utilisent les patrons d'adaptation des interactions PA7.6.x liés au patron « Faiblement couplé ».

Référence du patron : **PE6.x (ou EP6.x)**
Nom du patron : **Evolution (x) du faiblement couplé**
Description : Impliquer un nouveau partenaire dans un WFIO obéissant au patron « Faiblement couplé – PCBS6 »
Nouveau WF : WF3 Patrons utilisés : PA7.6.x (Ajout/modification/suppression d'interactions)
Niveau d'abstraction : « Sous-processus »




Contraintes: liées à l'ajout/suppression/modification des points d'interaction

**Figure VI. 31.** *Description des patrons d'évolution PE6.x – Expansion du couplage faible*

- **Propriétés de l'opérateur [LC]**

**[LC]** ((*WF1*, I1), (*WF2*, I2*)*, (*WF3*, I3)) = **[LC]** (**[LC]** ((*WF1*, I1) (*WF2*, *I2))*, (WF3, I3))

**[LC]** ((*WF1*, I1), (*WF2*, I2*)*, (*WF3*, I3)) = **[LC]** ((*WF1*, I1), **[LC]** ((*WF2*, *I2)*, (*WF3*, *I3*)))

- **Exemple d'évolution du patron « faiblement couplé »**

Reprenons l'exemple du WFIO obéissant au patron « Faiblement couplé » présenté au chapitre V. Le WFIO impliquant initialement un client et un fournisseur de produits peut être étendu à un troisième partenaire « Producteur » dont le WF consiste à répondre à une demande de production émanant de la part du « Fournisseur », si la quantité commandée par le client n'est pas disponible en stock. Le processus de WFIO étendu est représenté dans la figure VI.32, il est obtenu conformément au patron d'évolution PE6.a présenté dans la figure VI.31.

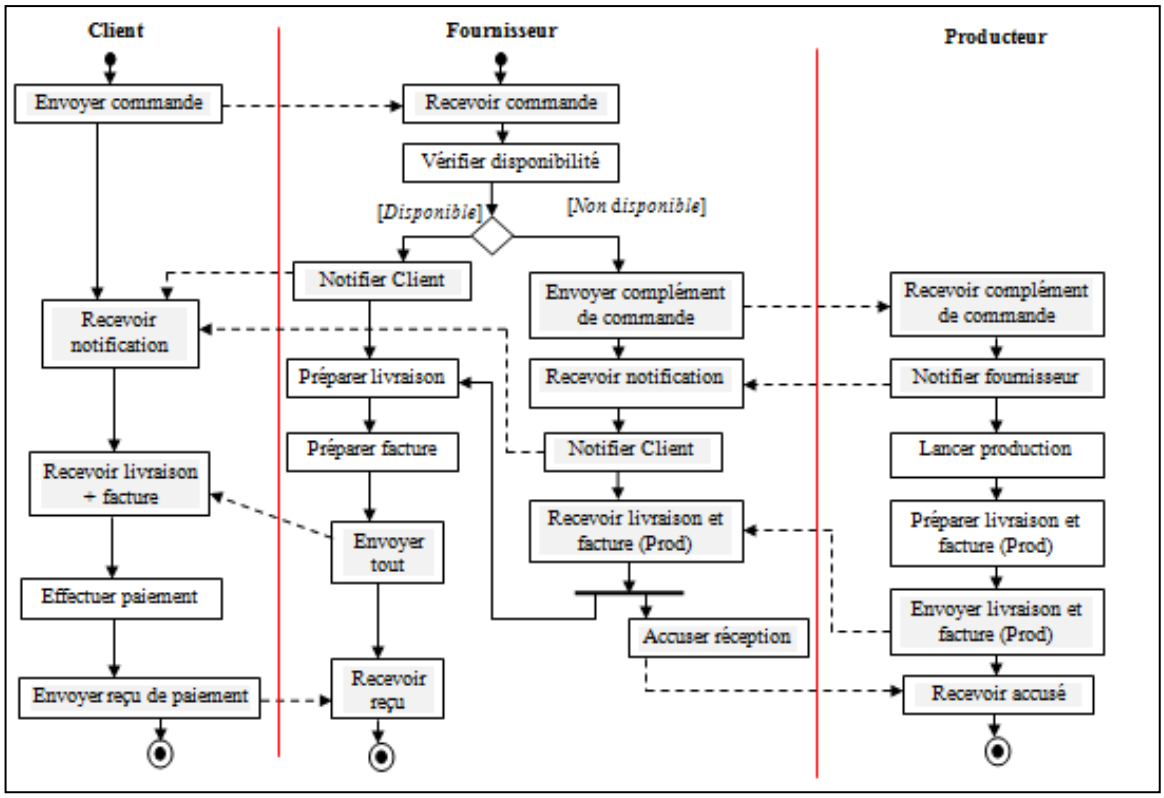



**Figure VI. 32.** *Exemple d'une évolution de WFIO - Expansion du « Faiblement couplé »*

Le processus consiste à gérer des commandes de clients pour un certain type de produits et implique trois partenaires : un client, un fournisseur et un producteur. Le client envoie sa commande au fournisseur qui vérifie la disponibilité des produits pour satisfaire la commande. Si la quantité du produit est suffisante, le client est notifié par un message « Préparer Commande », sinon il est notifié par un message « Attendre production » une fois que le fournisseur envoie une demande de production au producteur. Ce dernier, après avoir reçu la demande de production, notifie le fournisseur par un message « Production lancée». Quand la quantité de produit est disponible, le producteur envoie la livraison au fournisseur qui se charge de l'envoyer au client accompagnée d'une facture et envoie simultanément un accusé de réception au producteur. Le client réceptionne la livraison, effectue le paiement et envoie le reçu de paiement au fournisseur.

En résumé, les différentes évolutions sus-décrites peuvent être supportées par des patrons de coopération généralisés décrits de manière globale dans le tableau VI.3 ci-dessous.

**TAB VI.3.** *Description des patrons de coopération généralisés*

| Type d'évolution | Nom du patron | But | Description |
|---|---|---|---|
| Expansion du « *Partage de charge* » | **PCBS1** Généralisé | Impliquer de nouveaux partenaires dans une orchestration globale de services | - Mettre à jour les liens de partenaires<br>- Mettre à jour la fonction d'orchestration globale |
| Expansion de l'« *Exécution chainée*» | **PCBS2** Généralisé | Ajouter des phases en séquence dans un WFIO existant<br>Impliquer de nouveaux partenaires | - Mettre à jour les liens de partenaires<br>- Mettre à jour les fonctions d'orchestration locales<br>- Mettre à jour les points d'interaction entre les fragments consécutifs |
| Expansion de la « *Sous-traitance* » | **PCBS3** Généralisé | Déléguer d'autres activités du processus à de nouveaux partenaires secondaires | - Mettre à jour les liens de partenaires<br>- Définir de nouveaux services abstraits<br>- Remplacer les services abstraits services par des services externes dans la fonction d'orchestration globale |
| Expansion du « *Transfert de cas* »<br>Expansion du « *Transfert de cas étendu*» | **PCBS4** Généralisé<br>**PCBS5** Généralisé | Impliquer de nouveaux partenaires dans le WFIO global pour diminuer la charge de travail sur les différents sites. | - Mettre à jour les liens de partenaires<br>- Créer de nouveaux sous-processus encapsulés dans des services<br>- Mettre à jour la politique de transfert : mettre à jour les points de transfert dans la fonction d'orchestration globale |
| Expansion du « Faiblement couplé» | **PCBS6** Généralisé | Impliquer de nouveaux partenaires dans le WFIO en vue d'étendre les aspects fonctionnels du processus | - Mettre à jour les liens de partenaires<br>- Mettre à jour le protocole de communication : définir de nouveaux services d'interaction<br>- Insérer les services d'interaction dans les fonctions d'orchestration locales |

### VI.3.2 Evolution par réutilisation de modèles de WFIO : Les patrons composites

Un autre aspect de l'évolution des WFIO consiste à étendre un modèle en y intégrant un modèle de WFIO déjà existant, ce qui revient à une évolution par réutilisation de modèles. Pour ce faire, nous introduisons le concept de *patron de coopération composite* (Boukhedouma et al., 2015) dans le sens où un modèle de WFIO obéissant à un patron de coopération x est combiné à un autre modèle de WFIO obéissant à un autre patron de coopération y.

- **Définition VI.9 :** Patron de coopération composite (PC composite)

*Un patron de coopération composite noté PCBScmp(i,j) est un patron de coopération défini par combinaison de deux patrons de coopération PCBSi et PCBSj (i différent de j) parmi les six patrons de coopération de base.*

C'est un concept que nous introduisons pour supporter l'évolution par réutilisation de modèles de WFIO existants en vue de construire de nouveaux modèles de WFIO plus complexes. Par exemple *PCBScmp(2,3)* définit un modèle de coopération qui combine le PCBS2 (Exécution chainée) et le PCBS3 (Sous-traitance). Avec ce concept, nous aurons une coopération entre *groupes de partenaires* au lieu d'une coopération entre partenaires simples. Dans un PC composite, nous distinguons un patron de coopération prédominant et un patron de coopération secondaire. Le premier contrôle l'exécution du WFIO et constitue l'objet de l'évolution et le second est soumis au contrôle du premier. La figure VI.33 décrit le méta-modèle de définition d'un PC composite.

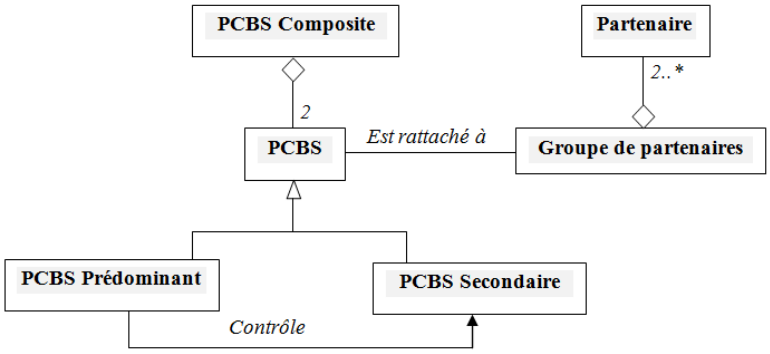



**Figure VI. 33.** *Méta-modèle de définition d'un patron de coopération composite*

En considérant deux PCBS (PCBSi et PCBSj), nous pouvons définir deux patrons de coopération composites PCBScmp(i,j) et PCBScmp(j,i) en inter-changeant les rôles de patron *prédominant* et patron *secondaire* entre les patrons PCBSi et PCBSj. Avec cette vision, nous pouvons définir vingt patrons de coopération composites en combinant, deux à deux, les patrons de coopération à base de services (PCBS1, PCBS2, …, PCBS6) décrits au chapitre V. Le tableau VI.4 donne une vue d'ensemble des patrons de coopération composites ; il s'agit d'une matrice où la première ligne et la première colonne comporte les patrons de coopération de base. Un croisement entre une ligne i et une colonne j donne le patron de coopération composite PCBScmp(i,j) sauf pour la diagonale (i = j) où l'on retrouve les patrons de coopération généralisés. Les lignes comportent les patrons prédominants et les colonnes comportent les patrons secondaires. Pour chaque patron composite, nous spécifions le type d'interaction qui est soit synchrone, soit asynchrone soit mixte ; de même pour le contrôle d'exécution qui peut être centralisé, décentralisé ou mixte, c'est-à-dire centralisé/décentralisé (C/D) ou centralisé/hiérarchisé (C/H) ou décentralisé/hiérarchisé (D/H)..

Le fait de décrire le WFIO comme un service composite à l'aide du concept de fonction d'orchestration permet de manipuler le WFIO lui-même comme un service composite pouvant être ajouté, supprimé, ou substitué en respectant les contraintes de chaque patron de coopération. A la suite du tableau VI.4, nous décrivons à titre d'exemple, deux patrons composites obtenus par combinaison des patrons « Sous-traitance » et « Exécution chainée » (Boukhedouma et al., 2014a) et deux autres patrons composites obtenus par combinaison des patrons « Partage de charge » et « Transfert de cas » (Boukhedouma et al., 2015).

**TAB VI.4.** *Vue d'ensemble des patrons de coopération composites*

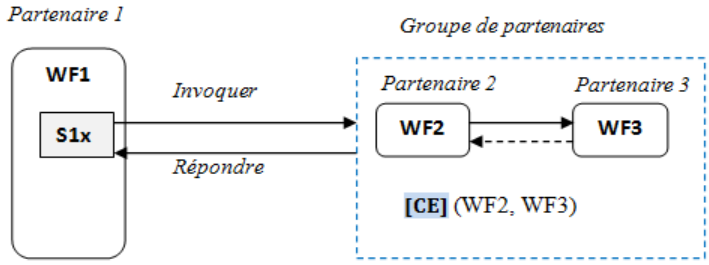
| | PCBS1 | PCBS2 | PCBS3 | PCBS4 | PCBS5 |
|---|---|---|---|---|---|
| PCBS1 | *« Partage de charge » Généralisé* | **PCBSCmp(1,2)**<br>- *"Exécution chainée" dans "Partage de charge"*<br>- Contrôle C/D<br>- interactions Synch./Asynch. | **PCBSCmp(1,3)**<br>- *"Sous-traitance" dans "Partage de charge"*<br>- Contrôle C /H<br>- Interactions synchrones | **PCBSCmp(1,4) - PCBSCmp(1,5)**<br>- *"Transfert de cas (étendu)" dans "Partage de charge"*<br>- Contrôle C/D<br>- Interactions Synch. / Asynch. | **PCBSCmp (1,6)**<br>- *"Faiblement couplé" dans "Partage de charge"*<br>- Contrôle C/ D<br>- Interactions Synch. / Asynch. |
| PCBS2 | **PCBSCmp(2,1)**<br>- *"Partage de charge" dans "Exécution chainée"*<br>- Contrôle D/C<br>- Interactions Asynch. / Synch. | *« Exécution chainée » Généralisée* | **PCBSCmp(2,3)**<br>- *"Sous-traitance" dans "Exécution chainée"*<br>- Contrôle D /H<br>- Interactions Asynch. / Synch. | **PCBSCmp(2,4) – PCBSCmp(2,5)**<br>- *"Transfert de cas(étendu)" dans "Exécution chainée"*<br>- Contrôle décentralisé<br>- Interactions asynchrones | **PCBSCmp(2,6)**<br>- *"Faiblement couplé" dans "Exécution chainée"*<br>- Contrôle décentralisé<br>- Interactions asynchrones |
| PCBS3 | **PCBSCmp(3,1)**<br>- *"Partage de charge" dans "Sous-traitance"*<br>- Contrôle H/C<br>- Interactions synchrones | **PCBSCmp(3,2)**<br>- *"Exécution chainée" dans "Sous-traitance"*<br>- Contrôle H/D<br>- Interactions Asynch. / Synch. | *« Sous-traitance » Généralisée* | **PCBSCmp(3,4) - PCBSCmp(3,5)**<br>- *"Transfert de cas (étensu)" dans "Sous-traitance"*<br>- Contrôle H/D<br>Interactions Asynch. / Synch. | **PCBSCmp(3,6)**<br>- *"Faiblement couplé" dans "Sous-traitance"*<br>- Contrôle H/D<br>- Interactions Asynch. / Synch. |
| PCBS4<br>PCBS5 | **PCBSCmp (4,1) - PCBSCmp(5,1)**<br>- *"Partage de charge" dans "Transfert de cas (étendu)"*<br>- Contrôle D/C<br>- Interactions Asynch. / Synch. | **PCBSCmp(4,2) - PCBSCmp (5,2)**<br>- *"Exécution chainée" dans "Transfert de cas (étendu)"*<br>- Contrôle décentralisé<br>- Interactions asynchrones | **PCBSCmp(4,3) - PCBSCmp (5,3)**<br>- *"Sous-traitance" dans "Transfert de cas (étendu)"*<br>- Contrôle D/ H<br>Interactions Asynch. / Synch. | *« Transfert de cas (étendu) » Généralisé* | **PCBSCmp(4,6) PCBSCmp(5,6)**<br>- *"Faiblement couplé" dans "Transfert de cas (étendu)"*<br>- Contrôle décentralisé<br>- Interactions asynchrones |
| PCBS6 | **PCBSCmp(6,1)**<br>- *"Partage de charge" dans "Faiblement couplé"*<br>- Contrôle D/C<br>- Interactions Asynch. / Synch. | **PCBSCmp(6,2)**<br>- *"Exécution chainée" dans "Faiblement couplé"*<br>- Contrôle décentralisé<br>- Interactions asynchrones | **PCBSCmp(6,3)**<br>- *"Sous-traitance" dans "Faiblement couplé"*<br>-<br>- Contrôle D/ H<br>- Interactions Asynch. / Synch. | **PCBSCmp(6,4) – PCBSCmp(6,5)**<br>- *« Transfert de cas » dans « Faiblement couplé »*<br>- Contrôle décentralisé<br>- Interactions asynchrones | *« Faiblement couplé » Généralisé* |

### VI.3.2.1 Les patrons composites « Exécution chainée – Sous-traitance »

- **Une « Exécution chainée » dans une « Sous-traitance »**

Ce patron composite peut être réalisé lorsqu'un partenaire principal sous-traite une partie de son WF auprès d'un groupe de partenaires alliés dans un WFIO de type « E*xécution chainée* ». Dans ce cas, le patron prédominant est la « *Sous-traitance* », le WFIO global est contrôlé par le partenaire principal qui invoque le WFIO du groupe de partenaires secondaires, ce dernier une fois invoqué est contrôlé de manière décentralisée par le groupe de partenaires. Ceci nécessite un retour de résultats dans le sens inverse de la séquence afin de communiquer les résultats de la sous-traitance au partenaire principal. On tombe sur un WFIO avec un contrôle mixte *hiérarchisé/décentralisé*. Sur la figure VI.34, le partenaire 1 sous-traite une partie de son WF auprès du groupe de partenaires *partenaire2 et partenaire3* fournissant un WFIO formé de la séquence de services *(WF2, WF3)*. Le partenaire 1 invoque le premier service (*S2*) de la séquence et attend le résultat d'exécution du partenaire 3.

Référence du patron : **PCBScmp(3,2)**
Nom du patron : **Une « Exécution chainée » dans une « Sous-traitance »**
Description : Sous-traiter un service d'un partenaire principal auprès d'un groupe de partenaires impliqués dans une coopération de type « Exécution chainée »
Patron Prédominant : PCBS3 - Niveau d'abstraction : « Sous-processus »
Patron secondaire : PCBS2 –Niveau d'abstraction « Processus » (WFIO)
Service à sous-traiter : S1x WF secondaire: **[CE]** (WF2, WF3)
Patron utilisé : PA3.x (substitution de services)




**[SC]** (WF1, (S1x, **[CE]** (WF2, WF3)))

$f^{S}1$ = **Sub** (f1, S1x, **S2**)
et $f^{S}2$ = **Sub** (f2, Sin2, **S'in2)**

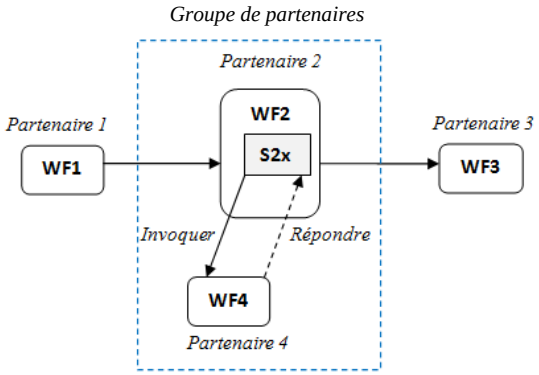
Contraintes: Input (S1x) = Input (WF2) et Output (S1x) = Output (WF3)

**Figure VI. 34.** *Description du patron de coopération composite PCBScmp(3,2)*

- **Une « Sous-traitance » dans une « Exécution chainée »**

Un partenaire impliqué dans un WFIO de type « *Exécution chainée* » peut sous-traiter une partie de son WF implémenté localement à un nouveau partenaire externe. Dans ce cas, le patron prédominant est l'*exécution chainée* et sur un fragment de la séquence, on a une sous-traitance. Le contrôle est mixte *décentralisé/hiérarchisé*. Il est hiérarchisé sur le fragment de WFIO concerné par la sous-traitance. Sur la figure VI.35, le partenaire *2* sous-traite une partie de son WF au partenaire *4*, une modification est apportée aux fonctions d'orchestration de WF2 et WF4, respectivement.

Référence du patron : **PCBScmp(2,3)**
Nom du patron : **Une « Sous-traitance » dans une « Exécution chainée »**
Description : Un groupe de partenaires impliqués dans une coopération de type « Exécution chainée » avec un des partenaires qui sous-traite une partie de son WF.
Patron Prédominant : PCBS2 - Niveau d'abstraction « Processus »
Patron secondaire : PCBS3 – Niveaux d'abstraction « Processus » / « Sous-processus »
Service à sous-traiter : S2x WF secondaire: WF4
Patrons utilisés : PA3.x




**[SC]** (WF2, (S2x, **WF4**))

**[CE]** (WF1, **[SC]** (WF2, (S2x, **WF4**), WF3)

Contraintes: Input (S2x) = Input (WF4) et Output (S2x) = Output (WF4)

**Figure VI. 35.** *Description du patron de coopération composite PCBScmp(2,3)*

Pour illustration, nous reprenons l'exemple de fabrication de circuits intégrés (voir Figure VI.36) déjà présenté au chapitre V impliquant deux partenaires et obéissant au patron « Exécution chainée ».

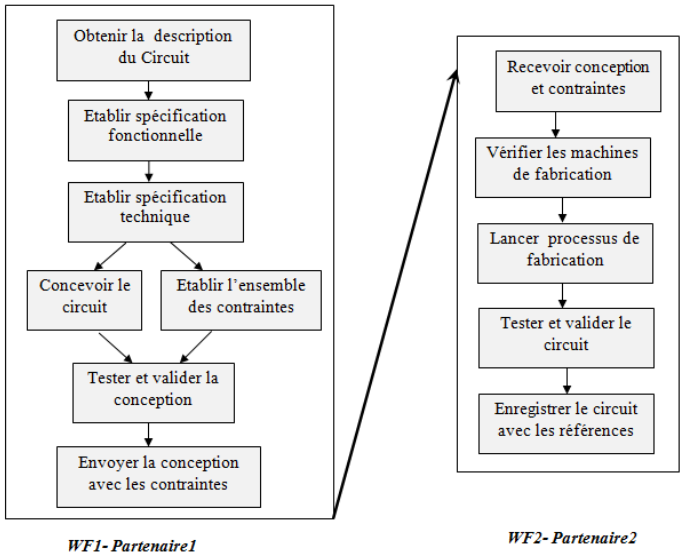



**Figure VI. 36.** *Exemple d'un WFIO obéissant au Patron « Exécution chainée » -PCBS2*

Si le partenaire 2 décide de sous-traiter la phase de test et de validation du WF2 à un troisième partenaire, cette phase sera remplacée par un service abstrait Sx et la coopération sera étendue à un partenaire supplémentaire donnant lieu à un WFIO impliquant trois partenaires et obéissant au patron de coopération composite PCBScmp (2,3) c'est-à-dire une « Sous-traitance » dans une « Exécution chainée » comme le montre la figure VI.37.

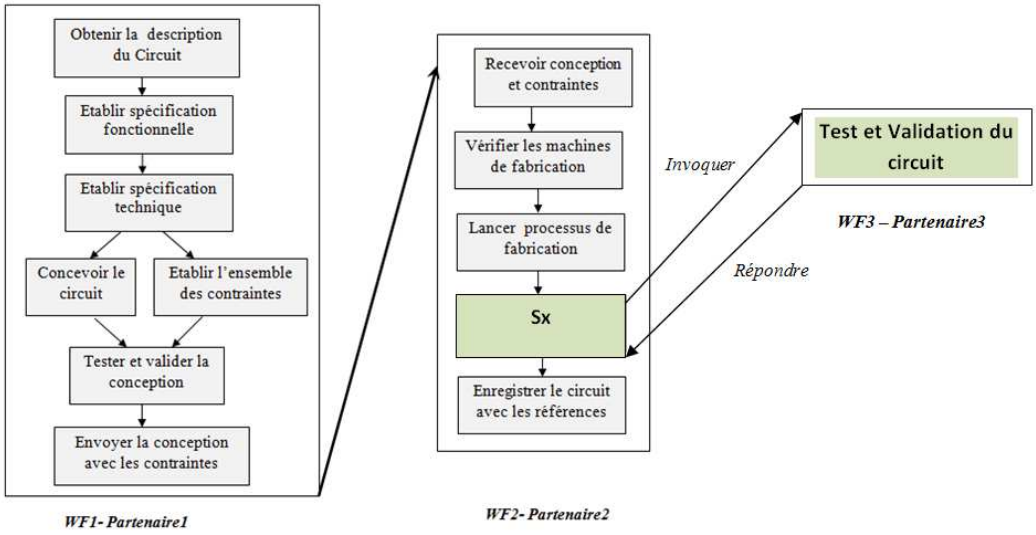



**Figure VI. 37.** *Exemple d'un WFIO obéissant au Patron composite -PCBScmp(2,3)*

### VI.3.2.2 Les patrons composites « Partage de charge – Transfert de cas »

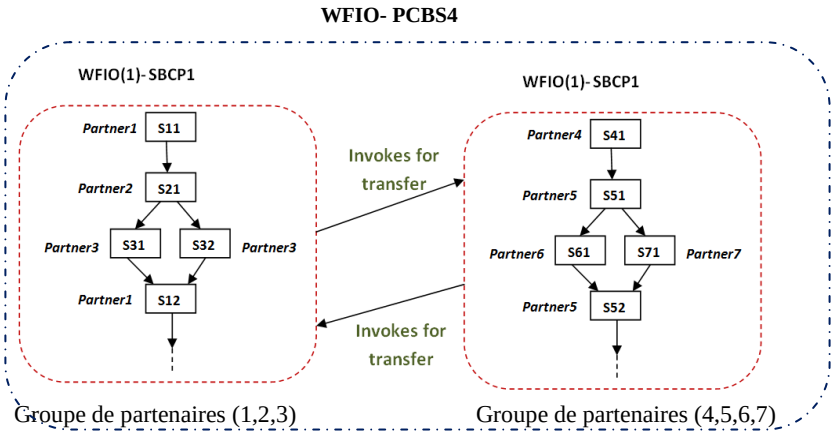
- **Un « Partage de charge » dans un « Transfert de cas »**

La réalisation de ce patron composite décrit dans la figure VI.38, repose sur l'existence de deux groupes de partenaires qui s'entendent entre eux pour implémenter le même WFIO obéissant au patron « Partage de charge » (PCBS1), et qui ensuite définissent des règles de transfert pour construire un WFIO basé sur le transfert de cas entre les WFIO impliquant les deux groupes de partenaires. Le patron prédominant est le « Transfert de cas » et le patron secondaire est le « Partage de charge ». Dans ce patron, le contrôle d'exécution est mixte (centralisé/décentralisé). Cette configuration pourrait être utile par exemple, dans le cas d'un processus lié à une chaine de production (d'un certain type de produit), implémenté selon un « Partage de charge » entre un fournisseur et un producteur constituant un groupe de deux partenaires ; et un autre groupe de deux autres partenaires (fournisseur, producteur) impliqués dans le même processus dupliqué sur un autre site. Pour satisfaire un grand nombre de clients, les deux groupes de partenaires peuvent s'entendre sur une coopération du type « Transfert de cas » pour réaliser un équilibrage de charge entre eux et satisfaire leurs clients dans les meilleurs délais.

- **Un « Transfert de cas » dans un « Partage de charge »**

La réalisation de ce patron repose aussi sur l'existence de deux groupes de partenaires. Le premier groupe qui se partage l'exécution d'un WFIO global obéissant au patron « Partage de charge » (PCBS1) en supposant qu'une partie de ce WFIO nécessite un transfert de cas au sein d'un deuxième groupe de partenaires selon des règles de transfert préalablement définies. Le patron prédominant dans ce cas est le « Partage de charge » (PCBS1) et le patron secondaire est le « Transfert de cas » (PCBS4). Le contrôle d'exécution est mixte (centralisé/décentralisé). La figure VI.39 donne une description de ce patron.

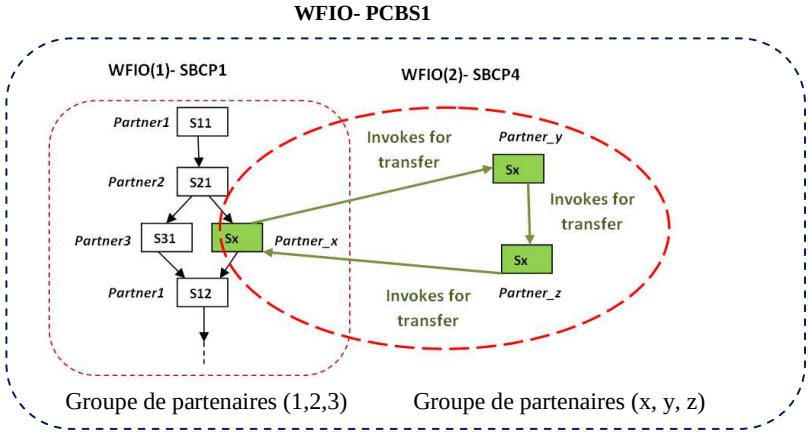
Référence du patron : **PCBScmp(4,1)**
Nom du patron : **Un « Partage de charge » dans un « Transfert de cas »**
Description : Deux groupes de partenaires impliqués dans une coopération de type « Transfert de cas » avec un partage de charge au sein de chaque groupe.
Patron Prédominant : PCBS4 Patron secondaire : PCBS1
Contrôle : décentralisé/centralisé




**[CT]** ( **[CS]**(WF1, WF2, WF3), **[CS]**(WF4, WF5, WF6, WF7), R)

**Figure VI. 38.** *Description du patron de coopération composite PCBScmp(4,1)*

Référence du patron : **PCBScmp(1,4)**
Nom du patron : **Un « Transfert de cas » dans un « Partage de charge »**
Description : Un transfert de cas entre un groupe de partenaires sur une partie de WFIO obéissant au patron « Partage de charge » impliquant un autre groupe de partenaires.
Patron Prédominant : PCBS1 Patron secondaire : PCBS4
Contrôle : centralisé/décentralisé
Service concerné par le transfert : Sx




**[CS]** ( WF1, WF2, WF3, **[CT]**(WFx, WFy, WFz, R))

**Figure VI. 39.** *Description du patron de coopération composite PCBScmp(1,4)*

Pour illustrer ce cas de figure, nous reprenons l'exemple du WFIO obéissant au partage de charge entre l'agence de voyage, la compagnie aérienne et l'hôtel déjà présenté au chapitre V de ce document. Ce WF pourrait subir une expansion de la coopération pour impliquer d'autres partenaires (par exemple d'autres compagnies aériennes) afin de traiter la demande de réservation du client. Les compagnies aériennes (C1, C2 et C3 sur la figure VI.40) implémentent le même WF

de réservation et opèrent selon un transfert de cas avec une règle simple : « si la requête de réservation ne peut être satisfaite au niveau d'un partenaire, elle est transférée pour traitement aux autres partenaires ». Le WFIO impliquera alors d'autres partenaires et obéira au patron composite PCBScmp(1,4) « un Transfert de cas dans un Partage de charge » comme le montre la figure VI.40.

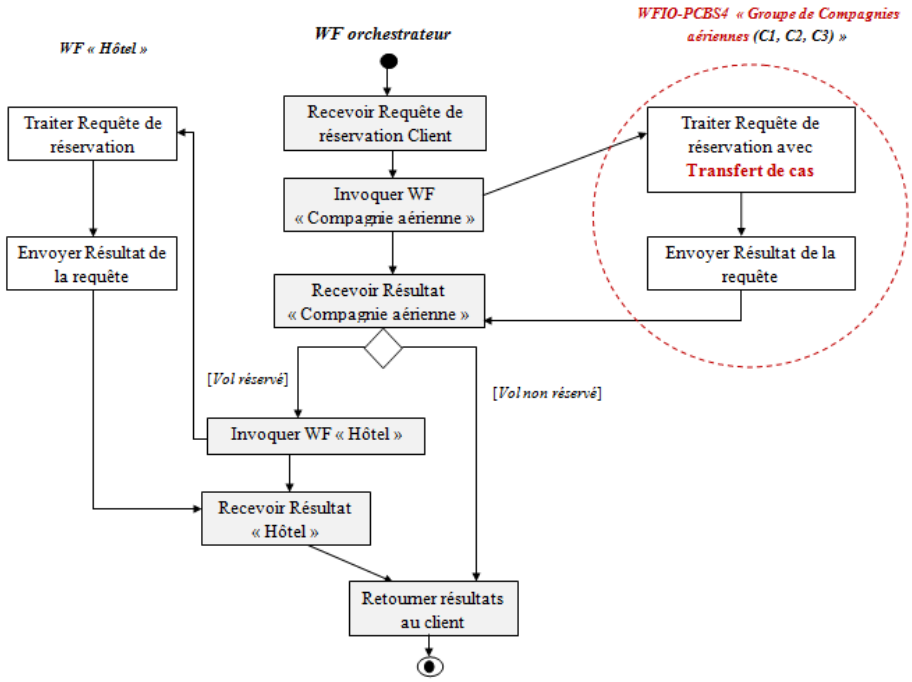



**Figure VI. 40.** *Exemple d'un WFIO obéissant au patron composite PCBScmp(1,4)*

Dans la suite, nous définissons brièvement quelques critères permettant d'évaluer notre approche d'adaptation/évolution.

## VI.4 Critères d'évaluation de l'adaptation/évolution

Une question sur laquelle nous nous sommes penchés mais que nous n'avons pas encore approfondie est la définition de métriques d'adaptation/évolution des modèles de WFIO. Nous avons dégagé un certain nombre de critères permettant une appréciation qualitative de notre approche d'adaptation/évolution. Parmi ces critères :

- **Coût de l'adaptation/évolution** : ce critère est lié déjà à l'approche orientée services de modélisation des WFIO. La principale caractéristique de couplage faible des services induit un coût d'adaptation réduit, notamment dans le cas des adaptations locales à un fragment de WF à savoir : les adaptations de services et les adaptations de contrôle de flux. En effet, le service à ajouter, substituer, supprimer ou réordonner n'est lié aux autres services qu'à travers des invocations via des interfaces, qui nécessitent une mise à jour du flux de données (et éventuellement une mise à jour des conditions) entre les services ayant été affectés par l'opération de changement. Quant aux adaptations des interactions et les opérations d'évolution, celles-ci s'avèrent plus coûteuses, notamment dans les WFIO obéissant aux patrons de coopération « Transfert de cas » (création de sous-processus) et « Faiblement couplé » (création des services d'interaction).

- **Nombre de modèles impliqués :** dans les adaptations locales, c'est-à-dire les adaptations de services et de contrôle de flux, un seul modèle est concerné par l'opération d'adaptation. Alors que dans le cas des adaptations d'interaction (ou d'évolution), deux modèles (ou plus) sont concernés par les opérations de changement. Une opération « invoke » d'un service de partenaire doit lui correspondre une opération « receive » du partenaire récepteur et éventuellement un « reply » dans le cas d'une communication synchrone.

- **Nombre et nature des opérations de changement :** le nombre et la nature des opérations de changement diffèrent d'un cas d'adaptation/évolution à l'autre. En effet ces opérations peuvent être un simple ré-ordonnancement des services dans le processus (cas de l'adaptation du contrôle de flux) et mise à jour des activités « assign » pour l'assignation des variables d'entrée/sortie entre les services. Elles peuvent nécessiter une mise à jour des liens de partenaires (Partner-Links) dans le cas d'ajout, substitution ou suppression de services, nécessitant éventuellement des opérations « import » si le service est externe au partenaire (cas de l'évolution par exemple). D'autres opérations de nature plus complexes sont par exemple la création de sous-processus dans le « Transfert de cas » et la création de services d'interaction dans le patron « Faiblement couplé ».

- **Impact sur la correction des modèles :** après chaque opération d'adaptation/ évolution, une vérification de la correction des modèles de WFIO adaptés est nécessaire. Cette vérification est relativement simple dans le cas des modèles obéissant aux patrons « Partage de charge », « Exécution chainée » et « Sous-traitance » et plus complexe pour des modèles obéissant aux patrons « Transfert de cas » et « Faiblement couplé ».

- **Complexité du modèle de WFIO :** ce critère est lié à l'approche de modélisation des WFIO. La complexité des modèles se trouve d'une part, réduite grâce à l'encapsulation de détails d'implémentation dans des services (en particulier, les services composites). D'autre part, grâce à la possibilité de considérer un processus BPEL entier comme un service composite (le service orchestrateur) qui peut être invoqué à partir d'un autre processus via uniquement son interface. Cette caractéristique permet de réduire considérablement, la complexité des modèles, notamment dans le cas des opérations d'évolution (expansion de la coopération) impliquant des WFIO entiers comme partie du modèle global.

- **Possibilité de défaire une adaptation :** il peut arriver qu'après une opération de changement sur un modèle de WFIO, on veuille revenir au modèle avant adaptation. Pour cela, BPEL offre un mécanisme de « versionning » permettant de garder trace de toutes les versions du modèle de WF après chaque opération d'adaptation, et de pouvoir revenir à une version antérieure du processus. Une autre manière consiste à défaire l'adaptation par une adaptation *inverse*. L'inconvénient de la première approche est le coût de stockage et d'indexation des différentes versions pour pouvoir les retrouver aisément et l'inconvénient de la seconde approche est le coût de l'adaptation/réadaptation. Nous pensons que la première est plus adéquate pour des modèles non denses et assez stables et la deuxième est pratique pour des opérations d'adaptation simples (non coûteuses) et aussi peu fréquentes.

- **Profondeur de l'adaptation** : dans notre cas, nous opérons une adaptation sur le schéma du processus puisqu'elle affecte directement la définition du modèle de processus. Les adaptations au niveau instance n'ont de sens que dans une approche d'adaptation dynamique.

- **Perspectives de l'adaptation :** notre approche d'adaptation/évolution opère directement sur les niveaux *fonctionnel*, *comportemental* et *interactionnel*, avec un impact sur les niveaux informationnel (mise à jour du flux de données) et organisationnel (ajout/suppression et substitution de partenaires).

Lors de la mise en pratique de la solution, d'autres critères plutôt quantitatifs peuvent être définies et incorporées au framework d'adaptation/évolution. Nous citons par exemple : la fréquence des adaptations sur un modèle, la fréquence d'un type d'adaptation sur un modèle donné ou sur un ensemble de modèles, la fréquence d'annulation d'une adaptation, le nombre d'instances affectées par une adaptation, …etc.

## Synthèse

Dans ce chapitre, nous nous sommes focalisés sur la deuxième partie de notre contribution, à savoir les aspects de flexibilité des modèles de WFIO que nous percevons sous trois perspectives complémentaires : adaptabilité, évolutivité et réutilisabilité. Notre objectif était de dégager les différentes situations (bien sur de manière non exhaustive) de changements susceptibles d'affecter les modèles de WFIO, de les catégoriser selon la portée du changement (local ou global) et la nature du changement (adaptation ou évolution) et bien sur offrir les mécanismes support de ces changements.

Pour ce faire, nous avons adopté une approche à base de patrons comme pour la partie interconnexion. Ainsi, nous avons défini des patrons d'adaptation, des patrons d'évolution qui sont d'une part soutenus par une généralisation des patrons de coopération de base où nous avons introduit les patrons de coopération *généralisés* ; et d'autre part, par une réutilisation des patrons de coopération existants pour la construction de modèles de WFIO plus complexes basés sur les patrons de coopération *composites*. Concernant les patrons d'adaptation, nous les avons classés en trois (sous-) catégories conformément aux trois dimensions de définition d'un PCBS : les patrons d'adaptation de services, les patrons d'adaptation de contrôle de flux et les patrons d'adaptation des interactions. Pour chaque patron d'adaptation illustré, nous avons fourni une description comportant un schéma générique, une formalisation à l'aide des opérateurs d'adaptation que nous avons définis. Ces derniers s'appliquent sur une fonction d'orchestration et nécessitent la vérification de certaines contraintes sur le flux de données dans le processus global. Quant aux patrons d'évolution, nous avons axé sur l'expansion de la coopération spécifique à chaque patron de coopération de base. Les patrons de coopération composites ont été illustrés à travers quelques exemples, ils reposent sur la notion de patron prédominant et de patron secondaire qui est contrôlé par le premier. L'intérêt de ces patrons composites est de pouvoir manipuler un WFIO comme une partie pouvant être intégrée à un autre WFIO. Ce qui permet d'atténuer la complexité de manipulation d'un modèle de WFIO en le représentant d'une manière modulaire. Cet aspect permettra par exemple, d'apporter séparément, des modifications sur un WFIO intégré à un autre sans perturber la cohérence du modèle global.

Par ailleurs, une question sur laquelle nous nous sommes penchés à la fin de ce chapitre consiste en la définition de critères d'évaluation de l'adaptation/évolution des modèles de WFIO. Les critères que nous avons pu identifier sont par exemple : le coût de l'adaptation/évolution, le nombre de modèles de WF impliqués, le nombre et la nature des opérations de changement, l'impact sur la correction des modèles, la possibilité de défaire une adaptation, les perspectives de l'adaptation…etc. La plupart de ces critères sont de nature qualitative. D'autres critères plutôt quantitatifs comme la fréquence des opérations d'adaptation, le nombre d'instances objet de l'adaptation/évolution, peuvent être définis dans le cas d'une approche d'adaptation/évolution dynamique. Côté implémentation, la plupart des patrons décrits dans ce chapitre ont été implémentés dans un framework d'adaptation/évolution pour s'appliquer à des modèles de WFIO issus du framework de coopération que nous décrivons dans le chapitre VII.

# CHAPITRE VII

# Les Frameworks de Coopération et d'Adaptation/Evolution

## Introduction

Dans le chapitre V, nous avons présenté une conceptualisation et une formalisation des patrons de coopération à base de services (PCBS) qui supportent la définition des modèles de WFIO. Ceux-ci sont obtenus par interconnexion de modèles de WF mis en coopération, conformément à un schéma de coopération choisi et bien identifié. Dans la description des PCBS, nous avons particulièrement ressorti les caractéristiques et les contraintes de l'opération d'interconnexion (ou de composition) pour nous guider dans la réalisation des patrons de coopération. Nous avons préalablement, déterminé les règles de transformation d'un modèle de WF à base d'activités en modèle de WF à base de services, en respectant selon les cas, le degré d'abstraction requis pour la mise en coopération des WF.

De la même manière, dans le chapitre VI, nous avons identifié différentes catégories de patrons d'adaptation à savoir, les patrons d'adaptation de services, les patrons d'adaptation de contrôle de flux et les patrons d'adaptation des interactions. Aussi, nous avons décrit deux catégories de patrons d'évolution, les patrons de coopération généralisés et les patrons de coopération composites, basés essentiellement sur les patrons de coopération décrits au chapitre V. Pour chaque patron d'adaptation/évolution, nous avons fourni une description qui renseigne notamment, sur les patrons (ré)utilisés pour son implémentation, un schéma générique du patron et un ensemble de contraintes devant être vérifiées sur le flux de données entre les services après une opération d'adaptation ou d'évolution.

Dans ce chapitre, nous nous concentrons sur les aspects d'implémentation de nos frameworks de coopération et d'adaptation/évolution. Dans une première partie, nous décrivons notre framework de coopération baptisé « *S-IOFLOW* » pour « **S**ervice based **I**nter-**O**rganisational work**FLOW** ». Nous enchainons ensuite, par la description du framework d'adaptation/évolution. Pour les deux frameworks, nous présenterons une architecture générale, l'organisation des différentes classes implémentées ainsi que les principales fonctionnalités offertes par les applications développées.

