Decentralisation des procédés métiers : qualité de services et confidentialité

Ustun Yildiz

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

Décentralisation des procédés métiers : qualité de services et confidentialité

THÈSE

présentée et soutenue publiquement le 8 septembre 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Ustun YILDIZ

Composition du jury

Rapporteurs :	Chihab Hanachi	Professeur, Université Toulouse I, France
	Heiko Ludwig	Directeur de Recherche, IBM Almaden Research Center, États-Unis
Examinateurs :	Isabelle Christment	Professeur, Université de Nancy I, France
	Claude Godart	Professeur, Université de Nancy I, France
	Thomas Tamisier	Chef de Projet, CRP Gabriel Lippmann, Luxembourg

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

Mis en page avec la classe thloria.

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

Décentralisation des procédés métiers : qualité de services et confidentialité

THÈSE

présentée et soutenue publiquement le 8 septembre 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Ustun YILDIZ

Composition du jury

Rapporteurs :	Chihab Hanachi	Professeur, Université Toulouse I, France
	Heiko Ludwig	Directeur de Recherche, IBM Almaden Research Center, États-Unis
Examinateurs :	Isabelle Christment	Professeur, Université de Nancy I, France
	Claude Godart	Professeur, Université de Nancy I, France
	Thomas Tamisier	Chef de Projet, CRP Gabriel Lippmann, Luxembourg

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

Mis en page avec la classe thloria.

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

Décentralisation des procédés métiers : qualité de services et confidentialité

THÈSE

présentée et soutenue publiquement le 8 septembre 2008

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Ustun YILDIZ

Composition du jury

Rapporteurs :	Chihab Hanachi	Professeur, Université Toulouse I, France
	Heiko Ludwig	Directeur de Recherche, IBM Almaden Research Center, États-Unis
Examinateurs :	Isabelle Christment	Professeur, Université de Nancy I, France
	Claude Godart	Professeur, Université de Nancy I, France
	Thomas Tamisier	Chef de Projet, CRP Gabriel Lippmann, Luxembourg

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

Mis en page avec la classe thloria.

Remerciements

Je remercie vivement les rapporteurs qui se sont intéressés à mon travail, et qui me font l'honneur de participer à ce jury, malgré leurs lourdes tâches.

Je remercie Monsieur Heiko Ludwig qui a accepté d'être rapporteur de ma thèse. Sa longue expérience sur la gestion des procédés métiers le désignait tout naturellement pour évaluer mon travail, et sa lecture critique a contribué à l'amélioration du manuscrit. Je lui en suis très reconnaissant.

Je tiens à remercier Monsieur Chihab Hanachi dont j'ai toujours apprécié la qualité de ses travaux. Il s'est investi sans retenue dans la relecture du manuscrit. Il a le mieux ressenti la difficulté qu'il y a à présenter un travail de recherche de qualité et m'a soutenu aussi bien moralement que scientifiquement.

Je remercie Madame Isabelle Christment qui m'a fait l'honneur de présider ce jury.

Je tiens tout particulièrement exprimer ma gratitude à Monsieur Claude Godart, pour sa confiance et son amitié. Il s'est investi sans retenue dans la correction de ce manuscrit.

Une thèse est autant le résultat d'un travail individuel que de l'environnement dans lequel elle s'est effectuée. Pour cela, je tiens à associer à ces remerciements, tous mes collègues du LORIA et du ECOO qui m'ont encouragé et aidé à finir ce travail. Je pense particulièrement à Rémi, Yannick, Laszlo, Sahbi, Gérald, Claudia, Adnène, Khaled, Jesse et Walid.

Je remercie le FNRS du Luxembourg pour le financement des 3 premières années de ce travail en relation avec le CRPGL.

Table des matières

Table des figures

Chapitre 1

Introduction

Ces 20 dernières années qui ont vu les sciences et technologies de l'information et de la communication (STIC) révolutionner la société, une des applications significatives en est le Web qui est un moyen de rendre toutes les connaissances humaines accessibles à tous. C'est un des plus beaux challenges qui soit, qui modifie la vie de chacun de manière essentielle. A l'époque, la vision originelle de Tim Berners-Lee, inventeur du Web et président du W3C (World Wide Web Consortium), était celle d'une ressource qui permettrait la collaboration. Aujourd'hui, quand on pense au futur, on peut estimer que le Web sera partout, encore plus qu'aujourd'hui. Les systèmes échangeront simplement entre eux de l'information. Nous pourrons poser à notre table de nuit des questions comme : Où ai-je laissé mes lunettes ? Ce yaourt est-il encore mangeable ? Où ma fille se trouve-t-elle ? Le Web a été l'une des avancées les plus visibles de l'informatique de ces dernières années. Avec l'avènement des outils portables, communiquant sans fil, et embarquant des ressources de calcul de plus en plus importantes, ainsi que la généralisation de la géolocalisation, nous pouvons prédire que chaque appareil du quotidien deviendra non seulement un récepteur potentiel de l'information disponible sur le Web mais également un émetteur. Avec une évolution de l'informatique ubiquitaire, ainsi que la mise en place de grandes bases de données représentant l'information à l'échelle du monde entier, nous pouvons aussi prédire que chacun pourra contribuer et enrichir en permanence cette représentation virtuelle du monde fournie par le Web.

La recherche qui porte sur le Web est un domaine de recherche à part entière qui suscite beaucoup d'intérêt de la part de chercheurs de communautés très variées depuis plus d'une dizaine d'années (la première conférence World-Wide Web fut organisée en 1994). Après une phase de bouillonnement un peu irrégulière due à un fort engouement, les fils rouges de cette nouvelle piste de l'informatique se sont rapidement structurés. Actuellement, un de ces fils rouges est clairement les Architectures Orientées Services (AOS). Les architectures orientées services, et en particulier les services Web, permettent l'accessibilité, la découverte et l'utilisation universelle de n'importe quel application logicielle sur le Web en utilisant des normes ouvertes. Ainsi, les logiciels codés dans divers langages de programmation et sur diverses plateformes d'exécution peuvent employer des services Web pour échanger des données à travers le Web. Il est à noter que les services Web portent sur la définition universelle des interfaces des applications logicielles et le format des communications avec ces derniers. En conséquence, les services Web n'introduisent pas de contraintes d'implémentation pour les applications qui sont rendues accessibles sur le Web. Une des raisons de l'intérêt porté aux AOS est le contexte actuel de l'évolution des entreprises qui est caractérisé par une concurrence économique planétaire et un environnement sauvage, souvent imprévisible en termes de besoins et de demandes du marché. Pour survivre dans un tel

contexte, les entreprises doivent être flexible, réactive et dynamique et pour ce faire, le Web leur fournit un intergiciel sans précédent. Grâce aux services Web, les entreprises peuvent encapsuler leurs procédés métiers et les publier comme des services, chercher et souscrire à d'autres services et échanger des informations au-delà des frontières des entreprises. L'approche des services Web promet d'être la technologie clé pour automatiser les interactions Business-2-Business (B2B) ainsi que Business-2-Consumer (B2C).

La contrepartie des avantages des AOS est l'extrême complexité de conception des applications orientées services. Les premières applications orientées services opérationnelles ont vu le jour il y a plus de 5 ans. Elles n'étaient en fait que de simples applications réparties du style client-serveur qui utilisaient des formats spéciaux pour invoquer des composants distants. Bien que la valeur de la technologie des services de Web ait été démontrée dans la pratique à l'aide des applications pilotes, il y a un désir intense d'employer cette approche pour adresser des problèmes plus généraux. Les chercheurs recherchent les perfectionnements qui élèvent le niveau et la portée de l'interopérabilité au delà de ce que permet l'échange de base de messages, exigeant l'appui pour l'interopérabilité des services de plus haut niveau d'infrastructure. Actuellement, les descriptions des services Web de seconde génération sont de plus en plus riches contrairement au Web traditionnel que nous considérons comme *syntaxique* puisque seule la structure, le contenu superficiel, et les propriétés élémentaires de l'information y sont disponible.

Bien que la technologie des services Web ne soit pas la seule manière pour réaliser une AOS, elle est la technologie la plus considérée par l'industrie. Avec les services Web, l'industrie adresse encore une fois le défi fondamental de l'informatique répartie : comment fournir une manière uniforme pour décrire des composants ou des services dans un réseau, les localiser, et y accéder. L'industrie aborde ce problème en utilisant des standards ouverts définis en association avec de larges consortiums tels que le W3C et l'organisation pour l'avancement des normes structurées de l'information (OASIS) [Dumas and Fauvet2006].

Un des points forts de la culture du Web est d'assurer une interaction très fructueuse entre les éléments décentralisés. Les gros problèmes de l'interaction ne sont pas seulement techniques mais ce sont aussi des problèmes de société : je dispose des services Web nécessaires pour terminer mon procédé, est-ce que je peux utiliser ces deux services ensembles ? Est-ce qu'ils exigent une interaction directe entre eux ou bien je peux les coordonner moi-même ? Si je les laisse travailler entre eux pour moi, peut-il y avoir des conséquences contre mes intérêts ? Est-ce qu'ils peuvent faire des choix pour moi si je ne suis pas accessible ? Dois-je surveiller l'interaction de ces services en permanence ? Les interactions sont souvent formalisées sous forme de procédés métiers exécutés par une organisation d'une manière centralisée ou bien par un ensemble d'organisation d'une manière décentralisée. Dans de nombreuses situations critiques, il est nécessaire d'assurer un certain degré de prépondérance entre ces deux modalités d'interactions. Les critères qui gouvernent les interactions sont nombreux allant de la cohérence des interactions mutuelles à l'implémentation des interactions en adéquation avec les relations de confiance des organisations. Les avancements dans les efforts de standardisation coïncident avec l'émergence des nouvelles exigences en termes de contrôle des interactions des services autonomes. Notre travail se place à l'intersection de ces deux contextes : les efforts de standardisation et les besoins de configuration des services composés qui sont régis par une multitude de besoins à mesure de la maturation de leur problématiques.

Dans ce cadre, le travail présenté dans cette thèse porte sur la définition et la mise en œuvre d'une méthodologie de gestion de procédés métiers destinés aux nouvelles exigences engendrées par l'AOS. Dans les grandes lignes, l'objectif visé est de fournir un environnement d'exécution de procédés métiers orienté services qui permet à un service, non seulement d'être composé au sein d'un procédé métier, mais aussi de pouvoir établir des interactions sophistiquées égal-à-égal

restreintes par des critères divers. Cette motivation est concrétisée par la production et le déploiement des procédés coopérants exécutés par les services composés. Ces derniers spécifient les interactions que les services doivent établir. Cette approche permet aux services composés de se comporter différemment pour chaque composition qu'ils font partie. En utilisant une approche délibérément formelle, la production des procédés coopérants se fait à partir d'une spécification centralisée de la composition qui est considérée comme une description de workflow. Par la suite, la méthode de production des procédés coopérants intègre alors, sous formes d'aspects non fonctionnels tels que les politiques du flux d'information, les restrictions qui gouvernent les interactions des services composés. L'exécution décentralisée du procédé avec les procédés coopérants produits après le traitement des politiques du flux d'information, montre la faisabilité des compositions qui ne peuvent pas être exécutées par des conceptions centralisés traditionnels. La dernière dimension de la méthodologie de décentralisation fournit un support pour le déploiement dynamique et décentralisé des procédés coopérants. La réalisation de l'approche développée consiste à implémenter le déploiement des procédés produits vers les services composés. Ceci considère une implémentation de service alternative aux implémentations actuelles. L'approche réside dans l'utilisation des standards pour la spécification et l'implémentation des compositions. Car les standards fournissent, non seulement un support pour l'interopérabilité des applications, mais aussi un environnement pour l'écheangéabilité (la mobilité) des spécifications codées en standards entre les services.

Organisation du manuscrit

Ce manuscrit se présente en 8 chapitres. Quelques annexes complètent le corps du texte avec notamment la prise en compte de détails de réalisations logicielles. Chaque chapitre commence par une introduction qui présente le propos et l'organisation de son contenu. En outre, une section de synthèse résume les contributions, l'intérêt, les différences par rapport aux travaux relatifs, et les limitations du travail présenté dans le chapitre. Les paradigmes, les concepts, les structures, et les algorithmes sont expliqués sur des exemples typiques pour mettre en évidence la difficulté des problèmes qu'ils formalisent et les voies pour les résoudre.

Le chapitre présent, **Introduction**, présente rapidement le contexte général.

Le chapitre 2, **Problématique générale et objectifs de la thèse**, est consacré à une présentation du contexte de notre travail et de la problématique générale de la thèse. Après une introduction au domaine de l'ingénierie des procédés métiers, les concepts appropriés du domaine de recherche sont brièvement expliqués. Par la suite, les problèmes principaux de la thèse sont clairement posés :

- comment fournir les procédés qui doivent être exécutés par les services composés afin d'établir des interactions égal-à-égal ?
- quelles sont les restrictions qui gouvernent les interactions des services du point de vue des flux d'information à respecter ?
- quelles sont les restrictions architecturales à respecter ? Comment un procédé coopérant peut être déployé vers un service ?

Un exemple illustratif est introduit pour faciliter la compréhension de la problématique. Le chapitre résume également les idées centrales, l'approche et les contributions principales de la thèse.

Le chapitre 3, **État de l'art**, est dédié au parcours de la littérature relative à notre travail

dans l'objectif de présenter un état de l'art. Le rôle de ce chapitre, en permettant une introduction progressive des paradigmes, des concepts et des technologies par rapport aux définitions habituelles dans le domaine, est essentiellement pédagogique. Une analyse critique montre leur intérêt et leur limites respectifs ainsi que leur caractère complémentaire. Compte tenu du foisonnement actuel du domaine, il est inévitable que l'état de l'art ne traite pas de manière exhaustive l'ensemble des travaux réalisés. Néanmoins, l'analyse faite couvre raisonnablement bien les principaux travaux pertinents, que ce soit en matière d'architectures et de technologies de services, de paradigmes et de modèles décentralisés et de workflow. Plus précisément, l'état de l'art est présenté en trois sections. Une première section présente les paradigmes de service, puisque l'utilisation des spécifications standards pour les services est considérée comme centrale dans la thèse, examine la diversité de ces paradigmes et des approches principales à la problématique de composition des services. La deuxième section de ce chapitre porte sur l'étude des paradigmes et à des systèmes décentralisés et égal-à-égal. Elle fait une comparaison critique des systèmes centralisés, des systèmes hybrides et des systèmes totalement décentralisés. Elle examine aussi les approches contradictoires en mettant en avant les avantages et les limitations au niveau de la conception et de la mise en œuvre, et en émettant les besoins que doivent satisfaire les procédés métiers et les compositions de services. Enfin, ce chapitre s'intéresse à l'initiative workflow qui est l'émergence informatique la plus générale du concept de procédé, et qui est aussi à l'origine de l'orchestration des services en commençant de façon un peu historique, et très rapidement, par la gestion décentralisée de workflow. Nous terminons cette section par un tableau qui synthétise les caractéristiques de chacune des propositions de gestion décentralisée de workflow.

Le chapitre 4, **La modélisation d'un procédé et sa décentralisation *par défaut***, décrit le cœur du travail de thèse sous la forme d'une technique de décentralisation d'une spécification de procédé centralisé qui compose un ensemble de services en un ensemble de procédés coopérants exécutés par les services. Les procédés coopérants mettent en œuvre la sémantique centralisée comme interactions d'égal-à-égal entre les services composés. Dans un contexte de workflow, une première partie approfondit les types de situations et les dépendances correspondantes que la technique de décentralisation doit traiter. Une seconde partie présente les algorithmes employés. D'abord, les contours du modèle de procédé qui est utilisé dans ce chapitre et le reste de la thèse sont clairement identifiés. Les hypothèses concernant l'infrastructure de communication sous-jacente et les procédés traités sont présentées. Les structures de données et les algorithmes sont accompagnés des exemples de motivations qui facilitent leur compréhension. Le chapitre discute également la complexité et les limitations des techniques présentées. L'approche présentée est une solution relativement complète aux problèmes de la production de procédés coopérants à partir d'une spécification centralisée. Elle considère la décentralisation des dépendances de contrôles, de données, et conversationnelles qui nécessite une considération plus profonde qu'une approche intuitive. Par rapport aux travaux liés à l'implémentation de la gestion décentralisée de procédés métiers qui portent sur la proposition des modules et de protocoles particuliers employés par les services, l'approche présentée considère le problème de décentralisation du point de vue de la production de procédés coopérants. En conséquence, les résultats du chapitre, qui seront suivis par les considérations architecturales correspondantes dans le chapitre 7, sont la contribution positive à une gestion décentralisée, universelle et peu coûteuse des procédés.

Le chapitre 5, **Contrôle du flux d'information : *Vers un modèle décentralisé efficace***, met les contributions du chapitre précédent dans un contexte inter-organisationnel plus applicatif. Le problème dont nous souhaitons nous affranchir est essentiellement l'adaptation de l'approche originale aux restrictions secondaires qui gouvernent les interactions des services

composés. Les restrictions que nous intéressent concernent le contrôle du flux d'information. Le contrôle du flux d'information est une dimension essentielle des compositions efficaces de services qui ont des interactions restreintes du point de vue des politiques de sécurité, d'intimité ou de confiance. Il consiste à spécifier et à configurer les interactions de services en considérant la propagation de l'information dans la composition qui les utilise. Les contributions présentées dans ce chapitre nécessitent la compréhension et l'intégration d'aspects qui sont souvent étudiés séparément comme le contrôle d'accès, la gestion de la confiance, ou la vérification d'une description de chorégraphie. Dans un premier temps, nous développons un support pour la spécification des politiques du flux d'information et les algorithmes pour traiter celles-ci. Le résultat du traitement des politiques de flux d'information est caractérisé par les chemins fiables au sein des services composés. Par la suite, nous intégrons les chemins trouvés à la production des procédés coopérants en révisant l'approche présentée dans le chapitre précédent. Soulignons qu'il n'est pas toujours aussi évident de donner un énoncé initial, une définition de ce type de restrictions. De par son caractère novateur, le contrôle du flux d'information selon cette approche permet une conception progressive d'une exécution décentralisée face aux restrictions principales qui gouvernent les interactions, et, facilite le passage d'une spécification à une exécution.

Le chapitre 6, **La réalisation des exécution décentralisées avec le support pour le choix dynamique et décentralisé des services**, s'intéresse au choix dynamique et décentralisé des services dans un contexte de composition décrit par les contributions précédentes. Le choix dynamique et décentralisé des services sont des préoccupations majeures dans les applications pervasives. Nous entendons par choix dynamique et décentralisé, le choix retardé d'un service par un autre service juste avant son invocation. Ce type de choix de service profite des avantages inhérents aux systèmes décentralisés. Pourtant, le choix dynamique des services peut engendrer des problèmes de synchronisation entre les services choisis d'une manière concurrente et décentralisée. Ce chapitre discute le choix dynamique et décentralisé des services et fournit une solution compréhensible aux problèmes de synchronisation liés. La solution proposée consiste à définir des dépendances entre les activités et les services de telle sorte que le choix d'un service par un autre ne cause pas des problèmes de synchronisation dans les étapes successives de l'exécution. Contrairement aux deux chapitres précédents qui se concentrent sur les aspects concernant les interactions des procédés coopérant, la contribution de ce chapitre porte sur les problèmes liés au déploiement des procédés produits. Ainsi, une autre dimension de la décentralisation est étudiée.

Le chapitre 7, **Mise en œuvre : *Des implémentations références***, présente les réalisations logicielles du travail. Dans ce cadre, l'implémentation considère deux réalisations logicielles séparées mais complémentaires. La première réalisation consiste à une implémentation de service, dite de *référence*, qui permet le déploiement d'un procédé vers un service par un autre service client. Elle se base sur l'implémentation des opérations d'administrations du moteur ActiveBPEL comme les opérations de l'interface du service permettant ainsi leur invocation par des clients. Comme nous l'avons supposé au début de notre travail, par rapport aux solutions existantes concernant la gestion décentralisée du workflow, cette implémentation est la preuve de l'existence d'une solution simple et élégante en exploitant les apports de la standardisation. La deuxième réalisation logicielle porte sur l'implémentation des algorithmes génériques développés. Le langage de procédé choisi est WS-BPEL qui est le standard de fait de l'industrie.

Le dernier chapitre, **Conclusions et perspectives**, rappelle les objectifs de la thèse ainsi que les résultats obtenus. Il contient également des perspectives dans des pistes en plein développement.

Chapitre 2

Problématique générale et objectifs de la thèse

Sommaire

2.1 Introduction

Dans ce chapitre, nous présentons le contexte de notre travail et la problématique générale de la thèse. Après une introduction au domaine de l'ingénierie des procédés métiers, les concepts appropriés du domaine de recherche sont brièvement discutés. Par la suite, les problèmes principaux de la thèse sont clairement posés. Un exemple illustratif est introduit pour faciliter la compréhension de la problématique. Le chapitre résume également les idées centrales, l'approche et les contributions principales de la thèse.

2.2 Contexte de la thèse

Le contexte général de cette thèse est l'ingénierie des procédés métiers. Un procédé métier est une procédure qui systématise l'organisation et la politique d'une organisation (ex. une entreprise) dans le but d'atteindre les objectifs bien précis de cette dernière. Cette définition n'est ni simple ni unique. On peut distinguer plusieurs types de procédés qui peuvent exister au sein des organisations : les procédés créatifs tels que les procédés de conception d'objets sophistiqués ou d'édition collaborative, les procédés automatiques tels que les chaînes de production sans intervention humaine, les procédés interactifs qui supportent les interactions homme-homme etc. L'ingénierie des procédés métiers peut être définie comme une branche de l'informatique qui

vise à gouverner le cycle de vie d'un procédé : il s'agit de le définir d'une manière formelle, fournir un environnement d'exécution pour l'exécuter, améliorer son agilité et observer sa performance.

Les procédés qui nous intéressent dans cette thèse sont structurés autour de l'approche workflow[Leymann and Roller2000]. Les concepts et les technologies de workflow sont introduits comme un ensemble de techniques et outils permettant la définition et l'exécution (et donc l'automatisation) des procédés métiers. Dans de tels systèmes, un procédé est généralement perçu comme un ensemble d'activités effectuées au sein de l'organisation. Le workflow définit les flots de contrôle et de données du procédé. Il intègre à la fois les acteurs humains et les systèmes logiciels de l'organisation pour les orchestrer au sein du procédé. L'ingénierie des procédés métiers, basée sur le workflow ou non, constitue aujourd'hui un enjeu stratégique pour les organisations commerciales. Les problèmes sous-jacents liés à l'analyse, la modélisation, l'exécution et le contrôle des procédés métiers ont fait l'objet de nombreuses recherches ces dernières années pour répondre aux besoins cruciaux dans la pratique.

Un de ces aspects est que le procédé métier d'une entreprise est souvent connecté aux procédés métiers d'autres organisations. Ces procédés sont souvent appelés des procédés inter-organisationnels. L'avènement des infrastructures réseaux et des technologies standardisées du Web pousse aujourd'hui les organisations à utiliser les procédés des autres organisations et à rendre leur procédés accessibles aux autres. Ce contexte actuel de l'évolution des organisations est caractérisé par une concurrence économique devenue planétaire et un environnement agressif et souvent imprévisible en termes de besoins et de demandes du marché. Les efforts de standardisation dans ce contexte ont permis l'émergence des services Web comme support concret au développement des procédés métiers inter-organisationnels. Un service Web est une application logicielle conçue pour supporter les interactions machines à machines sur le Web. Les services Web sont basés sur l'utilisation de normes ouvertes encodées en XML[Curbera *et al.*2002]. Une définition plus précise peut inclure l'usage exclusif de WSDL[Chinnici *et al.*2002] pour la description de l'interface d'un service et l'usage de SOAP[Box *et al.*2000] pour le traitement des messages. Par rapport à d'autres technologies conçues pour le développement d'applications distribuées telles que Java RMI[jav], l'avantage majeur des services Web et qu'ils n'imposent aucune contrainte de programmation spécifique pour aux applications rendues accessible sur le Web. De ce fait, les services Web répondent à un des aspects essentiels de la problématique bien connue de l'intégration des données[Halevy *et al.*2006]. Bien qu'il y ait plusieurs avantages qui expliquent leur adoption rapide, les caractéristiques des services Web posent de nombreuses problèmes. Outre les problèmes techniques de communication, pour lesquels les technologies des services Web existent, de nombreux problèmes de génie logiciel persistent pour l'utilisation des services Web : quel est le coût d'utilisation de ce service ? Puisse-je l'utiliser ? Pour combien de temps ? Je dispose des services nécessaires pour terminer mon procédé, est-ce que je peux utiliser ces deux services ensembles ? Si je peux les utiliser, est-ce qu'ils exigent une interaction directe entre eux ou bien je peux les coordonner moi-même ? Si je les laisse travailler entre eux pour moi, peut-y-il avoir des conséquences contre mes intérêts ? Est-ce qu'ils peuvent faire des choix pour moi si je ne suis pas accessible ? Dois-je surveiller ce service en permanence ?

En conséquence, il est pertinent de considérer un procédé inter-organisationnel orienté service comme étant constitué de deux niveaux problématiques : un niveau *communication* qui concerne les technologies sous-jacentes utilisées comme par exemple le format des messages échangés ou la représentation de l'interface et un niveau *application métier* qui concerne le choix des services à utiliser et leur combinaison en fonction de différentes critères.

Les contributions de cette thèse portent plutôt sur ce dernier point tout en pensant aux relations avec les aspects de communication sous-jacents. Plus précisément, nous nous intéressons aux aspects applicatifs des procédés métiers inter-organisationnels orientés services. Dans

ce contexte, notre travail est placé à l'intersection des pistes de recherche et des technologies suivantes :

Services et technologies Web La réalisation des procédés métiers inter-organisationnels s'appuie aujourd'hui de plus en plus sur des technologies standardisées du Web telles que les services Web. Il est à noter que les efforts de standardisation ont permit l'émergence des standards de bases (ex. WSDL, SOAP, UDDI) pour la réalisation des services Web simple ainsi que des standards plus compliqués comme les langages de spécification de procédés métiers(ex. WS-BPEL [OASIS2005]) qui spécifient la composition des services simples. Bien que la technologie de services Web ne soit pas la seule manière de réaliser un procédé inter-organisationnel, elle est la technologie essentielle considérée par l'industrie. Avec les services Web, un défi fondamental pour l'informatique répartie est : comment fournir une manière uniforme de décrire des composants ou des services dans un réseau, de les localiser, et de leur accéder. Ce problème est abordé en utilisant les concepts et les technologies qui sont développés d'une voie ouverte, utilisant des associations d'industrie et de larges consortiums tels que W3C et l'organisation pour l'avancement des normes structurées de l'information (OASIS). Ainsi, l'utilisation des technologies Web ainsi que les concepts associés sont incontournable pour la réalisation des procédés métiers efficaces.

Dans un premier temps, nous utilisons les services Web comme le support de réalisation de nos contributions conceptuelles. Par la suite, nous étudions les apports secondaires des efforts de standardisation comme la mobilité des spécifications entre différents outils et organisations. Notre approche est basée sur ce deuxième aspect. Celle-ci est expliquée dans la section *idées centrales de la thèse* (voir la section 2.4.1).

Workflow Le choix des modèles de conception pour les procédés métiers ou bien pour des systèmes d'information en général semble faire partie des sujets à polémiques, tant il est intimement lié à la culture scientifique des individus, au domaine d'application et aussi à la destination des aspects qu'on peut en dériver. L'idée de modélisation des procédés n'est pas nouvelle, on la trouve dans la description de procédés de fabrication de logiciel dès 1987[Osterweil1987], semblable aux langages de programmation, mais adaptés aux besoins spécifiques du domaine pour décrire ces procédés. Si l'émergence de l'approche workflow dans les systèmes d'information quotidiens est nouvelle, l'idée d'utiliser des approches similaires est bien plus ancienne. L'initiative *Worflow Management Coalition*[WfMC1996] qui regroupe alors un ensemble d'industriels et de chercheurs académiques, a proposé des standards pour le workflow suite aux indéniables besoins du marché et au manque de standardisation. Les efforts individuels comme ceux de Van Der Aalst et son groupe ont donné lieu à plusieurs résultats bien établis facilitant la compréhension et la mise en œuvre comme les patrons de workflow[van der Aalst *et al.*2000]. A bien des égards, les technologies des services Web et leur propres efforts de standardisation s'intéressent à la modélisation des procédés inter-organisationnels orientés services d'un point de vue worfklow. Ceci est tout à fait lié aux connaissances communes établies dans le cadre des efforts de standardisation du workflow. WS-BPEL[OASIS2005] qui est le langage *de-facto* de spécification des procédés métiers suit les mêmes voies que celles qui ont fait le workflow l'une des approches essentielles pour la modélisation des procédés métiers.

Dans cette thèse, nous pensons que plutôt que de définir de nouveaux langages de spécification de procédés, il est plus efficace de réutiliser des formalismes existants structurés autour de l'approche workflow, de les adapter et de les interfacer.

Systèmes décentralisés et égal-à-égal (P2P) La vision classique d'un système de gestion de procédés métiers est centralisée. En conséquence, elle est caractérisée par les limitations inhérentes des systèmes centralisés tant matériellement que conceptuellement. Depuis 1999, date à laquelle l'application Napster connut un succès, le modèle égal-à-égal attire l'attention des acteurs de la communauté de l'ingénierie des procédés métiers. Les scientifiques et les industriels voient en cette approche une véritable alternative aux approches centralisées qui sont les modèles actuellement utilisés au sein des organisations. L'établissement des interactions égal-à-égal entre les participants d'un procédé métier présente de nombreux avantages qui repoussent les limites induites par le modèle centralisé. Si les premières applications qui implémentent l'approche égal-à-égal était des applications de distribution de fichiers multimédias soumis à des droits d'auteurs, actuellement l'ensemble des systèmes de gestion de procédés métiers sont repensables et repensés pour être déployés selon la même approche. La décentralisation de l'exécution des procédés métiers est identifié comme une piste de recherche prometteuse depuis une dizaine d'année. Ainsi, l'utilisation des paradigmes de la décentralisation des procédés métiers précède l'apparition des systèmes égal-à-égal[Alonso *et al.*1997], mais avec cependant des lacunes qui réduisent sérieusement leur champ d'application. Dernièrement, les architectures orientés services introduisent de nouvelles caractéristiques qui ne sont pas en adéquation avec les anciennes technologies des workflows décentralisés comme par exemple OSIRIS[Schuler *et al.*2004], SELF-SERV[Benatallah *et al.*2002] ou PPWD[Fakas and Karakostas2004].

Même si l'exécution décentralisée des procédés métiers implémentés par les workflows a été identifiée comme une piste intéressante avant une dizaine d'année, la plupart des contributions précédent les architectures orientées services et les efforts de standardisation des procédés métiers. Les contributions qui suivent les efforts de standardisation et portant sur le workflows décentralisés suivent les mêmes approches que celles développées dans les travaux antérieurs. Plusieurs travaux récents ont identifiés des approches alternatives à la conception et l'implémentation des procédés métiers inter-organisationnels [King1983] [Schuler *et al.*2004] [Weerawarana *et al.*2005] [Fischer2007] pour dépasser les nombreuses limitations des systèmes centralisés. L'exécution décentralisée est une de ces approches alternatives. Il est à noter que notre compréhension de décentralisation est loin des applications de partage de fichier multimédia. Nous discutons les paradigmes liées à la décentralisation et d'égal-à-égal dans le contexte de l'ingénierie de procédés métiers. **La contribution de cette thèse portera sur les aspects de décentralisation des procédés métiers inter-organisationnels orientés services.**

2.3 Problématique

Dans cette section, nous nous penchons sur la problématique précise de cette thèse. La problématique générale porte sur la proposition d'un environnement décentralisé pour la conception et l'exécution des procédés métiers inter-organisationnels orientés services. Notre compréhension de la décentralisation se réfère aux exécutions où les services peuvent établir des interactions d'égal-à-égal. Cette proposition est motivée par les limitations des environnements existants face aux exigences des systèmes qui nous intéressent.

Dans un contexte inter-organisationnel qui intègre des organisations autonomes integrés, il est souvent souhaitable d'établir des relations égal-à-égal entre elles. Bien que considéré aujourd'hui comme étant un des outils les plus puissants pour supporter l'automatisation des procédés métiers, l'approche workflow souffre des limites conceptuelles, et matérielles de la centralisation. Pour expliquer nos motivations, nous utilisons l'exemple simple suivant.

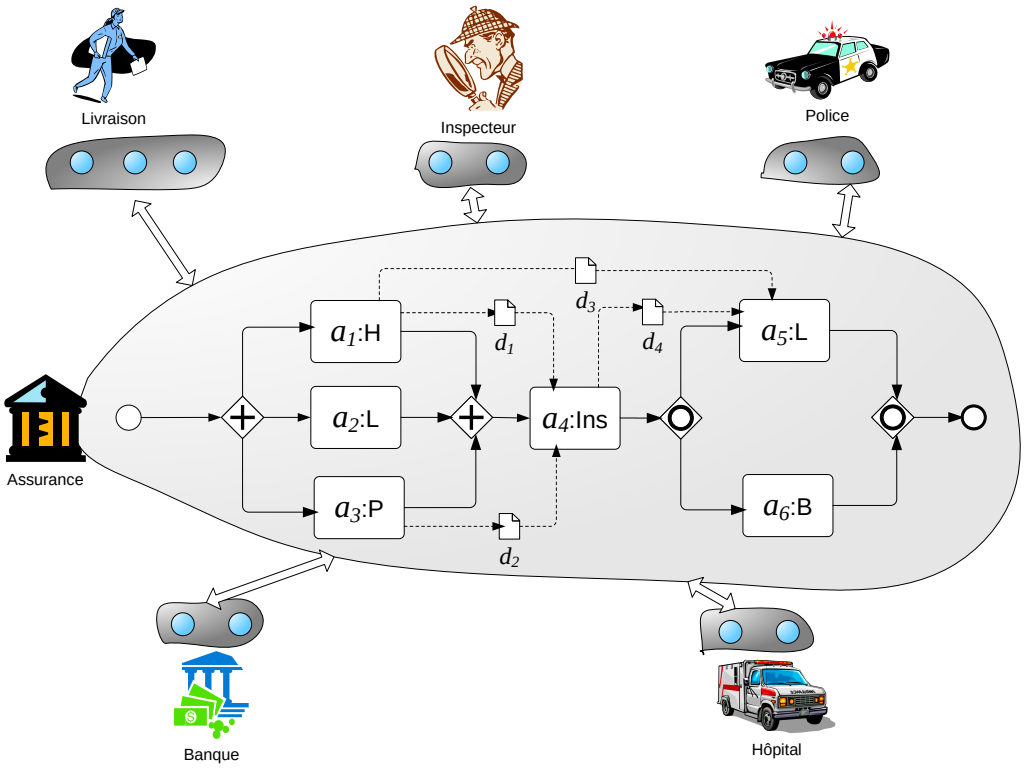


FIG. 2.1 – Un exemple de motivation

2.3.1 Un exemple de motivation simple

La figure 2.1 présente un procédé inter-organisationnel dans sa généralité. Il s'agit d'un procédé exécuté par une organisation d'assurance en vue de déterminer si la réclamation de sinistre d'un souscripteur est remboursable. Ce scénario peut être résumé comme suit. En cas de sinistre, le souscripteur appelle la police et les services d'urgence qui vont l'aider pour l'incident. La police et les services d'urgence gérés par l'hôpital local préparent respectivement deux rapports décrivant l'incident et l'état du souscripteur. Par la suite, le souscripteur doit déposer sa réclamation de remboursement dans un délai déterminé au siège de l'organisation d'assurance. Quand la réclamation est reçue, le procédé illustré dans la figure 2.1 est lancé par l'organisation d'assurance. Dans un premier temps, l'organisation d'assurance contacte la police et l'hôpital pour récupérer les rapports respectifs qui ont été préparés au moment de l'incident. Ces rapports sont importants pour déterminer la validité de la demande de remboursement. La réclamation et les rapports du sinistre sont examinés par un inspecteur qui s'engage à les traiter le plus rapidement et le plus objectivement possible. Si la réponse de l'inspecteur est affirmative, la réclamation est remboursée. L'indemnisation est réalisée via la banque avec laquelle l'organisation d'assurance travaille. Le rapport de l'inspecteur et les détails de l'indemnisation sont transmis au souscripteur par un service de livraison. Ce procédé inter-organisationnel est modélisé comme un workflow et exécuté par l'organisation d'assurance. Les autres organisations sont accessibles par les services Web. Ces services Web sont appelés : Hôpital, Police, Inspecteur, Banque, Livraison. On peut également ajouter Assurance comme un service comme les autres. Le workflow de la figure 2.1 illustre les dépendances de contrôle et de données des activités du procédé. Les

dépendances de contrôle caractérisent l'ordre partiel de l'exécution des activités alors que les dépendances de données caractérisent les relations de type entrées/sorties des activités. Dans l'illustration, chaque activité décrit son identité et le service évoqué pour son exécution.