Notons que le développement de nos frameworks de coopération et d'adaptation/évolution a été réalisé conformément au patron de conception MVC (Model-View-Controller) qui garantit une synchronisation entre les vues et les modèles qui subissent des changements via les contrôleurs. Les modèles conceptuels des frameworks comportent chacun, trois catégories de classes : les classes de la catégorie « Modèle » correspondant essentiellement aux fichiers BPEL, les classes de la catégorie « Vue » correspondant aux interfaces offertes à l'utilisateur pour visualiser différentes vues des modèles de processus et dialoguer avec l'application, et les classes de la catégorie « Contrôleur » qui implémentent les traitements effectués sur les modèles, ce sont les classes implémentant respectivement, les patrons de coopération dans le framework de coopération et les patrons d'adaptation/évolution dans le framework d'adaptation/évolution.

Nous avons réalisé le développement de nos frameworks sous la plateforme Windows 7. Nous avons utilisé BPEL comme langage de spécification de nos processus WF devant être interprétés et exécutés par le moteur de WF Open-ESB2.2 et le serveur GlassFich. Pour le développement des services Web, nous avons utilisé les EJB (Enterprise Java Beans) et pour le développement de nos applications, nous avons utilisé le langage java, l'IDE Net-beans et l'API Jdom. Ces différents outils sont décrits dans l'annexe B de ce document.

## VII.1 Le framework de coopération « S-IOFLOW »

Pour réaliser l'interconnexion entre deux processus BPEL, le framework de coopération doit permettre de :

- Charger les processus BPEL objet de l'interconnexion
- Sélectionner à chaque fois, un patron de coopération à réaliser
- Apporter les changements nécessaires au niveau des fichiers BPEL concernés par la mise en coopération
- Maintenir un flux de données cohérent dans le processus (vérifier les contraintes d'interconnexion)
- Déployer les processus de WFIO obtenus et suivre leur exécution

Dans la suite, nous présentons l'architecture générale de notre framework de coopération en décrivant les différents modules de la solution, nous montrons le découpage logiciel de notre solution ainsi que la logique d'utilisation des fonctionnalités offertes par le framework. Ce dernier est destiné à être utilisé par un concepteur de WFIO qui se charge de réaliser une coopération entre deux processus BPEL (ou plus), selon un schéma de coopération préalablement sélectionné. Le contrôle de flux exprimé dans le processus BPEL définit la fonction d'orchestration du processus.

### VII.1.1 Architecture générale du framework de coopération

La figure VII.1 décrit l'architecture générale de notre framework de coopération. L'architecture comporte les fichiers BPEL des différents partenaires ainsi que les services Web implémentés au niveau des sites de chaque partenaire. Les données manipulées sont stockées dans des bases de données locales et échangées entre les différents partenaires via les invocations de services externes ou d'interaction. Après interconnexion entre deux fichiers BPEL, ces derniers peuvent communiquer entre eux via des opérations d'invocation/réponse. Au centre de l'architecture se trouve l'application qui réalise l'interconnexion des modèles de processus BPEL. L'application comporte trois principaux modules : un *module de chargement*, un *module d'interconnexion* et un *module de validation et persistance*.

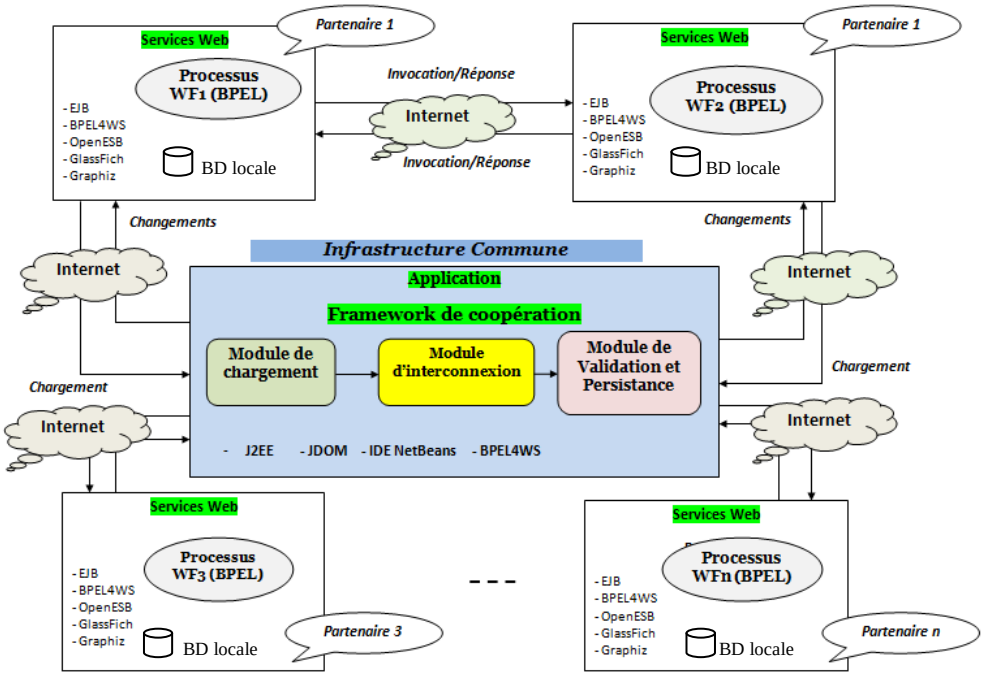



**Figure VII.1.** *Architecture générale du framework de coopération « S-IOFLOW »*

### VII.1.1.1 Module de chargement des processus

Le module de chargement consiste à charger les processus BPEL concernés par l'interconnexion à partir de leurs sites respectifs (les sites des partenaires qui les implémentent). Concrètement, il s'agit de créer une copie de chaque processus BPEL sur l'infrastructure commune et d'effectuer l'interconnexion (donc les changements sur les fichiers BPEL) de manière centralisée, sur cette infrastructure. Une fois la composition ou l'interconnexion validée, ces changements seront répercutés sur les fichiers BPEL d'origine grâce au module de validation et de persistance.

A titre illustratif, nous donnons dans la suite, le code java de la méthode `addFileBpel` qui permet de charger la liste des processus en vue de les composer (ou les interconnecter). Cette méthode se trouve dans la classe `ListBpelFile`, elle permet de créer une instance de la classe `BpelFile` en indiquant le chemin du fichier BPEL (une url de type String).

```
1  public boolean  addFileBpel(String url){
2          BpelFile bpelFile = new BpelFile(url);
3          if(this.exiteProcessus(bpelFile.getNomProcessus().getValue())== false)
4
5             {this.ListBpelFile.add(bpelFile);
6
7       //On ajoute le nom de ce nouveau processus à la liste des noms  ListNomProcessus
8              this.addNomProcessus(bpelFile.getNomProcessus().getValue());
9             }
10         else{
11             bpelFile = null;
12             return true;
12         }
13         return false;
14     }
```

**Commentaires:**

Ligne 2: création d'une instance de la classe BpelFile (qui permet réellement de charger le processus –fichier BPEL-)
Ligne 3 : on teste si on a déjà chargé ce processus ou non.
Ligne 5 : notre application doit composer plusieurs processus donc on doit garder trace de tous ces processus dans une liste (ListBpelFile).

**Listing VII.1.** *Code java de la méthode* `addFileBpel`

Pour la création d'une nouvelle instance (`new BpelFile`) du fichier BPEL sur l'infrastructure commune de coopération, nous avons utilisé des méthodes spécifiques de l'API JDOM qui permet d'encapsuler le fichier BPEL dans notre classe `BpelFile`, le code source du constructeur `BpelFile` est le suivant:

```
1  /**
2       * Constructeur
3       */
4      public BpelFile(String url){
5          this.url = url;
6          init();    // méthode permet d'initialiser les éléments de BpelFile
7      }
8
9      /**
10      * méthode qui permet d'initialiser les elements de BpelFile
11      */
12     public void init() {
13         try {
14             SAXBuilder builder = new SAXBuilder();
15             document = builder.build(new File(url));
16             racine = document.getRootElement();
17             nomProcess = racine.getAttribute("name");
             targetNamespace = racine.getAttribute("targetNamespace");
         } catch (JDOMException ex) {
```

```
            System.out.println("Erreur impossible de manipuler ce fichier");
            ex.printStackTrace();
        } catch (IOException ex) {
            System.out.println("Erreur impossible de manipuler ce fichier");
            ex.printStackTrace();
        }
        this.ListeVue = new ArrayList();      //Initialiser la liste des vues.
        this.ListeWSDLFile = new ArrayList();
                  //Initialiser la liste des fichiers WSDL associe au ce processus.

        initWSDLFile();
       //Methode qui permet de charger les différents fichiers WSDL appropriés pour ce processus
    }
```

**Commentaires**

Les méthodes utilisées pour charger le processus BPEL sont :
**SAXBuilder** : ligne 13
et **build** : ligne 14

définies dans l'API JDOM

**Listing VII.2.** *Code java du constructeur* `BpelFile`

### VII.1.1.2 Module d'interconnexion (ou composition) des modèles de processus BPEL

Le module d'interconnexion consiste à effectuer l'interconnexion entre les processus BPEL sélectionnés et dupliqués sur l'infrastructure commune. Il s'agit d'apporter les modifications nécessaires aux fichiers BPEL pour réaliser l'interconnexion selon les règles de spécification énoncées dans la description des patrons de coopération (cf. chapitre V) et qui dépendent du patron de coopération à réaliser. Une phase de mise à jour du flux de données est nécessaire afin de maintenir une cohérence dans le processus global.

### VII.1.1.3 Module de validation et de persistance

Le module de validation et de persistance consiste à valider l'opération de composition (ou interconnexion) effectuée. Les changements effectués en local (sur l'infrastructure commune) sur les fichiers BPEL dupliqués (des objets `BpelFile`) seront automatiquement répercutés sur les fichiers BPEL physiques d'origine hébergés dans les sites des partenaires. La méthode qui permet de répercuter les changements sur les fichiers physiques BPEL s'appelle `saveBpelFile()` définie dans la classe `BpelFile`. Cette méthode invoque aussi des méthodes de l'API JDOM.

```
/**
         * Méthode qui permet de sauvegarder l'objet bpelFile après modification
         */
        public void saveBpelFile(){
            XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
            try {
                 sortie.output(this.document,new FileOutputStream(this.url));
            } catch (Exception e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
            }
            this.IndicateurChangement = 4;
            this.notifyView(IndicateurChangement);
            //this.reloadBpelFile();}
```

**Commentaires**
Les méthodes de l'API JDOM utilisées pour charger le processus BPEL sont :
**XMLOutputter** (constructeur) : ligne 5
**et output**: ligne 7
Les lignes 12 et 13 correspondent à des instructions liées au patron de conception MVC pour la mise à jour de l'affichage des vues après la répercussion des modifications sur les fichiers BPEL.

**Listing VII.3.** *Code java de la méthode* `SaveBpelFile`

### VII.1.2 Architecture du framework selon le Patron MVC « Model-View-Controller »

Le patron MVC est un patron de conception qui permet de séparer la logique métier du traitement. Il englobe trois parties que l'on retrouve dans les composants distincts suivants:

- Le composant « Modèle » (modèle de données),
- Le composant « Vue » (présentation, interface utilisateur),
- Le composant « Contrôleur » (logique de contrôle, gestion des événements, synchronisation).

Pour appliquer ce patron à notre framework de coopération, nous avons d'abord identifié les trois composants essentiels de ce modèle (Modèle, Vue, contrôleur):

- Le *modèle* représente le cœur de l'application (les données concernées par les traitements) et dans notre cas, nous avons :
  - La classe `BpelFile` : cette classe encapsule tous les traitements manipulant un fichier de description des processus.
  - La classe `ListBpelFile` : permet de stocker dans une liste tous les processus chargés par le concepteur de WF en vue de les interconnecter.
- Une *vue* fait l'interface avec l'utilisateur de l'application. Sa tâche est d'afficher les données qu'elle a récupère auprès du modèle. Dans notre cas, les vues sont par exemple : `PrincipaleFrame`, `VueHierarchie`, `VueGraphique`, `VueDétails`, …etc. permettant d'afficher différentes vues des modèles de processus manipulés.
- Le *contrôleur* gère les interactions avec l'utilisateur, détermine quels traitements doivent être réalisés. Dans notre cas, l'utilisateur déclenche des évènements (cliquer sur un bouton, choisir une architecture, etc.) pour composer (ou interconnecter) les processus chargés.

  En ce qui concerne les implémentations des différents PCBS, nous avons les classes :
  - `PatronExécutionChainée`,
  - `PatronSousTraitance`,
  - `PatronTransfertDeCas`,
  - `PatronFaiblementCouplé`,
  - `PatronPartagedeCharge`.

L'avantage du patron MVC est de garantir la synchronisation entre le modèle et les vues qui observent le même modèle (`BpelFile`) mais le représentent de différentes manières. La synchronisation signifie que toutes les vues doivent préserver un état cohérent du modèle observé, par exemple à chaque fois qu'on ajoute une nouvelle activité à un processus toutes les vues observatrices doivent être notifiées qu'un changement est apparu sur le modèle en question. L'action de notification lance la mise à jour des vues. La figure VII.2 décrit l'architecture fonctionnelle du framework, conformément au patron MVC.

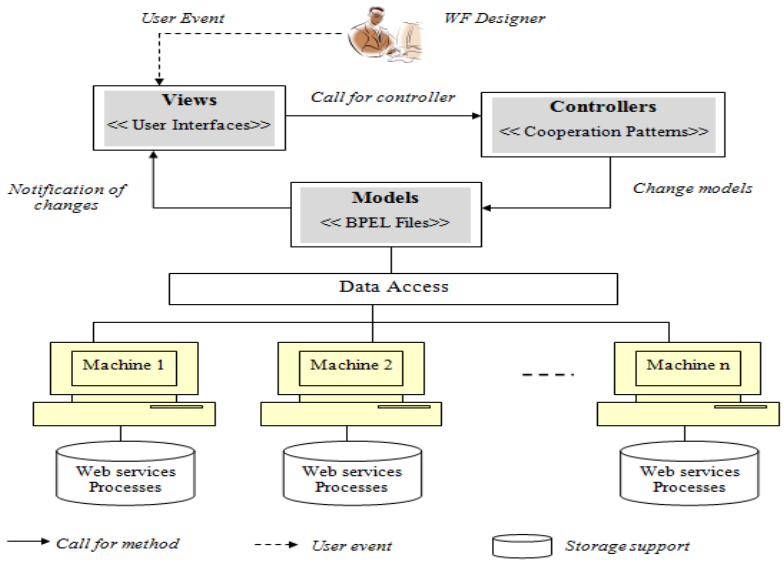



**Figure VII.2.** *Architecture fonctionnelle du framework de coopération selon le patron MVC*

### VII.1.3 Diagramme de classes du framework de coopération

Le diagramme de classes de la figure VII.3 donne une vue globale des classes implémentées dans le framework « S-IOFLOW ». Les principales classes de la catégorie « Modèle » sont `BpelFile` et `ListBpelFile` qui héritent de la classe « `observable` » et toutes les vues des modèles (`VueHiérarchie`, `VueCode`, `VueGraphique,…`) héritent de l'interface « `observer` ». Celle-ci est notifiée par la classe « `observable` » pour tous les changements qui affectent les modèles. Le contrôleur contient un ensemble de classes implémentant les patrons de coopération, qui sont :

- `PatronExécutionChainée,`
- `PatronSous-Traitance,`
- `PatronTransfertdeCas,`
- `PatronFaiblementCouplé,`
- `et PatronPartagedeCharge`

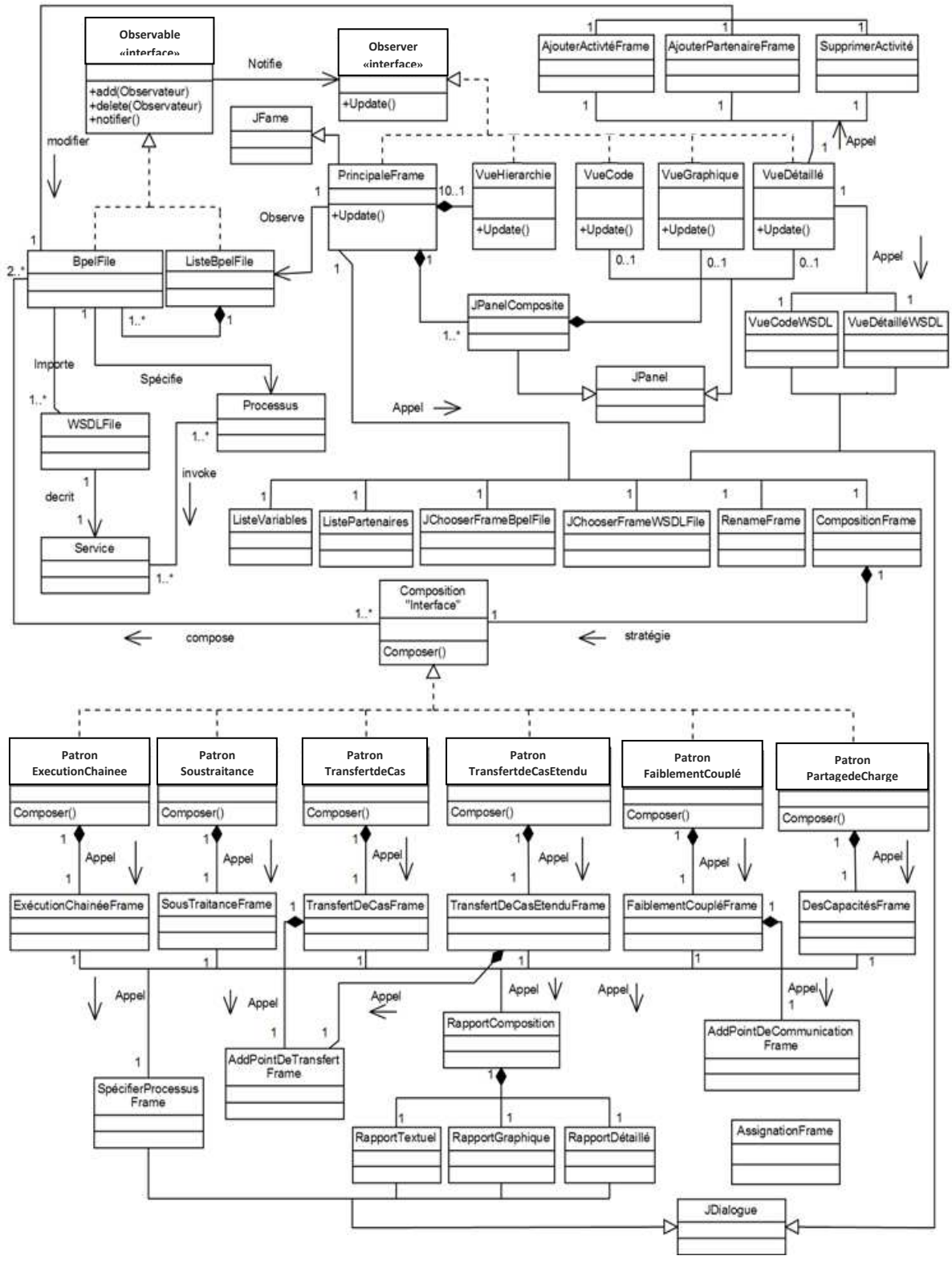



**Figure VII.3.** *Diagramme de classes global du framework de coopération « S-IOFLOW »*

### VII.1.3.1 Les classes de la catégorie « Modèle»

La partie « Modèle» contient les classes `BpelFile` et `ListBpelFile`. Celles-ci sont liées à d'autres classes telles que `WSDLFile` qui contient la description de chaque service Web invoqué par le processus en question. Les deux classes héritent de l'interface « `observable` » et

implémentent les méthodes `add (observer o)`, `delete (observer o)` et `notifyview (observer o)` en plus des méthodes spécifiques à chaque classe comme la méthode `init()` et `reloadBpelFile()` de la classe `BpelFile` et les méthodes `addFileBpel()` et `getListNomProcessus()` de la classe `ListBpelFile`.

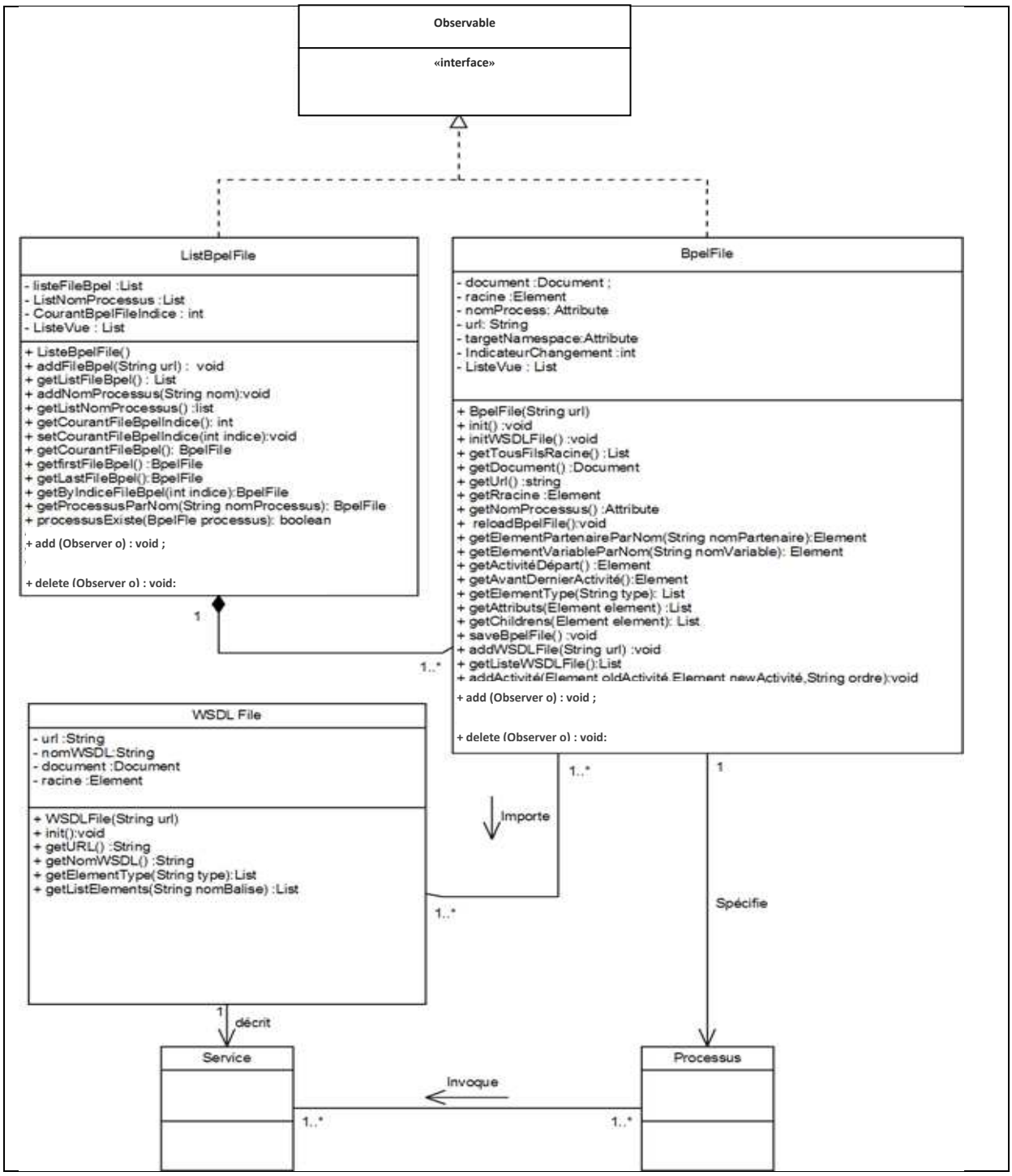



**Figure VII.4.** *Diagramme de classes de la catégorie « Modèle »*

A titre d'exemple, l'implémentation de la méthode `notifyview` se présente comme suit :

```
public void notifyview (int IndicateurChangement) {
Iterator it_ListeVue = this.ListeVue.iterator();
          while(it_ListeVue.hasNext()){
             Observer vue_courant=  (Observer)it_ListeVue.next();
             vue_courant.update(this,IndicateurChangement );
          }
 }
```

**Listing VII.4.** *Code java de la méthode* `notifyView` *de la classe* `BpelFile`

En cas d'un changement sur un modèle de processus (c'est-à-dire une instance de `BpelFile`), la méthode `notifyview` de la classe `BpelFile` notifie l'ensemble des vues observant ce processus. La liste `ListeVue` stocke ces vues, une instance de `BpelFile` notifie toutes les instances des vues observant le même processus.
`IndicateurChangement` est un entier qui spécifie le type de changement.

La méthode `notifyview` est invoquée par les méthodes:

- `reloadBpelFile` (recharger le processus),
- `saveBpelFile` (enregistrer le processus),
- `addWSDLFile` (ajouter un fichier WSDL),
- `addActivity` (ajouter une activité).

### VII.1.3.2 Les classes de la catégorie « Vue»

La partie « Vue » contient les différentes classes qui implémentent les vues (les interfaces) de notre framework de coopération, chaque vue est liée à un modèle et hérite de l'interface « `observer` ». A chaque notification émanant du modèle suite à un changement du modèle, toutes les vues sont mises-à-jour, chaque vue possède sa propre méthode de mise-à-jour `update().` La figure VII.5 ci-dessous montre les principales classes de cette catégorie qui héritent de l'interface « `observer` » contenant la signature de la méthode `update()`comme le montre le listing VII.5.

```
public void update(Observable o, int IndicateurChangement) {
        /**
     * 2://indicateur d'ajout d'un nouvel élément
     * 3://indicateur de rechargement (reload) d'un fichier BPEL
*/
        if(IndicateurChangement.equals(2) || IndicateurChangement.equals(3) ){
        this.modifierAffichageCode();
        }
    }
```

**Listing VII.5.** *Code java de la méthode « update » de la classe* `VueCode`

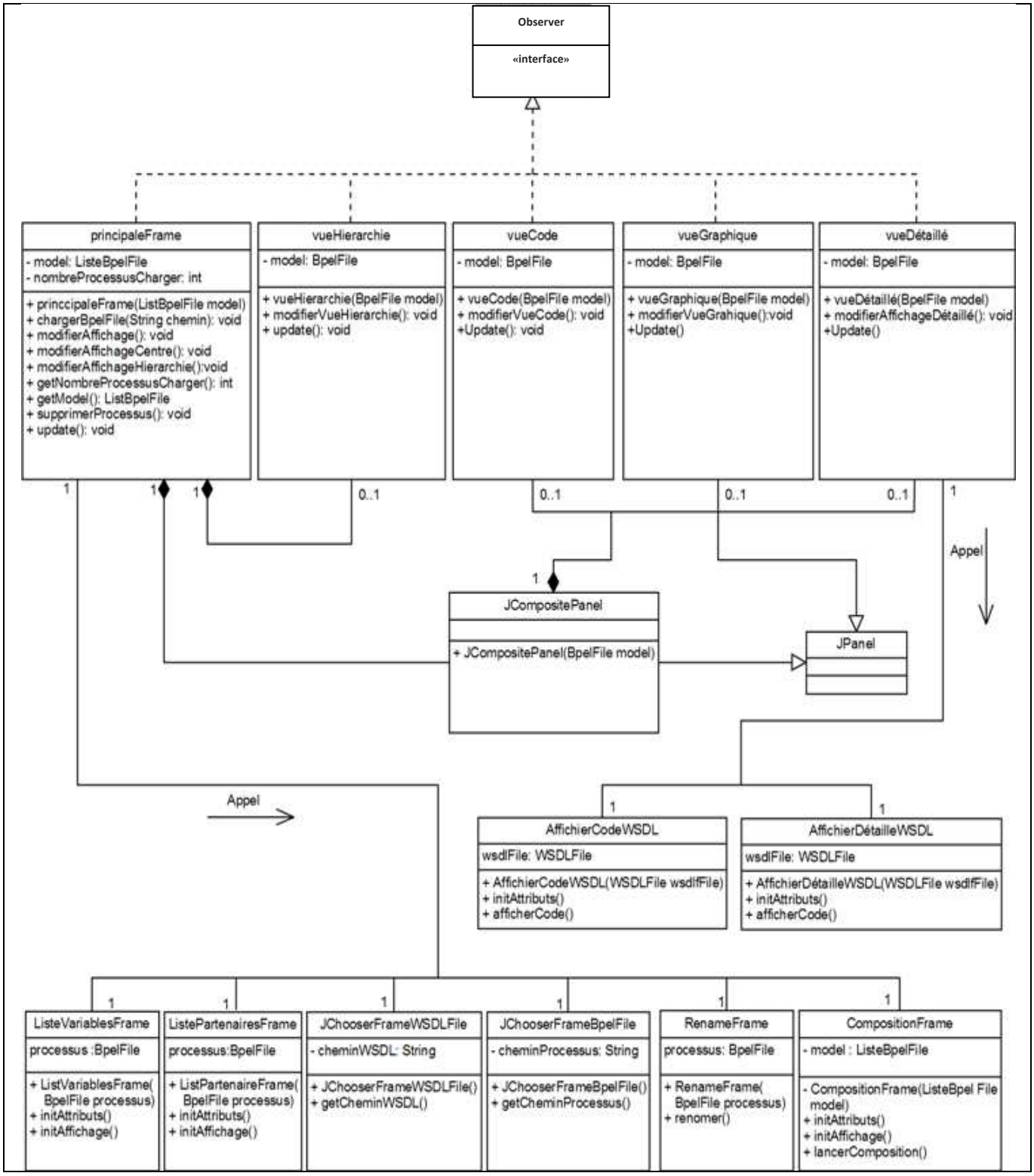



**Figure VII.5.** *Diagramme de classes de la partie « Vue »*

### VII.1.3.3 Les classes de la catégorie « Contrôleur »

La partie « Contrôleur» encapsule les traitements nécessaires pour chaque événement provenant du concepteur de WFIO, particulièrement les traitements concernant la composition (ou l'interconnexion) des modèles de WF qui permettent aussi de choisir la stratégie adéquate pour traiter cet événement. La figure VII.6 décrit le diagramme de classes associé à cette partie.

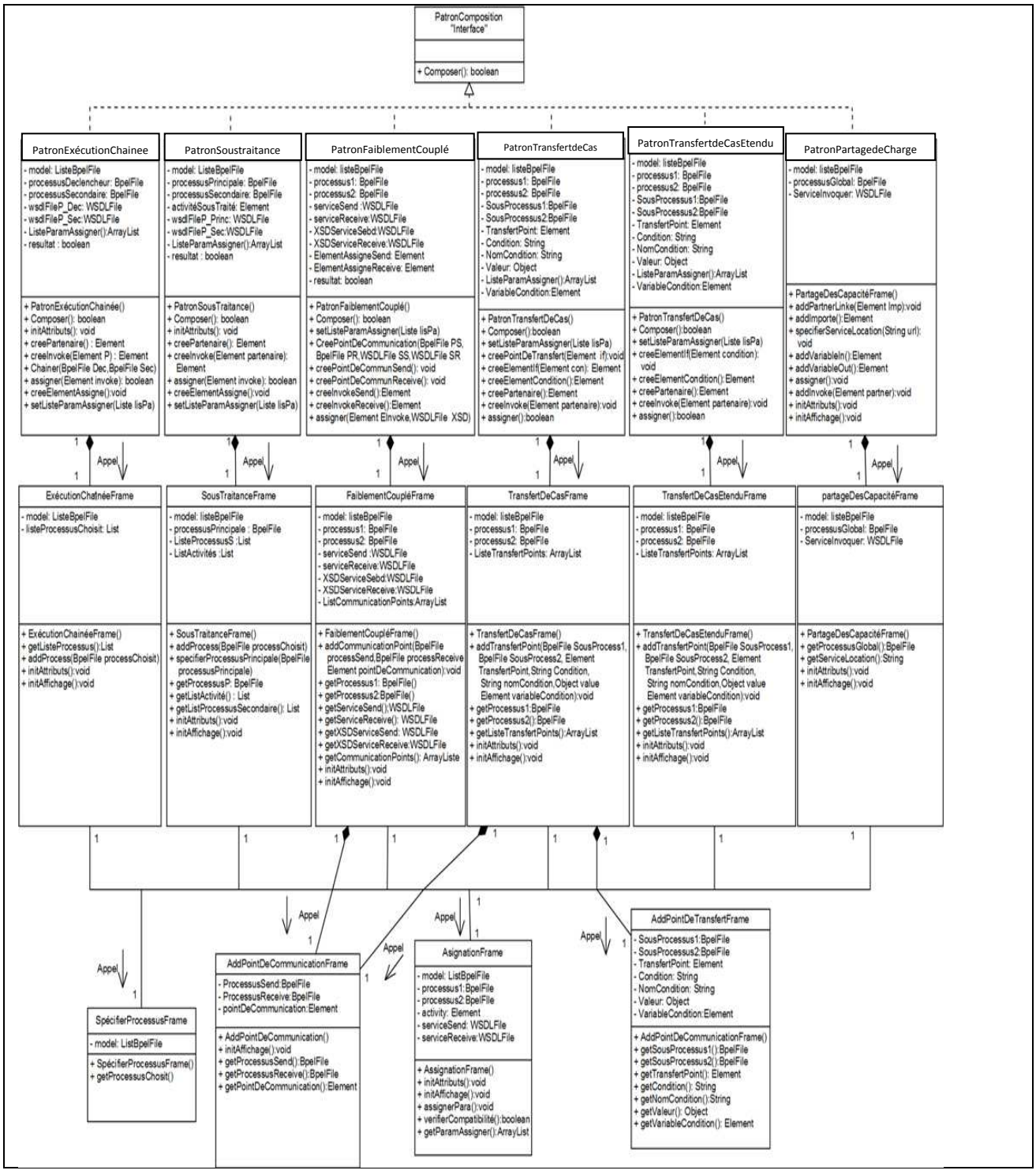



**Figure VII.6.** *Diagramme de classes de la partie « Contrôleur»*

- **Quelques méthodes des classes de la catégorie « Contrôleur»**

Les listings suivants fournissent quelques détails d'implémentation de certaines classes de la partie « Contrôleur», notamment les méthodes qui permettent de réaliser la composition entre deux fichiers BPEL.

```
Public boolean Chainer (BpelFile PremierProcessus, BpelFile SecondProcessus)
{
  Element  elementPartenaire= creePartenaire (PremierProcessus);
  Element  elementInvoke = creeElementInvoke(elementPartenaire,
PremierProcessus) ;
  Boolean resultat =  Assigner(elementInvoke, PremierProcessus,
SecondProcessus);
  return resultat;
}
```

**Listing VII.6.** *Code java de la méthode* Chainer *de la classe* PatronExecutionChainée

**Commentaires**

La méthode Chainer permet de réaliser une interconnexion entre deux processus BPEL nommés *PremierProcessus* et *SecondProcessus* pour réaliser un parton de type « Exécution chainée ». Il s'agit de créer un *élément* invoke dans le premier processus pour invoquer le second processus. La méthode Assigner permet d'assigner les paramètres d'E/S entre le processus invocateur et le processus invoqué afin de garantir un flux de données cohérent dans le processus global.

```
public boolean SousTraiter (BpelFile ProcessusPrincipal, BpelFile
ProcessusSecondaire, Element ActivitéASousTraiter)

{ Element  elementPartenaire= creePartenaire (ProcessusPrincipal) ;

Element  elementInvoke = creeElementInvoke
(elementPartenaire,ProcessusPrincipal) ;

Boolean resultat = Assigner (elementInvoke,
processusPrincipal,ProcessusSecondaire,
activitéASousTraiter) ;

return resultat;

}
```

**Listing VII.7.** *Code java de la méthode* Soustraiter *de la classe* PatronSousTraitance

**Commentaires**

La méthode SousTraiter permet de réaliser une interconnexion entre deux processus BPEL nommés *ProcessusPrincipal* et *ProcessusSecondaire* pour réaliser un parton de type « Sous-traitance ». Il s'agit de créer un *élément* invoke dans le processus principal (en remplacement de l'activité *ActivitéASousTraiter*) pour invoquer le processus secondaire. La méthode Assigner permet d'assigner les paramètres d'E/S entre le processus invocateur et le processus invoqué afin de garantir un flux de données cohérent dans le processus global.

La classe **PatronTransfertdeCas** implémente le patron de coopération Transfert de cas et contient les méthodes CreePointdetransfert, CreeElementIf, CreeCondition, CreePartenaire et Assigner. Par exemple, la méthode CreePointdetransfert sert à créer un élément de type action alternative correspondant à une condition de transfert au niveau d'un point de transfert dans le processus, en invoquant les méthodes CreeElementIf, CreeCondition et addElementIF implémentées dans la même classe.

```
public boolean CreePointdetransfert (Condition cond, Element pointTransfert,
BpelFile  Processus1, BpelFile processus2)
{
Element  elementCondition= creeCondition (cond);
Element  elementIF= creeElementIf (elementCondition, pointTransfert) ;
Boolean resultat= addElementIF (processus1, processus2) ;
return resultat;
}
```

**Listing VII.8.** *Code java de la méthode* `CreePointdetransfert` *de la classe* `PatronTransfertdeCas`

La classe **`PatronFaiblementCouplé`** contient aussi plusieurs méthodes dont la méthode `CreePointDeCommunication` qui permet de créer un point d'interaction entre deux processus BPEL appelés respectivement *processSend* et *processReceive* pour processus émetteur et processus récepteur. Cette méthode utilise les fichiers WSDL de description des services Web d'interaction appelés `SW_Send` et `SW_Receive`. Cette méthode fait appel à deux autres méthodes `creePointDeCommunicationSend` et `creePointDeCommunicationReceive` définies dans la même classe et permettant de créer les points d'interaction avec les activités « invoke » et « receive », respectivement.

```
public boolean creePointDeCommunication(BpelFile processusSend, BpelFile
processusReceive, WSDLFile SW_Send, WSDLFile SW_Receive)
{
      AssignationFrame assignerSendResponse = new
      AssignationFrame(processusSend, processusReceive, SW_Send, SW_Receive);
      creePointDeCommunicationSend (processusSend, SW_Send);
      creePointDeCommunicationReceive (processusReceive, SW_Receive);
}
```

**Listing VII.9.** *Code java de la méthode* `CreePointDeCommunication` *de la classe* ***`PatronFaiblementCouplé`***

## VII.1.4 Organisation des classes du framework

### VII.1.4.1 Diagramme de packages

Les trois catégories de classes su-décrites sont organisées en packages, à savoir un package « Modèle », un package « Vue » et un package « Contrôleur », en plus d'un package pour les classes utilitaires comme les boites de dialogue.

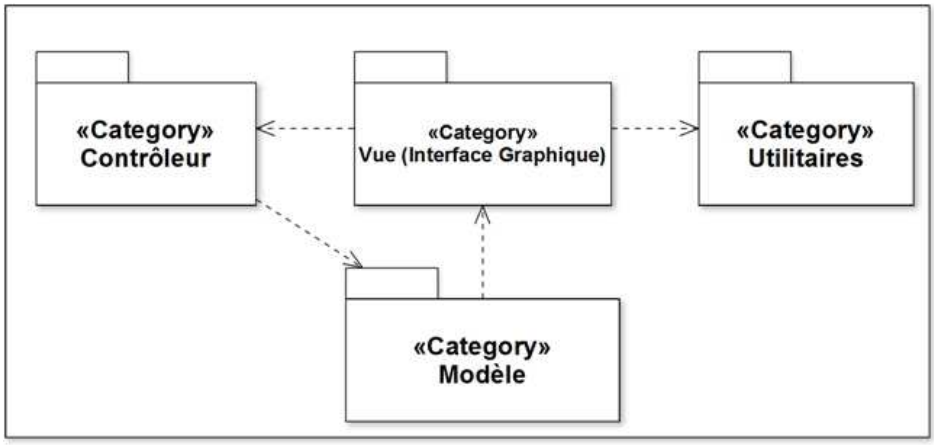



**Figure VII.7.** *Diagramme de packages du framework S-IOFLOW*

**TAB. VII.1.** *Description des packages dans le framework de coopération*

| Package | Les Classes principales |
|---|---|
| Package « *Modèle* » | BpelFile<br>ListeBpelFile<br>WSDLFile |
| Package « *Vue* » | PrincipaleFrame<br>VueHierarchie<br>VueCode<br>VueGraphique<br>VueDétail |
| Package « *Contrôleur* » | PatronExécutionChainée<br>PatronSousTraitance<br>PatronFaiblementCouplé<br>PatronTransfertDeCas<br>PatronTransfertDeCasEtendu<br>PatronPartageDeCharge |
| Package «*Utilitaire* » | Contient toutes les classes supplémentaires nécessaires pour le fonctionnement de l'application tel que les boites dialogue « ChooserBpelFileFrame », « ChooserWSDLFileFrame »...etc. |

### VII.1.4.2 Diagramme de composants

Nous illustrons sur la figure VII.8, l'architecture physique et statique de notre application en termes de composants. Les composants peuvent être organisés en paquetages, qui définissent des sous-systèmes. Ces derniers permettent de gérer la complexité, par encapsulation des détails d'implémentation.

Nous considérons trois composants de granularité élevé (Modèle, Vue, Contrôleur) pour notre framework de coopération, nous montrons pour chaque composant ses interfaces fournies et requises. Le composant contrôleur est composé d'autres composants qui encapsulent, chacun les classes spécifiques à la réalisation d'un patron de coopération parmi les six considérés.

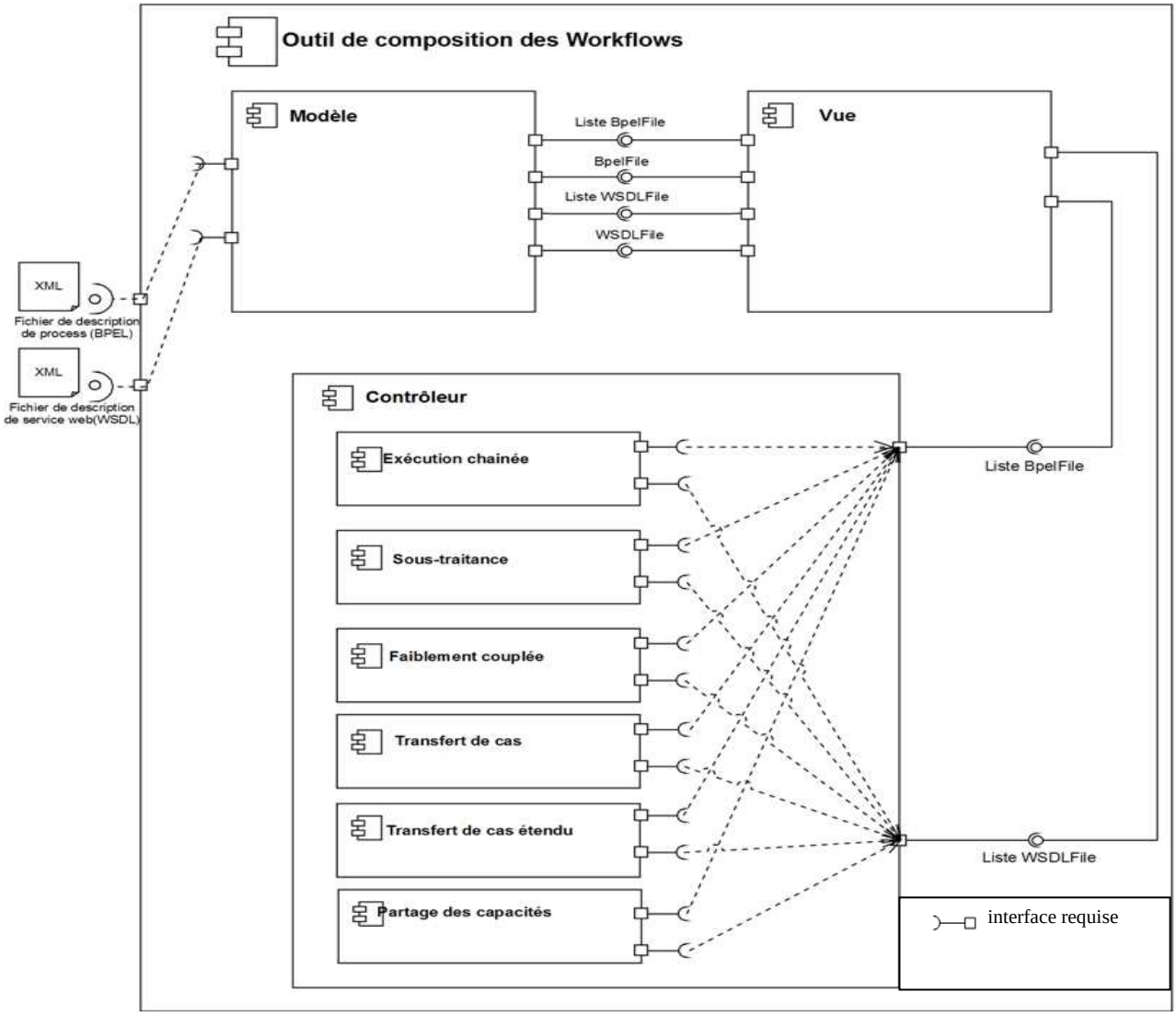



**Figure VII.8.** *Diagramme de Composants du framework « S-IOFLOW »*

## VII.1.5 Fonctionnalités de notre framework de coopération « S-IOFLOW »

Nous décrivons les fonctionnalités offertes par notre framework de coopération à travers le diagramme de cas d'utilisation global de la figure VII.9. Le framework permet entre autres de :

- Charger un processus BPEL,
- Renommer un processus BPEL,
- Afficher différentes vues d'un processus BPEL,
- Choisir un patron de coopération
- Réaliser l'interconnexion entre deux processus BPEL, selon le patron de coopération choisi
- Maintenir un flux de données cohérent dans le processus global
- Déployer et tester le processus

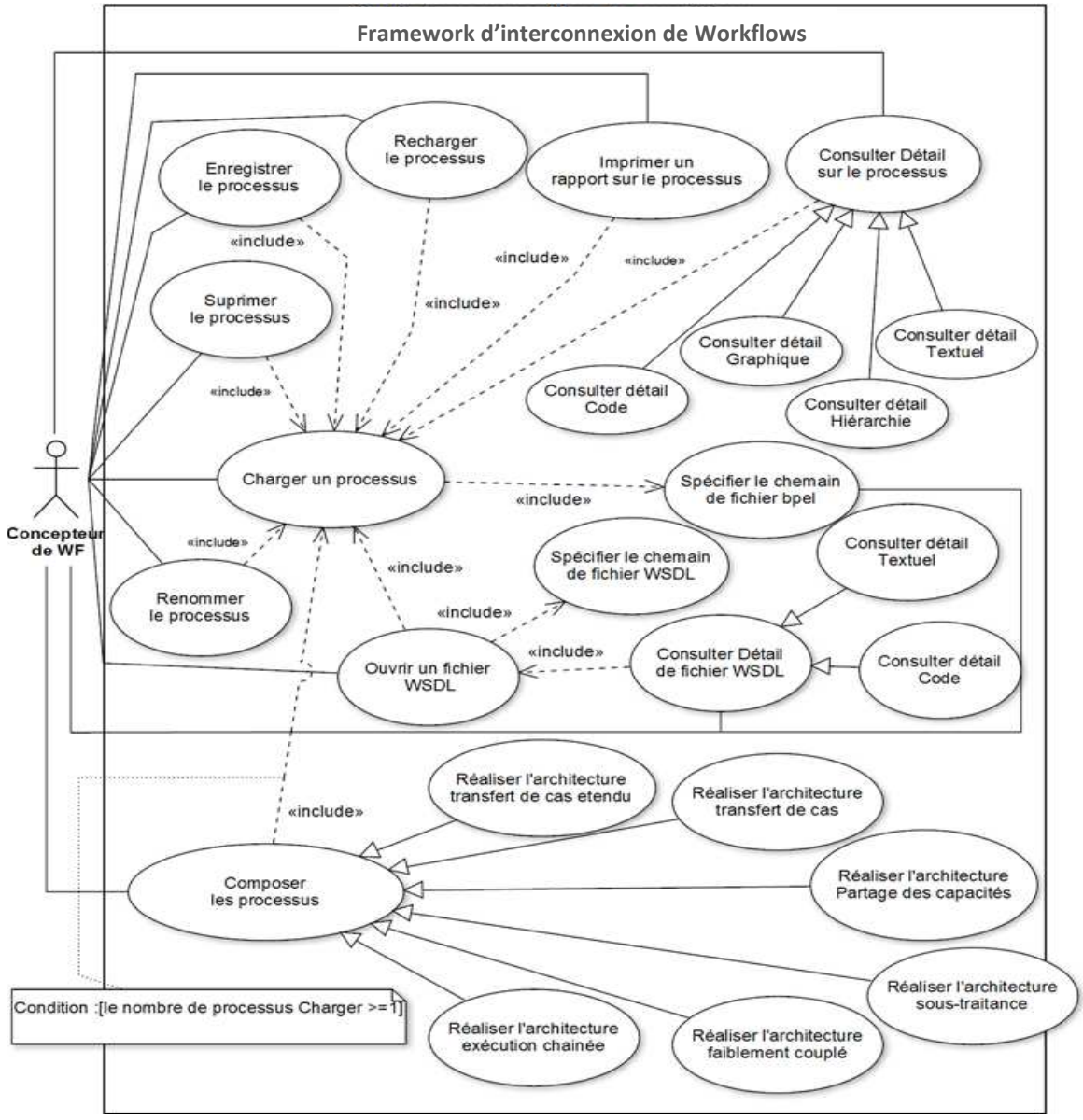



**Figure VII.9.** *Diagramme de cas d'utilisation global du framework « S-IOFLOW »*

### VII.1.5.1 Zoom sur la fonctionnalité « Réaliser l'architecture faiblement couplée »

A titre d'exemple, un zoom sur le cas d'utilisation « Réaliser l'architecture faiblement couplée » donne le diagramme de cas d'utilisation de la figure VII.10 ; ce cas d'utilisation en utilise d'autres tels que les cas d'utilisation « Ajouter Point de communication », « Spécifier les services d'interaction » et « Assigner les variables ».

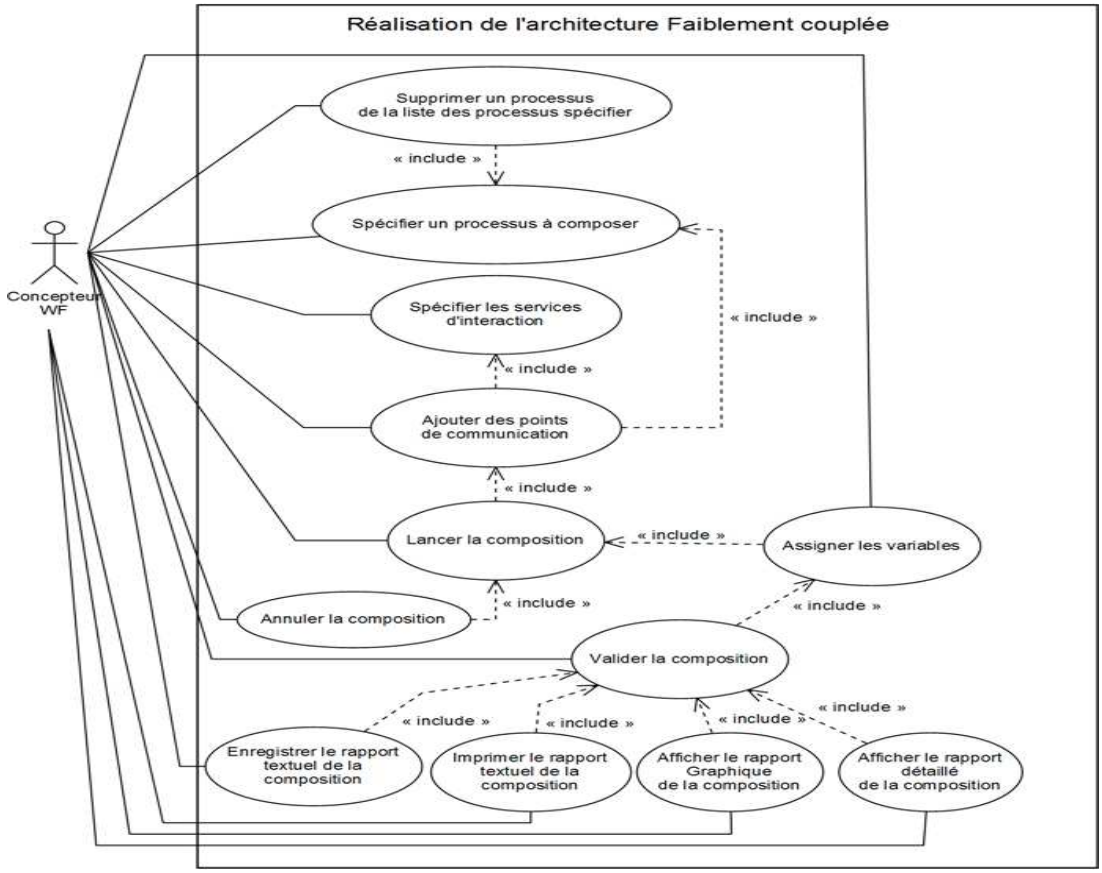



**Figure VII.10.** *Diagramme de cas d'utilisation « Réaliser l'architecture faiblement couplée »*

- **Diagramme de séquence du cas d'utilisation « Réaliser l'architecture faiblement couplée »**

Le diagramme de séquence de la figure VII.11 montre les interactions entre les objets du système pour le cas d'utilisation « Réaliser l'architecture faiblement couplée », ces objets sont des instances de classes appartenant à l'une des trois catégories « Modèles », « Vues » et « Contrôleurs ».

Les classes « `FaiblementCoupléeFrame` », « `DefinePointdeCommunicationFrame` » et « `AssignationFrame` » sont des classes de la catégorie « Vues » notifiées par les classes « `AddPointDeCommunication` » et « `PatronFaiblementCouplé` » de la catégorie « Contrôleurs » suite aux changements effectués sur les classes `BpelFile` et `ListBpelFile` de la catégorie « Modèles ». On peut voir sur ce diagramme de séquence, des références vers d'autres cas d'utilisation : « Spécifier les services d'interaction » et « Notifier les vues observant `BpelFile` »

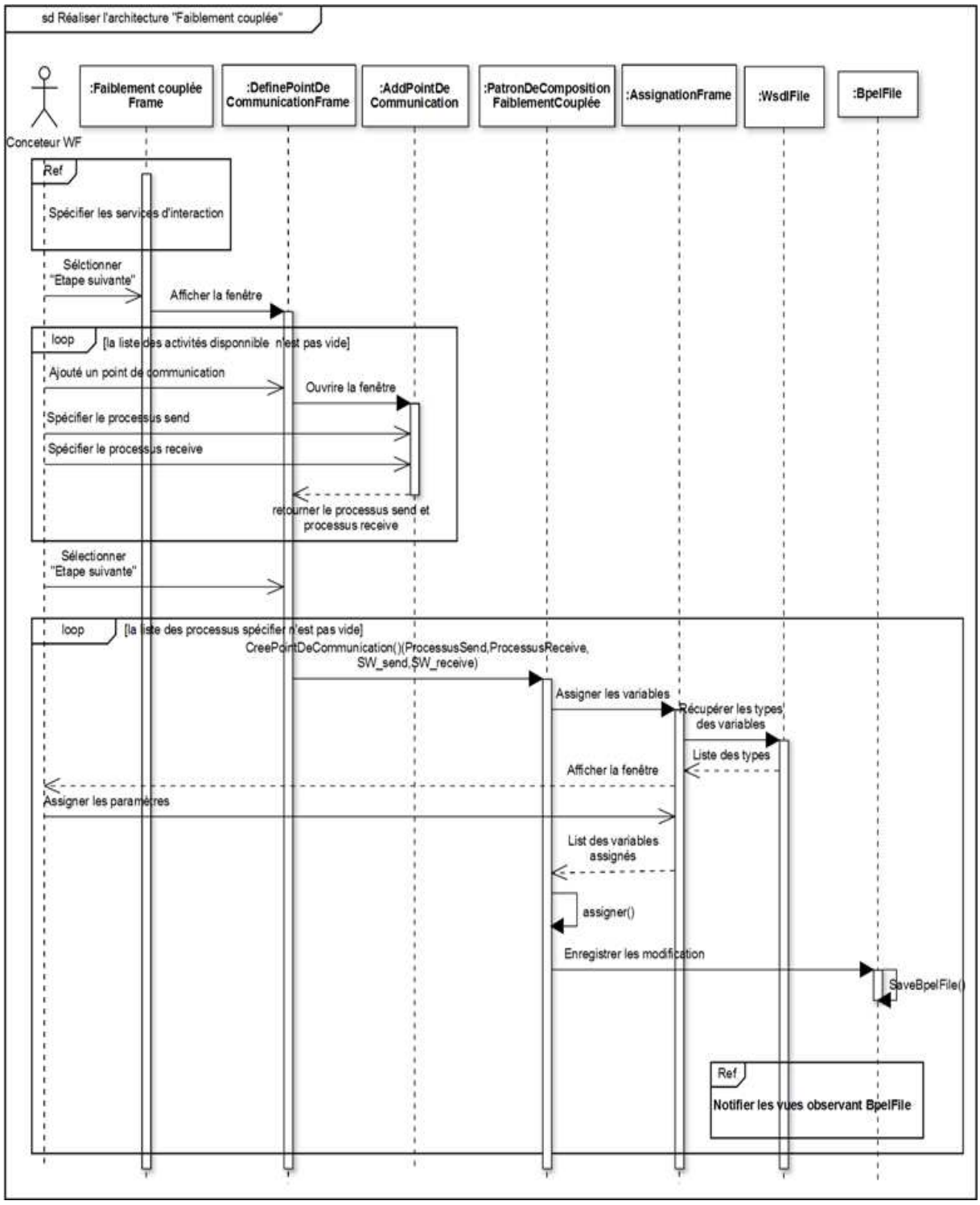



**Figure VII.11.** *Diagramme de séquence du cas d'utilisation « Réaliser l'architecture faiblement couplée »*

- **Diagramme de communication du cas d'utilisation « Réaliser l'architecture faiblement couplée »**

Le diagramme de communication de la figure VII.12 donne une autre vue des interactions entre les objets des différentes classes de ce même cas d'utilisation. Sur ce diagramme, on utilise le stéréotype « GUI » pour les classes de type interfaces (les vues), et les stéréotypes « Modèle » et « Contrôleur ».

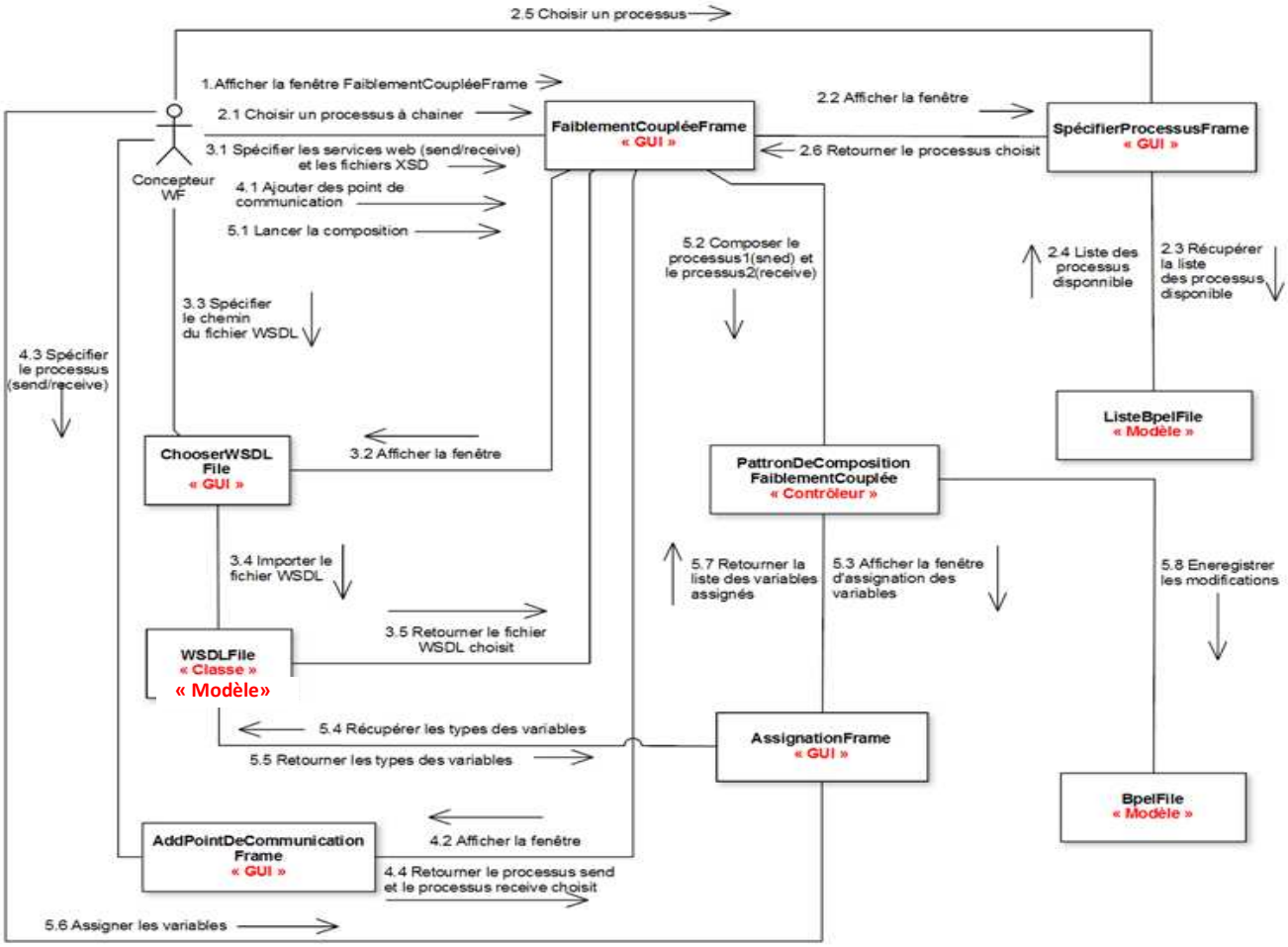



**Figure VII.12.** *Diagramme de communication du cas d'utilisation « Réaliser l'architecture faiblement couplée »*

## VII.1.6 Assistants de coopération

Le framework « S-IOFLOW » fournit un ensemble d'assistants de coopération comportant chacun, un ensemble d'étapes devant être suivies par le concepteur de WFIO pour la réalisation d'un patron de coopération donné. Donc, l'utilisation de notre framework de coopération se fait via cet ensemble d'assistants dont l'un est choisi, conformément au patron de coopération à réaliser. Le tableau VII.2 décrit les assistants de coopération en question en mettant l'accent notamment sur l'architecture de déploiement, c'est-à-dire les rôles de client/serveur sur les sites de partenaires, le type de contrôle (centralisé, décentralisé ou hiérarchisé), les étapes de chaque assistant de coopération et quelques détails d'implémentation. Ces derniers concernent notamment la mise à jour du flux de données entre les processus en coopération, à travers une phase d'assignation semi-automatique des paramètres d'entrée/sortie. D'autres détails concernent l'ajout des points de transfert dans l'architecture « Transfert de cas » et l'ajout des services d'interaction dans l'architecture « Faiblement couplée ».

A titre illustratif, nous présentons dans l'annexe C de ce document, des scénarios d'exécution permettant de montrer le déroulement des étapes des assistants « Transfert de cas » et « Faiblement couplé ».

**TAB VII.2.** *Description des assistants de coopération*

| Nom-Assistant | Architecture de deployment | Etapes | Détails d'implémentation |
|---|---|---|---|
| **"Partage de Charge"**<br>**"Capacity Sharing"**<br>**PCBS1 - SBCP1** | Un site **Client** (l'orchestrateur)<br>et<br>un ensemble de sites **Serveur**<br><br>- Contrôle **centralisé** | **Etape 1:** Sélectionner l'architecture<br>**Etape2:** Sélectionner les services<br>**Etape 3:** Définir le processus global -------<br><br>**Etape 4:** Assigner les paramètres ---------- | Spécifier le contrôle de flux entre les services à travers une interface appropriée pour l'ajout d'activités de type sequence, if, flow,…<br>Les paramètres d'**entrée** d'un service doivent être les paramètres de **sortie** du ou des services qui le précèdent directement |
| **"Exécution Chainée"**<br>**"Chained Execution"**<br>**SBCP2- SBCP2** | Tous les sites de partenaires sont **Client/Serveur** excepté le dernier qui est seulement **Serveur**.<br>- Contrôle **décentralisé** | **Etape 1:** Sélectionner l'architecture<br>**Etape 2:** Sélectionner les processus<br>**Etape 3:** Assigner les paramètres ----------<br><br>**Etape 4:**Valider la composition | Spécifier le premier processus et le second processus<br>Les paramètres d'**entrée** du second processus doivent être les paramètres de **sortie** du premier processus. |
| **"Sous-traitance"**<br>**"Subcontracting"**<br>**PCBS3 - SBCP3** | Le site du partenaire principal est **Client/Server**<br><br>et<br>les sites des partenaires secondaires sont des **Serveurs**<br><br>- Contrôle **hiérarchisé** | **Etape 1:** Sélectionner l'architecture<br>**Etape 2:** Sélectionner les processus ---------<br><br>**Etape3:** Faire la correspondance entre l'invocation d'un service **abstrait** (**vide,** dans le processus principal) et le processus secondaire<br><br>**Etape 4:** Assigner les paramètres ----------<br><br>**Etape5 :** Valider la composition | Spécifier le processus principal avec invocation de services abstraits (**vides** - **empty** services) et spécifier le processus secondaire.<br><br>Substituer l'invocation de tout service "**empty**" par l'invocation du processus secondaire approprié.<br><br>Les paramètres d'**entrée/sortie** du processus secondaire doivent être les paramètres de **sortie** du service abstrait correspondant dans le processus principal. |
| **"Transfert de cas"**<br>**"Case-Transfer"**<br>**PCBS4 - SBCP4**<br><br>**"Transfert de cas étendu"**<br>**"Extended Case-Transfer"**<br>**PCBS5 - SBCP5** | Tous les sites des partenaires sont **Client/Serveur**<br><br>- Contrôle **décentralisé ou mixte.** | **Etape1:** Sélectionner l'architecture<br>**Etape2:** Sélectionner les partenaires et le processus<br>**Etape 3:** Ajouter les **points de transfert** dans le processus et créer les **sous-processus**.<br><br>**Etape 4 :** Assigner les paramètres ----------<br>**Etape 5:** Valider la composition | Pour chaque **Point de transfert**, définir:<br>- sa localisation dans le processus (avant/après quelle service?),<br>- la condition (en utilisant un éditeur simple fourni) et<br>- l'action alternative dans le modèle de processus<br>- créer les **sous-processus**<br>Les paramètres d'entrée d'un service externe invoqué doivent être les paramètres de sortie du service qui précède l'action alternative. |
| **"Faiblement Couplé**<br>**"Loosely Coupled"**<br>**PCBS - SBCP6** | Tous les sites des partenaires sont **Client/Serveur**<br><br>Contrôle **décentralisé** | **Etape 1:** Sélectionner l'architecture<br>**Etape 2:** Sélectionner les processus<br>**Etape 3:** Insérer les services d'**interaction** dans chaque processus.<br>**Etape 4:** Assigner les paramètres ----------<br><br>**Etape 5:** Valider la composition | Deux types de services d'**interaction** sont définis: ***ServiceSend*** avec des données de sortie et ***Service*Receive** avec des données d'entrée.<br>Les paramètres d'**entrée** du processus récepteur (ServiceReceive) doivent être les paramètres de **sortie** du processus invocateur (ServiceSend). |

## VII.2 Le framework d'adaptation/évolution

Dans cette section, nous nous focalisons sur la description de notre framework d'adaptation/évolution, en présentant son architecture générale, le découpage logiciel de notre solution ainsi que la logique d'utilisation des fonctionnalités offertes par le framework. Comme le framework de coopération, le framework d'adaptation/évolution est destiné à être utilisé par un concepteur qui se charge de réaliser une adaptation sur un modèle de WFIO. L'adaptation peut être locale à un fragment de processus ou globale affectant la structure d'interaction entre deux fragments de processus. Dans le premier cas (adaptation locale), les adaptations se basent sur les patrons d'adaptation de services et les patrons d'adaptation de contrôle de flux. Par contre dans le deuxième cas (adaptation globale), les changements se basent sur les patrons d'adaptation des interactions et les patrons d'évolution. Le framework d'adaptation/évolution permet entre autres, de :

- Charger le (ou les) processus BPEL objet de l'adaptation/évolution
- Sélectionner à chaque fois, un patron d'adaptation/évolution à réaliser
- Apporter les changements nécessaires au niveau des fichiers BPEL concernés.
- Maintenir un flux de données cohérent dans le processus (vérifier les contraintes d'adaptation/évolution)
- Déployer les processus de WFIO obtenus et observer leur exécution

### VII.2.1 Architecture générale du framework d'adaptation/évolution

La figure VII.13 décrit l'architecture générale de notre framework d'adaptation/évolution. Au centre de l'architecture se trouve l'application qui réalise les adaptations/évolutions des modèles de processus BPEL. L'application comporte les modules suivants: un *module de chargement*, des *modules d'adaptation/évolution* et un *module de validation et persistance*. Remarquons que les modules d'adaptations locales sont aussi déployés sur les sites des partenaires. Ceux-ci vont servir comme nous l'avons déjà souligné, à apporter des modifications locales sur les fragments de WF, évitant ainsi le temps de chargement des processus sur l'infrastructure commune et le temps de répercussion des changements sur les sites des partenaires.

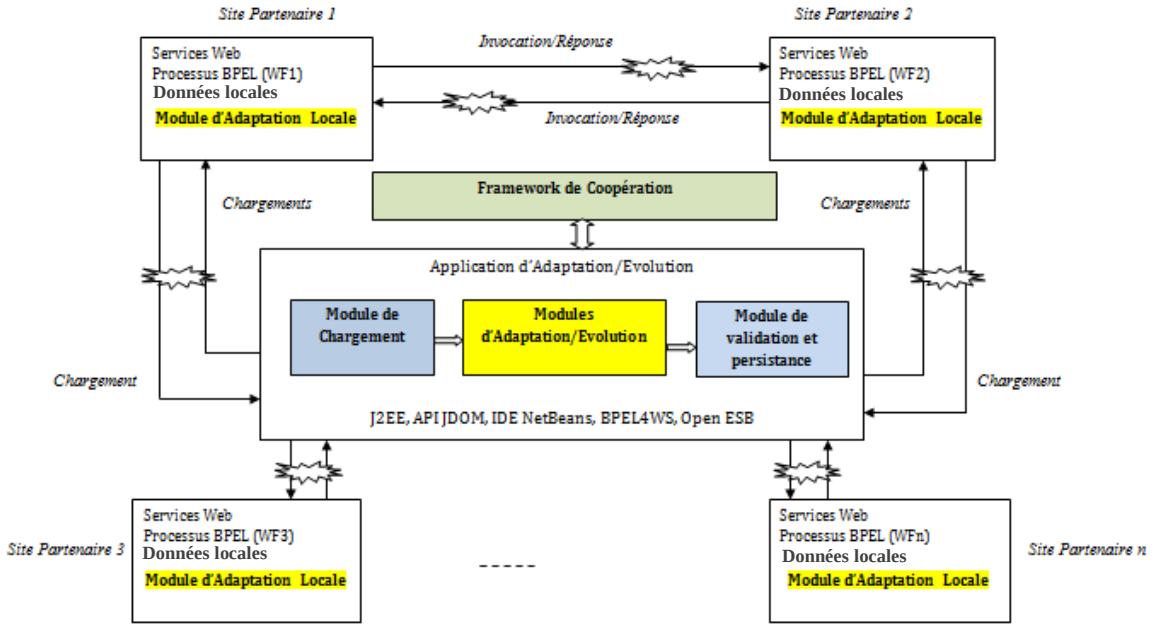



**Figure VII.13.** *Architecture générale du framework d'adaptation/évolution*

La même logique de traitement est donc reprise du framework de coopération. En effet, le module de chargement consiste à charger les processus BPEL concernés par les opérations d'adaptation/évolution à partir de leurs sites respectifs (création d'une copie des fichiers BPEL sur l'infrastructure commune). Notons que l'opération de chargement des fichiers BPEL est nécessaire lorsque les changements à opérer affectent la structure d'interaction entre deux fragments de WF ; ainsi les changements se font de manière centralisée afin de maintenir une cohérence du WFIO global. Autrement, dans le cas d'opérations affectant uniquement les services internes et leur contrôle de flux, l'opération de chargement sur l'infrastructure commune, n'étant pas nécessaire.

Les modules d'adaptation/évolution consistent à opérer les changements nécessaires sur les fichiers BPEL répliqués sur l'infrastructure commune, selon les règles de spécification énoncées dans la description des patrons d'adaptation/évolution (cf. chapitre VI). Une phase de mise à jour du flux de données est nécessaire afin de maintenir une cohérence dans le processus global.

Le module de validation et de persistance consiste à valider l'opération de d'adaptation/évolution effectuée. Les changements effectués de manière centralisée (sur l'infrastructure commune) sur les fichiers BPEL dupliqués (des objets `BpelFile`) seront automatiquement répercutés sur les fichiers BPEL physiques d'origine hébergés dans les sites des partenaires.

### VII.2.2 Diagramme de classes global du framework d'adaptation/évolution

Le diagramme de classes de la figure VII.14 donne une vue globale des classes implémentées dans le framework d'adaptation/évolution. Celles-ci sont organisées selon le patron MVC, la principale classe de la catégorie « Modèle » est `BpelFile` qui hérite de la classe « `observable` », toutes les vues du framework (`VuePrincipale`, `VueAdaptation`, `VueAjout`, `VueSuppr`, `VueEvolution` …) héritent de l'interface « `observer` ». Celle-ci est notifiée par la classe « `observable` » pour tous les changements qui affectent les modèles. Le contrôleur contient un ensemble de classes implémentant les patrons d'adaptation/évolution, ces classes sont : `AjoutServ`, `SupprServ`,`FusionServ`,`SequenceServ`,…`etc`.

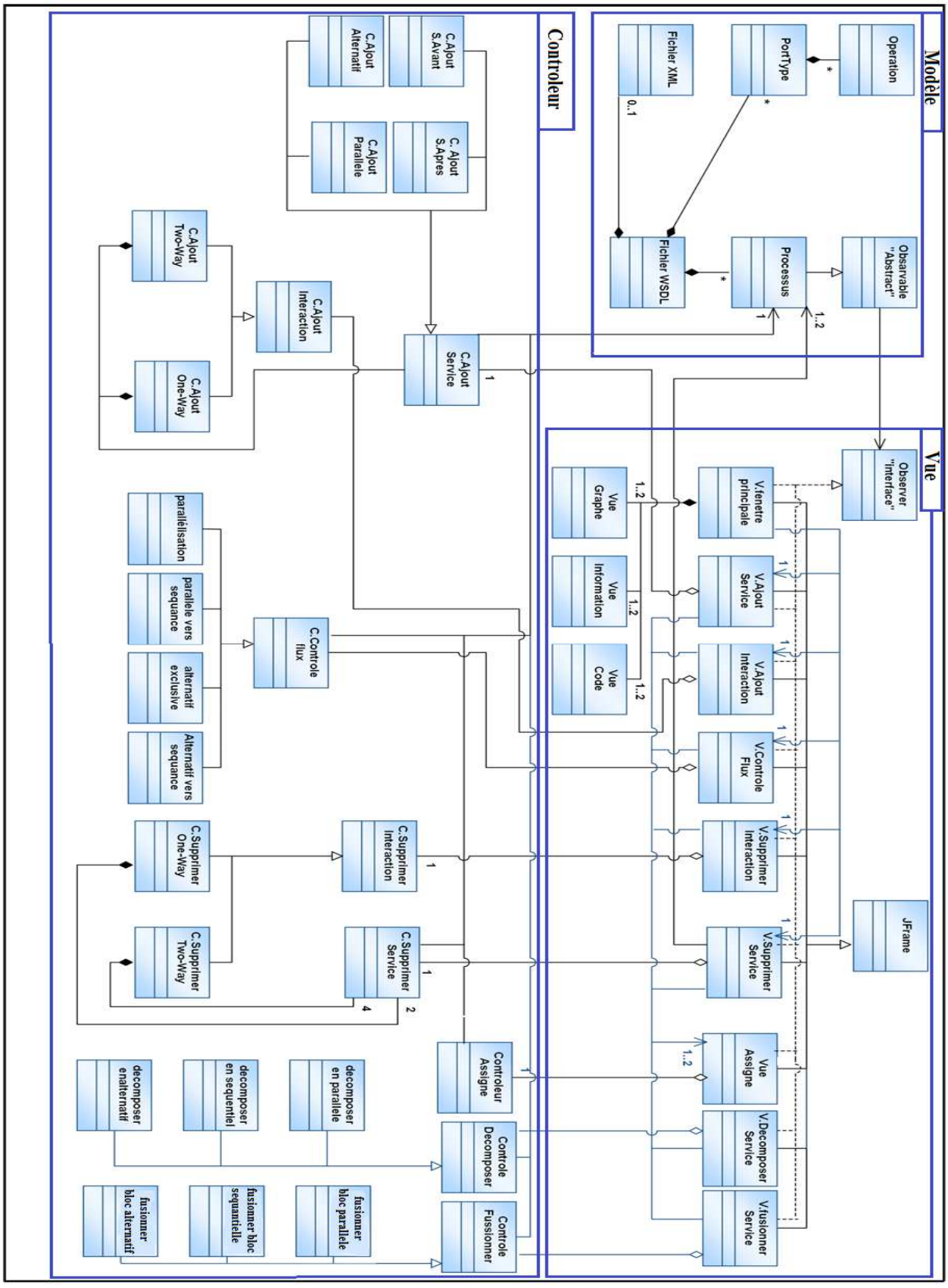



**Figure VII.14.** *Diagramme de classes du framework d'adaptation/évolution*

### VII.2.3 Fonctionnalités du framework d'adaptation/évolution

Nous décrivons les principales fonctionnalités offertes par notre framework d'adaptation/évolution à travers le diagramme de cas d'utilisation global de la figure VIII.15. Le framework permet entre autres de :

- Charger un processus BPEL,
- Adapter processus
- Faire évoluer processus -Visualiser différentes vues du processus
- Déployer et tester le processus

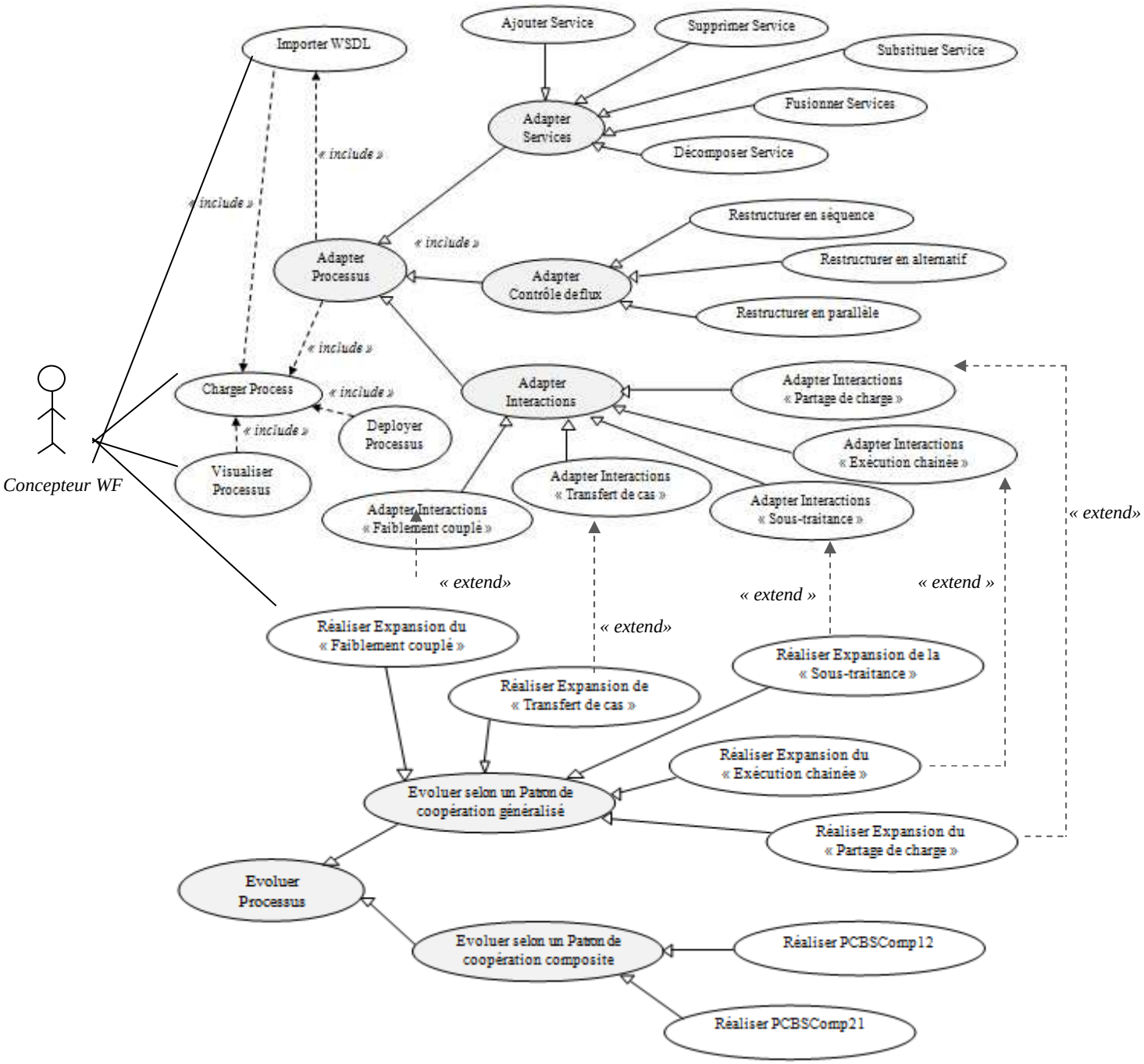



**Figure VII.15.** *Diagramme de Use-case global du framework d'adaptation/évolution*

### VII.2.3.1 Zoom sur la Fonctionnalité « Ajouter Service »

Le concepteur voulant effectuer un ajout de service sur un fragment de WF impliqué dans un WFIO ou non, commence d'abord par charger le processus BPEL, puis il choisit dans la fenêtre principale l'option « Adapter Processus », ensuite l'option « Ajouter Service », l'interface "Ajouter Service" récupère le processus et les services du processus objet de l'adaptation. L'utilisateur fournit un ensemble de données d'entrée : le service à ajouter, le service de référence et le mode d'ajout (avant, après, parallèle, alternatif), puis valide l'opération d'ajout. Cette adaptation s'effectue en utilisant la classe "`AjoutService`" de la catégorie « Contrôleur », celle-ci fournit des méthodes pour la mise à jour du code XML dans le fichier BPEL spécifiant le processus. Une phase semi-automatique d'assignation des variables est nécessaire avant de valider l'adaptation. Les vues observant le modèle de processus sont notifiées suite à toute opération d'adaptation. La figure VII.16 décrit le scénario d'exécution d'une adaptation de type « Ajouter service ».

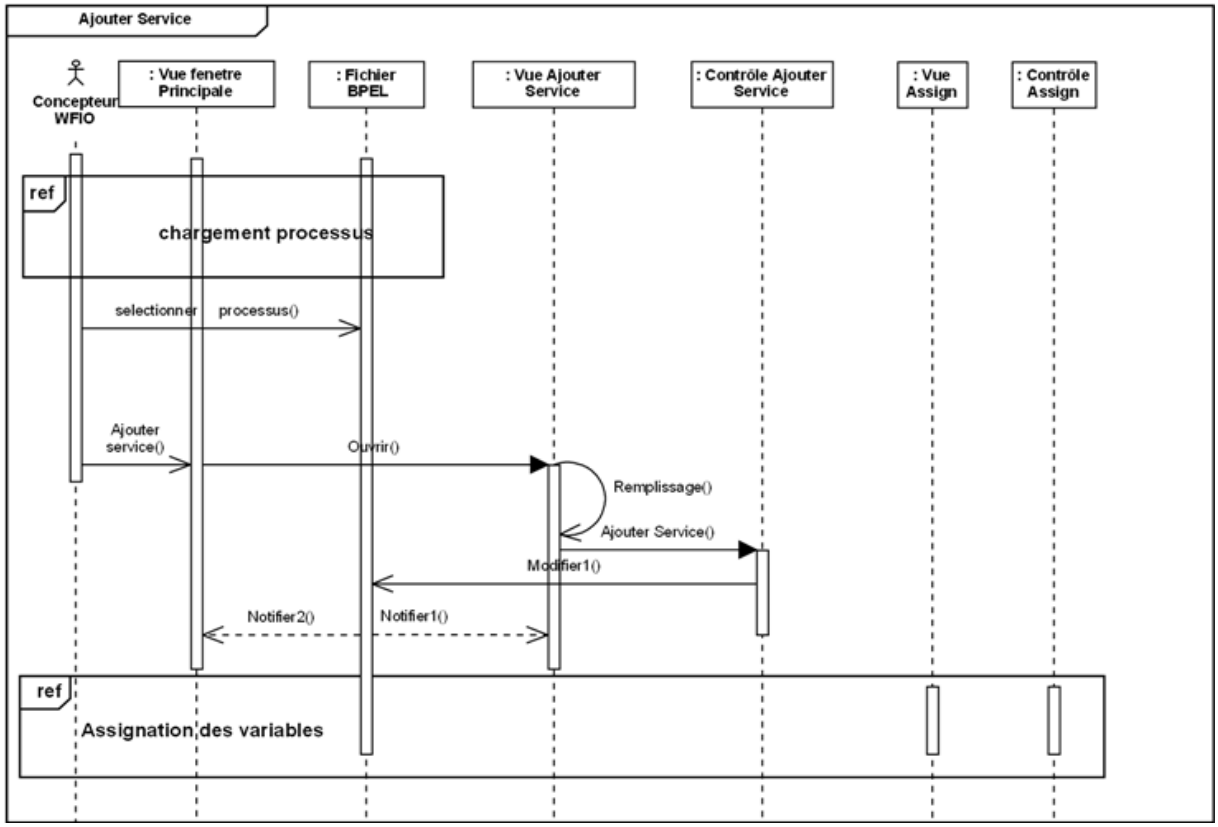



**Figure VII.16.** *Diagramme de séquence du use-case « Ajouter Service »*

### VII.2.3.2 Zoom sur la Fonctionnalité « Adapter contrôle de flux »

Pour adapter le contrôle de flux entre deux services S1 et S2 d'un processus BPEL, le concepteur de WFIO charge le processus sur lequel, il doit effectuer l'adaptation, puis il choisit l'option « Adapter Contrôle de Flux » dans la fenêtre principale. L'interface « Adaptation de contrôle de flux » récupère les services du processus BPEL sélectionné. Une fois que l'utilisateur sélectionne les deux services S1 et S2 à restructurer et le type de restructuration (en séquence, en parallèle ou bien en alternative), il valide l'opération de modification. Cette adaptation s'effectue via la classe contrôleur "`AdapterContrôleFlux`" qui modifie le code XML correspondant dans le fichier

BPEL du processus, elle sera suivie d'une sauvegarde physique du fichier BPEL modifié. Les vues observant le modèle de processus adapté, sont notifiées des changements effectués.