Dans une approche workflow classique, l'organisation d'assurance évoque les services appartenant aux organisations en suivant la spécification du modèle de workflow. Cela veut dire que les organisations impliquées sont coordonnées par l'organisation d'assurance et qu'elles sont isolées des unes des autres. Par exemple, les sorties des services Police et Hôpital sont utilisées comme entrées du service Inspecteur. Par contre ces données ne sont pas transmises de service producteur au service consommateur. Étant le coordinateur du procédé, Assurance fait l'opération de routage. Comme les données ne sont pas directement échangées entre les services concernés, la quantité globale d'information augmente et surcharge l'infrastructure. Cela est une limitation bien connue entre autres des systèmes centralisés (voir le chapitre *État de l'art*).

Comme nous l'avons souligné dans les sections précédentes, les interactions d'égal-à-égal entre les organisations sont souhaitables pour plusieurs raisons, mais la vision classique de l'approche worfklow ne permet pas de telles interactions.

Pour montrer l'intérêt d'avoir des interactions d'égal-à-égal, nous ajoutons les restrictions suivantes au scénario.

1. Hôpital peut fournir le rapport de l'accident seulement au souscripteur et à la Police,
2. Inspecteur peut recevoir les rapports seulement de Police et/ou Hôpital,
3. Banque peut être évoquée seulement par Assurance

La présence de ce type de restriction semblent très raisonnable dans le contexte où plusieurs services mutuellement non-confiants doivent être combinés. Par exemple, le rapport fourni par Hôpital peut contenir des informations confidentielles qu'un tiers ne doit pas obtenir. Pourtant, comme Police est un service appartenant au gouvernement, il peut être considéré comme fiable. Une situation similaire est valable pour Inspecteur car, les rapports de Police et Hôpital contiennent des informations sensibles qui peuvent être modifiées par des organisation d'assurance pour empêcher les remboursements. En conséquence, Inspecteur peut vouloir recevoir les rapports directement de la part des services qui les fournissent. Dans ce contexte, la présence de la première et de la deuxième restriction empêchent l'exécution du procédé par un système de gestion de workflow classique. Parce que quand le workflow évoque Police, il reçoit le rapport qu'il doit router à Inspecteur. Pourtant la restriction de ce dernier empêche cet opération (la restriction 2 ci-dessus). De même, Hôpital ne peut pas être évoqué par Assurance pour la récupération du rapport. Il est à noter que l'on peut enrichir la sémantique de ces types de restrictions en ajoutant des propriétés temporelles, spatiales ou conditionnelles similaires selon les restrictions similaires des systèmes de gestion de workflow [Bertino *et al.*1999] [Joshi *et al.*2005] [Damiani *et al.*2007] [Bertino2004]. Aussi, Il est clair que les environnements classiques de composition de services ou de gestion des procédés métiers actuels n'ont pas les bonnes fonctionnalités exigées par la décentralisation du contrôle.

Une première question qui vient alors est : *Comment les services composés peuvent connaître et établir des interactions avec les autres services du même procédé inter-organisationnel ? Comment peuvent-ils savoir à quel service envoyer leur sorties ou bien de quel service recevoir de données ?* Dans une composition centralisée comme présentée dans l'illustration, les services sont mutuellement isolés et répondent aux invocations venant d'un service client qui joue le rôle de coordinateur. Le coordinateur fait appel aux opérations de service déclarées dans son interface. Il est à noter que l'interface d'un service est statique et ne peut pas être modifiée dynamiquement même si l'organisation qui implante le service peut modifier son fonctionnement interne. Les interactions que nous considérons ont pour but d'échanger le contrôle et les données et ne sont

pas l'implantation des restrictions spéciales telles que les politiques déclarés par les services. En partant de cette question, on peut poser une deuxième question similaire. *Comment les services composés peuvent se comporter différemment pour chaque procédé dont ils font partie en fonction de critères différentes ?* Dans cet exemple, les restrictions qui doivent être respectés ne sont valable que pour ce scénario. Mais, étant donné qu'un service est une entité autonome, un autre client peut vouloir l'utiliser en imposant d'autres types de restrictions. Là encore, les implantations actuelles des services ne supportent pas ce type d'exécution où les services peuvent se comporter différemment en fonction des composition. Parlant des restrictions qui doivent être respectées, on peut poser la question suivante. *Comment peut-on spécifier les restrictions qui concernent les interactions d'égal-à-égal dans un procédé inter-organisationnel ? Qui les définit ? Est-ce qu'il y a des restrictions par défaut ?* Les besoins exprimés pour l'exécution des procédés métiers inter-organisationnels sont souvent proches de ceux exprimés dans des domaines variés comme la gestion décentralisée de base de données [Joshi *et al.*2005] [Bertino2004] ou la séparation des préoccupations dans les systèmes de gestion de workflow [Atluri and Warner2005] [Atluri *et al.*2001] qui ne sont pas adaptés à l'expression des relations d'égal-à-égal entre les services. Dans l'exemple de l'illustration de la figure 2.1, les interactions Hôpital/Police et Police/Inspecteur sont mutuellement en adéquation. En conséquence, Police peut évoquer Hôpital et recevoir les sorties de ce service. Par la suite, Police peut évoquer Inspecteur avec les sorties de Hôpital. Ainsi, un procédé inter-organisationnel qui n'est pas exécutable avec une approche centralisée, peut être faisable en révisant les interactions des services composés. Pourtant, cette approche nécessite la considération de restrictions diverses qui n'existe pas dans les exécutions conventionnelles. Par exemple, *est-ce que Police a le droit de fournir les sorties de Hôpital à un autre service à qui Hôpital ne fournit pas ses sorties ? Comment exprimer des restrictions similaires ? Quelles est la nature de ces spécifications ?*

En partant de ces observations, le but de notre contribution est de fournir des réponses compréhensibles et raisonnables aux question suivantes :

1. Comment les services peuvent établir des interactions d'égal-à-égal d'une manière systématique quand ils sont composés au sein d'un procédé inter-organisationnel ?
2. Comment les restrictions concernant les interactions d'égal-à-égal peuvent être spécifiées et traitées ?
3. Comment peut-on implémenter des services qui peuvent supporter des interactions d'égal-à-égal tout en respectant leur autonomie et en respectant les efforts de standardization ?

2.4 Objectifs de la thèse

Dans cette section, nous présentons les objectifs de cette thèse et nous résumons nos contributions. L'objectif général de cette thèse est de fournir un environnement d'exécution décentralisée pour les procédés métiers inter-organisationnels orientés services. Nous nous intéressons aux aspects conceptuels de la décentralisation ainsi qu'aux défis architecturaux pour son implémentation. Nous avons mentionné dans les sections précédentes la diversité des critères qui doivent être considérés. Dans un premier temps, notre objectif est de traiter les critères les plus essentiels et de donner un aperçu sur la satisfaction d'autre types de critères selon l'approche proposée.

2.4.1 Idées centrales de la thèse

La raison principale pour laquelle une exécution décentralisée n'est pas toujours faisable est que les services composés ne se comportent pas comme exigés par la décentralisation. En effet,

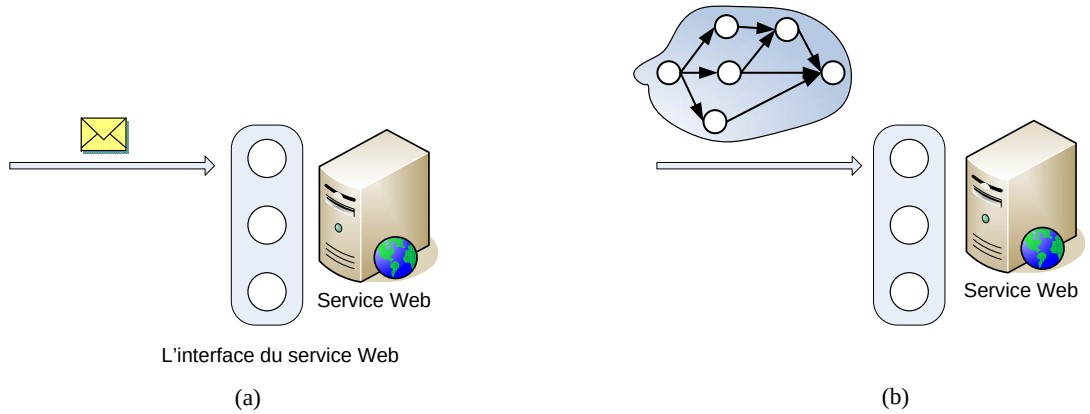


FIG. 2.2 – La figure(a) s'agit de l'utilisation d'un service classique, l'opération du service est évoquée par un message du client. La figure(b) illustre un service qui implémente notre approche, il est évoqué par un procédé qui sera exécuté par le service

un service établit des interactions comme son interface le décrit, pourtant les exigences en termes d'interactions peuvent varier d'un procédé inter-organisationnel à un autre. Une pratique qui nous a semblée intuitive pour résoudre ce problème est d'adapter le comportement d'un service en fonction des critères de décentralisation au moment où il est choisi pour faire partie d'un procédé inter-organisationnel. Pour ce faire, un service peut recevoir un procédé métier quand il est choisi pour un procédé inter-organisationnel et exécuter ce procédé reçu pour établir les interactions appropriées avec d'autres services qui exécutent leur propre procédé reçu. Il est à noter que les procédés incluent les activités appropriées pour l'établissement des interactions exigées[Yildiz and Godart2007f]. Ce mode de travail est illustré dans la figure 2.2. Avec cette approche, la composition illustrée dans la figure 2.1 est exécutée comme décrite dans la figure2.3. Dans cette dernière, Assurance définit un procédé centralisé qui compose les services utilisés. Par la suite, il utilise ce procédé pour en dériver des fragments qui seront exécutés par les services composés. Les fragments de procédé sont déployés vers les services composés par le service Assurance. Cela leur permet d'établir des interactions égal-à-égal en les exécutant.

La gestion décentralisée des procédés métiers inter-organisationnels où les participants peuvent avoir des interactions configurables n'est pas une idée nouvelle. Pour ce faire, les propositions existantes intègrent plusieurs niveaux de couches logicielles dont les participants doivent disposer. Le but de ces derniers est supporter des protocoles de coordination sophistiqués entre les participants. Par nature, ces couches d'applications sont souvent tellement complexes que assumer leur existence dans le contexte du Web est utopique. Sans aller plus loin dans les limitations des approches existantes qui sont largement approfondies dans la suite, nous pouvons dire que la façon de voir une exécution décentralisée avec les procédés métiers déployés au près des services composés est tout à fait nouvelle. La motivation globale de cette approche qui nous servira de fil directeur tout au long de cette thèse découle de ces différents efforts de standardisation.

Les efforts de standardisation des services Web sont fondés sur XML (eXtensible Markup Language). XML est un standard de représentation de données semi-structurées. Dans une description XML, la structure de données est fournie par le biais de l'utilisation de balises. La représentation des documents XML n'est pas aussi rigide que celle d'une base de données relationnelle par exemple. En se basant sur XML, les standards des technologies services Web

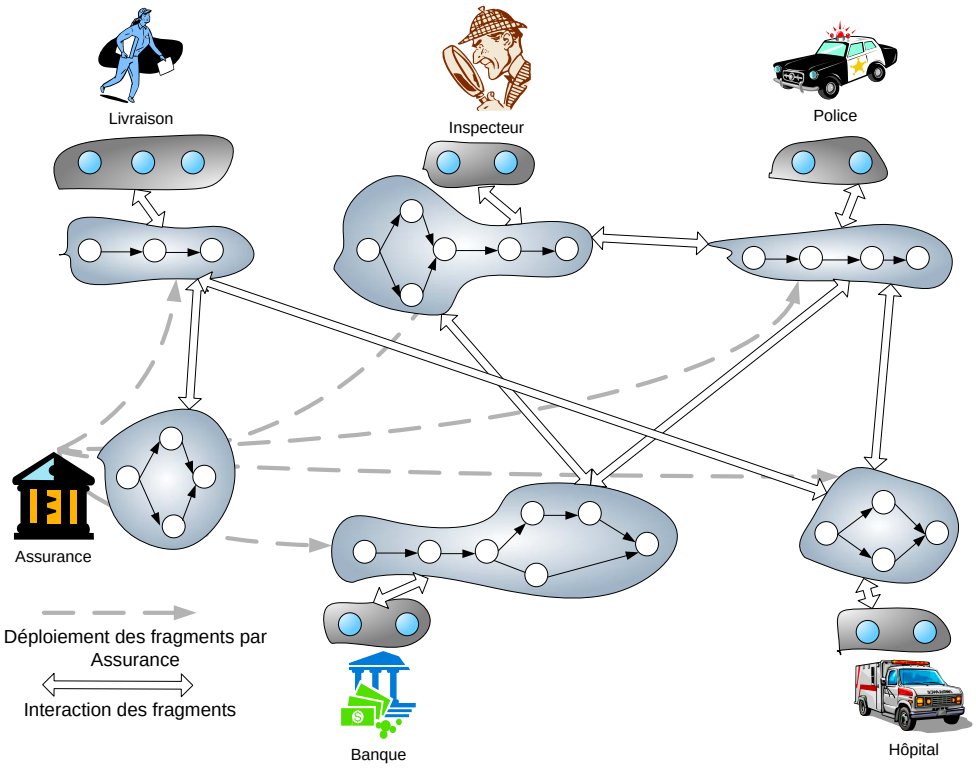


FIG. 2.3 – L'application de l'approche au procédé de l'exemple de motivation

fournissent une base solide pour le développement des procédés métiers inter-organisationnels. De plus, les spécifications codées en ces standards peuvent être échangées entre différentes plate-formes et même différentes organisations. Par exemple, un procédé inter-organisationnel codé dans un langage standard comme WS-BPEL, peut être transmis d'une organisation vers une autre. Comme il s'agit d'une spécification standard, l'organisation qui reçoit le procédé peut l'exécuter facilement sans avoir un niveau de logiciel additionnel à part un moteur d'exécution WS-BPEL. Ceci est valable pour les procédés WS-BPEL instanciés. La caractéristique de cette approche est illustrée dans la figure 2.2.

Cette façon de voir les choses est totalement nouvelle comparée aux environnements de composition de services classique. La plupart des outils de composition ne sont que des applications qui cherchent la cohérence syntaxtique entre les interfaces des services mais ne fournissent qu'une aide très relative à la spécification et à la satisfaction du comportement attendu du service dans le procédé inter-organisationnel. De la même façon, les services ne cherchent pas à s'impliquer dans un procédé d'une manière collaborative en interagissant avec les autres services. Ils ont une approche passive d'où ils répondent aux appels reçus.

Conformément à ce qui a été requis, cette approche ne nécessite pas la présence de niveaux logiciels sophistiqués que les services doivent implémenter, parce qu'un moteur d'exécution de procédé standardisé qui existe derrière les interfaces des services peut être facilement fourni par les organisations. Les questions essentielles de cette approche sont *comment fournir les procédés qui doivent être exécutés par les services (organisations) composés dans le cadre du procédé inter-organisationnel ? Quelles sont les restrictions à respecter ? Comment un procédé peut être déployé vers un service ?* Cela est d'autant plus vrai qu'il n'existe pas actuellement de

consensus sur la façon de développer un procédé métier d'une manière automatique en utilisant des services existants. Il suffit pour s'en convaincre de constater la multitude des propositions existantes. Dans la section suivante, nous discutons les étapes essentielles de notre démarche pour donner un aperçu compréhensible de notre contribution.

2.4.2 Les grandes lignes de la démarche

Dans cette section, nous présentons les grandes lignes de notre démarche. Comme souligné dans les sections précédentes, notre objectif est de fournir un environnement d'exécution décentralisée pour les procédés métiers inter-organisationnels. Pour ce faire, nous utilisons des procédés dynamiquement déployés au près des services composés. La conception des techniques implique le choix d'un ensemble de formalismes adaptés à la fois à leur expression et à la maîtrise de leur élaboration. Notre approche nécessite une description du procédé inter-organisationnel qui compose les services d'une manière centralisée. Cette description centralisée peut être fournie en utilisant différentes techniques de composition automatique [Berardi *et al.*2005] [Bultan *et al.*2006] [Pistore *et al.*2005]. Nous supposons que cette description est faite dans une étape antérieure qui ne nous concerne pas dans cette thèse. Similaire à une description d'un workflow, cette description exprime le flux de contrôle des invocations des opérations appartenant aux services composés. Dans un premier temps, il est nécessaire de fournir une technique qui fournit les procédés coopérants qui correspondent à la description centralisée tout en gardant la même sémantique. Cette technique doit démontrer comment un procédé inter-organisationnel peut être décentralisé sans considérer des restrictions spéciales comme les politiques du flux d'information de notre exemple de motivation. Cette étape nécessite une considération formelle de la description centralisée. Parce que la préservation de la sémantique dans une exécution décentralisée n'est pas une tâche triviale et nécessite de lever les ambiguïtés. De la même façon, une deuxième considération formelle doit porter sur les restrictions qui spécifient les interactions exigées des service composés. Ces restrictions peuvent être des restrictions par défaut qui doivent être considérées pour n'importe quelle opération de décentralisation qui fournit des procédés coopérants ou bien des restrictions qui caractérisent les exigences spéciales des acteurs principaux comme les services et les concepteurs. La définition des restrictions qui gouvernent l'opération de décentralisation doit être flexible pour permettre leur extensibilité vers des restrictions diverses qui ne seront pas détaillées dans cette thèse. Soulignons aussi que les procédés décentralisés, les restrictions ainsi que le opérations qui les traitent, doivent être décrits d'une manière générique indépendamment des langages de spécification. Conformément à ce qui a été présenté dans la figure 2.2, le transfert d'un procédé d'un client vers un service nécessite une architecture de service spéciale. Même si cette dernière n'est pas une architecture qui nécessite des protocoles aussi sophistiqués que ceux des travaux liés, elle nécessite néanmoins d'être approfondie pour valider le concept.

2.4.3 Contributions de la thèse

Cette section résume les contributions de cette thèse.

1. Nous introduisons une méthodologie et des outils pour décentraliser un procédé inter-organisationnel centralisé en respectant la sémantique de ce dernier et en allant vers des interactions d'égal-à-égal entre les services composés. Dans un premier temps, nous introduisons une technique qui formalise cette opération sans se baser sur des critères précis. Il s'agit des structures de données et des algorithmes qui respectivement caractérisent et

traitent les procédés métiers en vue de fournir les procédés coopérants que les services composés doivent mettre en œuvre.

2. Ensuite, nous introduisons un langage de base pour exprimer les restrictions imposées aux interactions d'égal-à-égal des services composés. Ce langage est basé sur la modélisation de la séparation des devoirs dans les systèmes de gestion de workflow. Nous présentons également les algorithmes qui traitent ce langage en vue de spécifier les interactions des services. Par la suite, les techniques développées dans la première contribution sont adaptées à cette deuxième contribution qui s'agit de configurer en permettant la configuration des interactions des procédés coopérants fournis.
3. La troisième contribution de cette thèse se penche sur la sélection dynamique des services pendant l'exécution décentralisée d'un procédé inter-organisationnel avec les procédés coopérants. Dans un premier temps, il s'agit d'identifier les situations problématiques que peuvent exister entre les procédés coopérants déployés dynamiquement. Nous proposons ensuite une technique pour empêcher de telles situations.
4. La dernière contribution de cette thèse concerne la validation des propositions conceptuelles ci-dessus. Outre la production des procédés coopérants en utilisant la spécification WS-BPEL, nous présentons une architecture de référence pour les services qui peuvent supporter ce mode de travail. Nous présentons également les résultats de nos expérimentations.

Il est à noter que les contributions conceptuelles nécessitent une base formelle solide pour fournir les preuves formelles concernant l'équivalence de l'exécution centralisée et l'exécution décentralisée ainsi que la complétude des restrictions et la complexité de leur traitement. Nous avons également complété nos contributions par des analyses formelles.

2.5 Synthèse

La difficulté de mise en œuvre des systèmes de gestion de procédés métiers tient aux problèmes de spécification, d'organisation, de développement, d'intégration et de vérification d'ensemble de facteurs nombreux et compliqués. Dans un contexte inter-organisationnel, ces problèmes sont aggravés pour les services répartis par la difficulté de maîtriser dynamiquement leur autonomie et la multitude de leur interactions. Même si les technologies de services Web aident à maîtriser les problèmes sous-jacents de communication, l'analyse et la configuration des interactions sont d'autant plus difficiles à contrôler que les exigences des services et des concepteurs peuvent être variées, contradictoires, et qu'un fonctionnement réparti empêche toute vision globale.

Les systèmes de gestion décentralisés du workflow introduisent des avantages intéressants pour maîtriser les difficultés actuelles des procédés métiers inter-organisationnels orientés services. Une telle exécution décentralisée peut correspondre aux réels besoins des procédés inter-organisationnels si elle combine les avantages respectifs des initiatives complémentaires. Les efforts de standardisation des services Web facilitent la réalisation des exécutions décentralisées.

Cette thèse s'inscrit dans ce contexte. Nous introduisons une nouvelle approche pour l'exécution décentralisée des procédés inter-organisationnels orientés services. Cette approche est motivée par les efforts de standardisations et leur apports secondaires. Il s'agit de prendre une description centralisée d'un procédé inter-organisationnel et de dériver des procédés coopérants qui implémentent les sémantique de ce dernier avec des interactions d'égal-à-égal entre les services composés. Apparemment simple, cette approche cache un grand nombre de concepts et de

sous-entendus. Elle suppose tout d'abord que les services composés peuvent recevoir des procédés qui seront déployés par leur client et les exécuter. Les services développés à l'heure actuelle ne fonctionnent pas de cette manière. En conséquence, notre approche nécessite des considérations architecturales même si ces dernières sont relativement simples à réaliser. Il est tout à fait naturel que la conception des exécutions décentralisées implique le choix d'un ensemble de critères adaptés aux domaines d'application visés par la décentralisation. Pour ce faire, nous introduisons un langage simple mais flexible et extensible pour caractériser les critères qui peuvent exister dans un contexte inter-organisationnel. Ces critères portent sur des aspects qui concernent, entre autres, la sécurité, l'intimité et la fiabilité des services ainsi que le point de vue des concepteurs. Pour ce faire, le procédé inter-organisationnel ainsi que les critères imposés à la décentralisation doivent être bien compris par ceux qui devront fournir une exécution décentralisée équivalente. Aussi, notre approche décompose le problème en sous-problèmes plus faciles à appréhender.

Chapitre 3

Etat de l'art

Sommaire

3.1 Introduction

Il existe à l'heure actuelle peu de propositions qui permettent de comprendre et de maîtriser l'exécution décentralisée d'un procédé qui compose un ensemble de services. Les raisons de la difficulté de compréhension sont multiples. Une difficulté essentielle tient aux problèmes de spécification d'un procédé inter-organisationnel comme une composition de services. Les problèmes sous-jacents de communication au niveau de l'intégration des services composés sont d'autant plus difficiles à comprendre que les paradigmes de modélisation sont abstraits. Parce que la conception et l'implémentation impliquent le choix d'un ensemble de technologies adapté à la fois à l'expression des procédés et à leur exécution sur un réseau de communication. La compréhension se complique considérablement si on pense aux motivations des exécutions décentralisées.

Dans ce chapitre, nous faisons le parcours de la littérature relative à notre travail dans l'objectif de présenter un état de l'art. Dans la première partie de ce chapitre, nous présentons l'approche orientée service et ses technologies qui constituent le niveau d'implémentation de notre approche. Il est à noter qu'une idée clé de notre approche est l'utilisation des standards introduits pour l'approche orientée service. Le rôle de la première section, en permettant une introduction progressive des concepts par rapport aux définitions habituelles dans le domaine, est essentiellement pédagogique. Nous cherchons à expliquer l'apport des efforts de standardisation à l'exécution décentralisée d'un procédé.

La deuxième partie de ce chapitre porte sur la présentation des systèmes d'information décentralisés et pair-à-pair. Dans un premier temps, nous présentons un ensemble de caractéristiques qui spécifient des systèmes décentralisés. Nous cherchons à en souligner les avantages dans un contexte aussi bien organisationnel qu'opérationnel. Ensuite, nous discutons judicieusement les contraintes, les défis et les limitations de la décentralisation.

La troisième partie est consacrée à la présentation des travaux qui portent sur la modélisation des procédés par le biais de l'initiative workflow. Cette partie s'appuie, d'un côté sur la modélisation des procédés par des worflows, d'un autre côté sur l'étude des différents modalités d'exécution d'un worflow. Nous décrivons enfin les travaux similaires à nos contributions. Par une analyse critique, nous montrons les intérêts et les limites respectifs ainsi que les aspects complémentaires des différents travaux liés.

Compte tenu du foisonnement actuel du domaine, il est inévitable que l'état de l'art ne traite pas de manière exhaustive l'ensemble des travaux réalisés. Néanmoins, nous croyons que l'analyse faite dans ce chapitre couvre raisonnablement bien les principaux travaux pertinents, que ce soit en matière de systèmes d'information décentralisés et égal-à-égal (P2P pour peer-to-peer en anglais), de modèles de workflows, ou d'approche orientée service.

3.2 L'approche orientée service et ses technologies : *Le dogme*

Les systèmes d'information sont basés aujourd'hui de plus en plus sur des technologies standardisées de l'Internet. Les efforts de standardisation dans ce contexte ont permis l'émergence des services Web [Alonso *et al.*2004] comme support concret de développement des applications accessibles par Internet. Ainsi, les technologies associées aux services Web sont devenues incontournables pour le développement des applications interagissant les unes avec les autres par le biais de l'Internet [Weerawarana *et al.*2005].

La notion de service désigne une application logicielle mise à disposition sur l'Internet par un fournisseur, et accessible par des clients à travers des protocoles standards [Curbera *et al.*2002]. Le service définit une abstraction. Il procure une application logicielle à des clients. Il possède une référence unique qui l'identifie sur l'Internet. Un service est encapsulé. Les clients ne peuvent accéder, utiliser, modifier un service que par l'appel aux opérations mises à leur disposition par le service lui-même. Chaque service est ainsi propriétaire et responsable de ses données et ses opérations. Historiquement, la notion d'objet, qui est apparue il y a une quinzaine d'années, donnait le premier élément de réponse à la problématique mais la notion de service l'a porté à un nouveau niveau [Aoyama *et al.*2002]. La programmation par objets a apporté un ensemble de principes de conception dont la notion de service a hérité. Pourtant, il y a des différences importantes entre la programmation par objets et l'approche service. Par exemple, les objets sont dépendants des langages et d'un environnement d'exécution. Les objets sont crées et détruits localement après leur utilisation alors que les services ne les sont pas. Les objets sont locaux au système qui les gère alors que les services sont accessibles sur un réseau. En conséquence la

gestion des exceptions est plus sophistiquée [Curbera *et al.*2003]. On peut multiplier les exemples qui séparent le paradigme service et le paradigme de la programmation par objet.

3.2.1 Pourquoi et comment employer l'approche orientée service ?

L'approche orientée services est un nouveau style de conception et d'implantation permettant aux entreprises de créer rapidement de nouvelles applications distribuées. Plusieurs définitions des services et en particulier des services Web sont mises en avant par plusieurs auteurs. Pour donner une définition précise, nous considérons la définition proposée par le consortium W3C : *A Web service is a software system designed to support machine-to-machine interaction over a network. it has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP [Box* et al.*2000] messages, typically conveyed using HTTP with an XML serialization in conjuntion with other Web-related standards.*

L'objectif visé avec le développement des technologies des services Web est une meilleure considération de l'interopérabilité des applications tournant sur des plate-formes hétérogènes. Par rapport à d'autres plates-formes pour le développement d'applications distribuées telles que CORBA [cor] et Java RMI[jav], l'une des différences primordiales est que les services Web n'imposent pas de modèles de programmation spécifiques. En d'autres termes, les services Web ne sont pas concernés par la façon dont les messages sont produits ou consommés par des applications orientées services Web. Ceci permet aux vendeurs d'outils de développement d'offrir différentes méthodes et interfaces de programmation au-dessus de n'importe quel langage de programmation, sans être contraints par des standards comme c'est le cas de CORBA qui définit des ponts spécifiques entre le langage de définition IDL et différents langages de programmation. Ainsi, les fournisseurs d'outils de développement peuvent facilement différencier leurs produits avec ceux de leurs compétiteurs en offrant différents niveaux de sophistication. Par exemple, il est possible d'offrir des environnements pour des méthodes de programmation de services Web minimalistes au-dessus de plates-formes relativement légères comme c'est le cas de NuSOAP pour le langage PHP ou d'Axis. En revanche, des plates-formes plus lourdes telles que BEA WebLogic, IBM Websphere ou .Net offrent un environnement pour des méthodes de développement très sophistiquées [Dumas and Fauvet2006]. Bien qu'elle facilite le développement des applications réparties, l'aspect technique n'est pas la seule raison qui explique l'adoption rapide des technologies des services Web. Nous allons détailler ces caractéristiques dans les sections suivantes.

La plupart des entreprises produisent et vendent leur produits à leur clients afin de réaliser les objectifs du métier tels que les objectifs financiers. Leur procédés métiers spécifient comment les produits sont conçus, produits, mis sur le marché, distribués et vendus. D'un point de vue abstrait, le procédé métier d'une organisation est une projection des produits qu'elle offre. Donc, la vitesse du changement de l'infrastructure ou de la création de nouveaux procédés métiers correspondent à la vitesse du changement ou création des produits. Afin de survivre dans un marché hautement dynamique et ouvert aux changements réguliers, les entreprises doivent être flexibles. Ici, la flexibilité veut dire la capacité de réagir rapidement, de préférence plus vite que la compétitivité, aux exigences des clients, nouveaux produits offerts par les compétiteurs existants et de nouveaux et aux avancement de technologies. En conséquence la flexibilité d'une entreprise est la projection de sa capacité d'adaptation de son procédé métier. Les constituants de la définition d'un procédé métier peuvent être traduits directement en actions qui sont nécessaires pour changer le procédé : changer le flux des activités du procédé, changer les acteurs qui exécutent les activités ; changer les outils qui supportent ces activités, ainsi de suite. Cela peut

aussi fournir un contexte où l'exécution des activités du procédé doit être déplacé vers les partenaires. Dans ce contexte, un certain nombre d'aspect organisationnels expliquent l'acceptation rapide des technologies services Web.

Le besoin d'un couplage faible La possibilité d'établir des communications avec les services, qui sont découverts dynamiquement, exige un couplage faible entre le client et le fournisseur du service à évoquer. La notion de couplage faible est une des caractéristiques essentielles des technologies des services Web. Cette notion exclut toutes les hypothèses à propos des plate-formes spécifiques que le client ou le fournisseur impose. Ceci est valable pour le format et le protocole qui caractérisent l'interaction de ces derniers. Ces caractéristiques sont à base de l'adaptation des services Web pour leur utilisation au sein des procédés métiers.

Le besoin d'intéropérabilité Aujourd'hui, différentes technologies pour réaliser des applications distribuées existent telles que Common Object Request Broker Architecture (CORBA) [cor], Java 2 Platform, Enterprise Edition (J2EE) [Haugland *et al.*], et Component Object Model (COM)[com1993], chacune en se basant sur différent concepts et modèles d'objet. En conséquence, l'intéropérabilité entre ces plate-formes est difficile. Dans un premier temps, la difficulté tient aux formats de messages échangés et aux protocoles de communication employées. Par la suite viennent des problèmes d'intéropérabilité de haut-niveaux tels que le support transactionnel, la sécurité et l'intimité. Par exemple, une simple tâche qui est exécutée sur plusieurs plate-formes avec des supports transactionnels différent ne peut pas mettre en œuvre une caractéristique transactionnelle cohérente. Les services Web fournissent un ensemble de technologies (WS-*) adaptées à l'expression des exigences dans un contexte de programmation répartie.

Le besoin pour la composition de services Si on compare aux technologies existantes telles que les technologies client/serveur, les applications Web et de nombreuses autres que nous avons mentionnées dans les sections précédentes (i.e. CORBA, COM), les services Web permettent les échanges entre entreprises, indépendamment d'un langage de programmation donné, selon une orientation messages tout en supportant différent modes de transport de ces dernier. Par rapport aux autres applications Web, les services Web fonctionnent sur des interactions applications-applications. Cela permet de les intégrer dynamiquement par composant, et naturellement, offre la possibilité d'agréger des services. Cette nouvelle tendance pour gérer ces procédés inter-organisationnels consiste à composer les services. L'impact sur les procédés métiers est double, car d'une part les procédés orientés service sont exposés comme des services, et d'autre part, ils sont également composés de services.

Le besoin pour la mobilité des spécifications entre différentes plate-formes et outils A l'heure actuelle, il existe un grand nombre de propositions et standards qui concernent la spécification des services Web. Nous allons nous intéresser aux standards dans les sections suivantes. Dans un premier temps, l'objectif de ces standards est de proposer un langage formel pour la définition et le raisonnement sur les services d'une manière non-ambiguë. Au niveau d'un service atomique, un standard doit permettre de décrire son comportement et ses interfaces ; il doit permettre la spécification de flots de contrôle ou de comportements transactionnelles quand il s'agit d'une composition. Une des caractéristiques majeures de ces spécifications standards concerne la mobilité des spécifications entre les différents outils, plate-formes et entreprises. Ceci est une conséquence secondaire de leur structure basée sur XML.