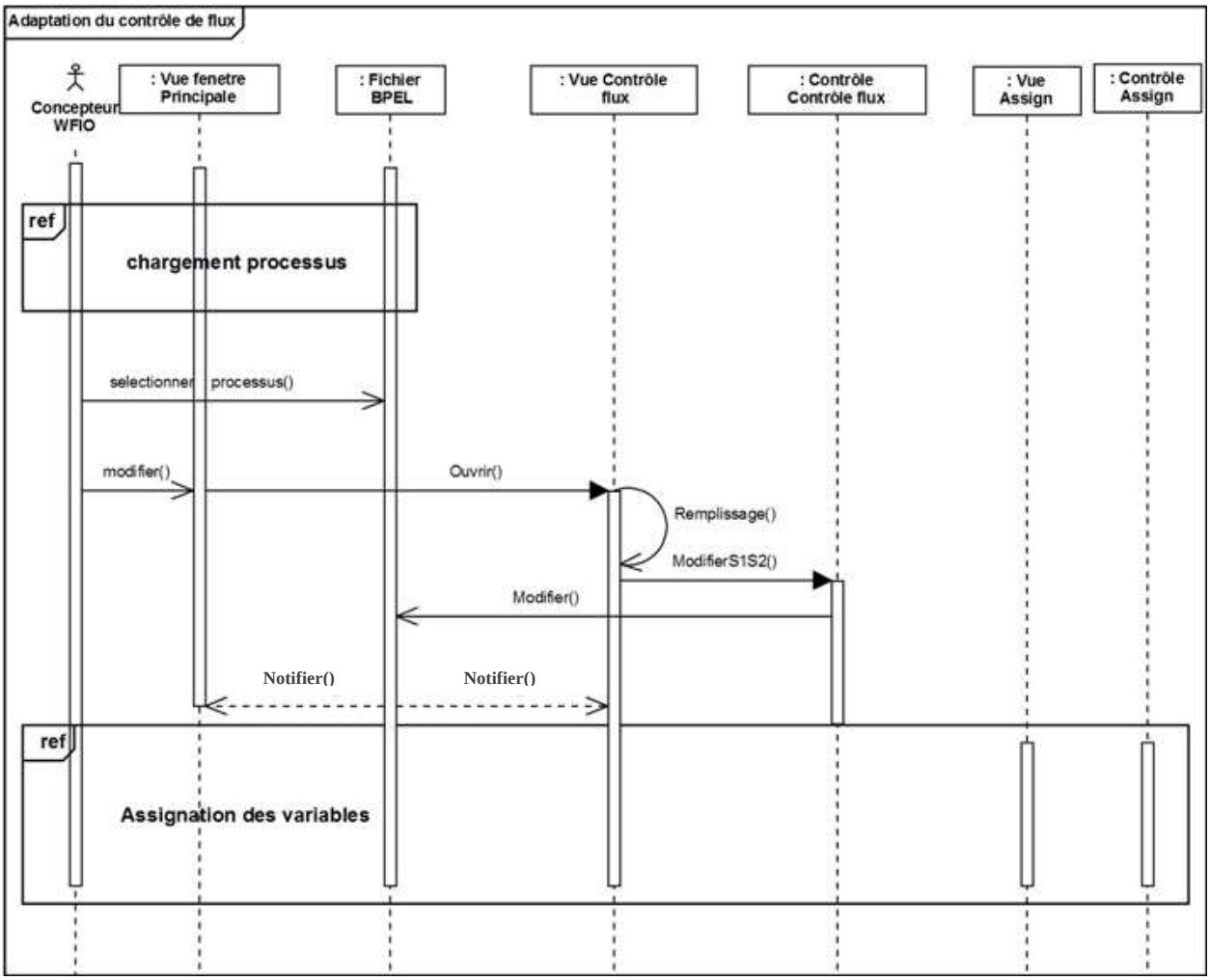



**Figure VII.17.** *Diagramme de séquence du use-case « Adapter Contrôle de flux»*

### VII.2.3.3 Zoom sur la Fonctionnalité « Ajouter point d'interaction One-Way »

Ici, nous décrivons un scénario d'adaptation des interactions lié au patron de coopération « Faiblement couplé ». Le cas d'utilisation « Ajouter point d'interaction One-Way » permet d'ajouter un point d'interaction entre deux fragments de WF interconnectés via le patron « Faiblement couplé ». Il représente une réalisation du cas d'utilisation « Adapter Interactions Faiblement couplé ». Le concepteur charge les deux processus objet de l'adaptation et choisit l'option « Adapter les interactions Faiblement couplé », ensuite « Ajouter Point d'interaction One-way». L'interface associée récupère les services constituant les processus BPEL et les données (Service-send, service de référence, Service-receive, service de référence) nécessaires à l'adaptation. Celle-ci s'effectue via les méthodes de la classe contrôleur « `Ajout-OneWay` », qui sauvegarde le point d'interaction dans un fichier XML « Trace » pour garder la trace des interactions dans le WFIO et ajoute le code XML correspondant dans les fichiers BPEL des deux processus. Cette opération est suivie d'une assignation de variables des deux processus via l'interface « Assignation de variables » et d'une sauvegarde physique des fichiers BPEL adaptés. Enfin, les vues observant les modèles BPEL sont mises à jour à travers la méthode *Notifier()*.

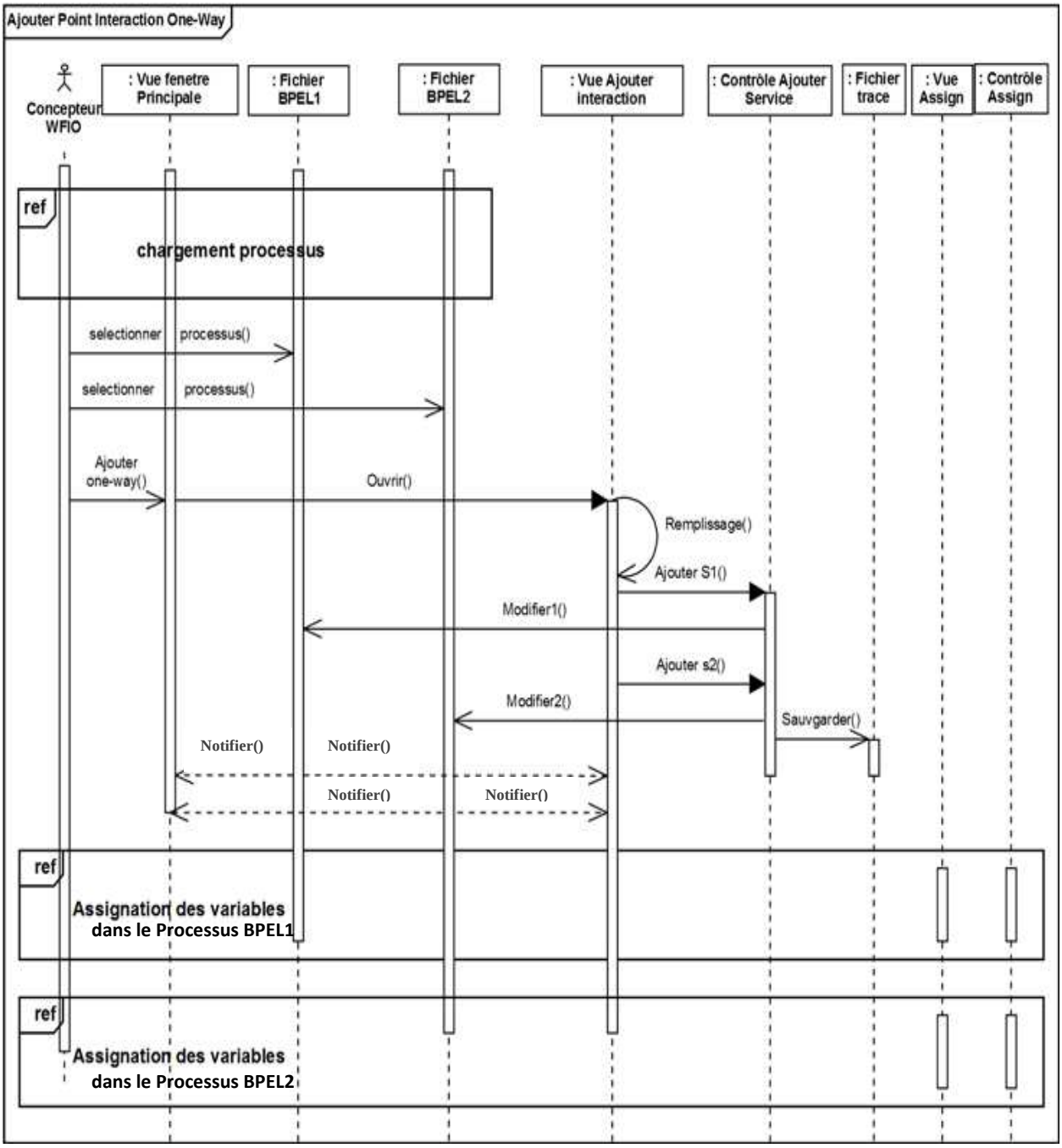



**Figure VII.18.** *Diagramme de séquence du use-case « Ajouter Point d'interaction One-way»*

## Synthèse

Dans ce chapitre, nous nous sommes intéressés aux aspects d'implémentation de nos frameworks de coopération et d'adaptation/évolution. Nous avons mis en évidence les principaux aspects de développement de nos frameworks dont l'architecture générale de déploiement, l'organisation des classes implémentées et les principales fonctionnalités offertes. Certaines d'entre elles sont illustrées dans l'annexe C de ce document.

Le découpage des classes selon le patron de conception MVC permet une maintenabilité et une extensibilité aisée des applications développées, en cas de besoin. A titre illustratif, nous avons jugé nécessaire de montrer quelques portions de code relatives à des méthodes particulières telles que la méthode `notifyview()` implémentée dans la classe `BpelFile`, la méthode `update()`

implémentée dans les différentes vues des frameworks ainsi que certaines méthodes des classes contrôleurs permettant de réaliser les patrons de coopération.

Les principales fonctionnalités des frameworks ont été mises en évidence à l'aide d'un diagramme de cas d'utilisation global pour chacun d'eux. Des zooms sur des cas particuliers accompagnés de diagrammes de séquence ont été présentés afin de modéliser une partie de l'aspect dynamique du framework.

La logique de fonctionnement des frameworks se fait selon une exécution en séquence de trois phases principales : (i) chargement des processus, (ii) interconnexion (ou adaptation/évolution) des processus et (iii) validation et persistance des changements. L'interconnexion des processus qui constitue le cœur du traitement dans le framework de coopération, se fait en utilisant un assistant de coopération choisi, conformément au patron de coopération à réaliser. Chaque assistant comporte un ensemble d'étapes devant s'exécuter en séquence pour guider le concepteur de WFIO dans la construction du modèle de processus inter-organisationnel. De même, les modules d'adaptation/évolution se trouvent au cœur de l'architecture du framework d'adaptation/évolution qui offre au concepteur des interfaces dédiées devant le guider dans chaque opération de changement qu'il doit effectuer sur les modèles de processus BPEL.

Il faut noter que les frameworks développés comportent des phases de traitement semi-automatiques et nécessitent l'intervention du concepteur de WFIO pour l'introduction de certaines informations qui doivent obligatoirement être fournies par un acteur humain comme la localisation d'un point de transfert, la spécification de la condition et l'action à entreprendre dans l'architecture « Transfert de cas », par exemple. De même, dans l'architecture « Faiblement couplée », la localisation des points d'interaction doit se faire par un concepteur de WF. La phase d'assignation des variables qui doit être effectuée suite à chaque opération d'interconnexion ou d'adaptation/évolution est aussi réalisée de manière semi-automatique. Cette phase est indispensable pour le maintien d'un flux de données cohérent dans le processus global, en respectant les contraintes spécifiées dans la conceptualisation des différents patrons (cf. Chapitre V et Chapitre VI).

Suite à chaque opération d'interconnexion ou d'adaptation/évolution, le processus de WFIO obtenu est exécuté en utilisant des applications test sous J2EE, afin d'observer s'il s'exécute correctement de bout en bout. Cependant, nous jugeons nécessaire de développer un outil de vérification formelle devant s'interfacer avec les frameworks développés, afin de garantir la vérification de propriétés comportementales (comme le non-blocage, l'atteignabilité) sur nos processus, après chaque changement qui leur soit apporté. Ce point constitue une perspective importante à notre travail.

# CHAPITRE VIII

# Bilan et perspectives

## VIII.1 Rappel du cadre et des objectifs de la thèse

Dans ce travail de thèse, nous nous sommes intéressés à la coopération B2B, particulièrement le cas d'une *coopération planifiée* soutenue par des concepts de workflow inter-organisationnel (WFIO). Ce type de coopération se trouve répandu dans les relations de partenariat durables visant la réalisation de projets importants à long terme. Notre point de départ était les schémas de coopération (ou architectures de WFIO) proposés par Van der Aalst (Van der Aalst, 99), (Van der Aalst, 2000) pour supporter une coopération entre des processus WF issus de différentes organisations. Ces schémas de coopération bien reconnus et référencés dans les travaux liés au B2B sont caractérisés par trois dimensions principales : le partitionnement du processus, le contrôle d'exécution et la structure d'interaction qui diffèrent d'un schéma de coopération à l'autre. Cette spécificité leur permet de couvrir un grand nombre de processus métiers existants dans le domaine du B2B et constitue notre principale motivation pour les considérer comme point de départ de notre travail.

Cependant, dans leur forme initiale, le principal inconvénient de ces schémas de coopération est que les modèles de WFIO sous-jacents sont rigides et difficilement adaptables face aux changements qui peuvent survenir, notamment les changements affectant deux (ou plus) fragments de processus en interaction. Ce qui constitue une véritable contrainte dans un environnement métier instable et en perpétuelle évolution, exigeant de suite une réactivité et une souplesse d'adaptation des processus métiers tant au niveau intra-organisationnel qu'au niveau inter-organisationnel. A partir de là, nous nous sommes fixés deux principaux objectifs :

- **Objectif 1** : Proposer de nouveaux schémas de coopération équivalents aux premiers mais suffisamment flexibles pour supporter les changements.
- **Objectif 2 :** Proposer des mécanismes support de la flexibilité (vue sous trois aspects complémentaires : adaptabilité, évolutivité et réutilisabilité) des modèles de WFIO obéissant aux nouveaux schémas de coopération.

Pour atteindre ces objectifs, nous avons identifié un ensemble de sous-problématiques pour lesquelles nous avons essayé d'apporter des solutions dans le cadre de cette thèse :

(1) Construction de processus métiers inter-organisationnels suffisamment flexibles et obéissant à des schémas de coopération bien identifiés.

(2) Adaptation des processus métiers inter-organisationnels, prenant en compte les changements intra-organisationnels et inter-organisationnels, notamment au niveau de la structure d'interaction entre les processus en coopération, tout en maintenant la cohérence du processus global.

(3) Evolution des processus métiers inter-organisationnels dans une perspective d'ouverture vers d'autres partenaires éventuellement, sans perturber la cohérence du processus WFIO déjà existant.

(4) Réutilisation de modèles de processus métiers inter-organisationnels pour la construction de nouveaux modèles de processus plus complexes et à forte valeur ajoutée.

## VIII.2 Nos contributions

Partant du constat que la flexibilité d'un modèle dépend fortement des entités qui le composent et les liens entre eux, nous nous sommes orientés vers une *approche à base de services* (SOA) pour la construction de modèles de processus WFIO avec un degré de flexibilité supplémentaire et obéissant aux schémas de coopération de base. Ce choix nous a permis de tirer profit des caractéristiques essentielles du paradigme SOA dont : le couplage faible, l'interopérabilité, la facilité d'adaptation, d'évolution et de réutilisation des services et des applications qui les utilisent. Nous avons rappelé que l'utilisation des services pour la définition et l'interconnexion des processus métiers n'est pas nouvelle, beaucoup de travaux ont adopté cette approche dans le cadre des coopérations interentreprises. Notre approche d'interconnexion diffère de ces dernières par le fait que les processus métiers que nous considérons ne sont pas appréhendés comme des « boites noires » mais doivent dans certains cas, selon les schémas de coopération à réaliser, exhiber certains détails nécessaires à l'interconnexion. Il s'agit, d'exhiber les points d'interaction entre les fragments de WF à faire coopérer, conformément à la règle générale suivante: « *présenter les modèles de WF avec le maximum d'abstraction (en vue de préserver le savoir-faire des partenaires) et le minimum de visibilité requise pour l'interconnexion* ». De plus, nous avons vu à travers la synthèse des approches de coopération existantes (cf. Chapitre II), que celles-ci ne supportent pas tous les schémas de coopération que nous avons considérés, mais seulement un sous-ensemble d'entre eux.

Comme première étape de notre travail, nous avons proposé une démarche de restructuration et d'interconnexion s'articulant autour de trois phases principales : une phase de « mapping », une phase d' « abstraction » et une phase d' « interconnexion » des modèles de WF. La phase de mapping est nécessaire pour transformer un WF à base d'activités en un WF à base de services, en supposant que les partenaires n'implémentent pas forcément leurs WF locaux selon une approche SOA. Pour cela, nous avons établi une correspondance de concepts entre les deux types de WF et nous avons dégagé les associations suivantes : (*activité, service élémentaire*), (*sous-processus, service composant*) et (*processus, service orchestrateur*). L'automatisation de cette phase relève des travaux liés au MDA (Model Driven Architecture) mais nous ne l'avons pas traité dans le cadre de cette thèse.

La phase d'abstraction repose trois niveaux de visibilité que nous avons définis : le niveau « activité », le niveau « sous-processus » et le niveau « processus » allant de la plus fine jusqu'à la plus grosse granularité en passant par une granularité intermédiaire. Dans le premier cas, les processus WF sont appréhendés comme des « boites blanches » où tous les détails de l'enchainement des activités (ou services élémentaires) sont visibles. Dans le cas d'une granularité intermédiaire, le processus WF est appréhendé telle une « boite grise » car il est découpé en sous-processus (encapsulés dans des services) au niveau des points d'interaction qui doivent rester visibles pour l'interconnexion avec d'autres processus. Dans le cas d'une granularité « processus », le processus WF est perçu comme une « boite noire » (encapsulé entièrement dans un seul service composite - orchestrateur) avec seules les entrées et les sorties visibles.

La phase d'interconnexion repose sur un nouveau concept que nous avons introduit, celui du *Patron de Coopération à Base de Services* (PCBS). Celui-ci repose sur trois dimensions principales : la distribution des services sur les sites des partenaires, le contrôle d'exécution (centralisé, décentralisé, hiérarchisé ou mixte) et la structure d'interaction (synchrone ou asynchrone) entre les services des différents partenaires. Nous avons proposé un méta-modèle support de la définition d'un PCBS dans lequel nous avons fait ressortir le concept de *fonction d'orchestration.* Celle-ci détermine la nature du contrôle d'exécution selon que le WFIO repose

sur une fonction d'orchestration globale et/ou sur plusieurs fonctions d'orchestration locales implémentées sur les différents sites de partenaires.

Nous avons proposé un PCBS pour chacun des schémas de coopération considérés. La description de chaque PCBS comporte une conceptualisation centrée sur un méta-modèle de définition du patron, un ensemble de règles d'abstraction, un ensemble de règles et caractéristiques liées à l'opération d'interconnexion (ou composition) et enfin, un ensemble de contraintes visant à maintenir un flux de données cohérent dans le processus de WFIO global. Une formalisation de chaque patron a été proposée en définissant un opérateur de coopération spécifique à chacun d'eux.

Concernant le deuxième volet de notre travail, celui de la *flexibilité* des modèles de WFIO obéissant aux différents PCBS définis, nous avons commencé par présenter nos définitions de la flexibilité que nous percevons sous trois aspects complémentaires : adaptabilité, évolutivité et réutilisabilité. Ensuite, nous nous sommes focalisés sur chacun des aspects, pour identifier les différentes situations de changement, pouvant altérer les modèles de processus WFIO en vue de décrire nos solutions support de ces changements. Pour l'aspect « adaptation », nous avons identifié trois catégories d'adaptation, conformément aux trois dimensions de définition d'un PCBS : les adaptations de *services*, les adaptations de *contrôle de flux* et les adaptations des *interactions*. La première catégorie vise essentiellement une adaptation au niveau fonctionnel et comportemental du processus, la deuxième catégorie concerne le niveau comportemental uniquement et la troisième catégorie est liée au niveau interactionnel du processus. Nous avons affirmé que les deux premières catégories d'adaptation s'appliquent sur un fragment de WF de n'importe quel partenaire sans altérer la cohérence du WFIO global et ne dépendent pas du patron de coopération qui relie les différents fragments de WF. Tandis que la dernière catégorie (adaptation des interactions) s'applique au WFIO en entier et dépend du PCBS auquel obéit le modèle de WFIO. En effet, les adaptations de cette catégorie sont spécifiques à chaque PCBS ; aussi nous avons identifié les différentes situations de changement affectant la structure d'interaction, auxquelles peuvent être confrontés les processus des partenaires obéissant à tel ou tel autre patron de coopération.

Pour l'aspect « évolution », nous avons défini l'évolution des modèles de WFIO selon deux perspectives principales : l'*expansion des fonctionnalités* et l'*expansion de la coopération*. Cette dernière concerne l'ouverture du WFIO à de nouveaux partenaires en vue d'étendre ses fonctionnalités et/ou impliquer de nouvelles compétences. A cet effet, nous avons introduit deux nouveaux concepts : *patron de coopération généralisé* (PCG) et *patron de coopération composite* (PCC). Le premier repose sur le patron de coopération à base de services (PCBS) généralisé à plus de deux partenaires (au moins trois) ; il est utilisé dans le cas où l'expansion de la coopération préserve le PCBS initialement existant. Bien sûr, l'expansion de la coopération dépend du PCBS et nécessite la définition de patrons d'évolution spécifiques à chaque patron de coopération. Le deuxième concept, celui du PCC est défini par combinaison de deux patrons de coopération différents et est utilisé dans le cas où l'expansion de la coopération induit l'introduction d'un nouveau WFIO obéissant à un autre patron de coopération, au sein du WFIO objet de l'évolution. Dans ce cas, nous avons introduit la notion de patron prédominant et la notion de patron secondaire, représentant les rôles joués par chaque WFIO dans la coopération. En effet, le patron prédominant est celui qui contrôle l'exécution du processus global et le patron secondaire est soumis au contrôle du patron prédominant. Par conséquent, en combinant les différents PCBS deux à deux et en inter-changeant les rôles de patron prédominant et patron secondaire, nous obtenons une vingtaine de patrons composites.

L'intérêt du concept de PCC peut être formulé intuitivement et plus simplement, il s'agit d'une *évolution par réutilisation* de modèles de WFIO ou d'une *réutilisation pour l'évolution* de modèles de WFIO, en vue de construire de nouveaux processus plus complexes et à forte valeur ajoutée.

Les différents patrons d'adaptation/évolution ont été décrits à l'aide d'informations générales (nom, référence, situation d'utilisation, patrons utilisés,…etc.), un schéma générique et un ensemble de contraintes visant également à maintenir un flux de données cohérent dans le processus après adaptation. Une formalisation de chaque patron d'adaptation/évolution a été proposée à l'aide d'opérateurs spécifiques que nous avons introduits.

Côté implémentation, nous avons mis en œuvre un framework de coopération baptisé « S-IOFLOW » (Service based Inter-Organizational workFLOW) présentant un ensemble d'assistants de coopération devant guider le concepteur de WFIO dans la construction d'un modèle de WFIO obéissant à un des patrons de coopération nouvellement définis , en considérant des processus WF spécifiés à l'aide du langage BPEL. Aussi, une grande partie des patrons d'adaptation/évolution proposés ont été implémentés dans un framework d'adaptation/évolution pouvant s'interfacer avec le framework de coopération. Nos frameworks ont été conçus et développés conformément au patron de conception MVC (Model-View- Controller) en vue de garantir la synchronisation entre le modèle et les vues, suite aux traitements fournis par les contrôleurs. Dans nos applications, nous avons plusieurs vues (ex : VueCode, VueHierarchie, VueDétail, …etc.) qui observent le même modèle BpelFile, pour le présenter de différentes manières. Les classes contrôleurs encapsulent de manière très modulaire les différents patrons de coopération et les différents patrons d'adaptation/évolution, respectivement dans les frameworks de coopération et d'adaptation/évolution. Cette modularité permet d'une part, une forte réutilisation des classes implémentées notamment pour les patrons d'adaptation/évolution et d'autre part, une extensibilité et une maintenabilité aisée des applications implémentées.

## VIII.3 Tests des frameworks développés

Nous avons testé les principales fonctionnalités fournies par nos frameworks sur un ensemble de modèles de WF que nous avons interconnectés (conformément aux différents patrons de coopération proposés) et sur lesquels, nous avons opéré un certain nombre d'opérations d'adaptation/évolution. Grâce aux applications tests composites générées sous J2EE, nous avons pu observer l'exécution des processus de WFIO obtenus à chaque fois.

## VIII.4 Limites de nos contributions et Perspectives

Les principales limites de ce travail de thèse se résument dans les points suivants et donnent lieu à des perspectives de recherche:

- **Perspective 1 :** Automatisation des phases de mapping et d'abstraction

La non-automatisation des phases de *mapping* et d'*abstraction* représentant les deux premières phases de notre démarche de restructuration et d'interconnexion de WF. Ces deux phases, notamment la première, requiert l'utilisation d'une approche MDA (Model Driven Approach) pour la transformation automatique d'un WF à base d'activités en un WF à base de services. Notre approche d'interconnexion et d'adaptation/évolution ne peut donc pas être utilisée pour des WF à base d'activités déjà mis en place dans les organisations voulant bâtir une coopération entre eux. De ce fait, notre première perspective est de nous pencher sur l'automatisation des premières

phases de la démarche que nous avons proposée, en utilisant une approche MDA basée sur les correspondances de concepts entre WF à base d'activités et WF à base de services.

- **Perspective 2 :** Automatisation de l'assignation des variables

La phase semi-automatique d'assignation des variables dans les frameworks de coopération et d'adaptation/évolution, représente aussi une limite du présent travail. En effet, à chaque opération d'interconnexion entre deux modèles de WF ou d'adaptation/évolution, le concepteur de WF se trouve confronté à une étape semi-automatique d'assignation de variables, pour faire correspondance les variables de sortie d'un service avec les variables d'entrée d'un autre service, selon les contraintes spécifiées dans la conceptualisation de nos patrons. Dans le but d'automatiser cette correspondance de variables, l'utilisation d'une *ontologie* serait un choix judicieux pour établir une correspondance sémantique entre les données manipulées par les services des différents partenaires. Par conséquent, notre deuxième perspective est de nous focaliser sur l'assignation automatique de variables d'entrée/sortie des services interconnectés.

- **Perspective 3 :** Vérification formelle de la correction des modèles

Une troisième perspective de notre travail consiste en la vérification formelle des modèles de WFIO après toute opération d'interconnexion ou d'adaptation/évolution. En effet, dans notre cas, une cohérence des modèles est garantie à travers les contraintes imposées dans la conceptualisation des patrons (qui sont forcément respectées lors des opérations d'interconnexion ou d'adaptation/évolution) et la phase (semi-automatique) d'assignation des variables qui vise à maintenir un flux de données cohérent dans le processus global. Mais, nous jugeons qu'il reste indispensable de vérifier la correction des modèles de WFIO en utilisant un outil de vérification formel et en définissant des propriétés à vérifier. Il faut noter que des travaux tels que (Schmidt et al., 2004), (Hinz et al, 2005), (Lohmann et al., 2009) se sont concentrés sur la transformation des modèles de processus BPEL vers les RdP en vue de vérifier formellement, des propriétés sur une composition de services orchestrée via un processus BPEL.

Pour notre part, nous travaillons actuellement sur l'aspect correction des modèles de WFIO, en utilisant le formalisme des réseaux de Petri (RdP). Nous nous focalisons sur les aspects comportemental et interactionnel des processus ; les principales propriétés à vérifier se résument dans une propriété dite de « soundness » initialement introduite par Van der Aalst (Van der Aalst, 97), (Van der Aalst, 2008) et qui consiste à vérifier que :

- A partir de l'état initial du réseau, l'état final est forcément atteint
- Absence de blocage (deadlocks)
- Absence de transitions mortes.

Dans notre cas, une transition du réseau modélise l'invocation d'un service et une place modélise le lien entre deux transitions (donc l'enchainement des invocations de services). Notre démarche pour la question de correction des modèles est la suivante : (1) Proposer d'abord une démarche de translation d'un processus BPEL en un modèle de RdP (en faisant une projection sur les aspects comportemental et interactionnel uniquement). (2) pour chacun des patrons de coopération considérés dans cette thèse, nous posons des règles d'interconnexion et des règles de vérification. Pour l'interconnexion de deux modèles de WF, nous nous basons sur la modélisation d'une communication synchrone ou asynchrone entre deux modèles de RdP. En effet, pour la communication asynchrone (le cas des patrons « Exécution chainée », « Transfert de cas » et « Faiblement couplé »), il s'agit d'utiliser une place partagée entre les deux transitions modélisant l'envoi et la réception de messages. Dans le cas d'une communication synchrone (le cas du patron

« Sous-traitance »), il s'agit de substituer la transition modélisant l'invocation du service abstrait par le service encapsulant le WF secondaire.

Pour les règles de vérification, nous examinons dans quel cas est-il possible d'effectuer une analyse modulaire des modèles de WF basée sur l'analyse modulaire des RdP (Christensen et al., 2000), pour vérifier les propriétés sur les fragments de WF séparément et translater ensuite le résultat de l'analyse sur le modèle global (les fragments interconnectés).

Pour les modèles obéissant aux patrons de coopération « Exécution chainée » et « Sous-traitance » par exemple; il sera possible de vérifier chaque fragment séparément et de translater le résultat de la vérification sur le modèle de WFIO global, si on arrive à démontrer que la substitution d'un service par un autre service (encapsulant un processus) préserve la propriété de « soundness ». Concernant le « Partage de charge », la vérification est plus simple puisqu'il s'agit d'un seul modèle de WF représentant le WFIO implémenté sur un seul site et contrôlé de manière centralisée. Quant au patron « Faiblement couplé », la vérification s'avère plus complexe puisqu'il s'agit de vérifier les propriétés sur chaque modèle de processus local à chaque partenaire et vérifier aussi le modèle global afin de décider si le protocole de communication (asynchrone) est consistant ou non. Pour le patron « Transfert de cas », il s'agit de vérifier la propriété sur un même modèle de WF qui est dupliqué au niveau des sites des différents partenaires et aussi sur les différents modèles de sous-processus créés, suite à l'ajout des points de transfert.

- **Perspective 4 :** Evaluation des adaptations/évolution

Une quatrième perspective concerne l'aspect évaluation de notre approche d'adaptation/ évolution sur les modèles de WFIO. En effet, dans le chapitre VI, nous avons émis quelques critères pour la plupart qualitatifs, permettant d'apprécier notre approche d'adaptation/évolution qui sont :

- Le coût de l'adaptation/évolution des modèles
- Le nombre de modèles de WF impliqués dans l'opération d'adaptation/évolution
- Impact des opérations de changement sur la correction des modèles
- Nombre et nature des opérations de changement pour chaque cas d'adaptation/évolution
- Complexité du modèle de WFIO, notamment suite à une opération d'évolution
- La possibilité de défaire une adaptation/évolution
- La profondeur de l'adaptation
- Aspects du processus affectés par l'adaptation/évolution

Nous envisageons d'approfondir ce point d'abord par le développement d'un outil permettant de mesurer les paramètres qui peuvent l'être.
L'ajout de nouveaux critères qui ne peuvent avoir de sens qu'avec une approche d'adaptation/évolution dynamique constitue une autre perspective à ce travail. Dans le cas d'une adaptation dynamique, des critères pourraient être définis et incorporées à l'outil d'adaptation/évolution. Nous citons par exemple : la fréquence des adaptations sur un modèle, la fréquence des adaptations par type d'opération sur un modèle spécifique ou sur l'ensemble des modèles, la fréquence d'annulation d'une adaptation, le nombre d'instances affectées par une adaptation, …etc.

# Références Bibliographiques


**(Afflerbach et al., 2014)**: Afflerbach P., Kastner G., Knause F., Roglinger M. (2014), The Business Value of Process Flexibility, Journal of Business and Information Systems Engineering, Springer, Vol. 6, n°4, pp. 203-214.

**(Ahmed et al., 2014)** : Ahmed T., Tripathi A., Srivastava A., (2014), Rain4Service: An approach towards decentralized web service composition. Proceedings of the IEEE International Conference on Services Computing (SCC'14), pp. 267-274. IEEE.

**(Ait-Cheikh-Bihi, 2012)**: Ait-Cheikh-Bihi W., (2012), Approche orientée modèles pour la vérification et l'évaluation des performances de l'interopérabilité et des interactions de services. Thèse de doctorat, Université de Belford-Montbeliard, France.

**(Ait-Cheikh-Bihi et al., 2012)** : Ait-Cheik-Bihi W., Nait-Sidi-Moh M., Bakhouya J., Gaber M., (2012), Performance Study of Workflow Patterns-Based Web Service Composition. Procedia Computer Science, Vol. 10, pp 728-735.

**(Aksit et al., 2003):** Aksit M., Choukair Z., (2003), Dynamic, Adaptive and Reconfigurable Systems Overview and Prospective Vision. Proceedings ICDCSW'03, Providence, Rhode Island, USA, pp. 84-92.

**(Alexander et al., 77)**: Alexander C., Ishikawa S., Silverstein M., (1997), A Pattern Language, Oxford University Press, New York.

**(Alonzo et al., 2004)**: Alonso G., Casati F., Kuno H., (2004), Web services: concepts, architectures and applications. Springer Verlag, Germany.

**(Alur et al., 94):** Alur R., David L. D., (1994), A theory of timed automata. Theoretical Computer Science, vol. 126, pp. 183–235.

**(Amsden et al., 2004)**: Amsden J., Gardner T., Griffin C., Iyengar S.,. UML 1.4 profile for automated business processes with a mapping to BPEL. Technical report, IBM, July 2004.

**(Anzböck et al., 2004)**: Anzböck R., Schahram D., Gholami M., (2004), Modeling ad-hoc medical imaging workflows with BPEL4WS. Technical report, IBM, July 2004.

**(Ardissono, 2007)**: Ardissono L., et al. (2007), Context-aware workflow management. Web Engineering. Springer Berlin Heidelberg. pp. 47-52.

**(Arenaza, 2006)** : Arenaza, N. (2006). Composition semi-automatique de Services Web. Rapport de Projet de Master, Ecole Polytechnique Fédérale de Lausanne.

**(Babin & al., 2003)**: Babin G., Leblanc M., (2003), Les Web services et leur impact sur le commerce B2B. CIRANO Burgundy, Reports 2003 rb-07.

**(Baghdadi, 2012)**: Baghdadi, Y., (2012), A methodology for Web services-based SOA realization', *Int. Journal of Business Information Systems*, Vol. 10, No. 3, pp.264–297, Inderscience Editors.

**(Baouab, 2013)** : Baouab K., (2013), Gouvernance et supervision décentralisée des chorégraphies inter-organisationnelles.Thèse de doctorat, Université de Lorraine, France.

**(Barba et al., 2013)**: Barba I., del Valle C., Weber B., Jiménez A., (2013), Automatic generation of optimized business process models from constraint-based specifications. Int. Journal of Cooperative Information Systems, Vol. 22, No. 2, ScientificWorld.

**(Barker et al., 2008)**: Barker A., Van Hemert J., (2008), Scientific Workflow: A Survey and Research Directions. Proceeding of the seventh Int. Conf. of Parallel Processing and Applied Mathematics., Wyrzykowski R. et al., eds., pp. 746-753.

**(Barros et al., 2005)**: Barros A., Dumas M., ter Hofstede A., (2005), Service Interaction Patterns. Proceedings of Business Process Management Conference, BPM'05, pp.302- 318.

**(Bartoli, 1996)**: BARTOLI J.A., (1996). L'entreprise virtuelle peut-elle être apprenante ? Thèse de recherche sur l'organisation apprenante, Université d'Aix en Provence.

**(Bastida et al., 2008)**: Bastida L., Nieto F. J., Tola, R. (2008), Context-aware service composition: a methodology and a case study. Proceedings of the 2nd international workshop on Systems development in SOA environments, SDSOA'08, pp. 19–24, New York, USA.

**(Bastos et al.,, 2002**): Bastos R. M., Ruiz D. D. A. (2002), Extending UML activity diagram for workflow modeling in production systems. Proceedings of the 35th Annual Hawaii International

Conference on System Sciences. HICSS'02, pp. 3786-3795). IEEE.

**(Baudry et al., 2007)**: Baudry, B., Ghosh, S., (2007), Providing Support for Model Composition in Metamodels. In D. H. Akehurst, R. Vogel, and R. F. Paige (eds.), LNCS, vol. 4530, pp. 32-42, Springer, Heidelberg.

**(Beauche et al., 2008)**: Beauche S., Poizat P., (2008), Automated Service Composition with Adaptive Planning. Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC '08, Sydney- Australie.

**(Belhajjame et al., 2005**): Belhajjame K., Vargas-Solar G, Collet C. (2005), Pyros - An environment for building and orchestrating open services. Proceedings of the IEEE International Conference on Services Computing, ICSC'03, pp.155–164, USA.

**(Benali, 2005)** : Benali, M., (2005), Une modélisation des liens de coopération et des trajectoires d'évolution des réseaux d'entreprises. Thèse de doctorat. Ecole nationale supérieure des mines de Saint-Etienne.

**(Benatallah et al., 2003)**: Benatallah B., Hacid M.-S., Rey C., Toumani, F., (2003), Semantic Reasoning for Web Services Discovery. Proceedings of the 2nd International Semantic Web Conference ISWC'03, pp. 242-257.

**(Bendraou et al., 2005)** : Bendraou R., Gervais M. P., Blanc X., (2005), Uml4spm : a UML 2.0-based metamodel for software process modelling. Proceedings of the 8th international conference on Model Driven Engineering Languages andSystems, MoDELS'05, pp.17–38, Berlin, Heidelberg, Springer-Verlag.

**(Benhassine, 2006):** Benhassine A., Shigeo M., Ishida T. (2006), Constraint-based Approach for Web Service Composition. International Semantic Web Conference , ISWC'06), LNCS 4273, pp. 130-143, Springer-Verlag.

**(Bernard et al., 2003):** Bernard C., Legalles X., Maesano L., (2003), Services Web et J2EE et .Net : Conception et implémentation, 1087p, Informatique, Eyrolles.

**(Bernauer et al., 2003):** Bernauer M., Kappel G., Kramler G., Retschitzegger W., (2003), Specification of Interorganizational Workflows — A Comparison of Approaches. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics, pp. 30-36. Orlando, Florida.

**(Bernstein, 2003)**: Bernstein P.A., (2003), Applying Model Management to Classical Meta Data Problems. Proceedings of the Conference on Innovative Database Research (CIDR'03), pp. 209-220

**(Berre et al., 2007)**: Berre A.-J., Elveaster B., Figay N., Guglielmina C., Johnsen S., Karlsen D., (2007), The ATHENA Interoperability Framework, Enterprise interoperability: New challenges and approaches II, Knothe T., Lippe, S. (Eds.), Springer.

**(Bézivin et al., 2006)**: Bézivin J., Bouzitouna S., Didonet M., Fabro D., Gervais M. P., Jouault F., Kolovos D., Kurtev I., Paige R.C., (2006), A canonical scheme for model composition. Proceedings of the European Conference in Model Driven Architecture (EC-MDA'06), Bilbao, Spain.

**(Bigne-Alcaniz et al., 2009)**: Bigne-Alcaniz E., Aldas-Manzano J., Andreu-Simo L., Ruiz-Mafe C., (2009), Business-to-Business e-commerce adoption and perceived benefits: evidence from small and medium Spanish enterprises. International Journal of Electronic Business, Vol 7, n°6, pp. 599 –624, Inderscience.

**(Blay-Fornarino, 2009)**: Blay-Fornarino M., (2009), Interprétations de la Composition d'activités. Thèse de doctorat, Université de Nice - Sophia Antipolis.

**(Bolcer et al. 96)** : Bolcer G.A., Taylor R.N., (1996), Endeavors: A Process System Integration Infrastructure. Proceedings of the International Conference on Software Process, ICSP'04. pp. 2-6, Brighton, U.K.

**(Bolognesi, 87)** : Bolognesi T., Brinksma E., (1987), Introduction to the ISO specification language LOTOS. Computer Networks. ISDN Systems, Vol.14, pp. 25–59.

**(Boukadi, 2009)** : Boukadi K., (2009), La coopération interentreprises à la demande: une approche flexible à base de services adaptables. Thèse de doctorat, ENSM, Saint-Etienne. France.

**(Boukhebouze, 2010)** : Boukhebouze M., (2010), Gestion de changement et vérification formelle de processus métiers : une approche orientée règles. Thèse de doctorat. Institut National des Sciences Appliquées de Lyon, France.

**(Boukhedouma et al., 2008)** : Boukhedouma S., Alimazighi Z., (2008), Modelling and Distributing Individual and Collective Activities in a Workflow System. Proceedings of the international conference of enterprise information systems, ICEIS'08 (3-2), pp. 290-297, Barcelona, Spain.

**(Boukhedouma et al., 2010a)**: Boukhedouma S, Alimazighi Z., (2010) Un Méta-modèle de processus métiers inter-organisationnels : une approche basée workflow et SOA. Actes de l'atelier SIRE

(SI des organisations étendues) associé au congrès INFORSID'2010. Marseille, France.

**(Boukhedouma et al., 2010b)**: Boukhedouma S., Alimazighi Z., (2010), A process meta-model based method for the development of collaborative applications built on workflow and SOA. In proceedings of the European and Middle Eastern Conference on Information Systems (EMCIS'2010), Dubai, UAE.

**(Boukhedouma et al., 2011a)** : Boukhedouma S., Alimazighi Z., Oussalah M., Tamzalit D., (2011), Une approche basée SOA pour l'interconnexion de Workflows: Application au transfert de cas. Actes du congrès INFOSID'2011, pp. 43-56, Lille, France.

**(Boukhedouma et al., 2011b)**: Boukhedouma S, Alimazighi Z, Oussalah M, Tamzalit D. (2011), SOA-Based Approach for Inteconnecting Workflows According to the "Subcontracting" Architecture. Proceedings of IADIS – Collaborative Technologies, Rome, Italy.

**(Boukhedouma et al., 2012a)**: Boukhedouma S., Alimazighi Z., Oussalah M., Tamzalit D., (2012), Interconnecting workflows using services: an approach for case transfer with centralized control. Proceedings of ICISTM'2012, S. Dua et al. (Eds.): CCIS 285, pp.396–401, Springer-Verlag Berlin Heidelberg.