Dans un environnement caractérisé par l'ouverture, la normalisation et la concurrence, le commerce électronique permet de dématérialiser les flux dinformation qui composent les échanges

inter-entreprises et accélèrent ainsi les transactions en les automatisant et en réduisant leurs coûts. Mais il faut des intérêts communs et une volonté partagée pour mettre en oeuvre des échanges inter-entreprises. Il faut aussi disposer de normes et de standards pour partager des données décrites dans un vocabulaire compréhensible par tous les partenaires impliqués. C'est là le fondement des modèles normatifs orientés échanges de données (EDI) qui ont démontré leur validité malgré une complexité et un temps de mise en oeuvre qui a limité leur déploiement aux grandes entreprises et à leur fournisseurs. Pour impliquer les PME dans ces chaînes d'échanges, d'autres normes, plus simples à implémenter et adaptées à l'Internet, sont aujourd'hui proposées. Le format d'échange qui fait l'unanimité repose sur XML, plus simple, plus flexible et plus interopérable que les formats EDI EDIFACT (en Europe) ou X.12 (au États-Unis). Malgré cet effort, le taux de pénétration de l'EDI de génération XML reste faible et a des difficultés à convaincre. Les raisons sont multiples, parmi lesquelles on peut citer :

- Une rigidité des offres très verticalisées (catalogue et processus d'achats, appels doffres) identique à celle du monde EDI,
- Une valeur ajoutée insuffisante pour justifier l'équipement ou le remplacement d'un existant EDI,

Allant dans le sens de la valeur ajoutée, des projets beaucoup plus ambitieux ont vu le jour tels que ebXML de l'OASIS et l'UN/CEFACT (organisme des Nations Unies à l'origine de EDIFACT). Ces projets complètent le modèle EDI en terme d'interopérabilité et de couverture fonctionnelle, mais ne réduisent pas réellement les coûts et délais de mise en œuvre de par leur complexité et leur côté fortement normatif. Pour ces raisons les services Web trouvent également leur place dans un contexte de mise en œuvre d'échanges B2B car ils apportent une alternative très peu contraignante aux entreprises et une implémentation moins coûteuse compatible avec la démarche d'expérimentation. Les services Web sont ainsi une étape pragmatique vers la mise en œuvre de solutions de type ebXML en permettant aux entreprises de préparer leur système d'information aux futures normes d'échanges B2B. Le système d'information de la plupart des entreprises est hétérogène et reflète l'évolution technologique des quinze dernières années : du mainframe au client/serveur puis du client/serveur au Web. Dans ce contexte, de plus en plus de projets « transversaux » voient le jour allant de simples besoins d'échange de données entre applications jusqu'aux collaborations étroites de ces applications au sein de procédés métiers. Pour répondre à ces besoins ont été conçues les plates-formes d'EAI (Enterprise Application Integration), outils d'infrastructures logicielles qui, sans remise en cause de l'existant, permettent d'en assurer l'ouverture et la flexibilité. Les services Web permettent de franchir une nouvelle étape en proposant un modèle d'architecture adapté à ces besoins.

3.2.2 Technologies relatives basées sur XML

Dans cette section, nous discutons les technologies liées à la réalisation des services Web. Par la suite, nous présentons la problématique liée à la composition des services ainsi que les standards correspondants. Les services Web distinguent plusieurs niveaux d'implémentation et de technologies correspondantes. Le premier niveau couvre les fonctions minimales de publication, découverte et liaison de services. Les spécifications présentées sont fondés sur XML (eXtensible Markup Language) [W3C]. XML est un standard de représentation de données semi-structurées. Dans un document XML, la structure des données est fournie par le biais de l'utilisation de balises (comme en SGML Standard Generalized Markup Language [Goldfard1990], mais en s'affranchissant des aspects liés à la présentation des données). Cette structure n'est pas aussi rigide que celle d'une base de données relationnelle par exemple. Dans un document XML, les données peuvent être définies selon un schéma, mais cela n'est pas obligatoire et le schéma peut

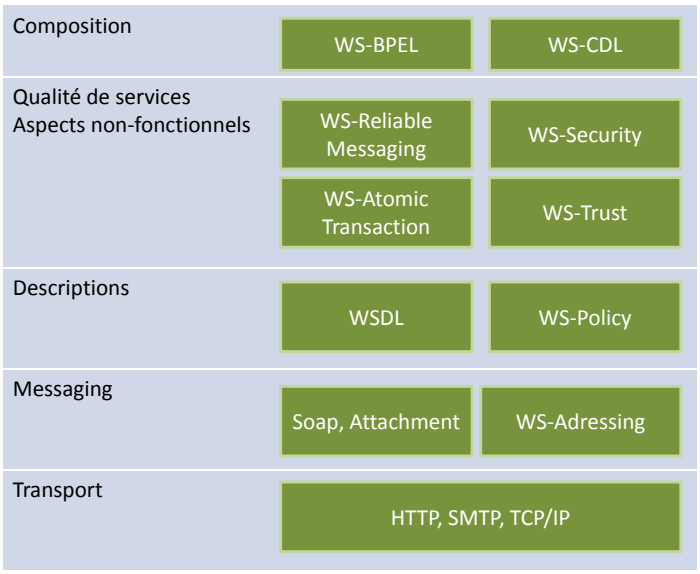


FIG. 3.1 – Les exemples de technologies et de standards des services Web

laisser quelques parties partiellement spécifiées.

XML que nous avons expliqué brièvement ci-dessus, est utilisé à plusieurs niveaux. La description de l'interface d'un service Web va permettre d'apporter les informations techniques sur le service. Ces descriptions seront échangées entre les différents acteurs pour qu'ils puissent prendre connaissance de ces informations techniques, étape préalable et nécessaire à l'intégration des services dans le système d'information.

WSDL

WSDL *(Web Services Description Language)*[Chinnici *et al.*2002] est un langage de la famille XML permettant de décrire les types des données echangées et les fonctions fournies par un service Web. L'objectif est de fournir la description des services indépendamment de leur implémentation sous une forme que des personnes ou des programmes peuvent interpréter. Pour cela, WSDL joue un rôle primordial dans les technologies de services Web. Les descriptions WSDL sont en fait équivalentes à des méthodes publiques d'une classe Java. La définition de WSDL est composée de deux parties : La description *abstraite* et la description *concrète*. La partie *abstraite* contient la définition des messages qui peuvent être reçus et fournis par le service et la signature des opérations supportées sans aucune considération sur la technologie utilisée pour effectivement implémenter le service (i.e. où le service est disponible physiquement). La partie *concrète* de la définition du WSDL décrit ce que fait un service en termes de messages échangés. Dans un premier temps, l'objectif de ce niveau est de décrire comment formater les messages pour interagir physiquement avec un service donné. Elle contient essentiellement l'indication du protocole utilisé pour échanger des messages avec le service (SOAP au-dessus de HTTP) et les associations entre la description de l'interface abstraite du service et les types de messages supportés par le protocole de communication sous-jacent (par exemple SOAP).

La production d'une description WSDL qui correspond à une application logicielle est une tâche fastidieuse dont le développeur peut être déchargé par l'utilisation d'outils pour la génération automatique de la description WSDL. L'exemple par excellence peut être un outil qui

prend en entrée la description du service d'une classe Java ou d'un objet COM et fournit la description équivalente en WSDL en sortie. La deuxième version de WSDL (WSDL 2.0) inclut des fonctionnalités que la première version ne supporte pas. Comme synthèse de WSDL, on peut noter les points suivants. WSDL décrit essentiellement l'accessibilité d'un service et pas qu'est-ce que le service peut faire. WSDL ne décrit pas comment un service effectue le service.

La technologie de base de la couche communication des services Web est le protocole SOAP. Ce protocole permet de mettre en oeuvre deux types d'échanges : le mode RPC (Remote Procedure Call) et le mode message/document. Un certain nombre de technologies complémentaires permettent de gérer plus finement le routage des messages dans une infrastructure réseau, ainsi que la sécurité.

SOAP

Les interactions entre services Web s'effectuent par le biais d'envois de messages structurés au format XML. Le protocole SOAP (Simple Object Access Protocol) [Box *et al.*2000] fournit le cadre permettant ces échanges. SOAP est originellement issu de tentatives précédentes visant à standardiser l'appel de procédures à distance, et en particulier de XML-RPC [Winer 1999]. Mais à la différence des technologies RPC, SOAP n'est pas fondamentalement lié à la notion d'appel de procédure. En effet, SOAP vise à faciliter l'échange de messages XML, sans se limiter à des messages dont le contenu encode des paramètres d'appel de procédure et sans favoriser des échanges bidirectionnels de type requête-réponse comme c'est le cas des protocoles RPC. Dans le jargon des services Web, SOAP permet d'encoder des interactions orientées-RPC mais aussi des interactions orientées-document. Une autre caractéristique de SOAP est de faire abstraction de la couche de transport sous-jacente. Bien que la pratique la plus répandue soit d'utiliser SOAP au-dessus de HTTP, il existe aussi des implantations de SOAP au-dessus d'autres protocoles tels que le protocole d'échange de messages électroniques SMTP, et les protocoles de transport orientées-message de Microsoft et IBM, à savoir MSMQ et MQSeries respectivement. La manière d'implanter SOAP au-dessus d'un protocole de transport donné est appelée une liaison SOAP (<SOAP binding > en anglais). Une liaison SOAP définit, en particulier, l'encodage des messages (nécessaire en particulier lorsque le protocole sous-jacent utilise un format binaire), la méthode pour l'échange de messages, l'encodage des noms d'opérations (appelés <SOAP Actions >), et la façon dont différents messages (y compris les messages d'erreur) appartenant à la même interaction sont corrélés. Par exemple, la liaison SOAP au-dessus de HTTP définit que les messages sont encodés dans un < type de média > appelé < application/soap+xml > (c'est-à-dire en XML avec quelques extensions), que le nom de l'opération correspondant à une requête est donné dans une en-tête HTTP appelée <SOAPAction >, et que les messages dans une interaction sont échangés au travers des méthodes POST et GET fournies par HTTP. D'autres règles (dont nous ne donnons pas les détails) définissent la manière dont les messages appartenant à un même échange (y compris les messages d'erreur) sont corrélés en exploitant les caractéristiques des méthodes POST et GET. Outre la définition du moyen des messages en faisant abstraction de la couche de transport, SOAP définit une structure standard de messages dans laquelle le contenu des messages est séparé des méta-données liées à l'échange des messages. Ainsi, un message SOAP est constitué de deux parties : un en-tête et un corps. L'en-tête indique l'objet du message (l'appel d'une opération ou le retour de résultats), la description de l'expéditeur, et l'information requise pour acheminer le message au destinataire. Le corps du message peut contenir : (i) un document quelconque ; (ii) l'appel d'une opération offerte par le service destinataire, avec les valeurs pour les paramètres d'entrée ; (iii) les valeurs produites en résultat d'une appel ; ou bien (iv) un message d'erreur. Ainsi, SOAP offre diverses possibilités d'interactions entre les services :

soit des échanges de documents, soit des interactions de type RPC. Il existe plusieurs mécanismes pour construire, analyser, et échanger des messages SOAP dans des langages variés tels que Java, C++, Perl, C, etc. Ces implantations permettent de générer les en-têtes de messages SOAP et de mettre en correspondance le contenu du corps du message avec les structures de données définies dans le langage.

UDDI

Une problématique généralement associée à la description de services est celle de leur publication et leur découverte. Un standard proposé pour s'attaquer à cette problématique est UDDI [Draft]. UDDI définit une interface de programmation pour publier des descriptions de services dans des répertoires dédiés, pour soumettre des requêtes à base de mots-clés sur ces répertoires et pour naviguer au travers des descriptions obtenues par le biais de ces requêtes. Etant donné l'existence de moteurs de recherche sophistiqués aussi bien pour des Intranets qu'au niveau de l'Internet tout entier, ainsi que l'existence d'autres technologies pour gestion de répertoires telles que LDAP [Koutsonikola and Vakali2004], la valeur ajoutée apportée par UDDI est difficile à identifier [Dumas and Fauvet2006]. Par conséquent, nous ne détaillons pas UDDI dans l'état de l'art comme il est plutôt lié à la découverte des services et pas à leur composition.

3.2.3 La composition des services

La composition des objets, des modules logiciels et même des procédés métiers est, un sujet d'étude actuel depuis de nombreuses années pendant lesquelles les développeurs ont créé de nombreuses propositions de langages et de systèmes. Pourtant, aucune proposition à jour n'est conçue pour être utilisée dans le contexte des architectures orientées services. Les architectures orientées services exigent de nouvelles approches qui pour maîtriser, les interactions entre des entités faiblement couplées, la composition dynamique sur demande, les changements fréquents tels que la localisation, la disponibilité, les qualités de service, et la manque de contrôle sur la plate-forme et l'implémentation des services composés.

Dans un tel contexte, plusieurs exigences ont été identifiées pour un modèle de composition idéal :

- **L'intégration flexible :** Le modèle de composition doit être suffisamment riche pour exprimer les scénarios que les organisations partenaires veulent implémenter. Plus important, le modèle doit s'adapter rapidement aux changements des services qu'il compose.
- **La composition récursive :** Fournir un procédé métier comme un service doit permettre de représenter plusieurs vues du même service aux différents clients.
- **La séparation et la composition des préoccupations :** En considérant la composition des services dans une plate-forme, la logique du procédé métier doit être découplée des mécanismes secondaires telles que la qualité de service, la coordination des protocoles et les messages.
- **Les conversations et la gestion du cycle de vie :** La composition doit définir le cycle de vie dans lequel les interactions conversationnelles sont intégrées.
- **La gestion des échecs :** Les modèles de composition ont besoin de systèmes de gestion d'échecs pour détecter et gérer les exceptions pendant la période de conception et d'exécution.

Les efforts de standardisation concernant la composition des services sont structurées autour de deux approches essentielles. La figure 3.3 illustre l'évolution des standards. La première approche qui suit l'approche centralisée du workflow s'appelle *Orchestration*. La deuxième ap-

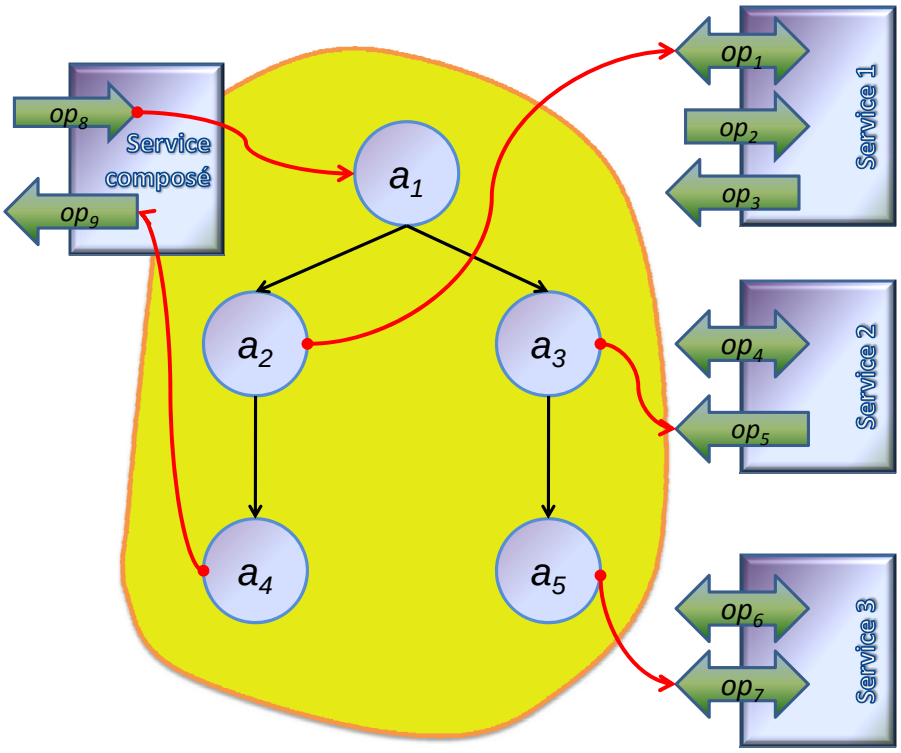


FIG. 3.2 – Exemple d'orchestration

proche qui est plus récente et en conséquence moins comprise est la *Chorégraphie*. Ces deux approches peuvent être présentées comme ci-dessous.

Orchestration

L'objectif de l'orchestration est de définir une spécification de la composition d'un point de vue d'une entité centralisée qui exécute la spécification et effectue la coordination des services. Comme souligné dans les sections précédentes, c'est la vision de l'approche workflow classique. Selon l'approche orchestration, une composition est spécifiée en un langage impératif décrivant les dépendances de contrôle et de donnée des activités. Les activités sont des invocations des services composés. La figure 3.2 illustre un exemple d'orchestration où la coordination des services composés (Service 1, Service 2, Service 3) est effectuée par un service (Service composé). La composition considère cinq activités (a_1, a_2, a_3, a_4, a_5) et leur dépendances de contrôle. Les activités correspondent aux invocation des opérations appartenant aux services composés. L'orchestration peut être implémentée en utilisant les spécifications existantes et standards tels que WS-BPEL (Web Services - Business Process Execution Language)[OASIS2005]. WS-BPEL est un des standards prometteurs. Dans le reste de cette section, nous présentons l'approche orchestration en utilisant WS-BPEL.

Connu sous le nom BPEL4WS(la première version BPEL4WS V1.0 date de juillet 2002 et a été proposée par BEA Systems, Inc. IBM Corporation et Microsoft), renommé WS-BPEL par la suite, cette spécification s'appelle aussi BPEL. Historiquement, cette version était la fusion

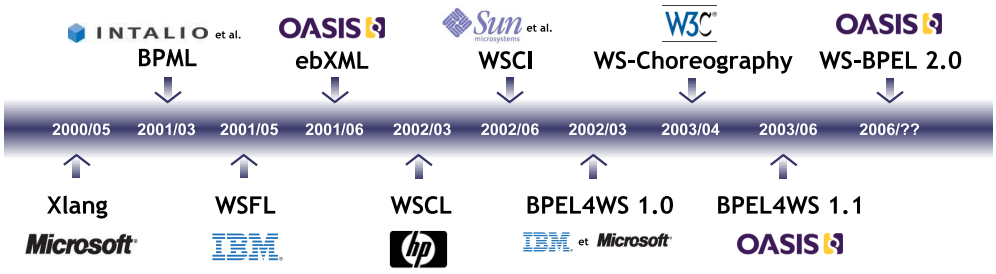


FIG. 3.3 – Evolution des efforts de standardization

de deux propositions antérieures faites par IBM et Microsoft qui s'appelaient respectivement Web Services Flow Langugage, WSFL [wsf] et XLANG [Thatte2001]. En 2003, la version 1.1 de BPEL est soumise à OASIS (Organization for the Advancement of Structured Information Standards) pour devenir un standard. Actuellement, une nouvelle version (version WS-BPEL 2.0) est en cours de définition. WS-BPEL est un langage de workflow extensible qui spécifie les interactions des services. La composition des services avec BPEL est récursive parce qu'une composition expose des interfaces WSDL, et que cette interface peut être utilisées d'autres compositions. BPEL est conçu pour être utilisé dans des environnements dynamiques dans lesquels les services composés peuvent rapidement changer. BPEL utilise les niveaux placés en dessus des WS spécifications ainsi que WSDL. Par exemple, en utilisant XPath, les données de BPEL peuvent être accédées et manipulées.

De même, les références des services composés peuvent être dynamiquement changées sous forme de WS-Addressing. Avec l'extensibilité de BPEL, un concepteur de composition peut attacher des aspects additionnels comme la qualité de service exigée avec WS-Policy et WS-PolicyAttachements. En BPEL, la logique des interactions entre un service et son environnement est décrite par le biais d'une composition d'actions élémentaires de communication (émission, réception ou émission/réception). Ces actions sont reliées les unes aux autres par un flot de contrôle exprimé par le biais de constructions telles que la composition parallèle (avec un nombre fixé de branches), séquentielle, et conditionnelle, des règles de type événement-action et des clauses de capture/transmission d'erreur. La manipulation des données s'effectue par le biais de variables, comme dans un langage de programmation impérative. Plus précisément, un processus exprimé en BPEL est construit à partir d'activités primitives qui peuvent être composées pour définir d'autres activités plus complexes. Les activités primitives sont : la réception (receive) qui bloque l'exécution jusqu'à l'arrivée d'un message correspondant à une opération ; l'appel (`invoke`) et la réponse (`reply`) qui chacune effectue un envoi de message correspondant à une opération ; l'affectation (`assign`) qui associe une expression (décrite en XPath ou XSLT) à une variable ; l'attente (`wait`) qui bloque l'exécution pour une période de temps fixée ; la transmission (`throw`) d'erreur et la fin (`exit`) pour terminer l'exécution. La plupart des opérateurs de composition proposés par BPEL correspondent à ceux déjà présents dans les langages de programmation impérative : la séquence (`sequence`) pour imposer un ordre dans l'exécution, le choix (`switch`) pour exprimer un branchement conditionnel, et l'itération (`while`). Le concept de bloc issu des langages de programmation structurée est présent dans BPEL par le biais de la notion de portée (`scope`). En plus de ces opérateurs de composition, BPEL offre plusieurs constructions pour la programmation concurrente : la composition parallèle (`flow`) pour exécuter plusieurs activités en parallèle ; la sélection (`pick`) pour choisir un message parmi plusieurs entrants ou les timers ; des dépendances de précédences (`link`) peuvent être définies entre des activités qui autrement

s'exécuteraient en parallèle. Finalement, des règles de type événement-action (event handlers) peuvent être associées à des blocs (`scope`). Ces règles sont déclenchées par la réception d'un message ou l'expiration d'un délai, à n'importe quel moment au cours de l'exécution du bloc correspondant. Dans le langage BPEL, tout est service : un processus exécutable est l'implantation d'un service qui s'appuie sur d'autres services. Lorsqu'une ressource, telle qu'une base de données, un fichier, ou une application patrimoniale, est utilisée par un service, il est nécessaire d'exposer le SGBD, le système de gestion de fichiers, ou l'application comme un ensemble de services. Pour pouvoir utiliser BPEL dans des environnements où toutes les ressources ne sont pas forcément exposées comme des services, il est quelques fois judicieux de casser le paradigme *tout service* de BPEL. C'est dans cet esprit qu'a été définie BPELJ [Blow et al. 2004], une extension de BPEL avec laquelle il est possible d'intégrer des bouts de code Java dans un programme BPEL. Cette approche est similaire à celle des JSPs (Java Server Pages) où du code Java est embarqué dans des programmes HTML. La considération de BPEL dans les deux plates-formes .Net et Java laisse à penser que d'autres dialectes proches de BPEL vont émerger.

BPEL est aujourd'hui un standard de facto pour implanter des services Web selon un point de vue orientée procédé. Des plates-formes matures et reconnues, telles que BEA WebLogic, IBM WebSphere, Microsoft BizTalk, SAP XI et l'outil de gestion de procédé BPEL d'Oracle supportent BPEL à des degrés divers démontrant ainsi l'intérêt réel de ce langage. ActiveBPEL est une autre proposition assez complète dans le domaine de l'open source. Bien que BPEL offre des constructions spécifiques pour le développement de services Web, sa complexité pose problème. Comme le montre [Wohed et al. 2003] l'opérateur link est dans une certaine mesure redondant avec les opérateurs switch et flow dans le sens que tout procédé BPEL écrit en utilisant switch et flow peut être reécrit en utilisant un seul flow et un certain nombre de links. Aussi, BPEL manque d'abstractions de haut niveau lorsqu'il s'agit de développer des services qui mettent en jeu des multicasts avec des conditions de synchronisation partielles, comme par exemple dans le cas d'un service client qui requiert des devis de plusieurs services fournisseurs. On peut s'attendre à ce que dans le futur, des efforts de recherche portent sur la résolution de ces problèmes et proposent soit des extensions de BPEL, soit des langages alternatifs.

Chorégraphie

La deuxième approche essentielle pour la composition de services Web est l'approche chorégraphie. L'approche chorégraphie considère une exécution décentralisée avec des interactions P2P entre les services composés. Comme expliqué dans la spécification du Consortium World Wide Web (W3C) qui a publié le premier document de travail du Langage de Description de Chorégraphies pour les Services Web, un des objectifs des services Web est la composition pour réduire leur coût de connectivité et optimiser l'utilité et, par conséquent, la valeur de l'information. Selon l'approche orchestration, la seule manière d'intégrer des services a consisté à les connecter en couplant des points d'intégration ou en utilisant du code impératif comme une spécification WS-BPEL. La chorégraphie, grâce à un modèle global, garantit un comportement contractuel entre services multiples sans recours à aucun couplage centralisé. Il est à noter qu'un autre objectif des services Web est la conformité tels que l'intégration d'applications de manière à partager les mêmes règles d'engagement, afin de garantir le résultat souhaité. Ainsi, les services bénéficient d'une mise sur le marché plus rapide grâce à une chorégraphie bien définie garantissant la conformité entre domaines d'applications. Il peut être démontré statistiquement qu'une chorégraphie est exempte de tout interblocage. Cela veut dire que les procédés des services s'arrêtent, chacun attendant l'un des autres, de cycle non productif (les procédés interagissant continuellement sans réaliser de tâches utiles) et de fuites (interférences de participants non au-

torisés). L'absence de fuites accroît la sécurité entre services connectés. L'absence d'interblocages et de cycles non productifs réduit les coûts de tests, et donc le coût total d'appropriation. Dans ce groupe, on peut donner comme exemple le langage WS-CDL [W3C2005]. Ce langage permet de spécifier les interfaces de services Web pour réaliser une chorégraphie. La collaboration se passe alors dune manière décentralisée, sans utiliser un procédé principal, mais plutôt plusieurs procédés distribués. La spécification WS-CDL définit la collaboration point-à-point entre services Web participants. Un client de service Web, automatisé ou non, peut être un autres service Web, une application ou un être humain. Dans WS-CDL, un ensemble d'interactions clientes peuvent être associées dans le temps dans un « groupe de collaboration ». Ce dernier est, par exemple, un ensemble de composants impliqués dans une opération commerciale ou une transaction de base de données. Le futur des applications de commerce électronique réside dans le couplage lâche et le caractère décentralisé du World Wide Web. Cet environnement requiert la possibilité de réaliser des collaborations point-à-point durables entre services participants, dans ou à travers les domaines de confiance d'une organisation. Les applications implémentant WS-CDL peuvent atteindre ce but commun, le groupe de travail ayant développé son cahier des charges en prenant en compte un large ensemble de besoins commerciaux concrets ainsi que de solides théories. La spécification WS-CDL regroupe des connaissances importantes du monde des entreprises et de la recherche. WS-CDL introduit non seulement les besoins des entreprises, mais aussi les travaux mathématiques séminaux sur le π-calcul, algèbre fondée sur la notion d'interaction et le mécanisme de nommage, et utilisée pour modéliser des systèmes physiquement ou virtuellement mobiles. En conséquence, la chorégraphie définir les interactions des services d'un point de vue décentralisé comme illustré dans la figure 3.4.

3.2.4 Synthèse sur l'approche orientée service

L'essor de l'Internet a permis de faire émerger de nouvelles perspectives dans la mise en place de collaborations entre organisations de toutes sortes (commerciales, gouvernementales ou individuelles). La réutilisation des services accessibles par l'Internet en les composant afin d'en construire de nouveaux, avec valeur ajoutée, est l'une de ces perspectives les plus prometteuses. Les services Web sont une technologie à part entière, dont le périmètre d'application très vaste ouvre de réelles opportunités au sein du système d'information des entreprises. Bien que la technologie de services Web ne soit pas la seule solution pour réaliser les architectures orientées services, elle est la technologie essentielle considérée par l'industrie. Avec des services Web, l'industrie adresse encore une fois le défi fondamental de l'informatique répartie : *comment fournir une manière uniforme de décrire des composants ou des services dans un réseau, de les localiser, et de les accéder ?* L'industrie aborde ce problème en utilisant la technologie et caractéristiques qui sont développées d'une voie ouverte, utilisant des associations d'industrie et de larges consortiums. Les systèmes d'information qui peuvent bénéficier des apports de la technologie des services Web sont nombreux. Parmi les principaux critères qui permettent de les identifier on trouve les besoins suivants :

- la mise à disposition des partenaires de services liés au cœur de métier d'une entreprise, pour permettre d'élargir le réseau de distribution et de créer des offres conjointes plus attractives en exploitant efficacement les synergies que relient chacun des acteurs,
- la commercialisation d'offres ASP (Application Service Provider) conçues pour permettre une étroite intégration avec le système d'information des clients,
- la définition de pratiques et services métiers partagés au sein d'une communauté interprofessionnelle, fédérés par un annuaire,
- la mise en œuvre d'échanges commerciaux avec flexibilité et faible investissement,

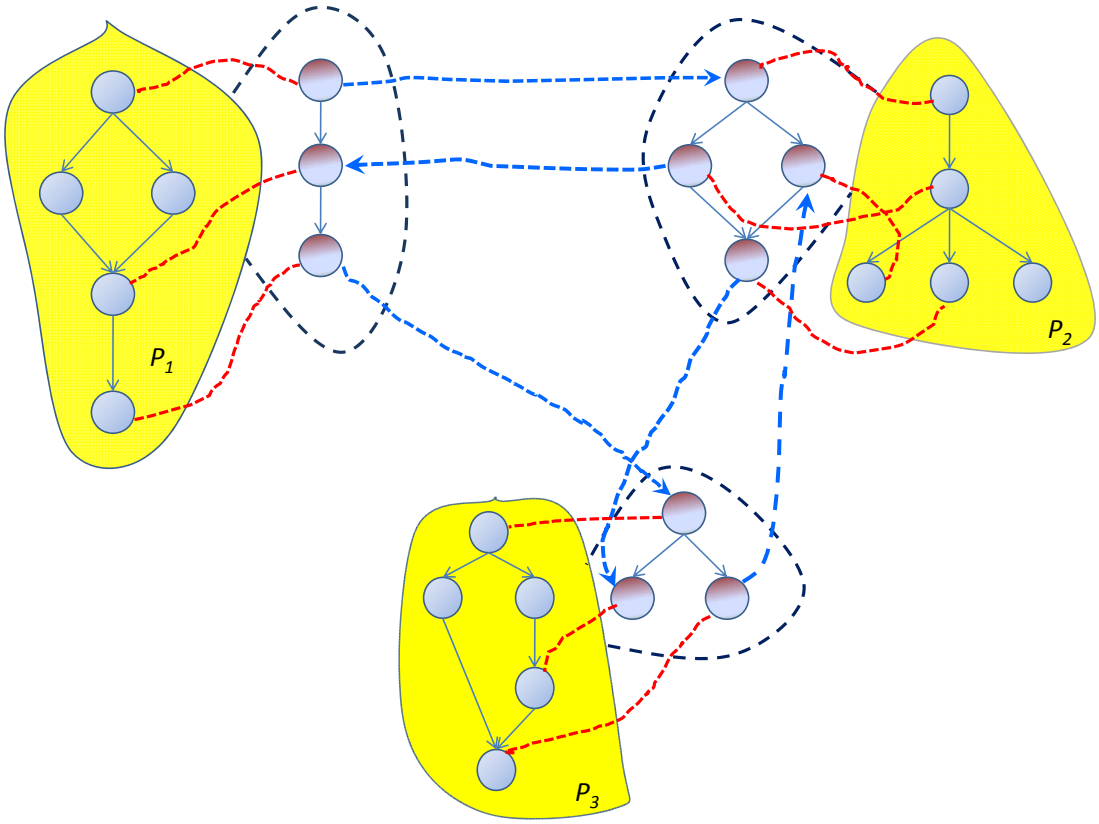


FIG. 3.4 – Exemple de chorégraphie

- les besoins d'échanges entre applications dans un système d'information hétérogène,
- la mise en œuvre de services applicatifs réutilisables et indépendants des évolutions du système d'information,
- la normalisation des interfaces d'accès aux sources d'information et référentiels de données,
- l'intégration d'applications, pour les besoins internes (Application-to-Application, A2A) et externes (Business-to-Business, B2B),

Cette liste n'est pas exhaustive, mais explique l'importance croissante des services Web au sein des offres métiers (progiciels...) et d'infrastructure (serveurs d'applications, EAI...). Cette adoption massive est amplifiée par un cycle d'apprentissage très court de la technologie des services Web. Les démarches d'étude et de mise en œuvre sont dans la continuité de celles utilisées pour les architectures à base d'objets distribués (CORBA, J2EE, .NET), et utilisent en particulier UML (Unified Modeling Language). Les technologies de développement ne sont pas remises en cause (les outils de développement de services Web sont souvent des compléments aux ateliers de développement habituels). La plate-forme technologique des services Web n'est pas totalement achevée ; cela doit être interprété comme une opportunité et confirme l'évolution du modèle vers une infrastructure complète et cohérente capable d'apporter les niveaux de services indispensables pour couvrir le cycle de vie des systèmes d'information :

- l'interface client pour le développement des applications clientes et des IHM (interfaces homme-machine),
- les mèthodes pour l'orchestration des procédés métiers, la gestion des transactions et des compensations,

- les registres pour l'annuaire des services et la gestion de leur cycle de vie,
- les composants pour la conception et la réalisation des services métier,
- la communication pour les services de transport et de sécurité,
- l'administration pour les outils et services d'administration, de déploiement et de supervision,

Les plates-formes de développement et de déploiement de services Web convergent aujourd'hui vers ce modèle et couvrent un sous-ensemble important de ces fonctionnalités. Les implémentations disponibles sont de bonne facture, interopérables entre elles grâce notamment à la disponibilité d'implémentations Open Source de référence, et permettant la réalisation d'applications services Web opérationnelles au sein des entreprises. Les services Web sont indéniablement un nouvel atout pour l'entreprise étendue.

On peut clairement constater que la technologie des services Web répond aux exigences essentielles de la programmation des systèmes distribués. Pourtant, soulignons aussi que les problèmes relatifs aux compositions des services tant pour le composition fonctionnelle que la composition vis-à-vis des aspects non-fonctionnels sont complexes.

3.3 Systèmes d'information décentralisés et pair-à-pair

Les systèmes d'information, réalisés en employant des architectures logicielles orientées services ou d'autres technologies, reposent sur des infrastructures matérielles dispersées dans l'espace. Ces infrastructures ont différentes caractéristiques les distinguant les uns des autres.

Les systèmes décentralisés et P2P ont pour but la mise en place des interactions directes entre les participants (personnes, machines ou systèmes de gestions de procédé métier) d'un système d'information. Tant d'un point de vue organisationnel que technique, la problématique des systèmes décentralisés et surtout pair-à-pair est très actuelle. Dans cette section, nous donnons une définition de notre compréhension de décentralisation et de P2P. Nous allons examiner les caractéristiques représentatives de ces systèmes, soit par leur influence historique sur l'évolution du paradigme, soit par l'importance qu'elles prennent par rapport aux exigences des systèmes d'information.