**(Boukhedouma et al., 2012b)**: Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D.(2012), A Service- based Approach for Adaptability of Workflow Models - The Subcontracting Architecture. Proceedings of ICEIS'2012 (3), pp. 224-231.

**(Boukhedouma et al., 2012c)**: Boukhedouma S., Alimazighi Z., Oussalah M., Tamzalit D., (2012), Patrons de coopération à base de services pour l'adaptabilité des modèles de workflow. Actes du congrès INFORSID'2012. Montpellier, France.

**(Boukhedouma et al., 2012d)**: Boukhedouma S, Alimazighi Z, Oussalah M, Tamzalit D., (2012), Adaptability of Service Based Workflow Models: the "Chained Execution" Architecture. proceedings of the int. Conf. of Business Information Systems (BIS'2012,. pp. 96-107.

**(Boukhedouma et al., 2013a)**: Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D., (2013), Adaptation patterns for Service-Based Inter-Organizational Workflows. Proceedings of RCIS'2013. Paris, France.

**(Boukhedouma et al., 2013b)**: Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D. (2013), Flexible loosely coupled workflows using SOA. Proceedings of AICCSA'2013, Fes, Maroc.

**(Boukhedouma et al., 2014a)** : Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D. (2014), Service-Based Cooperation Patterns to Support Flexible Inter-organizational Workflows. IJITCS 4(1), pp. 1-18, 2014. Published Online March 2014 in MECS (http://www.mecs-press.org/) DOI: 10.5815/ijitcs.2014.04.01

**(Boukhedouma et al., 2014b)** : Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D., (2014), Cooperation Patterns and Adaptation Patterns for Service-Based Inter-organizational Workflows. Book chapter, in "Uncovering Essential Software Artifacts through Business Process Archaeology". Eds Ricardo Pérez-Castillo, Chapter 10. pp. 250-283. Copyright, 2014. http://www.igi-global.com/chapter/

**(Boukhedouma et al., 2015)**: Boukhedouma S., Oussalah M., Alimazighi Z., Tamzalit D., (2015), A Pattern-Based Approach for Workflow Interconnection and Flexibility Support. International Journal of Business Information Systems (IJBIS). Vol 19, n° 3: pp. 375-402 (2015), Inderscience.

**(Bourdon, 2007)** : Bourdon, J., (2007), Multi-agent systems for the automatic composition of semantic web services in dynamic environments. Rapport de master, École des Mines de Saint Etienne - G2I & Université Joseph Fourier.

**(Bouzitouna et al., 2004)**: Bouzitouna S., Gervais M. P., (2004), Composition rules for PIM Reuse, Proceedings of the 2nd European Workshop on MDA with Emphasis on Methodologies and Transformations (EWMDA-MT'04), Canterbury, UK.

**(Brown et al., 2002)**: Brown A., Johnston S., Kelly K., (2002), Using Service-Oriented Architecture and Component- Based Development to Build Web Service Applications. A Rational Software White Paper, Copyright © 2002, Rational Software Corporation

**(Budinsky et al., 2003)**: Budinsky F., Steingerg D., Merks E., Ellersick R., Grose T. J., (2003), Eclipse Modeling Framework : A Developer's Guide, Addison Wesley, 2003.

**(Bussler, 2002)**: Bussler C., (2002), A Taxonomy of Adaptive Workflow Management, Yanbo Han, Amit Sheth 1, Large Scale Distributed Information Systems, The University of Georgia, Athens, 2002.

**(Camarinha-Matos et al., 99)** : Camarinha-Matos L. M., Pantoja-Lima C., (1999), A framework for cooperation in virtual enterprises. Proceedings of the IFIP TC5 WG5.3/5.7 Third International

Working Conference on the Design of Information Infrastructure Systems for Manufacturing , *DIISM '98*, pp. 305–322, Deventer, The Netherlands.

**(Canfora et al., 2008)** : Canfora G., Di Penta M., Esposito R., Villani M. L. (2008). A framework for QoS-aware binding and re-binding of composite web services. Journal of Systems and Software, Vol.81, pp. 1754–1769.

**(Capra et al., 2001)**: Capra L., Emmerich W., Mascolo C., (2001), Reflective Middleware Solutions for Context-Aware Applications. Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, pp. 126-133, Springer-Verlag London, UK.

**(Carman et al., 2003)** : Carman M., Serafini L., Traverso P., (2003), Web service composition as planning. Proceedings of the Workshop on Planning for Web Services, ICAPS, Trento, Italy.

**(Casati, 98)**: Casati F., (1998), Approaches to Handling Exceptions in Workflow. In proceedings of the Workshop Towards Adaptive Workflow Systems, CSCW'98, Seattle WA..

**(Casati et al., 96)**: Casati F., Ceri S., Pernici B., Pozzi G. (1996), Deriving Active Rules for Workflow Enactment. Proceedings of DEXA Database and Expert Systems Applications. Zurich, Suisse, pp. 9-13.

**(Casati et al., 2001)**: Casati F., and Shan M.C., (2001), Dynamic and adaptive composition of e-services, Journal of Information Systems, Vol. 26, n°3, pp. 143-163. Elsevier.

**(Chappelet et al., 97)** : Chappelet J. L., Snella J. J., (1997), Un langage pour l'organisation : l'approche OSSAD. 3ème édition « Presses Polytechniques et Universitaires Romandes », Lausanne, 2004.

**(Charfi et al., 2009)**: Charfi A., Dinkelaker T., Mezini, M., (2009), A Plug-in Architecture for Self-AdaptiveWeb Service Compositions. Proceedings of the IEEE International Conference on Web Services, ICWS '09: pp. 35–42. IEEE Computer Society.

**(Charif, 2007)**: Charif Y. (2007), Chorégraphie dynamique de services basée sur la coordination d'agents introspectifs. Thèse de doctorat, Université Pierre et Marie Curie Paris VI. France, 2007.

**(Charif et al., 2006)** : Charif Y., Sabouret, N. (2006), An overview of semantic web services composition approaches. Electronic Notes in Theoretical Computer Science, Vol. 146, n°1, pp. 33-41.

**(Chebbi, 2007) :** Chebbi I., (2007), CoopFlow : Une approche pour la cooperation ascendante dans les enterprises virtuelles. Thèse de doctorat. Institut National des Telecoms, France.

**(Chelli, 2003)** : Chelli H. (2003), Urbaniser l'entreprise et son système d'information, Editions Vuibert.

**(Chen et al., 2009a)** : Chen R.S., Sun C. M., Helms M. M. et al., (2009), Factors Influencing Information Systems Flexibility: An InterpretiveF lexibility Perspective. International Journal of Enterprise Information systems, Vol. 5, n°1, IGI Global.

**(Chen et al., 2009b)** : Chen K., Xu, J., Reiff-Marganiec S., (2009), Markov-htn planning approach to enhance flexibility of automatic web service composition. Proceedings of the IEEE International Conference on Web Services, ICWS'09, pp. 9-16.

**(Cheng et al., 2009)**: Cheng B. H. C., de Lemos R., Giese H., (2009), (2009). Software Engineering for Self-Adaptive Systems, volume 5525 of LNCS. Inverardi P. and Magee, J., (eds), Springer.

**(Christensen et al., 2000)**: Christensen S., Petrucci L. (2000), Modular Analysis of Petri Nets. The Computer Journal, Vol.43, n°3. Pp. 224-243. www.comjnl.oxfordjournals.org

**(Clarke, 2002)**: Clarke S., (2002), Extending Standard UML with Model Composition Semantics, Science of Computer Programming. Vol. 44, n°1, pp. 71-100. Elsevier.

**(Colombo et al., 2006)**: Colombo M., Di Nitto E., Mauri M. (2006), SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. Proceedings of the international conference of Service-Oriented Computing – ICSOC'06, Vol. 4294 of LNCS, pp.191–202, Springer, Berlin-Heidelberg.

**(Courbis and Finkelstein, 2005)**: Courbis C., Finkelstein, A., (2005). Towards aspect weaving applications. In Proceedings of the 27th international conference on Software engineering, ICSE '05, pp. 69–77, New York, USA.

**(Crusson, 2003)**: Crusson T. (2003), Business Process Management: de la modélisation à l'exécution positionnement par rapport aux architectures orientées services, Copyright 2003, Intalio.com, inc. www.intalio.com

**(Cruz Torres et al., 2010)**: Cruz Torres M. H., Noël V., Holvoet T., Arcangeli, J.P. (2010), MAS organisations to adapt your composite service. In Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond, MONA'10, pp. 33–39, New York, USA.

**(Curbera & al, 2001):** Curbera F., Nagy W. A., Weerawarana S., (2001), Web services: Why and how? Proceedings of OOPSLA'01 Workshop on Object-Oriented Web Services. ACM.

**(Damak et al., 2011)** : Damak Mallouli S., Assar S., Souveyet C., Pour une perspective comportementale dans les méta-modèles de processus. Actes du congrès INFORSID'2011, Lille, France.

**(Davidson et al., 2008)**: Davidson S. B., Freire J., (2008), Provenance of scientific workflows: challenges and opportunities. Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM.

**(Decker et al., 2007):** Decker G., Kopp O., Leymann F., Weske M., (2007), BPEL4Chor: Extending BPEL for Modeling Choreographies. Proceedings of the IEEE International Conference on Web Services (ICWS'07), Utah, USA.

**(Decker et al., 2009)**: Decker G., Kopp O., Leymann F., Weske M. (2009), Interacting services: from specification to execution. Data & Knowledge Engineering, Vol. 68, n°10, pp. 946–972.

**(Didonet et al., 2006)**: Didonet M., Del Fabro J., Bézivin, Valduriez P., (2006), Weaving Models with the Eclipse AMW plug-in. Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany.

**(Digiampietri et al., 2007)** : Digiampietri L. A., Pérez-Alcázar J. J., Medeiros C. B., (2007), AI-Planning in Web Services Composition: a review of current approaches and a new solution. SBC, 2007, 983-992.

**(Dohring et al., 2010)**: Döhring M., Zimmermann B., Godehardt E., (2010), Extended workflow flexibility using rule-based adapatation patterns with eventing semantics. Proceedings of INFORMATIK'10, pp. 216-226.

**(Dohring et al., 2011)**: Döhring M., Zimmermann B., Karg L., (2011), Flexible Workflows at design- and Runtime using BPMN2 Adaptation Patterns. In proceedings of the international conference of business information systems, BIS'2011- Springer.

**(Doshi et al., 2005)** : Doshi P., Goodwin R, Akkiraju R., Verma K., (2005), Dynamic workflow composition using markov decision processes. International Journal of Web Services Research,

Vol. 2, n°1, pp. 1-17.

**(Dumez et al., 2008)**: Dumez C., Gaber D., Wack M., (2008), Model-driven engineering of composite web services using UML-S. Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS '08, pp. 395–398, New York, NY, USA, ACM.

**(Eder et al., 96)** : Eder J., Groiss H., Liebhart W., (1996), Workflow Management and Databases. 2eme Forum International d'Informatique Appliquée, Tunis.

**(Eder et al., 98)** : Eder J., Liebhart W., (1998), Contributions to Exception Handling in Workflow Management. Proceedings of the EDBT Workshop on Workflow Management Systems, pp. 3-10, Valencia.

**(Eder et al., 2002)**: Eder J., Gruber W., (2002), A meta model for structured workflows supporting workflow transformations. Proceedings of the 6th East European Conference on Advances in Databases and Information Systems (ADBIS'02), pp. 326–339, Bratislava, Slovakia.

**(Edmond, 2000)** Edmond D., (2000), Applications of Reflection for Cooperative Information Systems. Mémoire de Thèse pour l'obtention du titre de "Doctor of Philisophy". Faculty of Information Technology, Queensland University of Technoloy, Australia.

**(Elfidoussis et al., 2014)**: Elfirdoussi S., Jarir Z., Quafafou M., (2014), Web service composition based on popularity. Proceedings of ICCSEA'14. Natarajan Meghanathan et al. (Eds) : pp. 251–263.

DOI : 10.5121/csit.2014.4729

**(Ellis et al., 98)**: Ellis C.A., Keddara K., Wainer W., (1998), Modeling workflow dynamic changes using timed hybrid workflow nets. Proceedings of the ICATPN'98, pp.109 –128.

**(Esper, 2010)**: Esper A., (2010), Integration des approches SOA et orientée objet pour modéliser une orchestration cohérente de services. Thèse de doctorat, INSA, Lyon, France.

**(Ferguson et al., 2005)**: Ferguson D. F., Stockton M. L., (2005), Service-oriented architecture: Programming model and product architecture. *IBM Systems Journal*, Vol. 44, n°4, pp. 753–780.

**(Ferrara, 2004)** : Ferrara A., (2004). Web services: a process algebra approach. Proceedings of the 2nd international conference on Service oriented computing, pp. 242-251, ACM.

**(Fleurey et al., 2007)**: Fleurey F., Baudry B., France R., Ghosh S., (2007), A Generic Approach For Automatic Model Composition.Workshop pf Aspect Oriented Modeling (AOM).

**(Font-Conte et al., 99) :** Front-Conte A., Giraudin J.P., Rieu D., Saint-Marcel C., (1999), Réutilisation et patrons d'ingénierie, Chapitre de l'ouvrage *Génie Objet : Analyse et Conception de l'Evolution*,

Editeur M. Oussalah, Hermès, 1999.

**(Foster et al., 2003)**: Foster H., Uchitel S., Magee J. Kramer J., (2003), Model-based Verification of Web Service Compositions. Proceedings of the18th IEEE International Conference on Automated Software Engineering (ASE'03), pp. 152-161, Montreal, Canada,

**(Fournier-Morel et al., 2006)** : Fournier-Morel X., Grojean P., Plouin G., Rognon C., (2006), SOA : Le guide de l'architecte. Editions Dunod.

**(Fryer, 94):** C. Fryer. "Move to workflow Provokes Business Process Scrutiny". Software Magazine. Avril 1994.

**(Gamma et al., 95)**: Gamma E., Helm R., Johnson R., Vlissides J., (1995), Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.

**(Geambasu et al., 2012)**: Geambasu C.V., (2012), BPMN vs. UML Activity Diagram for Business Process Modeling. Journal of Accounting and Management Information Systems , Vol.11, n° 4, pp. 637-651

**(Geebelen et al., 2010)** : Geebelen K., Kulikowski E., Truyen E., and Joosen W. (2010). A MVC Framework for Policy-Based Adaptation of Workflow Processes: A Case Study on Confidentiality. Proceedings of the IEEE International Conference on Web Services, ICWS '10, pp. 401–408.

**(Giachetti et al., 2003)**: Giachetti R. E., Martinez L. D., Saenz O. A. Chen, C. S., (2003), Analysis of the structural measures of flexibility and agility using a measurement theoretical framework, International Journal of Production Economics. Vol. 86, n°1, pp. 47-62. ScienceDirect.

**(Glassey et al., 2002)** : Glassey O., Chappelet J. L., (2000), Comparaison de trois techniques de modélisation de processus : ADONIS, OSSAD et UML. IDHEAP Working paper, Lausanne, 2002.

**(Gorton & al, 2009)**: Gorton S., Montangero C., Reiff-Marganiec S., Semini L., (2009), StPowla: SOA, Policies and Workflows. Proceedings of ICSOC'07 workshops, LNCS 4907, pp. 351-362.

**(Governatori et al., 2004)**: Governatori G., Rotolo A., Sadiq S., (2004), A model of dynamic resource allocation in workflow systems. Proceedings of the 15th Australasian database conference –Vol. 27 pp. 197-206. Australian Computer Society Inc.

**(Grefen et al., 2001)** : Grefen P., Aberer K., Hoffer Y., Ludwig H., (2001), CrossFlow : Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. IEEE Data Engineering Bulletin. Vol 24, n°1, pp. 52–57.

**(Grefen et al, 2009)**: Grefen P., Mehandjiev N., Kouvas G., Weichhart G., Eshuis R., (2009), Dynamic business network process management in instant virtual enterprises. Journal of Computers in Industry, Vol. 60, n°2, pp. 86–103

**(Grigori, 2001)** : Grigori D. (2001), Eléments de flexibilité des systèmes de workflow pour la définition et l'exécution des procédés coopératifs. Thèse de doctorat, Université d'Henri Poincaré, Nancy 1.

**(Hamadi et al., 2003):** Hamadi R., Benatallah, B., (2003). A Petri net-based model for web service composition. Proceedings of the 14th Australasian database conference, Vol. 17, pp. 191-200. Australian Computer Society Inc.

**(Han et al., 98)**: Han Y., Sheth A., Bussler C., (1998), A Taxonomy of Adaptive Workflow Management. Proceedings o the Workshop Towards Adaptive Workflow Systems CSCW'98, Seattle, WA.

**(Hashemian et al., 2005)**: Hashemian S. V., Mavaddat F., (2005), A graph-based approach to web services composition. Proceedings of the 2005 Symposium on the Internet and Applications, pp. 183-189. IEEE.

**(Havey, 2005)**: Havey M. (2005), Essential Business Process Modeling. 352p, O'Reilly Media Inc.

**(He et al., 2008)**: He Q., Yan Y., Jin H., (2008), Adaptation of web service composition based on WF patterns. Proceedings of the international conerence of Service Oriented Computing- ICSOC'08. Sidney, Australia.

**(Heorhi, 2012)** : Heorhi R., (2012), Service Composition in Dynamic Environments: From Theory to Practice, Phd Thesis, University of Trento.

**(Heubès, 2009)** : Heubès C., (2009), SOA : Du composant au service : Le contrat standardisé, article Xebia, cat. Architecture, mars 2009, http://blog.xebia.fr/2009/03/04/soa-du-composant-au-service-le-contrat-standardise/

**(Hinz et al., 2005)**: Hinz S., Schmidt K., Stahl, C., (2005), Transforming BPEL to Petri Nets. In Wil M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, Proceedings of the Third International Conference on Business Process Management (BPM'05), Vol. 3649 of LNCS, Nancy, France, pp. 220-235, Springer-Verlag.

**(Hoare, 83)**: Hoare C. A. R., (1983), Communicating sequential processes. *Commun. ACM*, Vol. 26, pp.

100–106.

**(Hoenish et al., 2013)**: Hoenisch P., Schulte S., Dustdar S., (2013), Workflow scheduling and resource allocation for cloud-based execution of elastic processes. Proceedings of the 6th International Conference on Service-Oriented Computing and Applications (SOCA'13), pp. 1-8, IEEE.

**(Hoffner et al., 2000)** : Hoffner Y., Ludwig H., Gülcü C., Grefen P. W. P. J., (2000), An architecture for cross organizational business processes. In Second International Workshop on Advance Issuesof E-Commerce and Web-Based Information Systems, Milpitas.

**(Hoffner et al., 2001)**: Hoffner Y., Field S., Grefen P.W. P. J., Ludwig H., (2001), Contract-driven creation and operation of virtual enterprises. Journal of Computer Networks, Vol. 37, n°2, pp. 111–136.

**(Hormosillo, 2012)**: Hermosillo G., (2012), Towards Creating Context-Aware Dynamically-Adaptable Business Processes Using Complex Event Processing. Thèse de doctorat, Université - Lille 1 Sciences et Technologies.

**(Hruby, 98)**: HRUBY P., (1998), Specification of Workflow Management Systems with UML. Workshop on Object Oriented WMS, OOPSLA'98.

**(Hu et al., 2014)**: Hu G., Wu B., Chen J. (2014). Dynamic adaptation of business process based on context changes: a rule-oriented approach. Proceedings of Service-Oriented Computing – (ICSOC' 13) Workshops, pp. 492-504. Springer International Publishing.

**(Huang et al., 2011)** : Huang Z., Van der Aalst W. M. P., Lu X., and Duan H., (2011), Reinforcement learning based resource allocation in business process management. Journal of Data & Knowledge Engineering, Vol. 70, no. 1, pp. 127–145. Sciencedirect.

**(Izza, 2006)** : Izza S., (2006), Intégration des systèmes d'information industriels: Une approche flexible basée sur les services sémantiques. Thèse de doctorat, École Nationale Supérieure des Mines de Saint-Étienne et de l'Université Jean Monnet, Saint-Étienne, p. 375.

**(Jérome, 2003)** : Jérome D. (2003), Services Web : Concepts, Techniques et Outils. Paris, Vuilbert.

**(Jeusfled et al., 2009)**: Jeusfled M.A., Jarke M., (2009), Metamodeling for Method Engineering, The MIT Press, Mylopoulos J., (Eds).

**(Jin et al., 2013)**: Jin T., Zhang F., Sun Q., Bui H., Parashar M., Yu H. et al., (2013), Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM.

**(Joeris, 2000)**: Joeris. G., (2000), Decentralized and Flexible Workflow Enactment Based on Task Coordination Agents. Proceedings of the Workshop on Agent Oriented Information Systems associated to CAISE'00 conference, Stockhlom, Sweeden.

**(Joosten 94)** : Joosten S., (1994), Trigger modelling for workflow analysis. Proceedings of CON'94 : Workflow Management, pp. 236-247, R. Oldenbourg (eds), Vienna, München.

**(Joosten et al., 95)**: Joosten S., Brinkkemper S., (1995), Fundamental Concepts for Workflow Automation in Practice. Proceedings of the ICIS'95 conference, Amsterdam.

**(Jordan et al., 2006)** : Jordan D., Evdemon J. et al. (2006) , Web services business process execution language version 2.0. W3C report, August 2006.

**(Justin et al., 2002)**: Justin O. S., Edmond D., Ter Hofstede A., (2002), What's in a service ? journal of Distributed Parallel Databases, Vol. 12(2-3), pp. 117–133.

**(Kaabi, 2007) :** Kaabi R. S. (2007), Une approche méthodologique pour la modélisation intentionnelle des Services et leur opérationnalisation. Thèse de doctorat, Université Panthéon-Sorbonne-Paris I.

**(Kammer et al., 98)**: Kammer P.J., Bolcer G.A., (1998), Techniques for Supporting Dynamic and Adaptive Workflow. Technical Report UCI-ICS-99-03. University of California Irvine.

**(Kanzow, 2004)**: Kanzow S., (2004), Approche pour l'ordonnancement distribué de Workflows dans le contexte d'entreprises virtuelles : méthodologie basée multi-agents pour la planification et l'exécution de processus distribués », Thèse de doctorat, Université Paris XII, Val de Marne.

**(Kapuruge et al., 2013)** : Kapuruge M., Han J., Colman A. (2013). Supporting Adaptation Patterns in the Event-Driven Business Process Modeling Paradigm.Web Information Systems Engineering– (WISE'2013), pp. 299-308, Springer Berlin Heidelberg.

**(Kavantzas et al., 2005)**: Kavantzas N., Burdett D., Ritzinger G., Fletcher T., Lafon Y., Barreto C. Web services choreography description language version 1.0. W3C. http://www.w3.org/TR/ws-cdl-10/. Dernière visite le 1/09/2015.

**(Ketfi et al., 2002)**: Ketfi A., Belkhatir N., Cunin P.-Y., (2002), Adaptation dynamique, concepts et expérimentations. International Conference in Software & Systems Engineering and their

Applications, ICSSEA'02, Paris, France.

**(Khadra, 2010)**: Khadka R., (2010), Model-Driven Development of Service Compositions: Transformation from Service Choreography to Service Orchestrations. Master thesis, University of Netherlands.

**(Khalaf et al., 2003)**: Khalaf R., Mukhi N., Weerawarana, S., (2003), Service-oriented composition in BPEL4WS. Proceedings of the 12th International World Wide Web Conference

**(Kiepuszewski & al., 2000)**: Kiepuszewski B., ter Hofstede A.H.M., Bussler C., (2000), On structured workflow modeling. Proceedings of the CAiSE '2000, LNCS, Vol.1789, pp.431 – 445, Springer, Berlin.

**(Kim et al., 2007) :** Kim D., Kim M., & Kim H. (2007). Dynamic business process management based on process change patterns. Proceedings of the International Conference on Convergence Information Technology. pp. 1154-1161. IEEE.

**(Kolar et al., 2013)** : Kolar J., Dockal L., Pitner T. (2013), A Dynamic Approach to Process Design: A Pattern for Extending the Flexibility of Process Models. In The Practice of Enterprise Modeling, pp. 176-190. Springer Berlin Heidelberg.

**(Kolovos et al., 2006)**: Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack, Merging Models with the Epsilon Merging Language (EML), Proceedings ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (Models/ UML 2006) (Genova, Italy), vol. LNCS, October 2006.

**(Komenda et al., 2009) :** Komenda J., Lahaye S., Boimond J.-L., (2009), Le produit synchrone des automates (max,+). Numéro spécial - Journal Européen des Systèmes Automatisés (JESA), Vol. 43 (7-9): pp.1033–1047.

**(Kon et al., 2002):** Kon F., Costa F., Blair G.S., Campbell R., (2002), The Case for Reflective Middleware: Building middleware that is flexible, reconfigurable, and yet simple to use, ACM Comminications, Vol. 45, no 6.

**(Koning et al., 2009)**: Koning M., Sun C.-a., Sinnema M., Avgeriou P., (2009). VxBPEL: Supporting variability for Web services in BPEL. Information and Software Technology, Vol. 52, n°2, pp. 258–269.

**(Krafzig et al., 2004)**: Krafzig D., Banke K., Slama D., (2004), Enterprise SOA: Service-Oriented Architecture Best Practices, Prentice Hall PTR, p.408, ISBN: 0-13-146575-9.

**(Kuzu et al., 2012)** : Kuzu M., Cicekli, N. K., (2012). Dynamic planning approach to automated web service composition. Journal of Applied Intelligence, Vol. 36, n°1, pp.1-28.

**(Lam, 2015)** : Lam, V. (2015). Constraint-based reasoning on declarative process execution with the logics workbench. Business Process Management Journal, Vol. 21, n°3. pp.586 – 609.

**(Laukkanen et al., 2003)**: Laukkanen M., Helin, H., (2003), Composing workflows of semantic web services. Proceedings of the Workshop on Web-Services and Agent-based Engineering (2003).

**(Lazcano et al, 2000)** : Lazcano A., Alonso G., Schuldt H., Schuler C.(2000), The WISE approach to electronic commerce. International Journal of Computer Systems Science & Engineering, special issue on Flexible Workflow Technology Driving the Networked Economy, Vol. 15, n°5.

**(Lazcano et al, 2001)**: Lazcano A., Schuldt G., Alonso G., Schek H.-J., (2001), WISE : Process based e-commerce. IEEE Data Engineering Bulletin, Vol. 24, n°1, pp. 46–51.

**(Leake et al., 2008)**: Leake D.B., Kendall-Morwick J., (2008), Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance. ECCBR'08, LNCS 5239, pp. 269-283, Springer.

**(Lécué et al., 2008)** : Lécué F., Silva E., Pires L. F., (2008), A framework for dynamic web services composition. Emerging Web Services Technology, Vol. II, pp. 59-75. Birkhäuser Basel.

**(Lee et al., 2007)**: Lee K., Sakellariou R., Paton N.W., Fernandes A., (2007), Workflow adaptation as an autonomic computing problem. Proceedings of the Second Workshop on Workflows in Support of Large-scale Science, Monterey, California, USA, ACM.

**(Lefebvre et al., 1999)** : Lefebvre L. A., Lefebvre É., (1999), Commerce électronique et entreprises virtuelles : défis et enjeux. Revue internationale de gestion, Vol.24, no.3, pp. 20-33.

**(Leitner et al., 2010)**: Leitner P., Michlmayr A., Rosenberg F., Dustdar S. (2010). Monitoring, Prediction and Prevention of SLA Violations in Composite Services. Proceedings of the 2010 IEEE International Conference on Web Services, ICWS '10, pp. 369–376,Washington, DC, USA, IEEE Computer Society.

**(Levan, 2000)** : Levan S.K., (2000), Le projet workflow- Concepts et Outils au Service des Organisations, 2eme tirage, Eyrolles, Avril 2000.

**(Leymann et al, 2002)**: Leymann F., Roller D., Schmidt M.-T., (2002), Web Services and Business Process Management. Journal of *IBM Systems, Journa*l, Vol. 41, No. 2.

**(Li et al., 2010)** : Li X., Fan Y., Madnick S., Sheng Q. Z. (2010), A pattern-based approach to protocol mediation for web services composition. Information and Software Technology, Vol.52, n°3, pp. 304-323.

**(Lins et al., 2007)**: Lins F. A. A., dos Santos Júnior J. C., Rosa, N. S. (2007). Adaptive web service composition. ACM SIGSOFT Software Engineering Notes, 32.

**(Liu et al., 98)** : Liu L., Pu C., (1998), Methodical restructuring of complex workflow activities. Proceedings of the ICDE 1998, IEEE Computer Society Press, Silver Springer, MD, pp.342 –350.

**(Liu et al., 2002):** Liu D., Wang J., Chan S. C., Sun J., Zhang L., (2002), Modeling workflow processes with colored Petri nets. Computers in Industry, Vol. *49*, n°3, pp. 267-281.

**(Lohmann et al., 2009)**: Lohmann N., Verbeek E., Dijkman R.M., (2009), Petri net transformations for business processes – a survey. Transactions on Petri Nets and Other Models of Concurrency 2 (2009).

**(Lopez-Velasco, 2009)** : Lopez-Velasco, C., (2009), Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation. Thèse de doctorat. UNIVERSITE JOSEPH FOURIER.

**(Lu et al., 2009)**: Lu R., Sadiq S., Governatori G., Xiaoping Y., (2009), Defining Adaptation Constraints for Business Process Variants. Procedings of the Int. Conf. of Business information systems (BIS '09), pp. 145-156. Springer.

**(Madhusudan et al., 2004):** Madhusudan T., Zhao L., Marshall B., A case-based reasoning framework for workflow model management. Data Knowledge Engineering, Vol. 50, no. 1, pp. 87–115, 2004.

**(Maesano et al., 2003)** : Maesano L. et al, (2003), Services web avec J2EE et .NET : Conception et implémentations, ISBN : 2-212-11067-7, Dunod, Paris.

**(Marca et al., 87)** : Marca D.A., McGowan C.L., (1987), SADT Structured Analysis and Design Technique, McGraw-Hill.

**(Marconi et al., 2009)**: Marconi A., Pistore M., Sirbu A., Eberle H., Leymann F., Unger T. (2009). Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation. In Service-Oriented Computing, volume 5900 of LNCS, pp. 445– 454. Springer Berlin / Heidelberg.

**(Markou et al., 2012)** : Markou G., Refanidis I. (2012). Towards an automatic non-deterministic web Service Composition platform. Proceedings of the 4th International Conference on Computational Aspects of Social Networks (CASoN'12), pp. 372-377, IEEE.

**(Martinez et al., 2004)** : Martinez E., Lesperance, Y., (2004). Web service composition as a planning task: Experiments using knowledge based planning. Workshop on Planning and Scheduling for Web and Grid services, ICAPS'04, Whistler, Canada.

**(Mayer et al. 95)** : Mayer J., Benjamin P.C., Caraway B.E., Painter M.K., (1995), A Framework and a Suite of Methods for Business Process Reengineering. in Grover, V., and Kettinger, W.J.(Eds.), Business Process Change: Concepts, Methods and Technologies, Idea Group Publishing, Harrisburg, PA, pp. 245-290.

**(Mayer, 2008):** Mayer P., Schroeder A., and Koch N., (2008), A Model-Driven Approach to Service Orchestration. Proceedings of the Int. Conference on Services Computing (SCC'08), Honolulu, USA.

**(Mayyad, 2009) :** Mayyad D. (2009), Architecture de système d'Information Distribué pour la Gestion de la Chaîne Logistique : Une Approche Orientée Services. Thèse de doctorat, INSA de Lyon.

**(Mbarki et al., 2010)** : Mbarki S. , Rahmouni M., Erramdani M., (2010), Transformation de modèles pour la génération de modèles Web MVC 2, Actes CARI'2010.

**(McCoy et al., 2006)**: McCoy D. W., Plummer, D. C., (2006), Defining, Cultivating and Measuring Enterprise Agility, Gartner Research.

**(McCready, 1992)**: McCready S., (1992), There is more than one kind of Workflow Software, Computerworld, Vol. 2.

**(Mehandjiev et al., 2005)**: Mehandjiev N., Stalker I., Fessl K., (2005), Weichhart G., Interoperability contributions of Crosswork. In invited short paper to Proceedings of INTEROP-ESA'05 Conference, Geneva, Springer-Verlag.

**(Meijler et al., 98)** : Meijler T.D., Kessels H., Vujist C., Le Comte R.. (1998), Realising Run-time Adaptable Wokflow by means of Reflectionin the Baan Workflow Engine. In proceedings of CSCW'98,Workshop Towards Adaptive Workflow Systems, Seattle, WA.

**(Meng et al., 2000)**: Meng J., Helal A., Su S., (2000), An Ad-Hoc Workflow Architecture Based on Mobile Agent and Rule-Based Processing. Proceedings of the International Conference on Artificial

Intelligence, Las Vegas, Nevada

**(Mesmoudi et al., 2011)** : Mesmoudi A., Mrissa M., Hacid, M. S. (2011), Combining configuration and query rewriting for Web service composition. Proceedings of the IEEE International Conference on Web Services (ICWS'11), pp. 113-120.

**(Milanovic et al., 2004)**: Milanovic, Malek., (2004), Current Solutions for Web Service Composition, IEEE Internet Computing, Vol. 8, n°6.

**(Milner, 89)**: Milner R. (1989), Communication and Concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

**(Minor et al., 2007)**: Minor M., Schmalen D., Koldehoff A., Bergmann R., (2007), Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. Proceedings of WETICE'07, pp. 370 – 375.

**(Minor et al., 2010)**: Minor M., Bergmann R., Georg S., Walter, K., (2010), Towards case-based adaptation of workflows. Proceedings of ICCBR'11, pp. 421– 435.

**(Minor et al., 2011)**: Minor M., Bergmann R., Georg S., Walter K., (2011), Reasoning on business processes to support change reuse. Proceedings of CEC'11, pp. 18–25.

**(Momotko et al., 2002)** : Momotko M., Subieta K., (2002), Dynamic Changes in Workflow Participant Assignment. ADBIS Research Communications, Vol. 2, pp. 175-184.

**(Monfort et al., 2003)** : V. Monfort, H Kadima. *Les Web Services.* Paris : Dunod, 2003.

**(Mosser et al., 2013)**: Mosser S., and Blay-Fornarino M., (2013), Adore: a logical meta-model supporting business process evolution. Journal of Science of Computer Programming, Vol. 8, n°1, pp.1035–1054, Elsevier.

**(Muller et al., 2004)**: Muller R., Greiner U., Rahm E., (2004), AGENTWORK: *A* workflow system supporting rule-based workflow adaptation. Data and knowledge engineering, pp: 223–256, Elsevier.

**(Muller, 2002):** Muller R., (2002), Event-oriented dynamic adaptation of workflows. Ph.D.Thesis, Department of Computer Science, University of Leipzig.

**(Muller, 98) :** P.A. Muller, (1998), Modélisation Object avec UML. Eyrolles.

**(Nejati et al., 2007) :** Nejati S., Sabetzadeh M., Chechik M., Easterbrook S., Zave P., (2007), Matching and Merging of Statecharts Specifications, ICSE '07 : Proceedings of the 29th international conference on Software Engineering (Washington, DC, USA), IEEE Computer Society, pp. 54-64.

**(Nguyen, 2008) :** NGUYEN T., (2008), Codèle: Une approche de composition de modèles pour la Construction de Systèmes à Grande Échelle. Thèse de doctorat. Université Joseph Fourier, Grenoble 1. France.

**(Nurcan, 96)** : Nurcan S., (1996), Analyse et conception de systèmes d'information coopératifs. Techniques et sciences informatiques, Vol. 15, n°9, pp. 1287 – 1315.

**(OASIS, 2007):** OASIS, Web Services Business Process Execution Language Version 2.0 – *OASIS Standard*, 2007.

**(OMG, 98):** OMG. Corba/iiop 2.3 specification, www.omg.org/docs/ptc/98-12-04.pdf. Technical report, http://www.omg/org, 1998.

**(OMG, 2005)** : Unified modeling language: Superstructure, Technical report., OMG, march, 2005.

**(OMG, 2011)**:Unified Modelling Language, V 2.4.1. August 2011, http://www.omg.org/spec/UML/2.4.1/

**(Omid et al., 2014)** : Omid M., Safi-Esfahani F., Nadimi-Shahraki M. H., (2014), A framework for context-aware web service composition using planning techniques. Multiagent and Grid Systems, Vol.10, n°4, pp. 185-197.

**(Osman et al., 2005)** : Osman T., Thakker D., Al-Dabass D., (2005), Bridging the gap between workflow and semantic-based web services composition. Proceedings of WWW Service composition with Semantic Web Services (Compiègne, France, Sept. 2005), pp. 13–23.

**(Oussalah et al., 2014)** : Oussalah M., et al., (2014), Architectures Logicielles, Editions Hermes Sciences Lavoisier ISBN 978-2-7462-4517-4, ISSN 2111-0360, 2014, Paris.

**(Ouvans et al., 2006)** : Ouvans C., Dumas M., Ter Hofstede A. H., Van der Aalst, W. M. (2006), From BPMN process models to BPEL web services. Proceedings of the int. conf. of Web Services (ICWS'06), pp. 285-292, IEEE.

**(Ouyang et al., 2007)** : Ouyang C., Verbeek E., Van der Aalst, W. M., Breutel S., Dumas M., Arthur H. M. ter Hofstede., (2007), Formal semantics and analysis of control flow in WS-BPEL. Science Computer Program, Vol. 67, pp. 162–198.

**(Pal et al., 2005)**: Pal N., Lim M., (2005), The Agile Enterprise, Springer.

**(Papazoglou, 2003)** : Papazoglou, M. P., (2003), Service oriented computing: concepts, characteristics

and directions. Proceedings of the 4th IEEE International Conference on Web Information Systems Engineering, Italy, pp. 3-12.

**(Papazoglou et al., 2007)**: Papazoglou, M. P, Van Den Heuvel W. J., (2007), Service Oriented Architectures: approaches, technologies and research issues. The VLDB Journal, Vol.16, pp 389-415.

**(Papazoglou, 2008)**: Papazoglou, M. P. (2008). Web Services: Principles and Technology. Pearson, Prentice Hall.

**(Patrick et al., 2005)** : Patrick A., Henocque L., Kleiner M., (2005), Composition de Workflows à l'aide de la Configuration. Premières Journées Francophones de Programmation par Contraintes.

**(Pedraza, 2009)**: Pedraza F. G., (2009), un canevas extensible pour la construction d'applications orientées procédé. Thèse de doctorat, Universite Joseph Fourier - GRENOBLE I.

**(Perrin et al, 2004)**: Perrin O. and Godart C., (2004), A model to support collaborative work in virtual enterprises. Data Knowledge Engineering, Vol.50, n°1, pp. 63–86.

**(Pesic et al., 2007)**: Pesic M., Schonenberg M. H., Sidorova N., Van der Aalst, W. M. P., (2007), Constraint based workflow models: Change made easy. Proceedings of the OTM Confederated International Conferences 2007, pp. 77-94.

**(Pessini, 2014)**: Pessini E. C., (2014), Expressiveness of Automatic Semantic Web Service Composition Approaches: A Survey based on Workflow Patterns. *Revista de Informática Teórica e Aplicada*, Vol. 21, n°1, pp. 45-76.

**(Pistore et al., 2005)**: Pistore M., Marconi A., Bertoli P., Traverso P., (2005), Automated Composition of Web Services by Planning at the Knowledge Level. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'05).

**(Pohl et al., 2005)**: Pohl K., Böckle G., Linden F. J., (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag.

**(Polyvyanny et al., 2009)**: Polyvyanny A. , Weske M. (2009) Hypergraph-based modeling of ad-hoc business processes. In Ardagna, Danilo, Mecella, Massimo, & Yang, Jian (Eds.) Lecture Notes in Business Information Processing : Business Process Management Workshops, Springer Berlin Heidelberg, Milano, Italy, pp. 278-289.

**(Qiqing et al., 2009)**: Qiqing F., Xiaoming P., Qinghua L., Yahui H.. A global QoS optimizing web services selection algorithm based on moaco for dynamic web service composition. Proceedings of the international Forum on Information Technology and Applications (IFITA). Vol 1, pp. 37 – 42. China.

**(Quirchmayr et al., 97)**: Quirchmayr, Tjoa A. M., (1997), A meta message approach for electronic data interchange. In DEXA '97 : Proceedings of the 8th International Conference on Database and Expert Systems Applications, pp. 377–386, London, UK, Springer-Verlag.

**(Rahman et al., 2008)**: Rahman S. S. u., Aoumeur N., Saake, G. (2008). An adaptive ECA-centric architecture for agile service-based business processes with compliant aspectual .NET environment. In iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, pp. 240–247. ACM.

**(Rainer et al., 2006)** : Rainer B., et al. (2006), Heuristics for QoS-aware web service composition. Proceedings of the Int. Conf. of Web Services, (ICWS'06). IEEE.

**(Ramadour et al., 2009)**: Ramadour P., Fakhri M., (2009), Modèle et langage de composition de services, LSIS (Laboratoire des Sciences de l'Information et des Systèmes), Université d'Aix-Marseille.

**(Rao et al., 2003)** : Rao, J., Kungas, P., and Matskin, M. Application of linear logic to web service composition. In Proceeding of the 1st International Conference on Web Services (Las Vegas, USA, June 2003).

**(Rao et al., 2005)** : Rao J., Su, X. (2005), A survey of automated web service composition methods. Proceedings of Semantic Web Services and Web Process Composition Conference, pp. 43-52. Springer Berlin Heidelberg.

**(Rational, 96)**: Rational Software Corporation. "UML Resource Center, Unified Modeling Language, Standard Software Notation V. 1.3". 1996.

**(Rational , 97)** : Rational Software Corporation. "UML Extension for Business Modeling version 1.1". 1997.

**(Raymond, 2011)** : Gilbert Raymond, SOA : Architecture Logique - Principes, structures et bonnes pratiques, Livre blanc, version 2.1, avril 2011.

**(Reddy et al., 2006)** : Y. Raghu Reddy, Sudipto Ghosh, Robert B. France, Greg Straw, James M. Bieman, N. McEachen, Eunjee Song, and Geri Georg, Directives for Composing Aspect- Oriented Design Class Models, Transactions on Aspect-Oriented Software Development I, LNCS 3880

(Springer-Verlag, 2006), 75-105.

**(Regev et al., 2006)** : Regev, G., Soffer P., Schmidt R.,(2006), Taxonomy of Flexibility in Business Processes. Proceedings of the 7th Workshop on Business Process Modeling, Development, and Support In conjunction with CAiSE'06

**(Reichert et al., 98)**: Reichert M., Hensinger C., Dadam P. (1998). Supporting adaptive workflows in advanced application environments. EDBT Workshop on Workflow Management Systems. Valencia.

**(Richardson et al., 2007)** : Richardson L., Ruby S., Services Web RESTful. O'Reilly Media. Oct. 2007.

**(Rolland, 2005)** : Rolland C., (2005), L'ingénierie des méthodes : une visite guidée , revue e-TI, n°1. http://www.revue-eti.net/document.php?id=726, dernière visite le 05/09/2015.

**(Rolland, 2007)**: Rolland C., (2007), Method Engineering : Trends and Challenges , Conférence invité à la WG8.1 Working Conference Situational Method Engineering, 12-14 sep. 2007, Genève, Suisse.

**(Romano et al., 2007)** : Romano P., et al. (2007), Biowep: a workflow enactment portal for bioinformatics applications. BMC bioinformatics 8.Suppl 1 (2007): S19.

**(Rostami et al., 2014)**: Rostami N. H., Kheirkhah E., Jalali M., (2014), An Optimized Semantic Web Service Composition Method Based on Clustering and Ant Colony Algorithm. *arXiv preprint arXiv:1402.2271*.

**(Russel et al., 2006a)**: Russell N., van der Aalst W.M.P., ter Hofstede A.H.M., Wohed P., (2006). On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. Proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Vol. 53 of CRPIT, pp. 95–104, Hobart, Australia, ACS.

**(Russel et al., 2006b)**: Russell N., Van Der Aalst W., ter Hofstede W.M.P., (2006), Exception handling patterns in process-aware information systems. Proceedings of CAiSE'06 (Luxembourg), pp. 288-302.

**(Sadiq et al., 2005)**: Sadiq S., Sadiq W., Orlowska M., (2005), A Framework for Constraint Specification and Validation in Flexible Workflows. Journal of Information Systems, Vol.30, n°5.

**(Saikali, 2001)**: Saikali, K., (2001), Flexibilité des workflows par l'approche objet : 2flows, un framework pour les workflows flexibles. Thèse de doctorat de l'école centrale de Lyon, France.

**(Sánchez et al., 2008)**: Sánchez M., Villalobos J., (2008). A flexible architecture to build workflows using aspect-oriented concepts. In AOM '08: Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling, pp. 25–30.ACM.

**(Sarkis, 2001)**: Sarkis J., (2001), Benchmarking for agility. Benchmarking Journal, Vol. 8, n°2, pp. 88-117.

**(Scheer et al., 1997)**: Scheer A.W., Borowsky S., Klabunde S. Traut., A., (1997), Flexible industrial applications through model-based Workflows. Proceedings of the International Conference on Enterprise Integration and Modeling Technology, pp. 439 – 448, Torino, Italy, Springer.

**(Scheer, 98)**: Scheer A.W., (1998), Intelligent Workflow – State of the Art. Workflow Business Process Engineering, Springer.

(**Schonenberg et al., 2007)** : Schonenberg M.H., Mans R.S., Russell N.C., Mulyar N.A., van der Aalst W.M.P, (2007), Towards a Taxonomy of Process Flexibility (Extended Version) . Internel BPM Center Report BPM-07-11, www.BPMcenter.or.

**(Seong-Jong et al., 2010)** : Seong-Jong Joo, Ik-Whan G. Kwon, Chang Won Lee, (2010), Future State of Outsourcing Supply Chain Information Systems: An Analysis of Survey Results. Int. journal of Enterprise information Systems, IGI Global, 2010, Vol. 6, n°3.

**(Serain, 97)**: Serain D., (1997), LE MIDDLEWARE: Concepts et Technologies, Edition MASSON. Paris.

**(Sharifi et al., 2000)**: Sharifi H. and Zhang Z. (2000), A methodology for achieving agility in manufacturing organizations. International Journal of Operations Production Management, Vol. 20, n°4, pp. 496-512.

**(Sheng et al., 2002)**: Sheng Q.Z., Benatallah, B., Dumas, M., O.I-Yan Mak E.,(2002), SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment. Proceedings of VLDB '02: the 28th international conference on Very Large Data Bases.

**(Singh et al., 2013)**: Singh R.K., Acharya P., (2013), Supply chain flexibility: a framework of research dimensions, Global Journal of Flexible Systems Management, Vol. 14, no. 3, pp.157–166, Springer.

**(Stricker, 2000)**: ACE-Flow: Deploying Agile Customer-Supplier Chain and Efficient Process Management with Federated WorkflowSystems, Swiss Priority Programme for Information and Communications Structures, first published in INFORMATIK/INFORMATIQUE 1/2000, SPP ICS project number 5003-054573

**(Strunk et al., 2009)**: Strunk A., Braun I., Reichert S., Schill A., (2009). Supporting Rebinding in BPEL. Proceedings of the 2009 IEEE International Conference on Web Services, ICWS '09, pp. 864–871, Washington, DC, USA. IEEE Computer Society.

**(Teway et al., 2013)**: Tewary A., Kosalge P., (2013), Implementing service oriented architecture – a case study. Int.Journal of Business Information Systems, Vol. 14, no. 2, pp.164 –181, Inderscience.

**(Tholkappia et al., 2014)**: Tholkappia A., Revathi V., (2014), Orchestration Of Web Services Based On Qos Using User And Web Services Agent. *IJRET:* International Journal of Research in Engineering and Technology. Vol. 3, special issue 7, Available at : http://www.ijret.org

**(Touzi, 2007)** : Touzi, J., (2007), Aide à la conception de Système d'Information Collaboratif support de l'interopérabilité des entreprises. Thèse de doctorat. Centre de Génie Industriel, Ecole des Mines d'Albi Carmaux.

**(Traverso et al., 2004)**: Traverso P., and M. Pistore., (2004), Automatic Composition of Semantic Web Services into Executable Processes. International Semantic Web Conference (ISWC'04).

**(Tripathy et al., 2012)**: Tripathy A. K., Patra M. R., Pradhan S. K., Rafique R., Mohanty P., Dash S., (2012). Graph based service selection for composition and adaptation. International Journal of ACM Jordan: The Research Bulletin of Jordan ACM, Vol.2, pp.83-89.

**(Tsai et al., 2010)** : Tsai C.-H, Huang K.-C., Wang F.-J., Chen C.-H., (2010), A distributed server architecture supporting dynamic resource provisioning for BPM-oriented workflow management systems. Journal of systems and Software, Vol. 83, no. 8, pp. 1538–1552.

**(Upton & Mc Affe, 1996)**: UPTON D. M., Mc AFFEE A., (1996), The real virtual factory, Harvard Business Review, July – August, 1996.

**(Van der Aalst, 97)**: Van der Alast W.M.P., (1997), Verification of workflow nets, LNCS, vol.1248, pp.407– 426.

**(Van der Aalst, 98)**: Van der Aalst. W.M.P., (1998), The Application of Petri Nets to Workow Management. The Journal of Circuits, Systems and Computers, Vol. 8, n°1, pp. 21-66.

**(Van der Aalst, 99)**: Van Der Aalst W. M. P, (1999), Process oriented architectures for electronic commerce and interogranizational workflow. Journal of Information systems, Vol. 24, n°9.

**(Van der Aalst , 2000**): Van Der Aalst W. M. P, (2000), Loosely Coupled Interorganizational Workflows : modeling and analyzing workflows crossing organizational boundaries. Journal of Information and Management Vol. 37, n° 2, Pp. 67-75.

**(Van der Aalst et al, 2001)**: Van Der Aalst W. M. P, Weske M., (2001), The P2P Approach to Interorganizational Workflows. Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag, pp. 140–156.

**(Van der Aalst et al., 2002)**: Van Der Aalst W. M. P, **Van Hee K., A.,** (2002), Workflow Management: Models, Methods and Systems. Copyright © 2000 W. M. P. van der Aalst/K. M. van Hee, *wwwis.win.tue.nl/~wvd**aalst**/publications/p120.pdf*

**(Van derAalst et al., 2003)**: Van Der Aalst W. M.P., ter Hofstede W.M.P, Kiepuszewski A.H.M, Barros B.A.P., (2003), Workflow Patterns. DAPD, Vol. 14, n°1, pp.5-51.

**(Van der Aalst, 2005)**: Van Der Aalst W. M.P., Ter Hofstede A. H. (2005). YAWL: yet another workflow language. Journal of Information systems, Vol. 30, n°4, pp. 245-275.

**(Van der Aalst, 2008)**: Van der Aalst W.M.P., van Hee K.M., ter Hofstede A.H.M., Sidorova N., Verbeek H.M.W., (2008), Voorhoeve M., Wynn M.T.. Soundness of Workflow Nets: Classification, Decidability and Analysis. BPM Center Report BPM-08-02, BPMcenter.org.

**(Verbeek et al., 2004)**: Verbeek H. M. W., Van der Aalst W.M.P., and Kumar A., (2004), XML/WOFLAN: Veriication and extensibility of an xml/petri-net-based language for inter-organizational workflows. Inf. Tech. And Management, Vol.5 (1-2): pp. 65–110.

**(Voorhoeve et al., 97)** : Voorhoeve M., Van Der Aalst W.M.P., (1997), Ad-hoc Workflow: Problems and Solutions. In R. Wagner, editor, Proceedings of the 8th. International Workshop on Database and Expert Systems Applications, DEXA'97, pp. 36–40, Toulouse, France.

**(Walid, 2011)** : Walid F., (2011), Décentralisation Optimisée et Synchronisation des Procédés Métiers Inter-Organisationnels. Thèse de doctorat, Université Henri Poincaré – Nancy 1.

**(Wang et al., 2009):** Wang M., Ramamohanarao K., Chen J., (2009). Trust-based robust scheduling and runtime adaptation of scientific workflow. Concurrency and Computation: Practice and Experience, Vol. 21, n°16, pp. 1982-1998.

**(Weber et al., 2004)** : Weber B., Wild W., Breu R., (2004), CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. Advances in CBR, Vol. 3155, LNCS, pp. 89-101, Springer.

**(Weber et al., 2008)** : Weber B, Reichert M, Rinderle-Ma S., (2008), Change patterns and change support features- Enhancing flexibility in process-aware information systems. Journal of Data & Knowledge Engineering,Vol. 66, pp. 438-466.

**(Weerawarana et al., 2005)**: Weerawarana S., Curbera F., Leymann F., Storey T., Ferguson D., (2005),Web Services Platform Architecture. Prentice Hall. Mai 2005.

**(Weske et al., 96)**: Weske M., Vossen G., Medeiros C., (1996), Scientific Workflow Management : Architecture and Applications. Fachbericht Angewandte Mathematik und Informatik.Universität Münste.

**(Weske, 2007)**: Weske M., (2007), Business Process Management : Concepts, Languages, Architectures. Springer Verlag, first edition, November 2007.

**(WFMC, 1995)**: WfMC. Workflow Reference Model, Workow Management Coalition, www.wfmc.org.

**(WFMC, 1999)**: WFMC, *Workflow Management Coalition Terminology and Glossary*, technical report WfMC-TC-1011, February 1999.

**(Wil, 2003)**: Wil M.P. et al., (2003), Web Service Composition Languages: Old Wine in New Bottles. Proceedings of the 29th EuroMicro Conference.

**(Winograd, 90)**: Winograd T., (1990), What can we teach about Human-computer interaction? Proceedings of the CHI'90 Conference on Human Factors in Computing, Seattle, pp. 443- 449.

**(Wohed et al., 2006)**: Wohed P., Van der Aalst W. M. P., Dumas M., ter Hofstede A. H., Russell, N., (2006). On the suitability of BPMN for business process modelling. Business Process Management, pp. 161-176. Springer Berlin Heidelberg.

**(Xiao et al., 2011)**: Xiao Z., Cao D., You C., Mei H., (2011), Towards a Constraint-Based Framework for Dynamic Business Process Adaptation. Proceedings of the 2011 IEEE International Conference on Services Computing, SCC'11, pp. 685–692.

**(Yan et al., 2010)**: Yan H., Zhijian W., Guiming L., (2010), A novel semantic Web service composition algorithm based on QoS ontology. Proceedings of the International Conference on Computer and Communication Technologies in Agriculture Engineering (CCTAE), Vol. 2, pp. 166-168. IEEE.

**(Yang et al., 2004)**: Yang J., Papazoglou M.P., (2004), Service components for managing the lifecycle of service compositions. Journal of Information Systems, Vol. 29, n°2, pp. 97–125.

**(Yang et al., 2005)**: Yang Y., Tan Q., Xiao Y., (2005), Verifying web services composition based on hierarchical colored Petri nets. Proceedings of the 1st International Workshop on Interoperability of Heterogeneous Information Systems, pp. 47-54, Bremen, Germany, ACM.

**(Yi et al., 2004 )** : Yi X., and Kochut K. J., (2004), A CP-nets-based design and verification framework for web services composition. Proceedings of the IEEE International Conference on Web Services, ICWS '04, pp. 756– 765, Washington, DC, USA, IEEE Computer Society.

**(Yusuf et al. 2003)**: Yusuf Y., Oadeleye E., Sivayoganathan, K., (2003), Volume flexibility: the agile manufacturing conundrum, Management Decision Support Systems, Vol. 41, n°7, p. 613.

**(Zeshan et al., 2011)**: Zeshan, F., & Mohamad, R. (2011). Semantic web service composition approaches: overview and limitations. International Journal of New Computer Architectures and their Applications (IJNCAA), Vol.1, n°3, pp. 640-651.

**(Zhai et al., 2008)** : Zhai Y., Su H., and Zhan S. (2008), A Reflective Framework to Improve the Adaptability of BPEL-based Web Service Composition.. IEEE International Conference on Services Computing, SCC '08, Vol. 1, pp. 343–350.

**(Zurawski, 2006)**: Zurawski R. (2006), Integration Technologies for Industrial Automated Systems, CRC Press, july 2006, Computers 600p.

# Partie III

# Annexes

# Annexe A

# Standards des Services Web et Langage BPEL

Cette annexe fournit un bref aperçu sur les standards de mise en œuvre des services Web qui représentent l'instanciation la plus répandue de l'architecture SOA. L'accent est mis sur la couche processus métier qui dans notre cas, repose sur le langage BPEL. Aussi, nous présentons les différents patrons de contrôle de flux et leur implémentation dans le langage BPEL.

## A.1 Les standards des services Web

L'implémentation des services Web repose sur la description de documents, de normes et de protocoles régissant l'échange d'information dans une architecture orientée service (SOA). La figure A.1 montre une architecture de base (ou de référence) des services Web avec trois principaux éléments : le registre UDDI pour la publication et la découverte de services, le langage WSDL pour la description des services et le protocole SOAP pour les invocations de services.

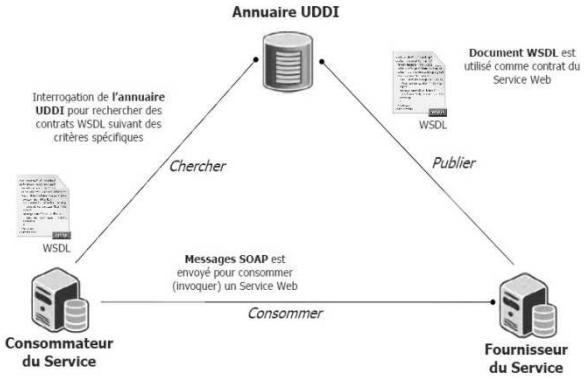



**Figure A.1.** *Architecture SOA basée sur les services Web*

- **Le protocole SOAP (Service Oriented Access Protocol)**

SOAP est le protocole qui assure l'échange de messages dans les SOA. Du fait qu'il est basé sur XML, il permet l'échange de données structurées indépendamment des langages de programmation ou des systèmes d'exploitation. SOAP permet l'échange d'informations dans un environnement décentralisé et distribué, comme Internet, indépendamment du contenu du message.

Il ne véhicule pas de modèle de programmation ou d'implémentation, mais fournit les outils nécessaires pour définir des modèles opérationnels d'échange (styles d'échange) aussi diversifiés que les systèmes de messagerie asynchrone et l'appel de procédures distantes (RPC).

Il peut être employé dans tous les styles de communication : synchrones ou asynchrones, point à point ou multipoints. SOAP utilise principalement deux standards qui sont HTTP comme protocole de transport des messages SOAP et XML pour structurer les requêtes et les réponses, indiquer les paramètres des méthodes, les valeurs de retours, et les éventuelles erreurs de traitements (Bernard et al., 2003). La figure A.2 montre la structure d'un message SOAP dans une requête de service weA. Le message est englobé dans une enveloppe et divisé en 2 parties : l'entête et le corps. L'entête (*Header*) offre des mécanismes flexibles pour étendre un message SOAP sans aucune préalable connaissance des parties communicantes. Les extensions peuvent contenir des informations concernant l'authentification, la gestion des transactions, le payement, etc. Le corps (*Body*) offre un mécanisme simple d'échange des informations mandataires destinées au receveur du message SOAP. Cette partie contient les paramètres fonctionnels tels que le nom de l'opération à invoquer, les paramètres d'entrée et de sortie ou des rapports d'erreur.

- **Le langage WSDL (Web Service Description Language)**

Basé sur XML, WSDL permet de décrire le service Web, en précisant les méthodes disponibles, les formats des messages d'entrée et de sortie, et comment y accéder (Weerawarane et al., 2005).

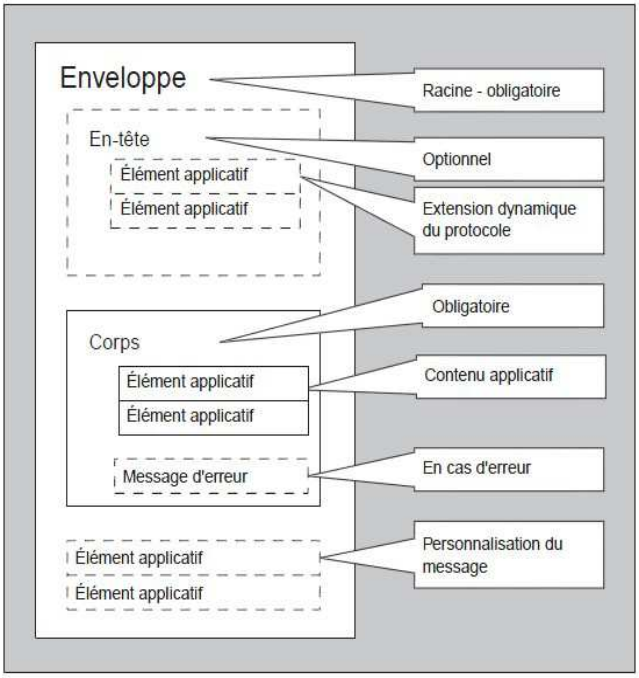



**Figure A.2**. *Structure générale d'un message SOAP*

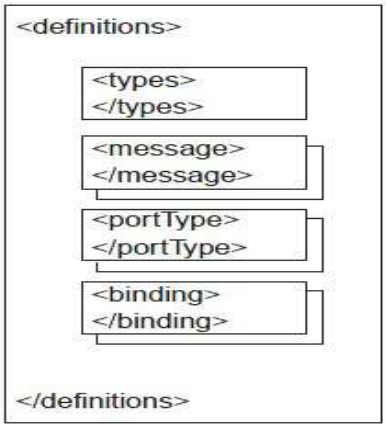



**Figure A.3.** *Structure générale d'un document WSDL*

La spécification de WSDL introduit quelques concepts essentiels à sa compréhension dont:

- Les types (*types*) : il s'agit de la définition des types de données qui structurent les messages, ceux-ci reposent sur un système de typage (tels que les schémas XML, par exemple).
- Les messages (*messages*) : représentent une définition typée abstraite des données échangées entre les nœuds de communication.
- Les opérations (*operations*) : définissent la description abstraite d'ensembles cohérents de messages (messages en entrée, messages en sortie) qui forment les unités d'interaction avec le service Web.

- Les types de ports (*port types*) : constituent des ensembles abstraits d'opérations prises en charge par un ou plusieurs nœuds de communication.
- Les liaisons (*bindings*) : décrivent les protocoles concrets et les formats de message pour chaque type de port.
- Les ports (*ports*) : ce sont les nœuds de communication particuliers, chacun étant défini comme une combinaison entre une liaison et une adresse réseau.
- Les services (*services*) : il s'agit de l'ensemble des ports exposés pour permettre l'accès aux services correspondants.

- **UDDI (Universal Description, Discovery, and integration)**

UDDI est une norme d'annuaire de services Web appelée via le protocole SOAP. Pour publier de nouveaux services Web, les indexer et faciliter ainsi leur identification et leur localisation afin d'interagir avec eux. Le protocole d'utilisation de l'UDDI contient trois fonctions de base: « publish » pour enregistrer un nouveau service, « find » pour interroger l'annuaire et « bind » pour effectuer la connexion entre l'application cliente et le service. Comme pour la certification, il est possible de constituer des annuaires UDDI privés, dont l'usage sera limité à l'intérieur de l'entreprise.

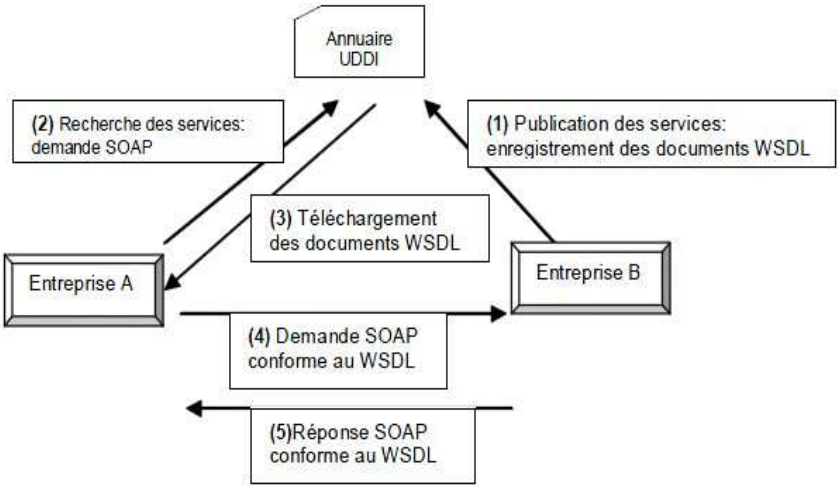



**Figure A.4.** *Schéma général d'UDDI*

## A.2 Architecture étendue des services Web

Différentes extensions de l'architecture de référence ont été proposées dans la littérature. Le groupe d'architecture du W3C travaille activement à l'élaboration d'une architecture étendue standard. Une architecture étendue est constituée de plusieurs couches se superposant les unes sur les autres, d'où le nom de *pile de services Web* ; la figure A.5 décrit un exemple d'une telle pile. Celle-ci est constituée de plusieurs couches, chaque couche s'appuyant sur un standard particulier. On retrouve, au-dessus de la couche de transport, les trois couches formant l'infrastructure de base décrite précédemment. Ces couches s'appuient sur les standards émergents SOAP, WSDL et UDDI.

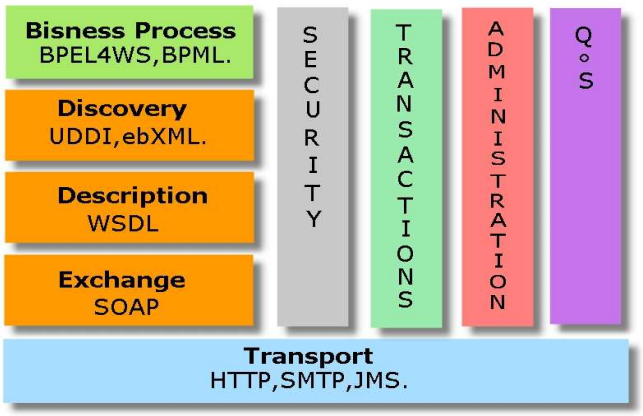



**Figure A.5.** *Architecture en pile des services Web*

L'infrastructure de base définit les fondements techniques permettant de rendre les processus métiers accessibles à l'intérieur d'une entreprise et au-delà même des frontières d'une entreprise. Dans ce contexte, deux types de couches permettent de compléter l'architecture de base :

- les couches dites transversales (par *exemple :* sécurité, administration, transactions et qualité de services (QoS)) rendent viable l'utilisation effective des services Web dans le monde industriel ;
- une couche « Business process » qui permet l'utilisation effective des services Web dans le domaine du e-business.

Pour supporter la couche « Business process », plusieurs langages de spécification de processus métiers ont été proposés : BPML, WSCI, BPEL, etc… Dans la suite, nous présentons brièvement le langage BPEL en mettant l'accent sur l'aspect comportemental des processus réalisé via les patrons de contrôle de flux.

## A.3 Eléments du langage BPEL

### A.3.1 Concepts de base

BPEL ajoute au WSDL des interactions avec état, les interactions de messages complexes, les interfaces de composants sont des processus abstraits arbitraires, des corrélations parmi des « conversations » longues multiples. Concrètement, un service Web peut être implémenté tel un processus BPEL4WS. Cela peut être le cas pour une orchestration, c'est-à-dire un processus de type exécutable. En fait, les points d'entrée, correspondant aux opérations qui peuvent êtres appelées dans l'interface WSDL, sont référencés dans BPEL4WS et vont utiliser les messages arrivants pour alimenter l'orchestration. Chaque étape du processus est appelée une activité (par exemple : <invoke> pour activer un service Web, <reply>pour générer une réponse dans le cas d'un échange de type input/ output, etc). Plusieurs activités primitives peuvent êtres ainsi regroupées en séquence pour former des activités structurées. BPEL permet de modéliser deux types de procédés :

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

Le grand avantage de BPEL est la possibilité de décrire les interactions entre les logiques métier des différentes entreprises à travers les services weA. Les éléments du procédé BPEL sont : les liens de partenaires, les activités et les données.

- **Les partenaires**

La relation entre le procédé et un partenaire est une relation "poste à poste" (peer-to-peer). Le partenaire est en même temps le consommateur d'un service que le procédé produit, et le producteur d'un service que le procédé consomme. Le lien de partenariat (*partnerLink*) définit le rôle que joue chacun des deux partenaires dans une conversation. Chaque lien de partenariat a un type (*partnerLinkType*).

- **Les propriétés de messages**

BPEL4WS ajoute des propriétés aux messages échangés entre partenaires. Un exemple de ces propriétés est la "corrélation", qui permet de faire le lien entre les messages appartenant à la même conversation.

- **Les activités**

**TAB A.1.** *Description des activités BPEL*

| **Les activités basiques** | | **Les activités structurées** | |
|---|---|---|---|
| **<invoke>** | pour invoquer une opération dans un service Web. | **<sequence>** | pour définir un ordre d'exécution. |
| **<receive>** | pour attendre un message d'une source externe. | **<switch>** | pour l'acheminement conditionnel. |
| **<reply>** | pour répondre à une source externe. | **<while>** | pour les boucles. |
| **<wait>** | pour attendre un certain temps. | **<pick>** | pour attendre l'arrivée d'un événement |
| **<assign>** | pour copier les données d'une place à l'autre. | **<flow>** | pour l'acheminement parallèle. |
| **<throw>** | pour lancer une erreur d'exécution. | **<scope>** | pour regrouper les activités afin qu'elles soient traitées par le même gestionnaire d'erreur. |
| **<terminate>** | pour terminer l'instance de service en entier. | **<compensate>** | Pour invoquer les activités de compensation par le gestionnaire d'erreurs, pour défaire l'exécution d'un regroupement d'activités. |
| **<empty>** | ne fait rien (utile pour la synchronisation des activités parallèles). | | |

- **Les variables**

Les variables permettent aux processus BPEL de garder l'état entre les interactions. Les types de variables peuvent être soit des messages WSDL, soit un simple schéma XML, soit un élément de schéma XML. Il y'a quatre façons d'utiliser les variables :

- La variable d'entrée recevant le message envoyé par un partenaire (doit connaitre le type de message) ;
- La variable de sortie contenant le message envoyé à un partenaire (doit connaître le type de message) ;
- La variable d'erreur retournée par une activité gérant les erreurs
- La variable temporaire utilisée pour des traitements intraprocessus

Les variables peuvent êtres globales (déclarées dans le cadre *scope* du processus global) ou locales (déclarées dans le cadre non global).

Le cadre peut contenir les déclarations suivantes :

- Les variables
- Les ensembles de corrélation

- Les gestionnaire d'erreurs (catch /throw)
- Le gestionnaire de compensation
- Le gestionnaire d'événements
- L'activité primaire (pouvant contenir des activités)

Le constructeur de flux de base permet la concurrence entre les activités. Les liens sont utilisés pour renforcer les dépendances (ordre partiel) entre activités dans un constructeur de flux.

En tant que langage pour composer un ensemble de services en un seul service, BPEL4WS consiste principalement à réaliser des invocations vers des services et/ou recevoir des invocations de la part de clients. BPEL4WS interagit avec ce que l'on appelle alors des « partenaires de processus ». Un partenaire est soit un service que le processus invoque (partenaire invoqué), ou bien un service qui invoque le processus (partenaire client). Il existe enfin un troisième type de partenaire, que le processus invoque, et qui invoque le processus de retour (ou bien le contraire) ; pour permettre l'exécution des processus, les services Web implémentés tels des processus BPEL4WS suivent un modèle de cycle de vie instancié. Cela signifie qu'un client de ces services interagit toujours avec une instance particulière d'un service (processus). Pour qu'un tel mécanisme soit rendu possible, des champs identificateurs ou « clés » sont émis dans les messages circulant pour spécifier l'instance spécifique impliquée dans l'interaction. Si une instance spécifiée dans un message en direction du processus n'existe pas, elle sera créée automatiquement.

### A.3.2 Invocation des services Web

La définition des processus BPEL est sous forme d'un document XML utilisant l'élément racine <process>. Juste après cet élément vient l'élément <sequence>, cette balise permet d'attendre la réception d'un message entrant pour commencer le processus. Cette attente est modelée avec l'élément <invoke> qui appellera les services Web relatifs. De telles invocations peuvent être faites de manière séquentielle ou parallèle.

Dans le cas séquentiel nous utilisons simplement <invoke> pour chaque invocation.

Cela est représenté dans ce qui suit :

```
<process…>…
…
   <sequence>
        // Attente de reception
        <receive…/>
        // Invocation
        <invoke…/>
        <invoke…/>
        <invoke…/>…
  </sequence>
</process>
```

Pour appeler des services Web concurrents nous pouvons utiliser l'élément <flow>. Dans l'exemple qui suit les trois *invoke* sont exécutés concurremment.

```
<process…>…
    <sequence>
        // Attente d'un processus arrivant
     <receive…/>

        //  invocation parallele de services Web
    <flow>
        <invoke…/>
        <invoke…/>
        <invoke…/>
    </flow>
…
    </sequence>
</process>
```

On pourrait également combiner les deux éléments *flow* et *sequence* afin de définir plusieurs ordres s'exécutant concurremment.

### A.3.3 Quelques Patterns de contrôle de flux avec BPEL4WS

Cette section introduit quelques uns des patterns de contrôle de flux de base (Control Flow Patterns) supportés par BPEL4WS, en montrant leurs implémentations possibles.

- **WCP-1: Séquence (Sequence)**

Une activité de la séquence est disponible une fois que les activités précédentes de la séquence soient terminées. Le *pattern de la séquence* est représenté dans la figure A.6.

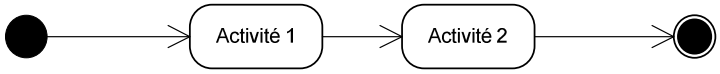



**Figure A.6.** *Diagramme d'activité du « Sequence Pattern »*

Dans cette séquence, l'activité *« Activité 2 »* est disponible une fois que l'activité *« Activité 1 »* a terminé son exécution.

```
<sequence>
   Activité1   Ordonnancement des activités: Activité1 et activité2
  Activité2
</sequence>
```

- **WCP-2: Parallélisme (Parallel Split)**

Le parallélisme représente un point dans le processus où un lien de contrôle simple est divisé en plusieurs liens de contrôle s'exécutant en parallèle. Le pattern du parallélisme est représenté dans la figure A.7.

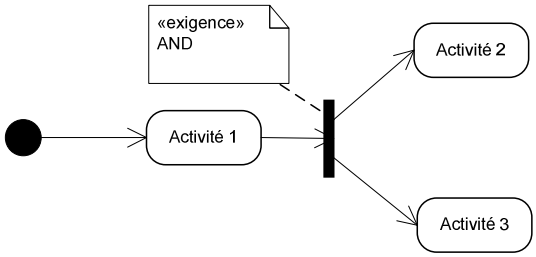



**Figure A.7.** *Diagramme d'activité du « Parallel split Pattern »*

Dans ce parallélisme, après l'exécution de l'activité *« Activité 1 »*, le lien de contrôle passe aux activités *« Activité 2 »* et *« Activité 3 »*. Ces activités seront exécutées de façon concurrente.

```
<sequence>
     Activity1
   <flow>
       Activité2   Definit les activités en parralèle.
       Activité3

 </flow>
</sequence>
```

- **WCP-3: Synchronisation *(Synchronization)***

A la jointure de deux (ou plus) branches parallèles, un point d'attente (de synchronisation) est défini, il représente un point d'arrêt d'une activité en attente de la terminaison d'une autre. Le *pattern de la synchronisation* est représenté dans la figure A.8.

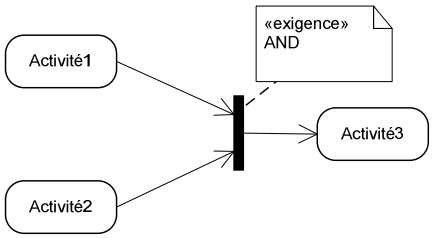



**Figure A.8**- *Diagramme d'activité du « Synchronization Pattern »*

Dans cet exemple de synchronisation, une fois les activités « *Activité 1* » et « *Activité 2* » terminées, l'activité « *Activité 3* » *est activée*.

```
<sequence>
   <flow>
     Activité1
     Activité2
  </flow>
     Activité3
 </sequence>
```

- **WCP-4: Choix Exclusif *(Exclusive Choice)***

Le choix exclusif est un point dans le processus où un chemin est choisi entre plusieurs. Ce choix est fait à l'aide d'une décision prise au moment de l'exécution, cette décision est basée sur une information ou une donnée du processus. Contrairement au Parallélisme, un seul lien dans le flux de contrôle est activé. Le *pattern du Choix Exclusif* est représenté dans la figure A.9.

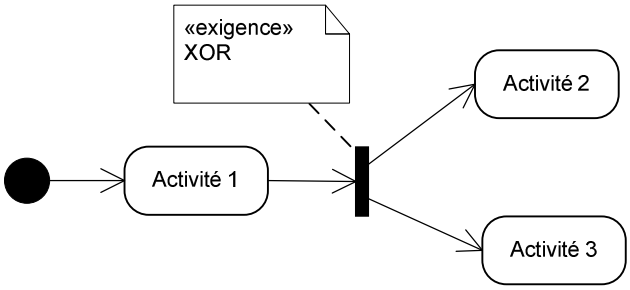



**Figure A.9.** *Diagramme d'activité du « Exclusive Choice pattern »*

Dans cet exemple de choix exclusif, une fois l'activité « *Activité 1* » terminée, une sélection, soit de l'activité « Activité 2 » soit de l'activité « Activité 3 », est faite pour continuer le processus.

```
<sequence>
    Activité1
     <switch>
        <case> //ou <case condition="C1">
            Activité2
        </case>
        <case> //ou <case condition="C2"> ou <otherwise>
            Activité3
        </case>
     </switch>

</sequence>
```

- **WCP-5: Fusion Simple *(Simple Merge)***

La fusion simple est un point dans le processus dans lequel une ou plusieurs branches du flux de contrôle se joignent sans nécessité de synchronisation. Le *pattern de la Fusion Simple* est représenté dans la figure A.10.

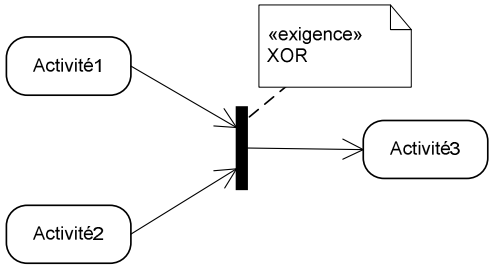



**Figure A.10.** *Diagramme d'activité du « Simple Merge pattern »*

Dans cet exemple, lorsque les activités « *Activité 1* » *ou* « *Activité 2* » *ou les deux sont terminées, le processus continue avec l'activité* « *Activité 3* ».

```
<switch>
   <case condition="C1">
     Activité1
   </case>
   <case condition="C2">
     Activité2
  </case>
</switch>
     Activité3
```

D'autres patrons de branchement et de synchronisation avancés sont aussi supportés dans le langage BPEL comme le choix multiple, la synchronisation multiple, la terminaison implicite,…etc.

# Annexe B

# Environnement de développement



Cette annexe fournit quelques précisions sur l'environnement de développement de nos frameworks ainsi que certains éléments relatifs à la création et le déploiement d'un processus BPEL sous NetBeans, l'invocation de services Web locaux et externes et la génération d'applications composites pour le test des processus.

## B.1 Environnement de développement des frameworks

Nos frameworks ont été développés sous la palteforme windows 7 en utilisant un ensemble d'outils permettant la description et l'exécution des WF, la création, le déploiement et l'invocation des services Web ainsi que des outils de développement et de test de d'applications.

### B.1.1 Outils de Workflow

- **Langage de définition de Workflows: BPEL4WS**

*BPEL4WS* (**B**usiness **P**rocess **E**xecution **L**anguage for **W**eb **S**ervices) est un langage de spécification de processus WF basé sur *XML*. Il permet de définir les interactions intra et interentreprises et de composer une multitude de services Web, synchrones ou non, au sein d'un flot de Workflow.

Un procédé *BPEL4WS* est formé d'un ensemble de partenaires, d'activités, de transitions et de gestion d'exception de compensation. Les partenaires de *BPEL4WS* peuvent être statiques et dans ce cas ils sont connus à priori et le comportement est fixé dès le début, ou bien dynamique et dans ce cas ils ne sont connus qu'à l'exécution. L'annexe A de ce document fournit plus de détails sur le langage BPEL.

- **Moteur de Workflow: Open ESB 2.2**

OpenESB est une implémentation Open Source d'un ESB (Enterprise Service Bus) développé par Sun et basée sur JBI (Java Business Integration), elle comprend un ensemble de composants fonctionnant avec GlassFish ainsi qu'un ensemble de plugin pour Netbeans. OpenESB est composé de plusieurs briques:

- Un outil d'orchestration de services (BPEL),
- Un outil d'IEP (Intelligent Event Processing), qui permet d'agréger et d'ordonnancer des messages dans le temps,
- Un outil d'administration (dit console ESB) qui n'est présent que si OpenESB est déployé sur le serveur d'application GlassFish (car il est un composant d'extension de la console d'administration GlassFish).

La solution GlassFish ESB (OpenESB déployé sur GlassFish server) est la plus aboutie dans le monde open-source, autant pour sa robustesse et sa fiabilité que pour ses fonctions d'administration.

- **Serveur d'application : GlassFish server v2**

Sun GlassFish server v2 est un serveur compatible avec la plate-forme Java™ Enterprise Edition (Java EE 5) permettant de développer et de déployer des applications J2EE et des services Web Java. GlassFish Server offre une flexibilité qui aide à diminuer les coûts d'exploitation en améliorant la productivité des développeurs, simplifiant l'architecture logicielle et en dynamisant le support des mises à jour.
Dans notre application, GlassFish joue deux rôles différents. Au début, on le considère comme un serveur d'application en déployant nos EJBs qui contiennent les différents services Web, ces services seront consommés par les processus BPEL. La deuxième utilisation de GlassFish consiste à le considérer comme un moteur de Workflow pour déployer nos processus BPEL, mais pour pouvoir jouer ce rôle, GlassFish doit intégrer le moteur de Workflow OpenESB (Moteur de Workflow pour BPEL).

### B.1.2 Outils de services Web – les EJB

Pour la création des services Web, nous avons utilisé les EJB (Enterprise Java Beans) qui sont des composants Java portables, réutilisables et déployables qui peuvent être assemblés pour créer des applications. L'architecture des EJB est une architecture de composants pour le développement et le déploiement d'applications d'entreprise distribuées basées sur des composants. Les applications

écrites en utilisant l'architecture des EJB sont évolutives, transactionnelles et sûres. Ces applications peuvent être écrites une fois, puis déployées sur toute plate-forme serveur qui supporte la spécification des EJB. Dans notre application, nous utilisons les EJB comme des conteneurs de services Web, un EJB peut contenir plusieurs services Web à la fois ; une fois l'EJB déployé, tous ses services seront prêt à être consommés.

### B.1.3 Outils de programmation

- **Le langage java** : pour la réalisation du framework de coopération, nous avons utilisé JAVA 6 (ou JAVA 1.6), une version qui apporte plusieurs améliorations avec de nombreuses nouvelles fonctionnalités concernant XML (évolution de JAXB qui utilise les schémas XML).

- **L'IDE Netbeans** : NetBeans est un environnement de développement intégré (EDI) développé par Sun Microsystems, il est spécialisé dans le domaine Java. C'est un logiciel qui propose un éditeur de texte, de quoi créer et gérer des projets, compiler, débugger le code source, tester les programmes et gérer les versions. Sa conception est complètement modulaire. Tout est module, même la plateforme ; ce qui fait de NetBeans une boite à outils facilement améliorable ou modifiable. Il intègre une large panoplie de fonctionnalités: XML, documentation, databases ...etc.

Tous les patrons de coopération implémentés dans notre application nécessitent une manipulation des fichiers XML (les fichiers BPEL décrivant les processus) et leurs éléments (balises et attributs), donc nous avons besoin d'un outil qui nous permet d'ouvrir des fichiers XML à partir d'un programme JAVA, de pouvoir manipuler son contenu et effectuer diverses opérations sur ses éléments comme l'insertion d'une balise et la modification d'une valeur d'attribut. Pour cela, nous utilisons l'API jDom2 de java.

- **L'API JDom2** (Java Document Object Model) : est une API open source du langage Java développée indépendamment de Sun Microsystems. Elle permet de manipuler des documents XML plus simplement qu'avec les API classiques. Son utilisation est pratique pour tout développeur Java et repose sur les API XML de Sun. JDOM n'est pas un parseur, elle a d'ailleurs besoin d'un parseur externe de type SAX ou DOM pour analyser un document et créer la hiérarchie d'objets relative à un document XML. Un document XML est encapsulé dans un objet de type *Document* qui peut contenir des objets de différents types encapsulés dans des classes dédiées : *Element*, *Attribute*, *Text*, P*rocessingInstruction*, *Namespace*, *Comment*, *DocType*, *EntityRef*, *CDATA*. Un objet de type *Element* peut contenir des objets de type *Comment* (commentaire), *Text* et d'autres objets de type *Element*.

A l'exception des objets de type *Namespace*, les éléments sont créés en utilisant leur constructeur. JDOM vérifie que les données contenues dans les éléments respectent la norme XML. Par exemple, il n'est pas possible de créer un commentaire contenant moins de deux caractères qui se suivent. Une fois un document XML encapsulé dans un arbre d'objets, il est possible de modifier cet arbre dans le respect des spécifications XML.

L'API JDom propose plusieurs fonctionnalités dont:

- La création de documents XML
- L'encapsulation d'un document XML sous la forme d'objets Java de l'API
- L'exportation d'un document dans un fichier, un flux SAX (**S**imple **A**PIfor **X**ML est une API générale pour la lecture d'un flux XML) ou un arbre DOM
- Le support de XSLT

### B.1.4 L'outil GraphViz[12]

Afin de visualiser graphiquement, les modèles de WF et montrer les changements apportés suite à une opération d'interconnexion ou d'adaptation/évolution, nous avons utilisé l'outil GraphViz v1.9 pour générer des graphes représentant les modèles de WF.

GraphViz est un Open-source conçu par une équipe des laboratoires de recherche AT&T (American Telephone & Telegraph). Cette application convient à la représentation de graphes très denses comprenant un très grand nombre de nœuds grâce à des algorithmes très puissants. Elle est très rapide à l'exécution et le rendu est optimisé, évitant le recouvrement des nœuds et le croisement des liens.

De plus, entièrement paramétrable, l'application permet de personnaliser le rendu des graphes par le choix des formes, couleurs et polices de caractères. Le format des fichiers d'entrée est simple et souple. Les formats de sortie sont très variés.