Les infrastructures distribuées dites "P2P" sont essentiellement conçues pour le partage de ressources d'informations (contenu, stockage, cycles d'unité centrale de traitement) par l'échange direct, plutôt que d'exiger l'intermédiaire ou l'appui d'un serveur ou d'une autorité centralisé [Androutsellis-Theotokis and Spinellis2004]. Ces infrastructures connaissent un très grand succès depuis 1999, date à la quelle Napster, une application P2P de partage de fichiers a permis le partage libre des fichiers multimédias entre les utilisateurs volontaires de l'application. Suite à l'arrêt du Napster qui n'est pas formellement une application P2P puisqu'il repose sur l'utilisation d'un serveur centralisé, un autre groupe de développeurs a réalisé une autre application de partage de fichier similaire à Napster mais qui est complètement distribué, échappant ainsi à tout contrôle [Doyen2005]. Cette application, Gnutella, montra qu'il est possible de fournir un service particulier à une communauté sans se baser sur une infrastructure physique concrète. Cette bonne propriété est la raison principale de l'attention particulière portée aux systèmes P2P. Si les premières applications de l'approche P2P étaient exclusivement liées à l'échange souvent illégal des fichiers multimédias soumis à des droits d'auteur, actuellement, l'ensemble des systèmes d'information sont repensables et repensés pour être déployés selon une approche P2P.

Dans la littérature, on trouve plusieurs définitions qui décrivent la notion P2P. Ci-dessous, nous donnons quelques définitions de la notion P2P qui soulignent ses différentes caractéristiques.

1. *Distributed computer architectures labeled "peer-to-peer" are designed for the sharing of computer resources (content, storage, CPU cycles) by direct exchange, rather than requiring the intermediation or support of a centralized server or authority. Peer-to-peer architectures are characterized by their ability to adapt to failures and accommodate transient populations of nodes while maintaining acceptable connectivity and performance.* [Androutsellis-Theotokis and Spinellis2004]
2. *Peer-to-peer is a class of applications that take advantage of resources (storage, cycles, content, human presence) available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers* [Schollmeier2001].
3. *A distributed network architecture may be called a Peer-to-Peer network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, . . .). These shared resources are necessery to provide the Service and content offered by the network. They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource providers as well as resource requestors* [Thain *et al.*2005].

De ces définitions, un certain nombre de caractéristiques peuvent être développées pour décrire les systèmes P2P. Nous nous intéressons à ces caractéristiques dans le reste de cette section.

Caractéristiques des systèmes P2P

Dans cette partie, nous présentons les caractéristiques typiques des systèmes décentralisés et systèmes égal-à-égal ;

1. **Passage à l'échelle :** L'idée de base des systèmes P2P est de fournir un service particulier à une communauté sur une infrastructure décentralisée. Une des limitations bien connue des systèmes centralisés est le passage à l'échelle comme une entité centralisée peut devenir facilement un goulot d'étranglement [King1983] [Sheth and Kochut1997] [Chen and Hsu2001] d'un point de vue de performance. En effet dans les systèmes centralisés, une entité centralisée comme par exemple un serveur d'application, concentre les données des processus et les mécanismes de coordination des participants de ces derniers. En conséquence, ceux-ci génèrent un trafic de réseau au niveau du serveur centralisé. On peut dire sans doute que les systèmes P2P sont des systèmes à *grande échelle* qui gèrent un nombre d'utilisateurs et des quantités de données variants sur de grands intervalles. Parce qu'un système P2P répartit la charge induite par des processus du système. Il est à noter que la gestion peut être effectuée à l'aide des serveurs partiellement centralisés. Dans le cas où un serveur est employé pour un certain nombre de services particuliers (par exemple des services de *booting*), le nombre d'utilisateurs ou la quantité de données gérés sont limités du fait des capacités du serveur partiellement centralisé. Augmenter la capacité du service particulier au delà de cette limite peut entraîner alors des coûts financiers en matériels et en logiciels importants. C'est pour cette raison que les systèmes P2P qui sont partiellement centralisés doivent faire un compromis entre les capacités nécessaires au fonctionnement global du système et celles des serveurs centralisés. L'une des possibilités d'augmentation de ces capacités d'une manière intuitive est la mise en place de plusieurs serveurs qui fournissent le même service particulier [Schuler *et al.*2004]. Cela pose alors le problème de la répartition de ces serveurs, de leur interconnexion et surtout de leur cohérence. Toutefois,

dans les systèmes P2P partiellement centralisés, les charges imposées aux serveurs spéciaux permettent de décharger les autres participants du système.

2. **Dynamicité et Instabilité :** Les systèmes P2P permettent de représenter des échanges sociaux au sens où ils lient des participants, utilisateurs ou machines, qui s'impliquent dans un système de manière humaine. Contrairement aux systèmes distribuées classiques, où les participants du système ne changent pas avec le temps, dans un système P2P, les participants du système peuvent changer d'une manière dynamique. C'est une conséquence naturelle de partage volontaire des ressources par les utilisateurs de l'application P2P. Dans un réseau de partage de fichier comme eMule [emu2002], les participants du système se connectent au système sans aucune incidence sur le fonctionnement global et ils peuvent le quitter sans avoir à se chercher d'autres participants remplaçants. Si on considère les applications de partage de fichiers, le manque des garanties transactionnelles [Gray and Reuter1993] n'est pas un soucis essentiel pour des participants. Pourtant dans les applications qui gèrent des procédés métier, l'instabilité est une exigence essentielle [Yu *et al.*2004]. Toutefois, la dynamicité dont l'instabilité peut être une conséquence est aussi un avantage dans le contexte des procédés métier réalisés et exploités chaque jour dans les entreprises [Benatallah *et al.*2006] [Nezhad *et al.*2006].
3. **Tolérance aux pannes :** Une autre conséquence de la dynamicité est la tolérance aux pannes. Dans un système d'information, les raisons des pannes peuvent être multiples. Pour donner des exemples de pannes qui peuvent intervenir dans un système d'information, centralisé ou décentralisé, on peut nommer l'arrêt de participants du processus ou d'une entité centralisé qui fournit un service particulier. Dans un processus où une entité centralisée est employée pour fournir un service particulier comme par exemple la coordination des participants, l'exécution correcte du processus repose intégralement sur celle de l'entité employée. Si celle-ci devient indisponible, le service qu'elle fournit devient indisponible. Dans un système purement P2P, il n'existe potentiellement aucun point central de faute. Si un pair tombe en panne, le processus peut continuer avec ceux qui restent. Cela illustre la sensibilité des systèmes centralisés face à une panne de l'entité centralisée qui assure la coordination des participants [Gauron2006]. En pratique, dans les systèmes de partage de fichiers, lorsque un nœud échoue, il peut être remplacé par un autre qui fournit le même fichier partagé. Dans les procédés métiers, cela n'est pas toujours valable.
4. **Échange direct :** Une des caractéristiques essentielles des systèmes décentralisés P2P est l'échange directe entre les participants de l'application. L'échange direct peut s'établir via un tiers participant qui a le rôle d'annuaire. Si un tiers qui a le rôle d'annuaire n'existe pas, les participants peuvent établir l'échange direct par une recherche propagée dans le réseau des participants. L'échange direct est la raison du passage à l'échelle.
5. **Double rôle :** Un système P2P est un modèle où les participants jouent le rôle de client, serveur ou un routeur pour le processus dans lequel ils sont impliqués. Un participant est un client quand il consomme le produit d'un service ou d'un processus exécuté par un ou plusieurs autre(s) participants. Un participant est un serveur s'il fournit un service aux autres participants. Un participant peut être considéré comme un routeur s'il fait l'intermédiaire entre deux participants qui ne peuvent pas établir une relation directe. Actuellement, on peut recenser plusieurs approches pour identifier des rôles des participants [Park and Hwang2003] dans un système P2P. L'identification des rôles peut aller des simples restrictions [Atluri *et al.*2001] jusqu'à d'autres plus sophistiquées [Bertino *et al.*1999]. Il est à noter que la spécification des restrictions concernant les rôles est intimement liée à la topologie du système P2P. Par exemple, la satisfaction des res-

trictions sophistiquées comme celles du [Bertino *et al.*1999] [Atluri and Warner2005] peut être irréaliste dans un système complètement décentralisé.

Différents types de systèmes P2P

Comme nous l'avons détaillé dans la section précédente, les systèmes P2P ont plusieurs caractéristiques qui les distinguent des systèmes totalement centralisés. La caractéristique essentielle, qui motive nos contributions, d'un système P2P est l'établissement d'interactions directes entre les participants. Nous détaillons ci-dessous les différentes topologies P2P dans un ordre décroissant de décentralisation. Plutôt que discuter les raisons qui ont poussé à leur création, nous détaillons leur intérêt et limites respectifs ainsi que leur caractère complémentaire.

1. **Systèmes P2P basés sur des entités centralisées :** Ce type de systèmes est le premier exemple de systèmes P2P. La première application de ce type est Napster [Carlsson and Gustavsson2001 qui vu le jour en juin 1999. Conçu pour partager des fichiers multimédias, il s'agit d'un serveur index qui contient les titres de fichiers partagés et l'adresse des participants qui les mettent à disposition. Lorsqu'un participant cherche un fichier, il envoie au serveur une requête concernant sa recherche. Le serveur répond alors par une liste de participants hébergeant le fichier recherché. L'échange du fichier se fait ensuite directement entre le participant qui cherche le fichier et celui qui le fournit. L'indexation des fichiers partagés et le processus de recherche qui sont les services particuliers du Napster sont fournis par une entité centralisée. En conséquence, Napster reste un système dans lequel les participants peuvent établir des interactions directes via une entité centralisée. Au delà de Napster, les avantages et les limites des systèmes P2P basés sur des entités centralisées peuvent être résumés comme suit : Les services particuliers sont fournis par des entités centralisées, qui sont responsable de les maintenir. La charge correspondant aux services particuliers est donc concentrée sur les liens attenants à l'entité centralisée dans le réseau physique. Par ailleurs, il se peut que le fonctionnement global du système soit limité par d'entité centralisée qui fournit les services particuliers. Il est à noter que ces modèles sont sensibles aux pannes. Cette partie de la problématique est intuitive. Parce que si l'entité centralisée tombe en panne, le système P2P peut cesser de fonctionner correctement comme certains services n'existent plus. Malgré ces limitations, ce modèle a des avantages. Les entités centralisées sont aisées de contrôler les services qu'elles fournissent. Dans le cas de partage libre de fichiers, ce fait peut être une désavantage comme l'entité centralisée qui indexe les fichiers ou les participants peut censurer le service de recherche. Mais dans les systèmes d'information plus sophistiqués, cela est une avantage [Fakas and Karakostas2004] [Fakas *et al.*2003]. Pour résumer, on peut dire que les systèmes P2P qui emploient des entités centralisées ont les mêmes avantages et limitations des systèmes centralisés. La topologie de ce type de systèmes est illustrée dans la figure 3.5(c).
2. **Systèmes P2P hybrides :** Chronologiquement, les systèmes P2P hybrides sont apparus après les systèmes P2P purement décentralisés. Le but essentiel des approches hybrides est de garder les bonnes propriétés des systèmes P2P basés sur des entités centralisées en multipliant celles-ci afin de diminuer les limitations de performance. En effet, dans ces systèmes hybrides, les services particuliers fournis par les entités centralisées sont répliqués ou partagés entre plusieurs entités. Apparemment simple, le modèle hybride cache un grand nombre de concepts et de sous-entendus. On peut immédiatement constater que les systèmes hybrides ne sont pas purement décentralisés. Ils reposent sur un nombre limité d'entités sur lesquelles la charge totale est distribuée. Ces systèmes sont plus tolérants aux

pannes si les services particuliers sont répliqués. Dans ce cas, afin d'assurer la cohérence des services particuliers, les entités qui les fournissent doivent être synchronisés d'une manière transactionnelle [Özsu and Valduriez1999]. La topologie de ce type de systèmes est illustrée dans la figure 3.5(b).

3. **Systèmes P2P purs :** Les systèmes P2P purement décentralisés, conçus pour le partage des fichiers, ont vu le jour après l'arrêt de Napster suite à un procès. Contrairement aux systèmes P2P hybrides ou basés sur des entités centralisées, les systèmes P2P purs répartissent la totalité des services entre les participants. Tous les participant ont alors le même rôle. Dans les applications de partage de fichiers, les modèles P2P purs sont très efficaces comme ils protègent leurs participants en réaction aux contrôles juridiques. De cette remarque, on peut facilement déduire qu'un système P2P pur est difficilement contrôlable par un tiers. Ce fait est en avantage pour des applications de partage de fichier lorsque il peut être une limitation pour des applications plus sophistiquées qui nécessitent la surveillance de l'un de ses participants. La topologie de ce type de systèmes est illustrée dans la figure 3.5(a).

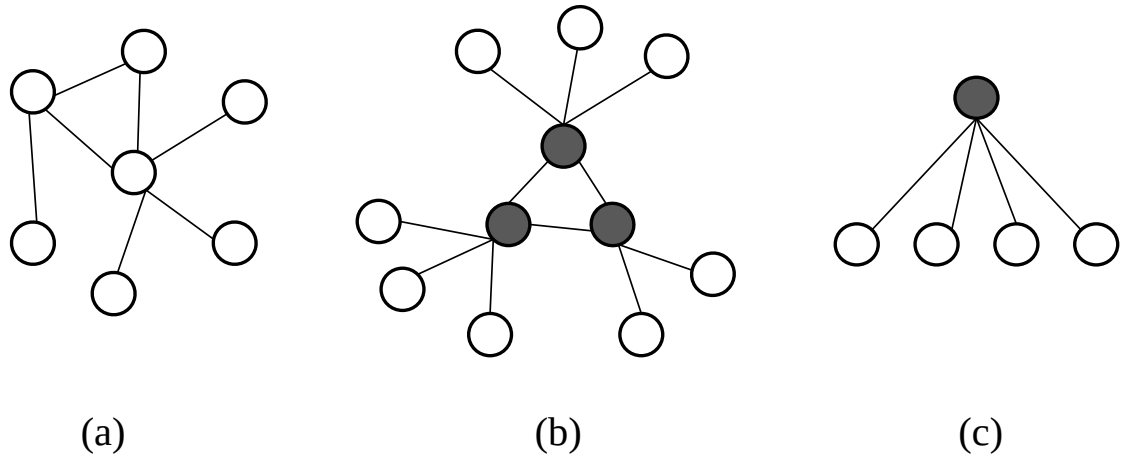


Fig. 3.5 – Les topologies des systèmes décentralisés P2P

3.3.1 Pourquoi avoir des systèmes décentralisés ?

Dans les sections précédentes, nous avons présenté les caractéristiques habituelles des systèmes P2P et les principales topologies qui méritent d'être mentionnées pour une meilleur compréhension de celles-ci. Dans cette section, nous allons mettre en évidence pourquoi la décentralisation et le paradigme P2P sont importants dans un contexte plus organisationnel au delà des applications de partage de fichier. Il ne s'agit nullement d'écrire une autre classification, mais d'écrire les aspects motivants de la décentralisation. Les avantages et les limites exprimées sont souvent proches des caractéristiques que nous avons décrites ci-dessus.

1. **Des économies matérielles :** Une des avantages, le plus indéniable et le plus souligné par la communauté scientifique [Benatallah *et al.*2002] [Nanda *et al.*2004] [Alonso *et al.*1997] [Alonso *et al.*1996], des systèmes P2P est la diminution de coûts pour l'achat et la maintenance des entités qui jouent des rôles centralisés.

   Nous pouvons mettre en évidence l'avantage des applications P2P en vue de fournir des économies matérielles avec l'exemple de motivation suivant. La figure 3.6(a) illustre trois

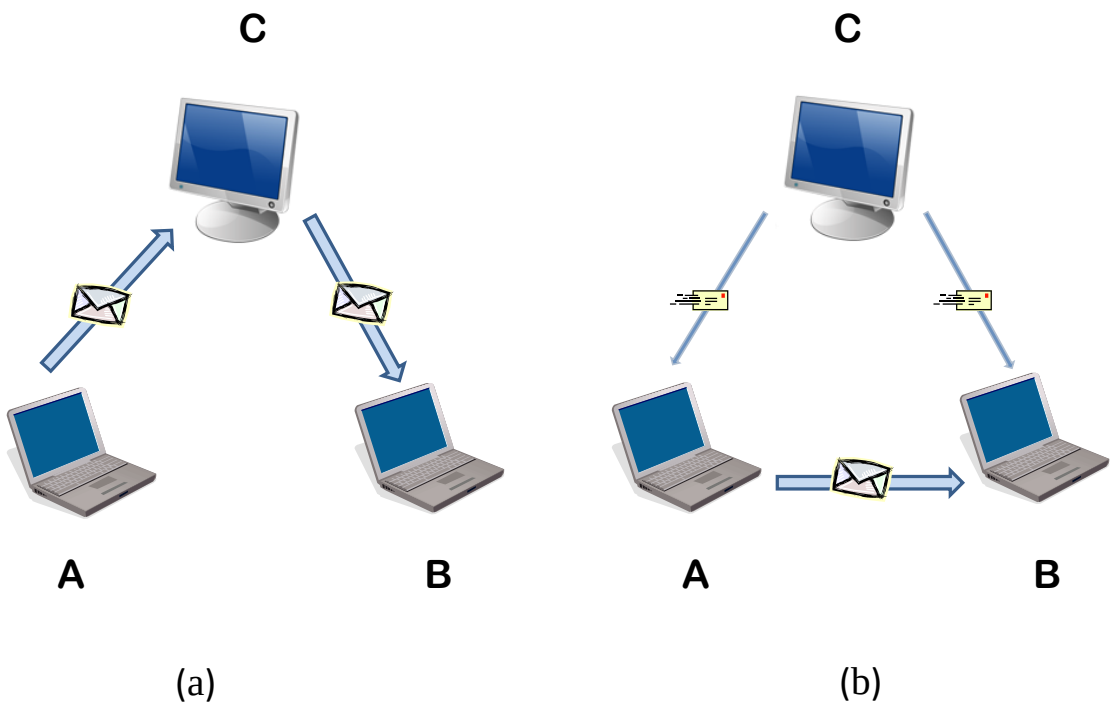


FIG. 3.6 – La comparaison de l'approche centralisée et décentralisée

organisations A, B et C. L'entité C a le rôle de coordinateur centralisé de la collaboration parce qu'elle route la sortie de A vers B. La figure 3.6(b) illustre une exécution alternative dans la quelle la sortie de A est envoyée à B par A lui-même. Il est à noter que A et B doivent *se connaître* dans un contexte dynamique. Pour ce faire C doit envoyer des messages à A et à B pour qu'ils puisse se coordonner.