## B.2 Création et déploiement d'un processus BPEL sous NetBeans

### B.2.1 Création d'un fichier BPEL

La création d'un fichier BPEL sous NetBeans commence par la création d'un nouveau projet de type « module BPEL » dans la catégorie SOA. A ce niveau, un processus BPEL vide est créé.

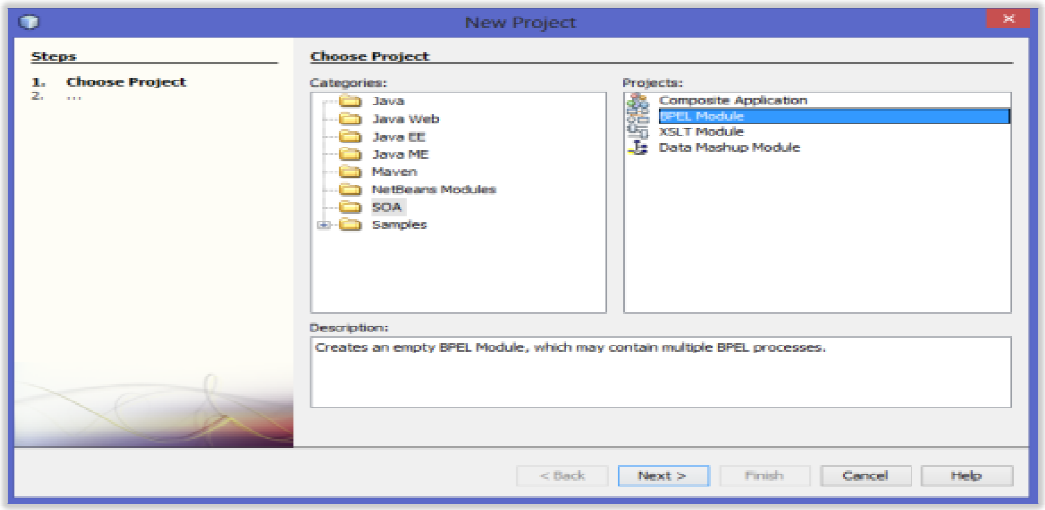



**Figure B.1.** *Création d'un processus WF sous NetBeans*

Pour ajouter de nouveaux éléments (activité « receive », « invoke », « assign » …) au processus, il suffit de sélectionner un élément depuis la palette et le glisser jusqu'à l'emplacement voulu.

[12] http://www.graphviz.org

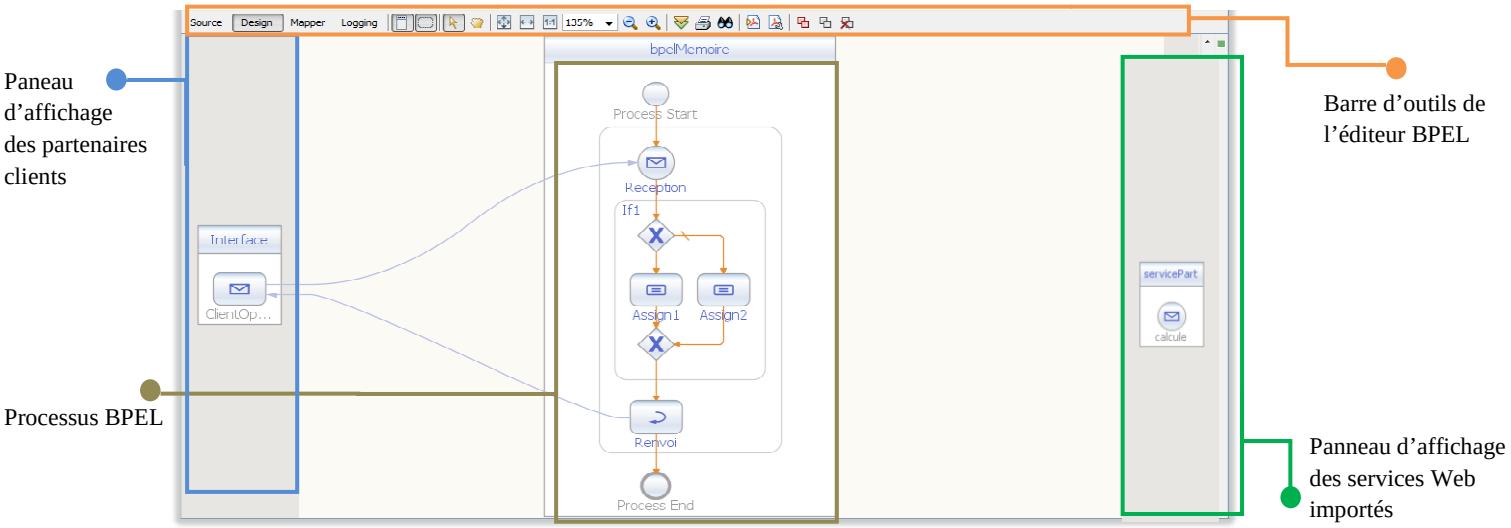



**Figure B.2.** *Schéma d'un processus BPEL*

### B.2.2 Création d'un fichier WSDL

Apres avoir construit le processus BPEL, il faut créer un partenaire qui représentera l'interface du processus. Un partenaire est représenté par un document WSDL de type « concrete WSDL » qui sera importé par le processus BPEL.

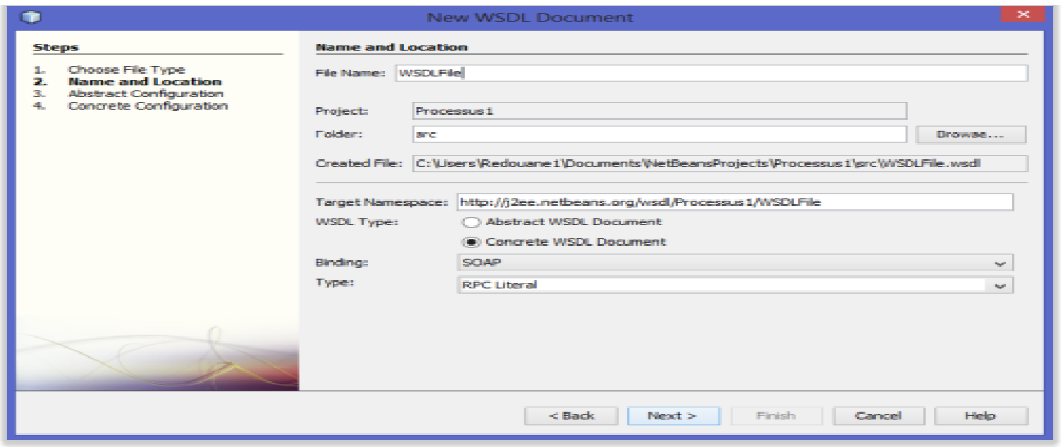



**Figure B.3.** *Création d'un fichier WSDL*

Il faut modifier le port SOAP du fichier WSDL de « ${HttpDefaultPort} » en « 9080 », cette modification permet d'utiliser ce fichier WSDL comme un partenaire externe dans un autre processus.

Il s'agira ensuite de spécifier les variables d'entrée/sortie du service.

```
<port name="WSDLFilePort"
      binding="tns:WSDLFileBinding">
    <soap:address
    location="http://localhost:9080/WSDLFileService/WSDLFilePort"/>
</port>
```

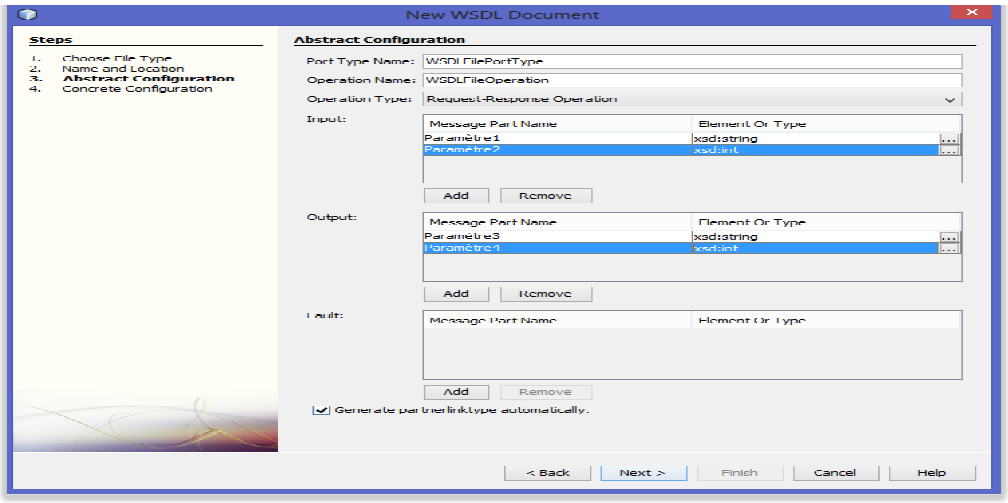



**Figure B.4.** *Définition des variables d'entrées/sorties*

Une fois le fichier WSDL créé, il faut le glisser dans le panneau d'affichage des partenaires qui montre le nouveau partenaire créé. L'activité « invoke » permet d'invoquer ce partenaire. Une étape d'assignation des paramètres est nécessaire pour assigner les paramètres des variables d'entrées / sorties du service invoqué avec les variables locales du processus.

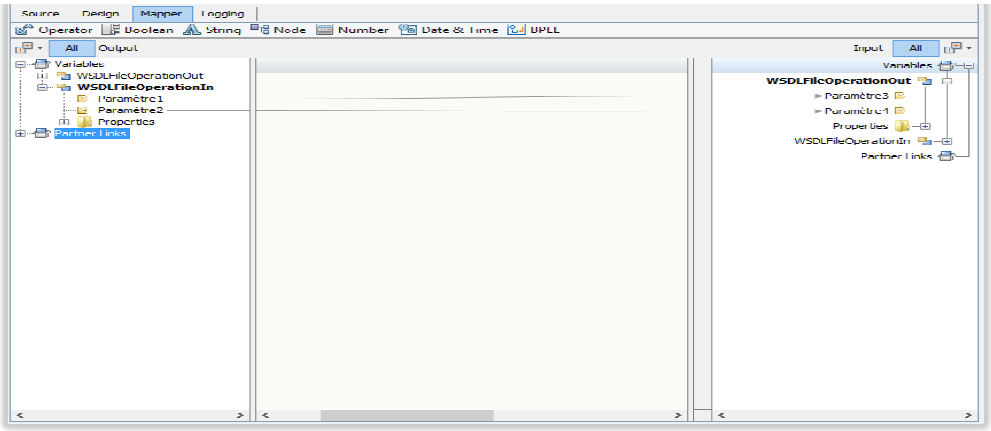
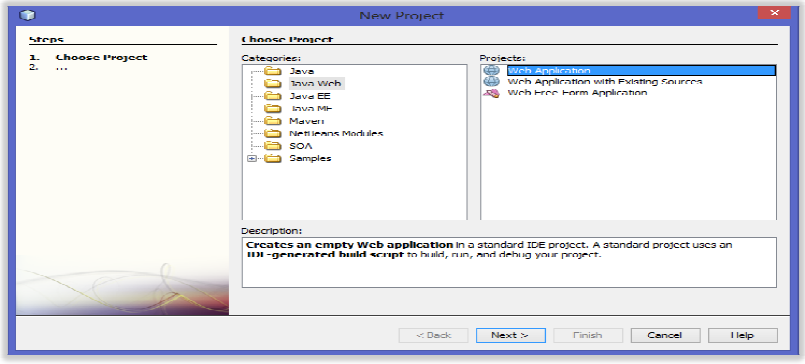



**Figure B.5.** *Assignation des paramètres*

## B. 3 Création et invocation d'un service Web interne (local)

Pour invoquer un service interne (local) à partir d'un processus BPEL, il faut créer un nouveau projet dans la catégorie « Java Web », en sélectionnant le type « Web Application ».




Une fois l'application web créée, il faut créer un nouveau service Web « Nouveau-SW » et une opération qui encapsule les traitements du service.

**Figure B.6.** *Création d'un service Web*

Le déploiement et le test du service Web dans le serveur GlassFich permet de récupérer l'URL du fichier WSDL associé. Pour invoquer ce service dans un processus WF, il suffit d'ajouter un nouveau «External WSDL File», et spécifier l'URL du fichier WSDL, soit : localhost:9080/ServiceWeb/Service1Service?wsdl

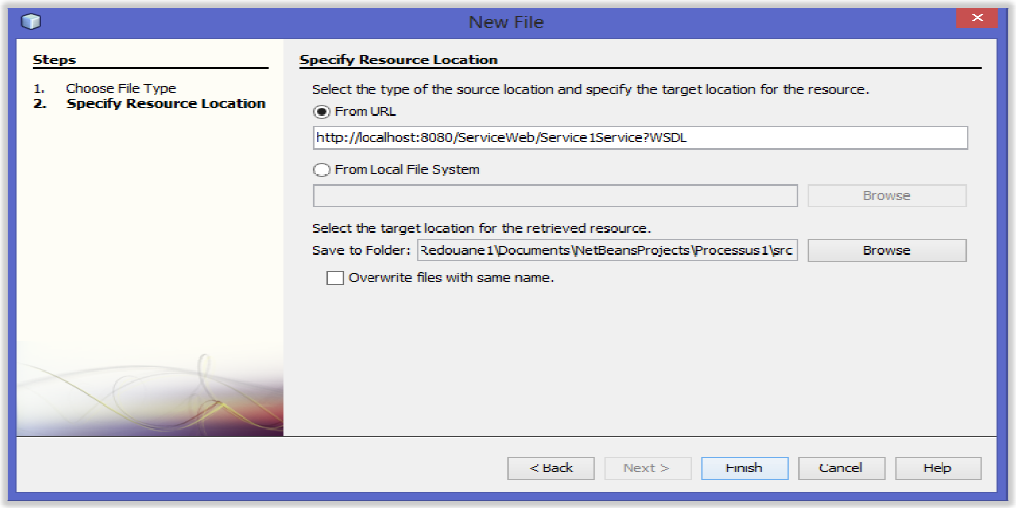



**Figure B.7.** *Création d'un fichier WSDL externe*

Un processus qui invoque un service Web interne (local).

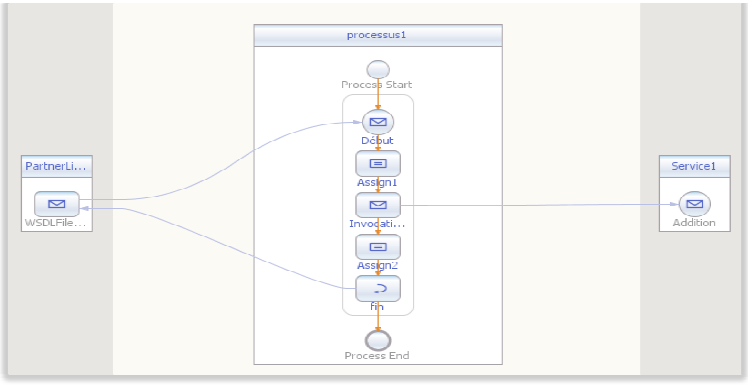



**Figure B.8.** *Invocation d'un service Web*

## B.4 Déployer et tester le processus BPEL

Pour déployer et tester un processus BPEL, il faut créer un nouveau projet dans la catégorie SOA, de type « Application Composite ». L'application composite contiendra le processus BPEL et les différents composants qui communiquent avec ce processus et les liaisons entre eux, dans notre cas on doit ajouter le processus qu'on vient de construire et l'EJB qui contient le service Web invoqué par le processus. Un « Build » permet de construire l'application composite montrée sur la figure B.9. L'application composite peut être déployée et exécutée sur GlassFish. Pour tester l'application composite, il suffit d'ajouter un « **test Case** » puis exécuter ce test.

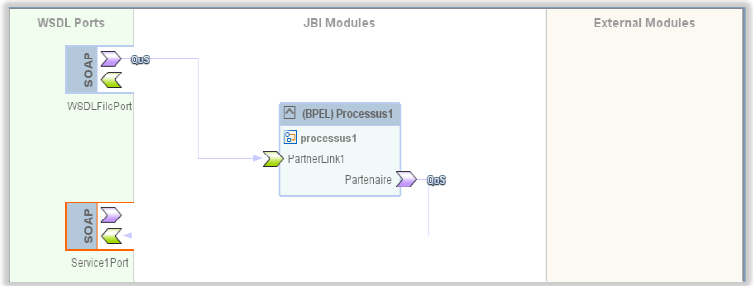



**Figure B.9.** *Application composite*

# Annexe C

# Eléments d'implémentation et Scénarios d'exécution



Cette annexe fournit des précisions sur la réalisation d'une communication synchrone et d'une communication asynchrone entre deux processus BPEL. De plus, elle met en évidence les étapes d'exécution des assistants de coopération « Transfert de cas » et « Faiblement couplé » sur des exemples de processus BPEL.

## C.1 Réalisation d'une communication synchrone entre deux processus BPEL

Dans le cas synchrone (ou communication par rendez-vous), l'envoi d'un message interrompt l'activité du processus émetteur jusqu'à la réception d'un « reply » du processus récepteur. Dans notre framework de coopération, la communication synchrone est utilisée dans les architectures « Partage de charge » et « Sous-traitance ».

Par exemple le patron « Sous-traitance » est réalisé par l'invocation d'un service Web externe depuis un processus BPEL. Le processus BPEL principal doit invoquer un processus BPEL secondaire (considéré comme un service Web composite puisque les points d'interaction encapsulent entièrement le processus secondaire). A travers l'exécution de l'assistant de coopération dédié au patron « Sous-traitance », quatre éléments principaux seront affectés par des modifications : (1) ajout du « *partnerLink* » du processus secondaire, (2) substitution d'un élément « *invoke* » dans le processus principal, (3) assignation des paramètres d'entrée, (4) assignation des paramètres de sortie. Le tableau suivant illustre ces quatre éléments.

**TAB. C.1.** *Exemples de portions de code BPEL illustrant une communication synchrone*

| Code BPEL | Description |
|---|---|
| `<partnerLink xmlns:tns="http://j2ee.netbeans.org/wsdl/Processus2/WSDL2" name="partnerLink_processus2invoke" partnerLinkType="tns:WSDL2" partnerRole="WSDL2PortTypeRole" />` | Le processus secondaire est représenté par un « PartnerLink ». |
| `<assign name="Assign4"> <copy> <from variable="WSDL1OperationIn" part="Message1_In" /> <to variable="WSDL2OperationIn" part="Message2_In" /> </copy> <copy> <from variable="WSDL1OperationIn" part="Déplacement1_In" /> <to variable="WSDL2OperationIn" part="Déplacement" /> </copy> </assign>` | Ce code permet d'assigner les paramètres d'entrée de l'activité sous-traitée avec les paramètres d'entrée du processus secondaire. |
| `<invoke xmlns:tns="http://j2ee.netbeans.org/wsdl/Processus2/WSDL2" name="Invoke_processus2" partnerLink="partnerLink_processus2invoke" operation="WSDL2Operation" portType="tns:WSDL2PortType" inputVariable="WSDL2OperationIn" outputVariable="WSDL2OperationOut" /> <assign name="Assign6">` | L'élément « invoke » permet d'invoquer le partnerLink « partnerLink_processus2invoke » qui représente le processus secondaire. |
| `<assign name="Assign6"> <copy> <from variable="WSDL2OperationOut" part="MessageChiffré" /> <to variable="WSDL1OperationOut" part="Message1_Out" /> </copy> </assign>` | Ce code permet d'assigner les paramètres de sortie du processus secondaire avec les paramètres de sortie de l'activité sous-traitée. |

## C.2 Réalisation d'une Communication Asynchrone entre deux processus BPEL

Dans notre framework de coopération, la communication asynchrone est requise dans les architectures « Exécution chainée », «Transfert de cas » et « Faiblement couplée ».

La forme la plus simple de communication asynchrone est celle utilisée dans « l'Exécution chainée », l'interaction est de type « One-Way » (un seul sens) et le processus à invoquer est entièrement encapsulé dans un service et est donc assimilé à un service Web composite. Dans ce cas, un processus i invoke un processus i+1 dans la séquence. L'exécution de l'assistant de coopération « Exécution chainée » apporte les modifications suivantes dans le processus invocateur : (1) Création d'un nouveau partenaire (PartnerLink) qui représente le processus à invoquer, (2) Ajout d'une activité « invoke » à la fin du processus invocateur et (3) assignation des paramètres de sortie du processus invocateur avec les paramètres d'entrée du processus invoqué. Ces trois éléments sont illustrés dans le tableau suivant :

**TAB. C.2.** *Exemples de portions de code BPEL illustrant une communication asynchrone simple*

| L'élément | Code BPEL de l'élément | Description |
|---|---|---|
| PartnerLink | `<partnerLink xmlns:tns="http://j2ee.netbeans.org/wsdl/Processus2/WSDL2" name="partnerLink_processus2" partnerLinkType="tns:WSDL2" partnerRole="WSDL2PortTypeRole" />` | Un nouveau partenaire qui représente le second processus (processus2) |
| Invoke | `<invoke xmlns:tns="http://j2ee.netbeans.org/wsdl/Processus2/WSDL2" name="Invoke_processus2" partnerLink="partnerLink_processus2" operation="WSDL2Operation" portType="tns:WSDL2PortType" inputVariable="VariableInProcessus2" outputVariable="VariableOutProcessus2" />` | Invoquer le processus2 |
| Assign | `<assign name="newAssignation"> <copy> <from variable="VariableOutProcessus1" part="Nom1_Out" /> <to variable="VariableInProcessus2" part="Nom2_In" /> </copy> </assign>` | Une assignation de paramètres selon ce que l'utilisateur spécifie depuis la boite de dialogue |

Dans les patrons « Transfert de cas » et « Faiblement couplé », la communication entre deux processus est de type asynchrone mais sous une forme plus élaborée que « l'Exécution chainée ». En effet, dans le patron « Transfert de cas », la communication entre deux processus se fait par ajout de blocs alternatifs au niveau des points de transfert (pour l'invocation éventuelle de services externes) et nécessite la création de sous processus à chaque point de transfert. Un sous-processus est lui-même décrit par un fichier BPEL et nécessite la création d'un fichier WSDL pour sa description. Ces éléments seront illustrés à travers un scénario d'exécution de l'assistant de coopération « Transfert de cas » sur un exemple.

## C.3 Scénario d'exécution de l'assistant « Transfert de cas » sur un exemple

Pour illustrer les étapes d'exécution de l'assistant de coopération « Transfert de cas », nous considérons un processus de WFIO impliquant deux organismes SAIDAL et Generic-Lab producteurs et fournisseurs de médicaments. Le processus en question concerne la commercialisation de médicaments à des clients potentiels et peut être implémenté selon l'architecture « Transfert de cas » avec une politique de transfert très simple. En effet, l'unité commerciale de SAIDAL possède un partenariat stratégique avec Generic-Lab lui permettant de transférer le traitement de certaines commandes clients à Generic-Lab. Le transfert de commandes se fait dans le cas d'indisponibilité de la licence de production au niveau de SAIDAL, concernant certains médicaments qui peuvent être fournis par Generic-Lab.
Ainsi, toutes les commandes des clients arrivent au niveau de l'unité commerciale (UC) de SAIDAL mais certaines d'entre-elles sont transférées à Generic-Lab qui doit les prendre en charge par le biais de son WF visiblement identique à celui de l'UC de SAIDAL.

Le diagramme d'activité de la figure C.1 décrit globalement, le processus inter-organisationnel reliant l'UC de SAIDAL et Generic-Lab, à travers l'invocation d'un *service Web externe* « Traiter BCT» (traiter bon de commande transféré).

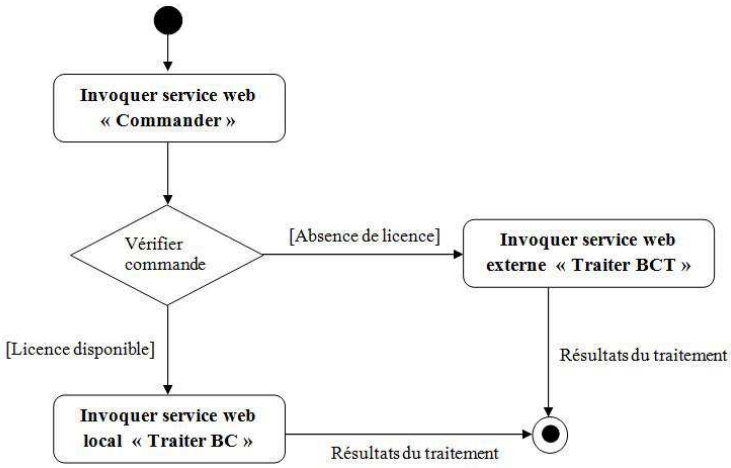



**Figure C.1.** *Exemple d'un processus obéissant au patron « Transfert de cas »*

Un client fait une commande via le site web de SAIDAL ; deux cas sont possibles : (1) la commande concerne des médicaments produits par SAIDAL, elle est traitée au niveau de SAIDAL à travers l'invocation du *service local* « Traiter BC ». (2) La commande concerne des médicaments produits par Generic-Lab, elle est aussitôt transférée pour traitement au niveau de Generic-Lab, via le *service externe* « Traiter BCT ». Le processus repose sur une politique de transfert très simple puisqu'il comporte un seul point de transfert. Toutes les activités sont encapsulées dans un seul service.

Les services « Traiter BC » et « Traiter BCT » encapsulent le même sous-processus implémenté au niveau de SAIDAL et de Generic-Lab respectivement. La figure C.2 donne une description de ce sous-processus qui englobe un ensemble d'activités semi-automatiques et automatiques. A noter que les activités manuelles ne sont pas sous le contrôle du SGWF et peuvent être simulées à travers des signaux de validation introduits par les acteurs concernés.

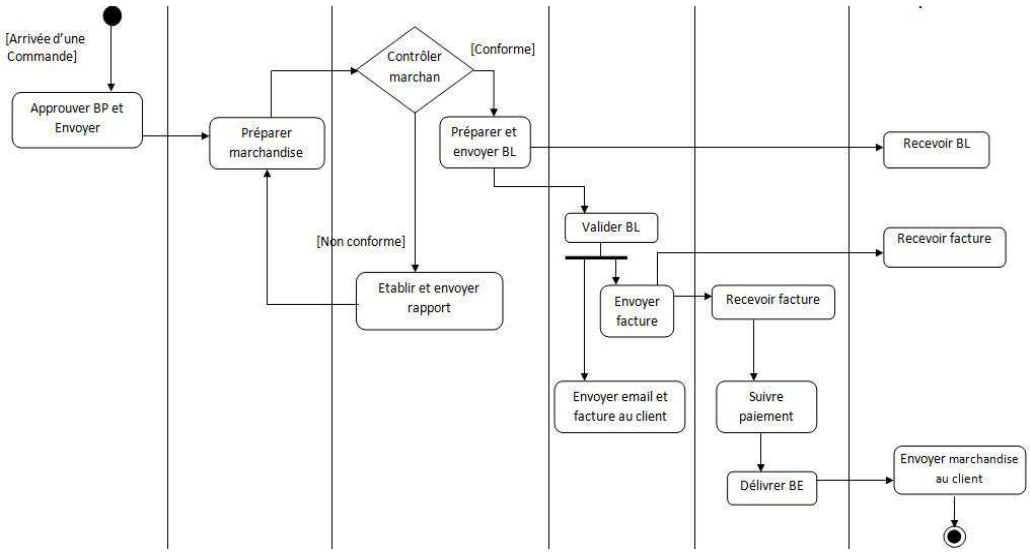



**Figure C.2.** *Détail du sous-processus « Traiter BC »*

L'assistant de coopération « Transfert de cas » permet l'interconnexion entre deux (ou plus) processus BPEL et opère en cinq étapes :

**Etape 1 « Sélectionner l'architecture »**: le concepteur de WF sélectionne le patron de coopération à réaliser, dans l'interface principale du framework de coopération, comme le montre la figure C.3.

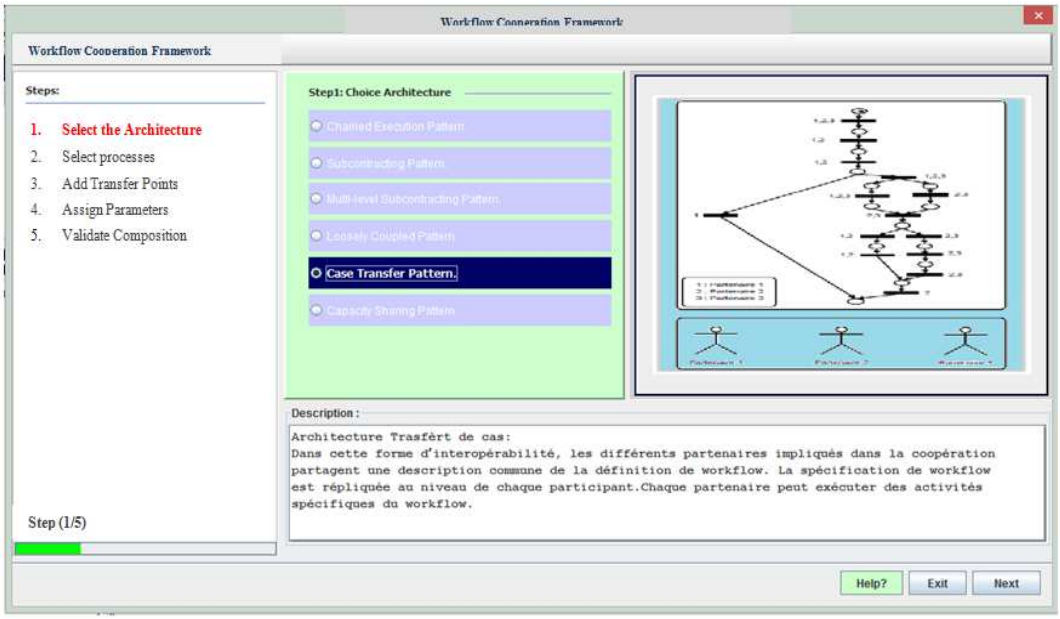



**Figure C.3.** *Interface de sélection du patron « Transfert de cas »*

**Etape 2 « Sélectionner les processus »** : une autre interface apparait afin de permettre au concepteur de WF de sélectionner les processus à interconnecter selon le patron « Transfert de cas » choisi. Lorsqu'un processus BPEL est sélectionné, une copie de son fichier BPEL est automatiquement créée sur l'infrastructure commune.

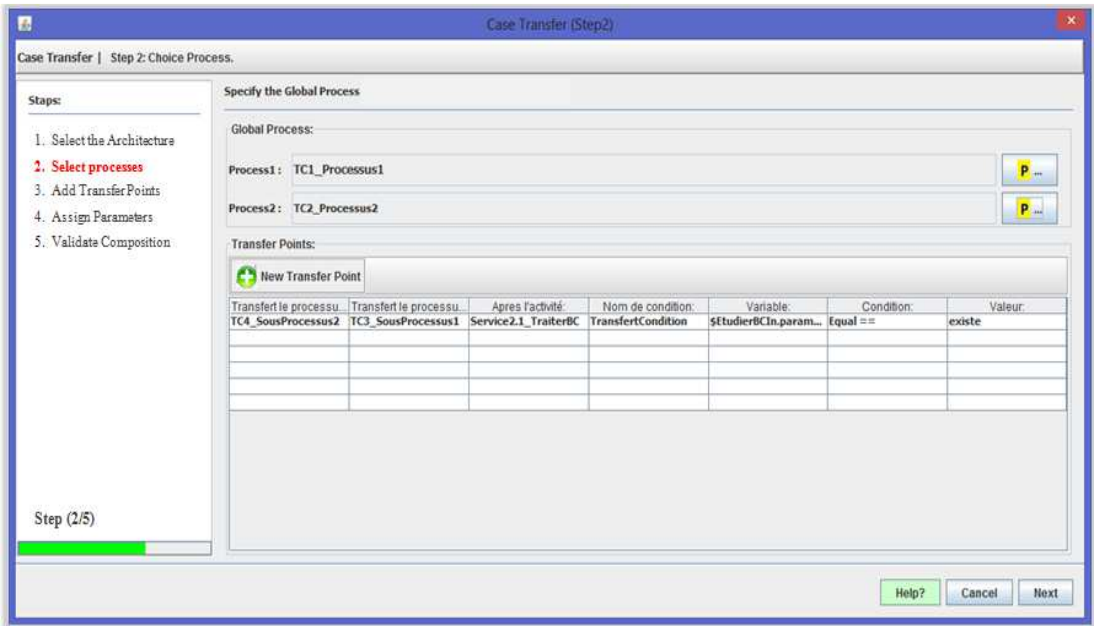



**Figure C.4.** *Interface de sélection des processus pour le « Transfert de cas »*

**Etape 3 « Insérer les points de transfert »** : sur la même interface affichée à l'étape 2, on retrouve un bouton « New Transfer Point » qui doit être activé à chaque fois que l'on veuille ajouter un point de transfert dans le processus. Initialement la table qui affiche les points de transfert est vide, elle est mise à jour à chaque nouvelle insertion de point de transfert. Pour ajouter un point de transfert, on doit spécifier sa localisation dans le modèle de processus (avant ou après quelle

invocation de service), la condition de transfert et l'action de transfert. Le modèle de processus est ensuite mis à jour par l'ajout d'une activité alternative (if) pour invoquer un service local ou un service externe selon la valeur de vérité de la condition de transfert. Le service (local ou externe) à invoquer encapsule un sous-processus qui commence au point de transfert et qui est spécifié en utilisant une interface spécifique fournie par l'assistant de coopération.

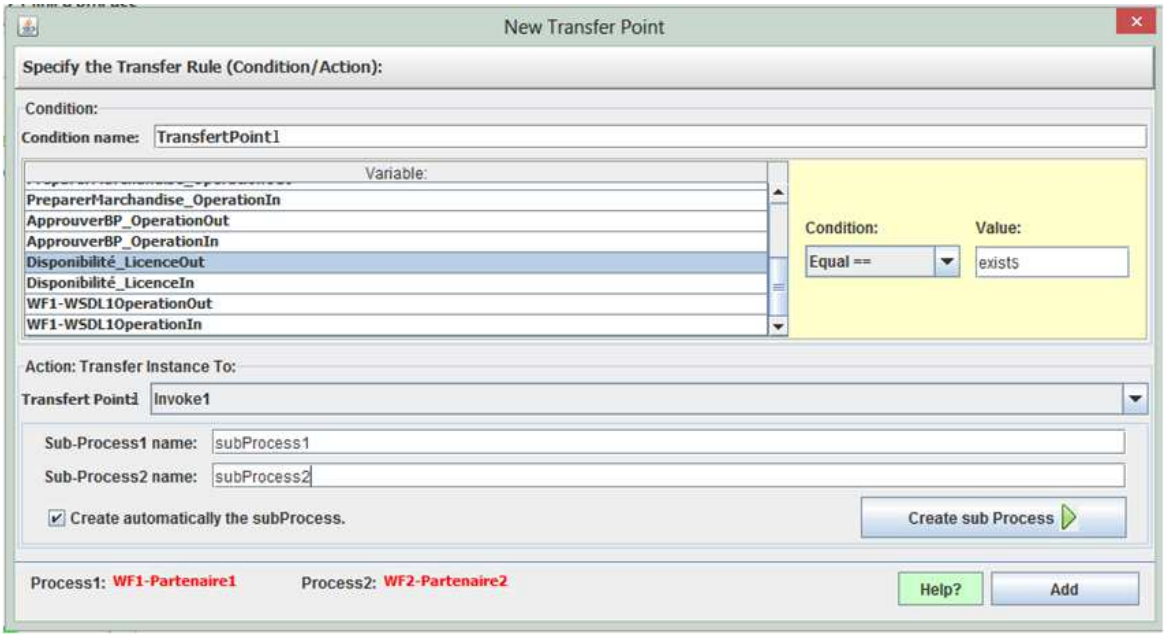



**Figure C.5.** *Interface d'ajout de points de transfert*

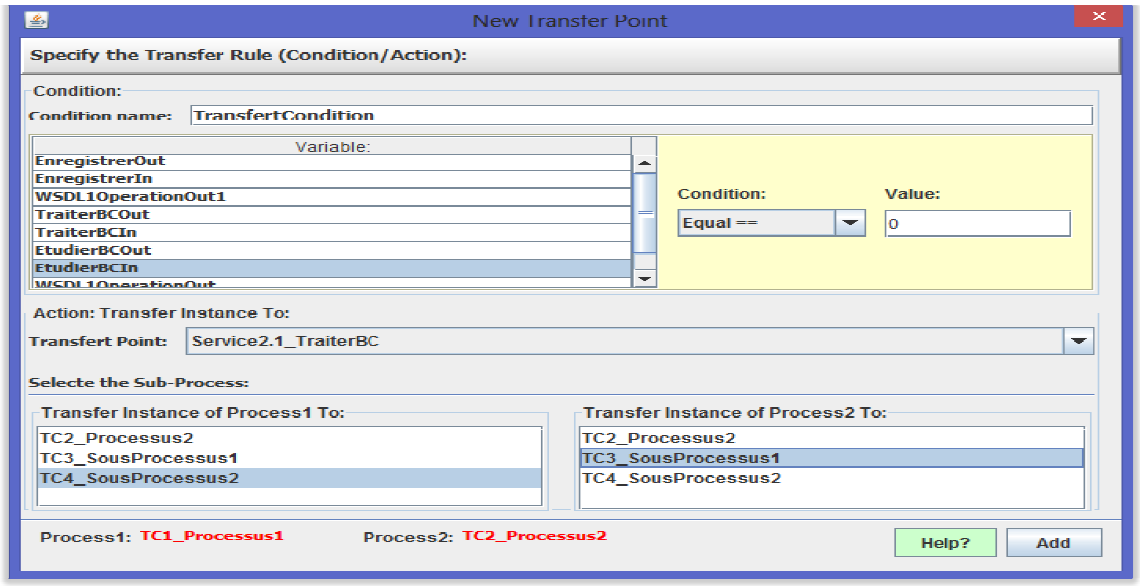



**Figure C.6.** *Spécification de la condition de transfert*

**Etape 4 « Assigner les paramètres »** : après la définition d'un point de transfert, une assignation de paramètres est effectuée d'une manière semi-automatique afin de maintenir un flux de données cohérent entre dans le processus global : les variables d'entrée du service invoqué doivent être les variables de sortie du service qui se situe avant le point de transfert dans le processus invocateur.

**Etape 5 « Valider l'interconnexion »** : dès que le concepteur de WF valide l'interconnexion, les modifications effectuées sur les copies des fichiers BPEL des processus interconnectés sont

automatiquement répercutées (enregistrées) sur les processus BPEL d'origine au niveau de leurs partenaires respectifs. Un rapport de l'opération d'interconnexion est affiché au concepteur de WF, l'interface correspondant à cette étape permet de visualiser différentes vues des processus : une vue « code », une vue « graphique » et une vue « détail ». La figure suivante est un zoom sur le point de transfert

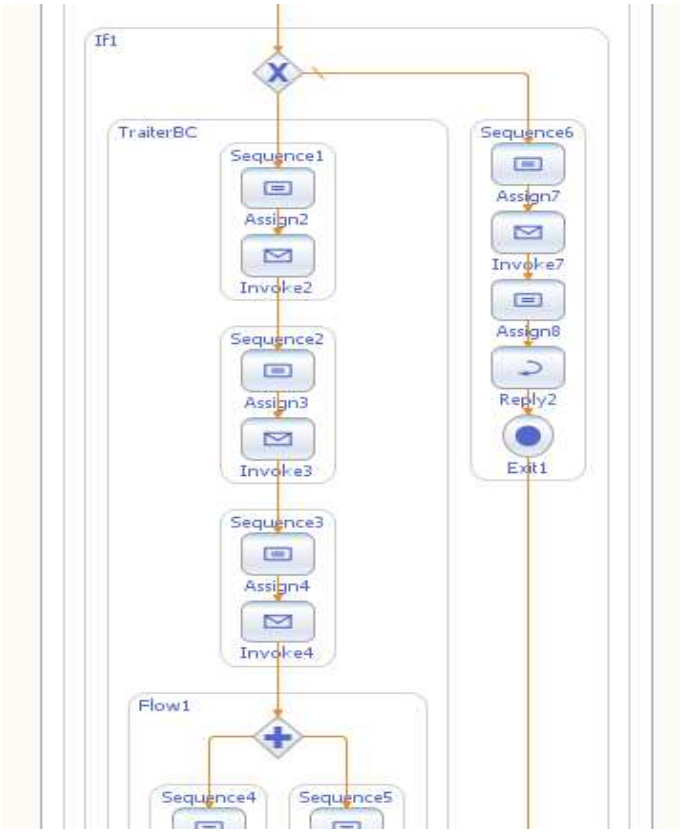



**Figure C.7.** *Zoom sur le point de transfert*

Le tableau suivant résume les modifications (les activités ajoutées dans le fichier BPEL du partenaire 1) suite à l'opération d'interconnexion.

**TAB. C.3.** *Portions de code BPEL illustrant une communication asynchrone dans le « Transfert de cas »*

| Activité IF (exprime la condition –disponibilité de la licence-) |
|---|
| `<if name="If1">`<br>`<condition>'existe' = $Disponibilité_LicenceOut.parameters/return</condition>` |
| **Activité Assign7- assignation des paramètres d'entrée du service externe** |
| `<assign name="Assign7">`<br>`<copy>`<br>`<from variable="WF1-WSDL1OperationIn" part="Reference"/>`<br>`<to variable="WSDL-SP1OperationIn" part="reference"/>`<br>`</copy>`<br>`<copy>`<br>`<from variable="WF1-WSDL1OperationIn" part="Qte"/>`<br>`<to variable="WSDL-SP1OperationIn" part="Qte"/>`<br>`</copy>`<br>`</assign>` |

| Invoke7- invocation du sous-processus (service externe) |
|---|
| `<invoke name="Invoke7" partnerLink="Sub-process" operation="WSDL-SP1Operation" xmlns:tns="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1" portType="tns:WSDL-SP1PortType" inputVariable="WSDL-SP1OperationIn" outputVariable="WSDL-SP1OperationOut"/>` |
| Assign 8 – assignation des paramètres de sortie du service externe |
| `<assign name="Assign8"> <copy> <from variable="WSDL-SP1OperationOut" part="part1"/> <to variable="WF1-WSDL1OperationOut" part="Rapport"/> </copy> </assign>` |
| Replay2 – retour du processus1 |
| `<reply name="Reply2" partnerLink="Partenaire1" operation="WF1-WSDL1Operation" xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1" portType="tns:WF1-WSDL1PortType" variable="WF1-WSDL1OperationOut1"/>` |
| Exit1-fin du processus1 |
| `<exit name="Exit1"/>` |

Dans l'annexe D, se trouve le code BPEL du processus complet après interconnexion avec le second processus. Les parties insérées dans le code suite à l'exécution de l'assistant de coopération sont surlignées.

## C.4 Réalisation du patron « Faiblement couplé »

### C.4.1 Les services d'interaction dans le patron « Faiblement couplé »

Le patron « Faiblement couplé » repose sur l'utilisation de services d'interaction « Service_Send » et « Service_Receive » permettant respectivement, l'envoi et la réception de messages.

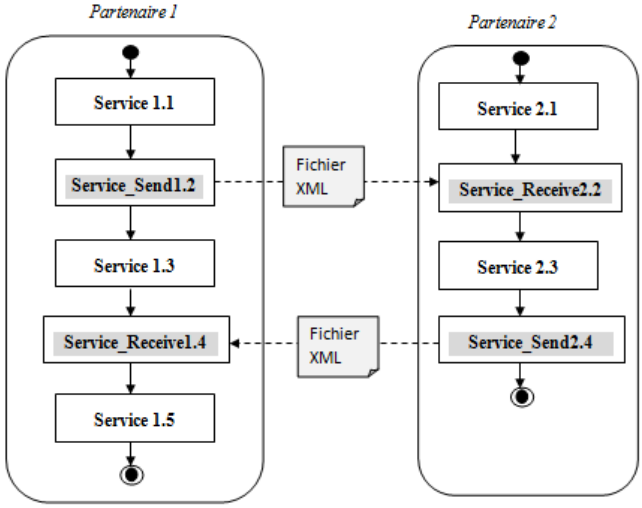



**Figure C.8.** *Schéma d'interaction asynchrone entre deux processus BPEL dans le patron « Faiblement couplé »*

A chaque fois que le concepteur WF veut établir une communication entre deux processus, il doit ajouter un point d'interaction entre eux, chaque point d'interaction est spécifié par trois composants : le service Web d'envoi « *Service_Send* », le service Web de réception « *Service_Receive* » et le fichier XML temporaire contenant le message envoyé.

- **Le service Web « Service_Send»**

Le service Web « *Service_Send*» permet d'enregistrer les messages envoyés par un processus dans un fichier XML, le code source suivant présente un exemple de service Web « *Send_ Service* », ce code permet d'enregistrer un message composé de deux paramètres (libelle produit et la quantité).

```
@WebService()
public class ServiceWebSend1 {
    @WebMethod(operationName = "Send1")
    public String Send1(@WebParam(name = "LibelleProduit")
    String LibelleProduit, @WebParam(name = "Qte")
    int Qte) {
        //TODO write your implementation code here:
        File file = new File("C:/EssieFaiblementCouplee/data/data.xml");
        Document document;
        Element racine=null;
        try{
            //récupération et déclaration des attributs:
            SAXBuilder builder = new SAXBuilder();
            document = builder.build(file);
            racine = document.getRootElement();
            //on va supprimer toutes les données.
            racine.removeContent();

            //associe les données:
            Element element1 = new Element("Produit");
            element1.setText(""+LibelleProduit);
            Element element2 = new Element("Qte");
            element2.setText(""+Qte);

            racine.addContent(element1);
            racine.addContent(element2);

            //enregistrement de fichier
            XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
            sortie.output(document,new FileOutputStream(file));
```

**Figure C.9.** *Code java d'un service Web « Service_Send »*

- **Le service Web « Service_Receive»**

Le service Web « Service_Receive » permet de récupérer le message envoyé par le processus émetteur, le code source ci-dessous présente un exemple d'un tel service.

```
@WebService()
public class ServiceWebPointDeCommunicationResponse2 {
    @WebMethod(operationName = "Response2")
    public data Response2() {
          File file = new File("C:/EssieFaiblementCouplee/data/data.xml");
          Document document=null;
          Element racine=null;
          data d = new data();
          try{
             //récupération et déclaration des attributs:
             List listeFils= new ArrayList();
             TimeUnit.SECONDS.sleep(6);
             while(listeFils.size() == 0){
                  SAXBuilder builder = new SAXBuilder();
                  document = builder.build(file);
                  racine = document.getRootElement();
                  listeFils =  racine.getChildren();

                  Iterator it_listeFils = listeFils.iterator();

                  Element nomProduit = (Element) it_listeFils.next();
                  String produit =  nomProduit.getText();
                  d.setLibelleProduit(produit);

                  Element qteProduit = (Element) it_listeFils.next();
                  String d2 = qteProduit.getText();
                  d.setQte(d2);
                  TimeUnit.SECONDS.sleep(5);
             }
```

**Figure C.10.** *Code java d'un service Web «  Receive Service »*

On distingue deux types d'interaction entre les processus (à travers l'invocation de  services ): une interaction « One-Way » et une interaction « Two-Way »

- **Interaction "One-Way"  et Interaction "Two-way"**

L'interaction One-Way signifie l'invocation d'un service Web ou d'un processus sans résultat de retour. L'interaction est de type « Two-way » si l'invocation d'un service lui correspond une autre invocation dans le sens inverse.

Pour spécifier qu'un partenaire soit « One-Way » ou « Two-way », il suffit de choisir le type d'opération : « One-Way » ou « Two-way » lors de la création du fichier WSDL.

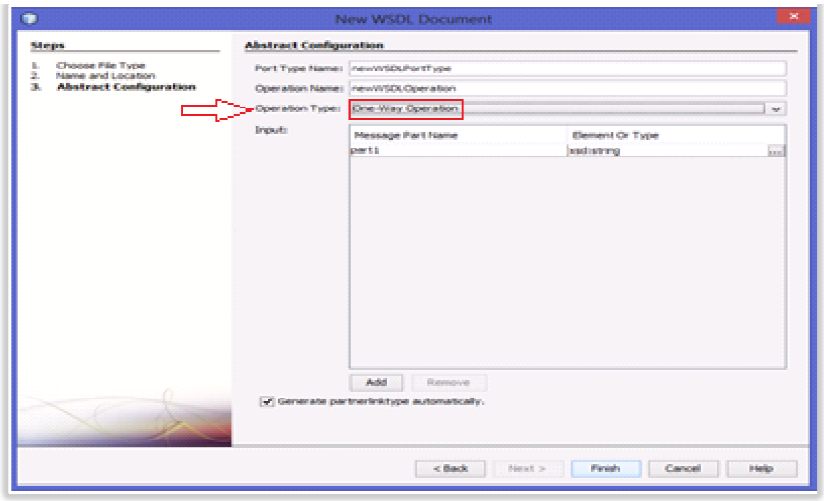



**Figure C.11.** *Spécifier le type d'interaction*

Après la création du fichier WSDL, on peut observer que le type d'interaction et One-Way.

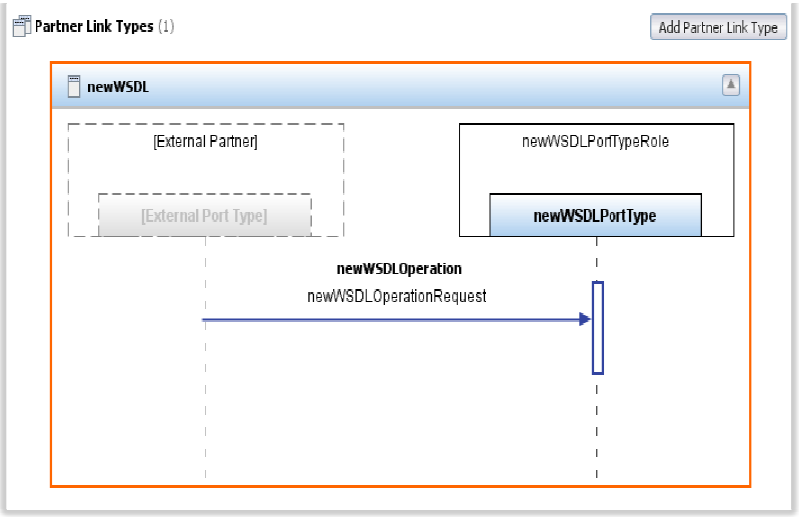



**Figure C.12.** *Schéma de l'interaction One-Way*

### C.4.2 Scénario d'exécution de l'assistant « Faiblement couplé »

Pour illustrer les étapes d'exécution de l'assistant « Faiblement couplé », on considère l'exemple de deux processus, un processus client « Customer » et un processus fournisseur « Supplier » qui doivent interagir entre eux pour l'échange des données de manière asynchrone.
Comme pour l'assistant « Transfert de cas », l'assistant « Faiblement couplé » s'exécute en cinq étapes illustrées par les captures d'écran suivantes.

**<u>Etape 1</u> : choix de l'architecture**

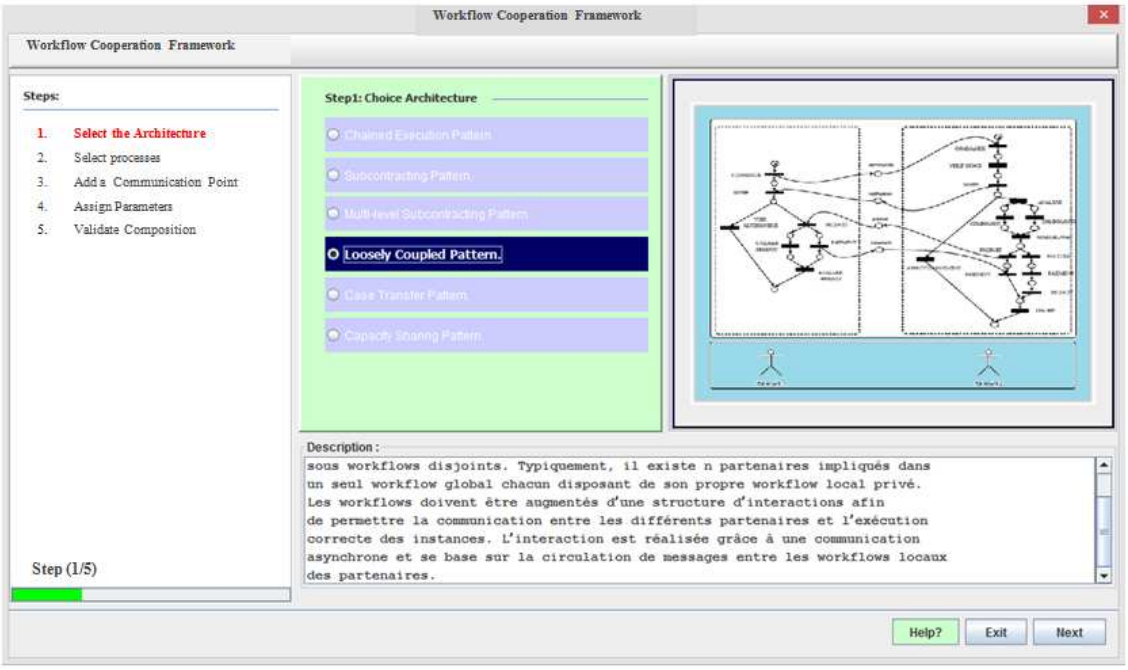



**Figure C.13.** *Interface de sélection du patron « Faiblement couplé »*

**<u>Etape 2</u> : Sélection des processus et spécification des services d'interaction**

L'interface suivante permet de spécifier les services d'interaction entre deux processus BPEL préalablement sélectionnés. En effet, il s'agit de préciser les fichiers WSDL et XSD associés à chaque service d'interaction (« Service-send » ou « Service-receive »).

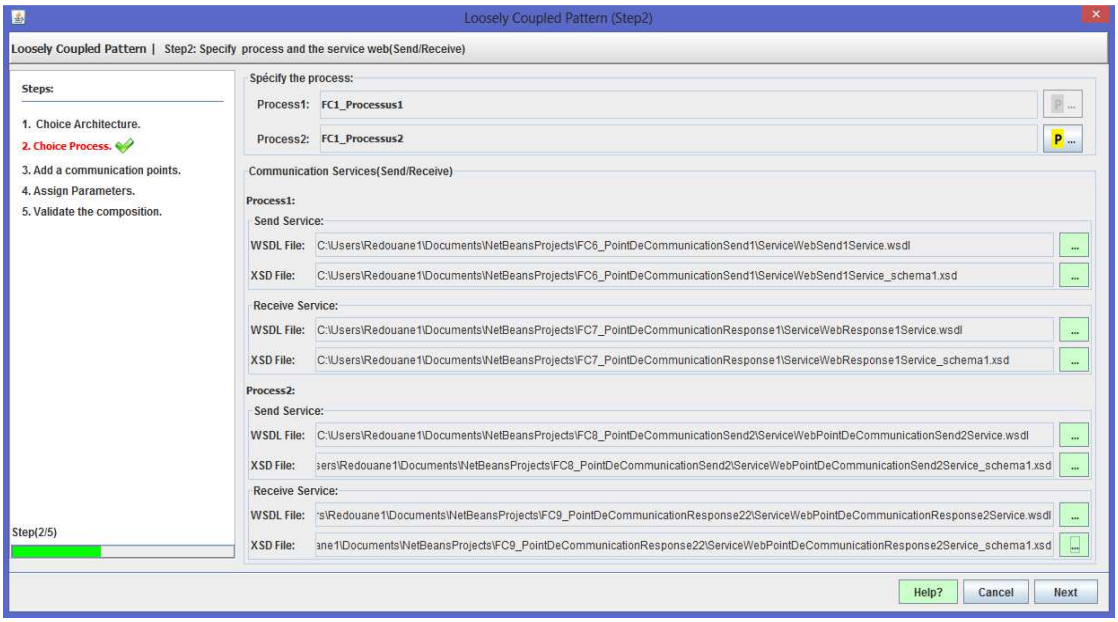



**Figure C.14.** *Interface de sélection des processus dans le patron « Faiblement couplé »*

**<u>Etape 3</u> : Ajout des points d'interaction (ou de communication)**

Cette interface permet d'insérer les services d'interaction préalablement sélectionnés au niveau des processus à interconnecter, en spécifiant les services de référence qui indiquent l'emplacement des points d'interaction dans les deux processus.

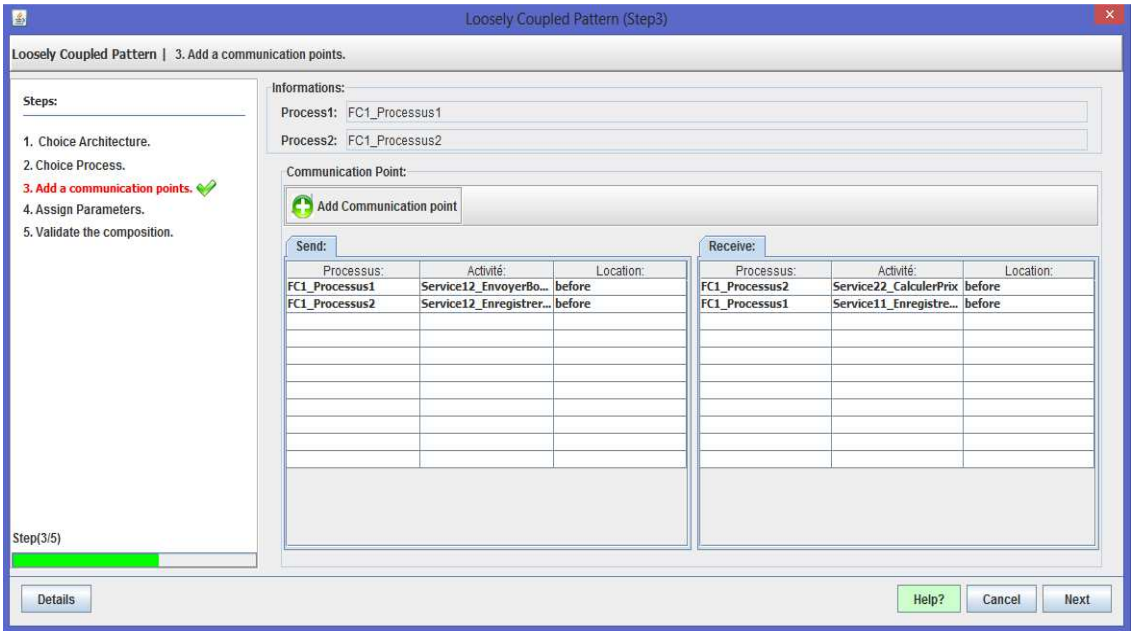



**Figure C.15.** *Interface d'ajout des points d'interaction dans le patron « Faiblement couplé »*

**<u>Etape 4</u> : Assignation des variables**

La figure suivante illustre l'étape d'assignation des variables entre le processus émetteur « Service_ Send » et le processus récepteur « Service_ Receive »

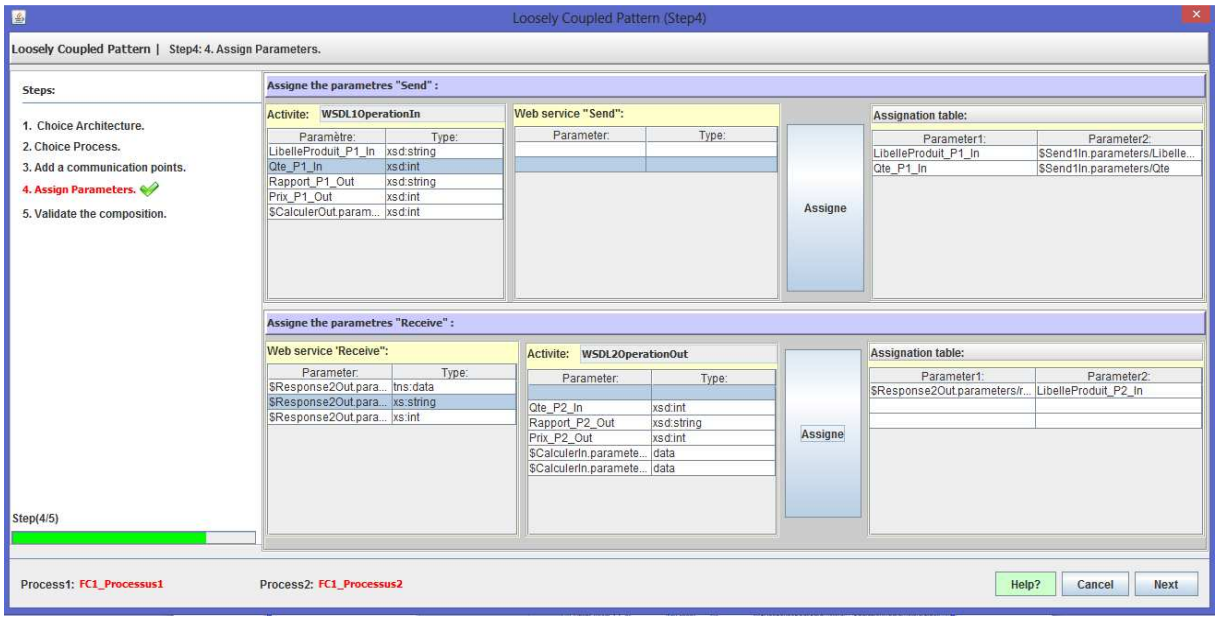



**Figure C.16.** *Interface d'assignation des paramètres dans le patron « Faiblement couplé »*

**Etape 5 « Valider l'interconnexion »** : La dernière étape de l'assistant consiste à valider l'interconnexion entre les processus. De ce fait, les modifications sont répercutées sur les fichiers BPEL des deux partenaires.

Dans la dernière partie de cette annexe, nous donnons le code BPEL complet des deux processus avant et après interconnexion.

## C.5 Test du processus

Le test du processus de WFIO obtenu nécessite la génération d'applications composites associées à chaque processus impliqué dans la coopération, la figure suivante montre l'application composite associée au processus1 de notre exemple.

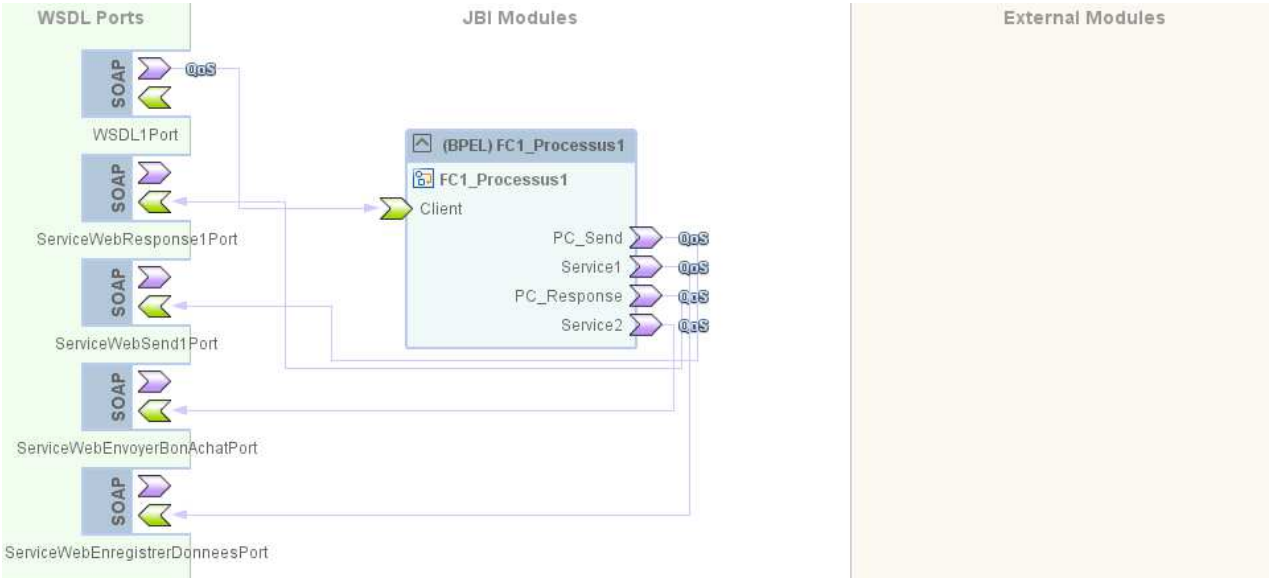



**Figure C.17.** *Schéma de l'application composite du processus1 (Customer)*

## C.6 Quelques illustrations d'adaptations

- **Ajout de service**

Pour montrer les différentes modifications apportées par une opération d'ajout de service sur un processus, nous avons construit un processus de test dans lequel nous ajoutons un service nommé « newService ». Ainsi, la représentation de ce processus sera comme suit :

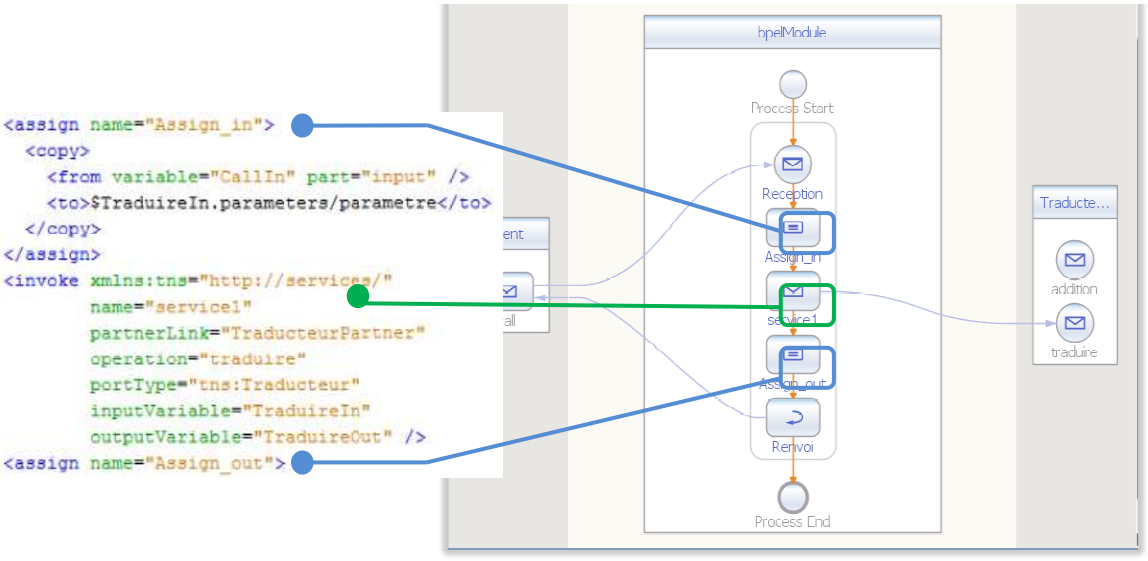
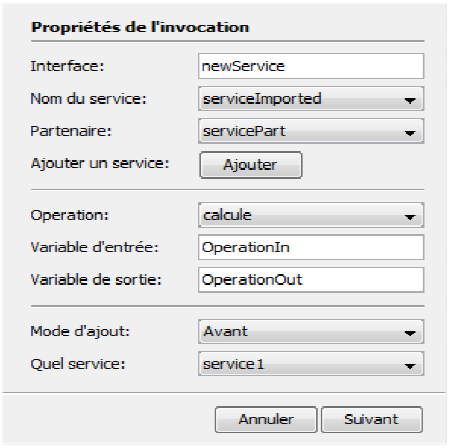



**Figure C.18**. *Schéma et code du processus avant l'ajout*

La figure C.19 montre l'interface fournie à l'utilisateur de notre framework d'adaptation/évolution, pour ajouter un service dans un fragment de processus.




L'utilisateur doit introduire les informations nécessaires pour une opération d'ajout de service telles que : le nom du service (interface), le partenaire qui propose ce service, l'opération et ses variables d'entrée et de sortie ainsi que le mode d'ajout de ce service (avant, après, en alternatif ou en parallèle).

**Figure C.19.** *Interface graphique de l'opération d'ajout*

- **Ajout dans un bloc alternatif**

Si l'on choisit d'insérer le nouveau service « newService » en exclusif avec le service « service1 », on aura le résultat suivant une vue code et une vue graphique du processus.

```
<if name="If_service1_service2">
    <invoke xmlns:tns="http://services/"
            name="service1"
            partnerLink="TraducteurPartner"
            operation="traduire"
            portType="tns:Traducteur"
            inputVariable="in"
            outputVariable="out" />
    <else>
        <invoke name="newService"
        partnerLink="ScripteurPartner"
        operation="operation"
        portType="ns0:testWebService"
        inputVariable="OperationIn"
        outputVariable="OperationOut"/>
    </else>
</if>
```

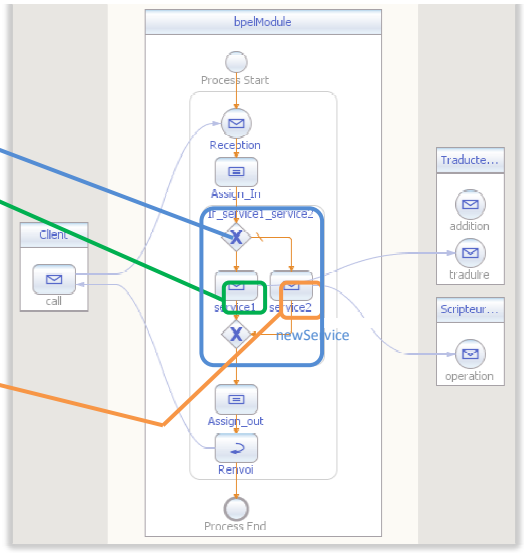
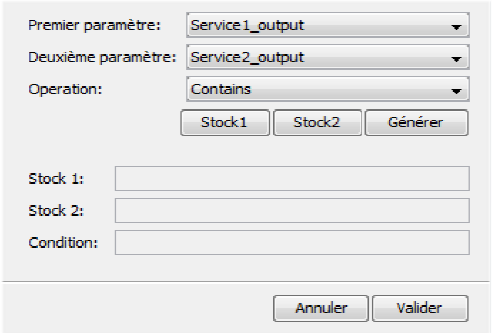



**Figure C.20** – *Schéma et code du processus après l'ajout du service2 en alternatif avec le service1.*

Une fois que l'utilisateur choisit d'ajouter un service pour créer un bloc exclusif, l'interface graphique de génération de condition s'affiche, cette interface permet à l'utilisateur de générer une condition, cette condition sera insérée dans la balise « condition » du bloc exclusif qui contient les deux services.




Pour créer une condition composée, l'utilisateur doit stocker ses conditions simples dans les Stock1 et Stock2 pour les réutiliser en les sélectionnant dans l'une des listes déroulantes, une fois la condition est complète, il doit la générer puis valider ses modifications.

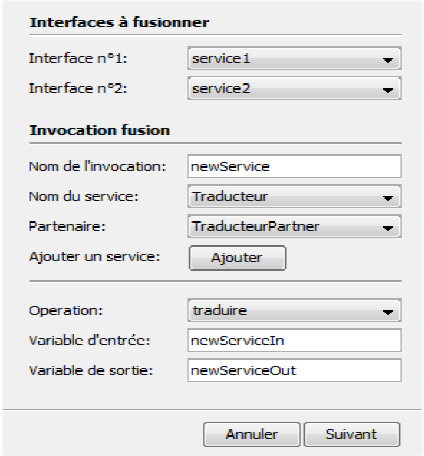
**Figure C.21.** *Interface graphique de création des conditions*

- **Fusion de services**

Dans une opération de fusion de services, il s'agira de spécifier les deux services objet de la fusion et le service résultat de la fusion. La figure C.22 montre le code de la méthode permettant de faire une opération de fusion et la figure C.23 montre l'interface de notre framework d'adaptation/évolution s'affichant eu concepteur de WF pour introduire les différents paramètres nécessaires à l'opération de fusion.

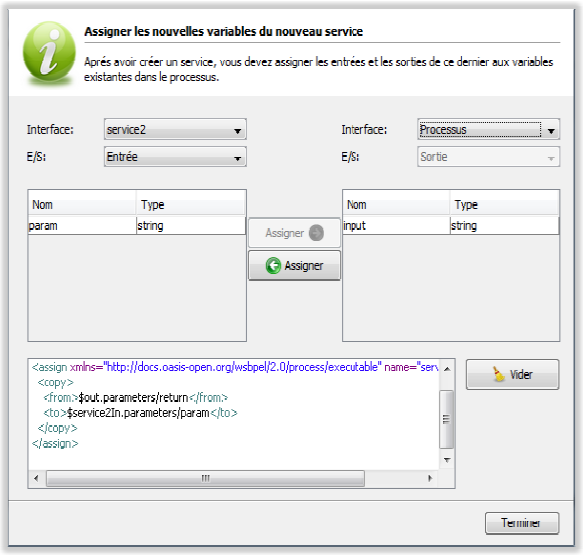
```
public static void fusion(BpelFile bpel, Element firstService, Element secondService, Element newService) {
    Element parentElement = firstService.getParentElement();
    int size = parentElement.getChildren().size();
    Inserer insert = new InsererAvant();
    if (size > 2) {
        insert.inserer(firstService, newService, null);
        Supprimer.supprimer(bpel, "invoke", firstService.getAttributeValue("name"));
        Supprimer.supprimer(bpel, "invoke", secondService.getAttributeValue("name"));
    } else if (size == 2) {
            insert.inserer(parentElement, newService, null);
            Supprimer.supprimer(bpel, parentElement.getName(), parentElement.getAttributeValue("name"));
    }
}
```

**Figure C.22.** *Code du patron de fusion de services dans un bloc parallèle*




Le concepteur de workflow doit sélectionner d'abord les deux services à fusionner, puis introduire les informations nécessaires pour le nouveau service qui est le résultat de la fusion.

**Figure C.23**. *Interface graphique de l'opération de fusion.*




Pour l'insertion des balises « assign » dans le code du processus afin de mettre à jour le flux de données entre services, l'application offre une interface au concepteur permettant la mise à jour des données d'entrée/sortie aux services. A chaque fois qu'un service est sélectionné, ses variables d'E/S sont affichées dans l'interface.

**Figure C.24.** *Interface Graphique de l'assignation des variables*

# Annexe D

# Code BPEL du Processus Exemple

**Code du processus BPEL « Traiter commandes » de l'assistant « Transfert de cas » après interconnexion**

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="WF1-Partenaire1"
    targetNamespace ="http://enterprise.netbeans.org/bpel/WF1-Partenaire1/WF1-Partenaire1"

    xmlns:tns="http://enterprise.netbeans.org/bpel/WF1-Partenaire1/WF1-Partenaire1"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
    xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"

xmlns:sxeh="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/ErrorHandling"
xmlns:sxed2="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor2">
    <import namespace="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1"
location="WF1-WSDL1.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SW_ApprouverBPServiceWrapper"
location="SW_ApprouverBPServiceWrapper.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/"
location="http://localhost:8080/ApprouverBP/SW_ApprouverBPService?WSDL"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SW_CommanderServiceWrapper"
location="SW_CommanderServiceWrapper.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/"
location="http://localhost:8080/Commander/SW_CommanderService?WSDL"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import
namespace="http://enterprise.netbeans.org/bpel/SW_PreparerMarchandiseServiceWrapper"
location="SW_PreparerMarchandiseServiceWrapper.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/"
location="http://localhost:8080/PreparerMarchandise/SW_PreparerMarchandiseService?WSDL"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SW_ValiderBLServiceWrapper"
location="SW_ValiderBLServiceWrapper.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/"
location="http://localhost:8080/ValiderBL/SW_ValiderBLService?WSDL"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import
namespace="http://enterprise.netbeans.org/bpel/SW_EnvoyerFactureServiceWrapper"
location="SW_EnvoyerFactureServiceWrapper.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/"
location="http://localhost:8080/EnvoyerFacture/SW_EnvoyerFactureService?WSDL"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SW_EnvoyerEMailServiceWrapper"
location="SW_EnvoyerEMailServiceWrapper.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://SW/" location="http://localhost:8080/EnvoyerE-
Mail/SW_EnvoyerEMailService?WSDL" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1"
 location="SubProcess1/WSDL-SP1.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="Sub-process"
xmlns:tns="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1" partnerLinkType="tns:WSDL-
SP1" partnerRole="WSDL-SP1PortTypeRole"/>
        <partnerLink name="Service1"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_CommanderServiceWrapper"
partnerLinkType="tns:SW_CommanderLinkType" partnerRole="SW_CommanderRole"/>
```

```
        <partnerLink name="Service2"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_ApprouverBPServiceWrapper"
partnerLinkType="tns:SW_ApprouverBPLinkType" partnerRole="SW_ApprouverBPRole"/>
        <partnerLink name="Service3"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_PreparerMarchandiseServiceWrapper"
partnerLinkType="tns:SW_PreparerMarchandiseLinkType"
partnerRole="SW_PreparerMarchandiseRole"/>
        <partnerLink name="Service4"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_ValiderBLServiceWrapper"
partnerLinkType="tns:SW_ValiderBLLinkType" partnerRole="SW_ValiderBLRole"/>
        <partnerLink name="Service5"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_EnvoyerFactureServiceWrapper"
partnerLinkType="tns:SW_EnvoyerFactureLinkType" partnerRole="SW_EnvoyerFactureRole"/>
        <partnerLink name="Service6"
xmlns:tns="http://enterprise.netbeans.org/bpel/SW_EnvoyerEMailServiceWrapper"
partnerLinkType="tns:SW_EnvoyerEMailLinkType" partnerRole="SW_EnvoyerEMailRole"/>
        <partnerLink name="Partenaire1" xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-
Partenaire1/WF1-WSDL1" partnerLinkType="tns:WF1-WSDL1" myRole="WF1-WSDL1PortTypeRole"/>
    </partnerLinks>
    <variables>
        <variable name="WF1-WSDL1OperationOut1"
xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1" messageType="tns:WF1-
WSDL1OperationResponse"/>
        <variable name="WSDL-SP1OperationOut"
xmlns:tns="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1" messageType="tns:WSDL-
SP1OperationResponse"/>
        <variable name="WSDL-SP1OperationIn"
xmlns:tns="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1" messageType="tns:WSDL-
SP1OperationRequest"/>
        <variable name="EnvoyerEMail_OperationOut" xmlns:tns="http://SW/"
messageType="tns:EnvoyerEMail_OperationResponse"/>
        <variable name="EnvoyerEMail_OperationIn" xmlns:tns="http://SW/"
messageType="tns:EnvoyerEMail_Operation"/>
        <variable name="EnvoyerFacture_OperationOut" xmlns:tns="http://SW/"
messageType="tns:EnvoyerFacture_OperationResponse"/>
        <variable name="EnvoyerFacture_OperationIn" xmlns:tns="http://SW/"
messageType="tns:EnvoyerFacture_Operation"/>
        <variable name="ValiderBL_OperationOut" xmlns:tns="http://SW/"
messageType="tns:ValiderBL_OperationResponse"/>
        <variable name="ValiderBL_OperationIn" xmlns:tns="http://SW/"
messageType="tns:ValiderBL_Operation"/>
        <variable name="PreparerMarchandise_OperationOut" xmlns:tns="http://SW/"
messageType="tns:PreparerMarchandise_OperationResponse"/>
        <variable name="PreparerMarchandise_OperationIn" xmlns:tns="http://SW/"
messageType="tns:PreparerMarchandise_Operation"/>
        <variable name="ApprouverBP_OperationOut" xmlns:tns="http://SW/"
messageType="tns:ApprouverBP_OperationResponse"/>
        <variable name="ApprouverBP_OperationIn" xmlns:tns="http://SW/"
messageType="tns:ApprouverBP_Operation"/>
        <variable name="Disponibilité_LicenceOut" xmlns:tns="http://SW/"
messageType="tns:CommaberResponse"/>
        <variable name="Disponibilité_LicenceIn" xmlns:tns="http://SW/"
messageType="tns:Commander"/>
        <variable name="WF1-WSDL1OperationOut"
xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1" messageType="tns:WF1-
WSDL1OperationResponse"/>
        <variable name="WF1-WSDL1OperationIn"
xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1" messageType="tns:WF1-
WSDL1OperationRequest"/>
    </variables>
    <sequence>
        <receive name="Receive1" createInstance="yes" partnerLink="Partenaire1"
operation="WF1-WSDL1Operation" xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-
Partenaire1/WF1-WSDL1" portType="tns:WF1-WSDL1PortType" variable="WF1-WSDL1OperationIn"/>
        <sequence name="Traité_Licence">
            <assign name="Assign1">
                <copy>
                        <from variable="WF1-WSDL1OperationIn" part="Reference"/>
                            <to>$Disponibilité_LicenceIn.parameters/refernce</to>
                    </copy>
                    <copy>
                        <from variable="WF1-WSDL1OperationIn" part="Qte"/>
                            <to>$Disponibilité_LicenceIn.parameters/Qte</to>
                    </copy>
            </assign>
```

```
            <invoke name="Invoke1" partnerLink="Service1" operation="Commander"
xmlns:tns="http://SW/" portType="tns:SW_Commander" inputVariable="Disponibilité_LicenceIn"
outputVariable="Disponibilité_LicenceOut"/>
        </sequence>
        <if name="If1">
            <condition>'existe' = $Disponibilité_LicenceOut.parameters/return</condition>
            <sequence name="TraiterBC" xmlns:tns="http://SW/">
                <sequence name="Sequence1">
                        <assign name="Assign2">
                                <copy>
                                        <from variable="WF1-WSDL1OperationIn"
part="Reference"/>

<to>$ApprouverBP_OperationIn.parameters/Reference</to>
                                    </copy>
                                    <copy>
                                        <from variable="WF1-WSDL1OperationIn" part="Qte"/>

<to>$ApprouverBP_OperationIn.parameters/Qte</to>
                                    </copy>
                            </assign>
                            <invoke name="Invoke2" partnerLink="Service2"
operation="ApprouverBP_Operation" xmlns:tns="http://SW/" portType="tns:SW_ApprouverBP"
inputVariable="ApprouverBP_OperationIn" outputVariable="ApprouverBP_OperationOut"/>
                    </sequence>
                    <sequence name="Sequence2">
                        <assign name="Assign3">
          <copy>
     <from variable="WF1-WSDL1OperationIn" part="Reference"/>

<to>$PreparerMarchandise_OperationIn.parameters/Reference</to>
   </copy>
<copy>
 <from variable="WF1-WSDL1OperationIn" part="Qte"/>

<to>$PreparerMarchandise_OperationIn.parameters/Qte</to>
                                    </copy>
                            </assign>
                            <invoke name="Invoke3" partnerLink="Service3"
operation="PreparerMarchandise_Operation" xmlns:tns="http://SW/"
portType="tns:SW_PreparerMarchandise" inputVariable="PreparerMarchandise_OperationIn"
outputVariable="PreparerMarchandise_OperationOut"/>
                    </sequence>
                    <sequence name="Sequence3">
                        <assign name="Assign4">
                                <copy>
                                        <from variable="WF1-WSDL1OperationIn"
part="Reference"/>

<to>$ValiderBL_OperationIn.parameters/Reference</to>
                                    </copy>
                                    <copy>
                                        <from variable="WF1-WSDL1OperationIn" part="Qte"/>
                                            <to>$ValiderBL_OperationIn.parameters/Qte</to>
                                    </copy>
                            </assign>
                            <invoke name="Invoke4" partnerLink="Service4"
operation="ValiderBL_Operation" xmlns:tns="http://SW/" portType="tns:SW_ValiderBL"
inputVariable="ValiderBL_OperationIn" outputVariable="ValiderBL_OperationOut"/>
                    </sequence>
                    <flow name="Flow1">
                        <sequence name="Sequence4">
                                <assign name="Assign5">
                                        <copy>
                                                <from variable="WF1-WSDL1OperationIn"
part="Reference"/>

<to>$EnvoyerFacture_OperationIn.parameters/Reference</to>
                                            </copy>
                                            <copy>
                                                <from variable="WF1-WSDL1OperationIn"
part="Qte"/>

<to>$EnvoyerFacture_OperationIn.parameters/Qte</to>
                                            </copy>
                                    </assign>
```

```
                                    <invoke name="Invoke5" partnerLink="Service5"
operation="EnvoyerFacture_Operation" xmlns:tns="http://SW/"
portType="tns:SW_EnvoyerFacture" inputVariable="EnvoyerFacture_OperationIn"
outputVariable="EnvoyerFacture_OperationOut"/>
                                </sequence>
                                <sequence name="Sequence5">
                                    <assign name="Assign6">
                                            <copy>
                                                    <from variable="WF1-WSDL1OperationIn"
part="Reference"/>

<to>$EnvoyerEMail_OperationIn.parameters/Reference</to>
                                                </copy>
                                                <copy>
                                                    <from variable="WF1-WSDL1OperationIn"
part="Qte"/>

<to>$EnvoyerEMail_OperationIn.parameters/Qte</to>
                                                </copy>
                                        </assign>
                                        <invoke name="Invoke6" partnerLink="Service6"
operation="EnvoyerEMail_Operation" xmlns:tns="http://SW/" portType="tns:SW_EnvoyerEMail"
inputVariable="EnvoyerEMail_OperationIn" outputVariable="EnvoyerEMail_OperationOut"/>
                                </sequence>
                        </flow>
                </sequence>
                <else>
                    <sequence name="Sequence6">
                        <assign name="Assign7">
                            <copy>
                                <from variable="WF1-WSDL1OperationIn" part="Reference"/>
                                <to variable="WSDL-SP1OperationIn" part="reference"/>
                            </copy>
                            <copy>
                                <from variable="WF1-WSDL1OperationIn" part="Qte"/>
                                <to variable="WSDL-SP1OperationIn" part="Qte"/>
                            </copy>
                        </assign>
                        <invoke name="Invoke7" partnerLink="Sub-process" operation="WSDL-
SP1Operation" xmlns:tns="http://j2ee.netbeans.org/wsdl/SubProcess1/WSDL-SP1"
portType="tns:WSDL-SP1PortType" inputVariable="WSDL-SP1OperationIn" outputVariable="WSDL-
SP1OperationOut"/>
                        <assign name="Assign8">
                            <copy>
                                <from variable="WSDL-SP1OperationOut" part="part1"/>
                                <to variable="WF1-WSDL1OperationOut" part="Rapport"/>
                            </copy>
                        </assign>
                        <reply name="Reply2" partnerLink="Partenaire1" operation="WF1-
WSDL1Operation" xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1"
portType="tns:WF1-WSDL1PortType" variable="WF1-WSDL1OperationOut1"/>
                        <exit name="Exit1"/>
                    </sequence>
                </else>
            </if>
            <reply name="Reply1" partnerLink="Partenaire1" operation="WF1-WSDL1Operation"
xmlns:tns="http://j2ee.netbeans.org/wsdl/WF1-Partenaire1/WF1-WSDL1" portType="tns:WF1-
WSDL1PortType" variable="WF1-WSDL1OperationOut"/>
        </sequence>
</process>
```