2. **Tolérance aux pannes :** Une autre bonne propriété engendrée par la décentralisation est la maîtrise de la tolérance aux pannes. Dans une architecture centralisée, la disponibilité d'un service fourni repose totalement sur la disponibilité de l'entité centralisée. Dans un cas dans lequel l'entité centralisée tombe en panne, le service qu'il fournit devient naturellement indisponible. Dans un contexte P2P, il n'existe potentiellement aucun point central de faute. Si un participant disparaît pour une raison quelconque, le service global continuera d'être fourni par ceux qui restent. Les causes de pannes peuvent être diverses, par exemple un afflux de messages plus important que d'habitude. Mais ces pannes peuvent être dues à des attaques pirates telles que le déni de service dont fut victime le service de DNS d'Internet. Cela illustre la sensibilité des systèmes en partie centralisé face à une charge importante sur quelques nœuds ou les arêtes attenantes. Un système décentralisé n'a en revanche aucune précaution particulière à prendre vis-à-vis de nœuds spécifiques, puisqu'aucun nœud particulier n'est nécessaire à son fonctionnement. Dans ce cas, pour assurer que l'on puisse toujours envoyer des messages, une solution classique consiste à vérifier régulièrement que les voisins de chaque nœud soient toujours connectés au réseau. L'une des méthodes les plus utilisés pour cela est de demander à chaque voisin d'envoyer régulièrement un message signalant sa présence. Il est aussi possible d'imposer aux nœuds du réseau un nombre de voisin important pour que la probabilité qu'un nœud soit déconnecté du réseau du fait de pannes de ses voisins puisse être considérée comme négligeable. En cas

de déconnexion d'un voisin, ses voisins le remplacent rapidement par un autre nœud, afin de diminuer le risque de déconnexion du réseau. En pratique, dans un système décentralisé P2P, la panne d'un participant ou un nœud doit continuer de fonctionner correctement tant que des participants y sont connectés. Même si les réseaux de partage de fichiers peuvent résister au tolérance aux pannes, les systèmes décentralisés peuvent tolérer les pannes des participants dans une certain mesure. Un système d'information décentralisé peuvent se retrouver fractionné en plusieurs réseaux connectés entre eux même si certains participants tombent en panne.

3. **Relations équitables :** Un système d'information, plus particulièrement, un outil de gestion de procédé automatise un procédé métier au sein d'une même organisation. Comme nous l'avons déjà souligné, ces procédés s'appellent des procédés intra-organisationnels. Le développement des technologies de l'information et de la communication facilitent la collaboration virtuelle des organisations et les procédés inter-organisationnels. A l'heure actuelle, la majorité des systèmes d'information comme des outils de gestion de workflow qui automatisent des procédés inter-organisationnels sont centralisés. L'exemple par excellence de procédé inter-organisationnel est la composition des services Web. Une importante limitation de la centralisation est l'empêchement de l'établissement des relations équitables au niveau inter-organisationnel. Pourtant, dans bon nombre de collaborations, intra ou inter-organisationnels, l'existence de la possibilité de pouvoir établir des relations équitables est essentielle[Port and Kaiser1998] [Grinter *et al.*1999]. Dans une collaboration, une relation équitable existe si aucune des organisations participantes ne contrôle les appels aux organisations participants, ne dispose de la totalité des données et ne gouverne l'enchaînement des activités. La gestion centralisée d'un procédé métier assume la présence d'une organisation dominante qui dispose de la spécification complète de la collaboration. Cela ne peut pas être toujours le cas. Parce que les vues de la collaboration peuvent être mutuellement dispatchées entre les organisations et aucune ne peut avoir une vue globale[Zhao *et al.*2005] [Port and Kaiser1998]. La gestion centralisée d'un procédé met les organisations participantes de la collaboration dans une situation récessive. Parce que les participants contribuent à la collaboration d'une manière isolée sans savoir leur rôle précis en répondant aux invocations reçues [Omicini and Ossowski2004].

4. **Intéropérabilité fiable :** Comme nous l'avons détaillé dans la section précédente, l'industrie propose un certain nombre de standards pour l'implémentation des systèmes d'information dont les problèmes sont relativement complexes. Un de ces efforts est l'utilisation des standards pour établir des échanges fiables entre les participants d'une interaction. Dans un premier temps, établir une interaction directe entre deux participants fait partie des exigences de confiance [Yildiz and Godart2007c]. Dans un second temps, l'utilisation des standards comme WS-Security est plus adaptée à supporter des interactions P2P qui ne nécessite pas un tiers comme coordinateur. Considérons l'exemple suivant pour expliquer la facilité d'établissement directe d'une interaction fiable entre deux participants. C'est une version simplifiée de l'exemple de motivation du deuxième chapitre. Nous considérons une organisation d'Assurance qui fournit les service d'assurances en composant les services fournis respectivement par l'Hôpital, la Police et l'Inspecteur. Comme indiqué cidessus, les interactions du service Assurance avec les autres services peuvent être supporté par un standard comme WS-Security. Après la réclamation d'un client, l'Assurance évoque l'Hôpital afin de recevoir le rapport médical du client. Dans une telle interaction, Assurance peut être obligé de faire une série de transformations sur les données reçues. Considérons qu'après être évoqué, Hôpital fournit un rapport qui contient un ensemble d'information

relatif au souscripteur comme par exemple les tests pathologiques, les radios, les empreintes dentaires, les détails de facturation etc. Pour transférer le rapport vers la Police, l'Hôpital peut l'encrypter. Si le rapport peut être seulement décrypté par la Police, l'interaction de l'Hôpital et la Police peut être considérée comme fiable même si celle-ci est effectué via Assurance. Maintenant supposons que l'Assurance soit intéressée par les détails de facturation qui sont inclus dans le même rapport. Naturellement, elle peut obtenir les détails de facturation du rapport, si le rapport est décryptable par elle-même. Cela n'est pas possible dans cet exemple comme il s'agit d'un couplage entre les services composés pour synchroniser les données relatives aux opérations de cryptage et de décryptage. Alternativement, si les différents fragments du rapport peuvent être décryptés par les services composés séparément, Assurance peut accéder aux informations nécessaires lorsqu'il route les fragments cryptés vers la Police. Il est à noter que dans cet exemple la dynamicité n'est pas prise en compte car une interaction directe entre deux entités nécessite une synchronisation avant l'échange d'information, comme par exemple l'échange des clés publiques et privées. En outre, comme coordinateur de la composition, l'Assurance ne peut pas extraire des données pour tester les conditions du flux de contrôle. Les standards comme WS-Security sont essentiellement proposés pour supporter les interactions point-to-point et ont un support limité pour les modalités de transformation de cryptologie sophistiquées. En conséquence, il est plus simple d'utiliser les standards pour des interactions P2P qui ne nécessitent pas un tiers. Si nous reprenons l'exemple réduit, une exécution centralisée peut supporter les interactions directes avec les standards :

– les transformations sophistiquées ne sont pas requises entre les participants et le coordinateur,
– chaque service peut fournir chacune de ses sorties en faisant un cryptage séparément.

Cet exemple explique l'intérêt de l'intéropérabilité fiable dans un certain mesure.

3.3.2 Quels sont les défis et les limitations des systèmes décentralisés ?

Comme expliqué dans la section, un système décentralisé P2P semble plus motivant qu'un système totalement centralisé ne l'est pas. Pourtant les systèmes décentralisés et surtout les systèmes P2P ont des désavantages qui peuvent les rendre limités. Ci-dessous, nous rappelons les plus importantes limitations de l'approche P2P, qu'intuitivement, sont les avantages des systèmes totalement centralisés.

1. **Manque de contrôle :** Les systèmes d'information visent à décentraliser les fonctionnalités essentielles des systèmes comme par exemple les interactions des participants, le traitement des requêtes et le routage d'information. C'est classiquement l'objectif de l'approche P2P où les services centralisés ne font souvent que satisfaire les tâches spéciales et pas celles qui sont essentielles. Cet aspect engendre une manque de contrôle qui n'est pas toujours souhaitable. Par exemple, les entreprises qui emploient des systèmes d'information ou plus précisément des systèmes de gestion de workflow dans le but d'automatiser leur procédés métiers, utilisent ces systèmes pour surveiller l'état de leur procédés en général. En outre, les informations collectées par la surveillance des procédés sont utilisées pour faire des analyses à posteriori pour améliorer les procédés. Le contrôle et l'analyse a posteriori caractérisent qualitativement et quantitativement l'exécution du système d'information, sa situation actuelle, pour déduire des conclusions par rapport à un ensemble de critères d'efficacité. Dans une exécution décentralisée dans laquelle les traces d'exécution sont collectées par des participants différents, faire une analyse globale et précise n'est pas triviale. La notion de manque de contrôle ne doit pas être confondue avec la notion d'*anonymat*.

L'anonymat est une fonction qui permet de cacher son identité dans une communauté. Dans le cadre des applications de partage de fichiers, l'anonymat concerne : l'auteur, l'éditeur, le client, le serveur, le document partagé et la requête [Gauron2006]. L'anonymat est une notion importante si les fichiers soumis aux droits d'auteur sont partagés d'une manière illégale.

2. **Instabilité :** D'une manière générale, pour assurer un service rapide et continu, un système P2P doit pouvoir transmettre des messages en permanence, que ce soit pour envoyer des requêtes, y répondre, en acheminer ou fournir un service d'une manière continue. Tout participant doit évidemment rester stable pendant ce processus. Le délai de connexion d'un nœud dépendant directement du nombre de participants voisins avec lesquels il doit garder des interactions, un degré faible est un avantage pour permettre un fort dynamisme des participants. Par ailleurs, tout comme pour la durée d'une requête, le délai de connexion d'un participant sera à mettre en rapport avec le temps durant lequel ce participant restera connecté au système. Dans les applications de partage de fichiers, l'instabilité d'un participant n'est pas extrêmement importante comme le même fichier peut être récupéré par un autre participant. Mais dans une application d'affaire où les participants ont des relations contractuelles, l'instabilité ne doit pas exister. En conséquence, la nature instable des réseaux P2P n'est pas adaptée aux applications métiers. Il est à noter que la dynamicité est une caractéristique du nouveau contexte des application distribuées comme les services Web. La dynamicité caractérise plutôt la possibilité de pouvoir accéder à des nouveaux services qui deviennent disponibles. Cela ne veut pas dire qu'un service évoqué peut être indisponible pendant son utilisation contrairement à des applications de partage de fichiers.
3. **Confiance :** Les applications de partage de fichiers sont capables de regrouper les participants par centres d'intérêts communs créant ainsi des communautés qui peuvent elles-mêmes s'organiser. On trouve ce type d'organisation dans les réseaux sociaux comme le Facebook. Partager des fichiers multimédias ou bien d'autre types d'information peut engendrer des problèmes de confiance entre les participants qui émettent ou reçoivent les informations. Une des raisons de problème de confiance ou de sécurité en général, est l'anonymat qui permet de cacher son identité. Comme nous l'avons souligné dans les sections précédentes, il y a plusieurs types d'anonymat concernant les entités impliquées dans une application P2P. Comme les participants peuvent cacher leur identité ainsi que l'identité des objets partagés, ceci peut causer des problèmes de sécurité. Les objets partagés peuvent contenir de l'information qui n'est pas souhaitée par le participant émetteur récepteur. Pourtant la notion de confiance peut être vue d'un autre point de vue. Cela est expliqué dans la section *Intéropérabilité Fiabilité*. Les standards de services Web comme WS-Security, WS-Trust ou WS-Adressing fournissent différent types de support pour établir des interactions sécurisés, chacun en se penchant sur un aspect différent de l'interaction en termes de sécurité et de confiance.

3.3.3 Synthèse sur les systèmes décentralisés pair-à-pair

Les systèmes P2P de partage de fichiers sont des systèmes visant à permettre à des utilisateurs la mise en commun d'objets et leur recherche en vue de leur récupération ou de leur utilisation. Les applications P2P de partage de fichiers multimédias sont des systèmes à *grande échelle* qui doivent permettre un nombre d'utilisateur variant sur de grands intervalles. Cette gestion peut s'effectuer au moyen de serveurs centralisés ou de manière décentralisée. Permettre la conception et la réalisation des systèmes P2P demande à répondre à plusieurs questions. Comment les

participants sont-ils connectés entre eux ? Comment la communication est-il établie ? Comment sont envoyés les messages et quel type d'information contiennent-ils ?

La réalisation de l'approche décentralisée P2P n'est pas seulement limitée aux applications de partage de fichiers. Du fait de leurs bonnes propriétés des systèmes P2P sont repensés pour être adaptés à toutes sortes de systèmes d'information. D'ailleurs, la thématique de recherche relative à la gestion décentralisée des systèmes d'information, comme par exemple les bases de données fédérées [Sheth1991] et les systèmes de gestion de workflow [Alonso *et al.*1997], précède les applications de partage de fichiers. Si les premières applications P2P servent principalement à la recherche décentralisée de fichiers, le champ d'application de l'approche est bien plus large : la décentralisation à faible coût peut remplacer des services centralisés, l'échange directe a plusieurs avantages aussi bien sociaux ainsi que technique, par exemple les relations de confiance directes sont plus simple à établir etc.

Quel que soit le système d'information, l'efficacité fait l'objet d'un compromis avec la décentralisation et la centralisation. Dans un contexte comme le Web où le nombre de services virtuellement disponibles est théoriquement illimité, le but d'un système d'information P2P est d'établir des interactions directes entre les services participants. C'est son principal intérêt. Un effort important de nos contributions a porté sur la définition des concepts P2P pour les procédés métiers orientés services. Nous soulignons que notre compréhension de l'approche P2P est loin des fichiers de partage de fichiers. Nous cherchons à utiliser le paradigmes P2P dans les procédés inter-organisationnels. Les problèmes appartenant aux différents types de systèmes P2P sont différents. Par exemple, dans les applications de partage de fichiers, les problèmes portent sur la réactivité des nœuds, ou encore la rapidité de traitement des requêtes face à la gestion de nombreux participants alors que le problème essentiel des base de données réparties est la cohérence des données distribuées. Nous allons voir dans la suite de cette thèse comment l'approche P2P a été abordée dans le contexte des procédés inter-organisationnels. Quels sont les caractéristiques et problèmes essentiels, lesquels peuvent être combinés, et à quel prix ?

3.4 Worflow

Cette section a pour propos la présentation de l'initiative workflow pour la modélisation et l'exécution des procédés métiers.

3.4.1 La modélisation des procédés métiers

Nous commençons cette section en faisant une revue de la problématique de la modélisation des procédés métiers en général. L'objectif de la modélisation des procédés métiers est de décrire ces procédés à des fins :

- d'évaluation à priori,
- de comparaison,
- d'automatisation,
- d'évaluation à posteriori,

Conformément à ce qui a été dit précédemment, tout procédé, devant être supporté par un environnement, commence par la description des procédés métiers utilisés. Toute description de procédé appliquée à un ensemble d'objets (données), tout procédé fait appel à l'interpréteur de procédés qui transforme le procédé en un enchaînement d'opérations effectives sur les opérandes. Les opérandes sont (au moins partiellement) désignées par les concepteurs et sont extraites de la base d'objets gérée par le système de gestion d'objet. L'exécution du procédé assure l'enchaînement, la coordination de ces opérations. En général, un procédé se décompose en sous-procédé.

Chaque sous-procédé peut être considéré comme une entité qui résout un sous-problème. La résolution de certains de ces sous-problèmes n'est pas automatisé et s'exécute dans la tête des agents humains. Une description de procédé doit distinguer les sous-procédés automatisées de ceux qui ne le sont pas. En général, une description de procédés métiers ne détermine pas une et une seule façon d'atteindre l'objectif d'un procédé, de résoudre le problème initial, qui peut d'ailleurs être lui même redéfini dynamiquement. Il appartient alors aux concepteurs de lever l'indéterminisme du procédé en choisissant la ou les prochains sous-procédés à exécuter en s'appuyant sur l'état d'avancement du procédé. Un procédé métier doit permettre de supporter l'automatisation de la production du produit, et en particulier la coordination entre les agents humains et machines, la mise en évidence des sous traitements, l'exécution de ces traitements et la connaissance de l'état du développement (jusqu'à quel point l'objectif du développement a été atteint). La description de ces éléments nécessite l'utilisation des formes concrètes. Les travaux qui sont à l'origine de l'initiative workflow, ont avancé l'idée qu'un procédé métier peut s'exprimer comme un programme informatique classique. Ils utilisent le vocabulaire de programme de procédé. Ces termes sont très controversés parce que l'idée d'une opération d'exécutant dans la tête de l'utilisateur, comme c'est le cas de bon nombre d'activités de travail coopératif, c'est à dire non mécanique, non automatique semble en opposition avec l'idée de programme, directement liée à l'exécution par une machine d'opérations totalement définies et le mot programmation peut alors sembler mal venu. Nous ne voulons pas entrer dans un débat philosophique. Nous conservons l'idée d'un procédé métier doit pouvoir s'exprimer en réutilisant des éléments similaires à ceux utilisés dans des langages de programmation existants, mais nous utilisons les termes de description ou modèle de procédés, plutôt que ceux de programme de procédé afin d'insinuer le fait qu'une description de procédé ne s'exécute pas comme un programme classique.

D'un point de vue historique, tous les procédés métiers créent et transforment des produits en appliquant des opérations à des opérandes. On peut donc penser que le besoin et donc une certaine capacité de description et de coordination de ces opérations ne sont pas entièrement nouveaux et que des prédécesseurs aux langages de description de procédés existent. C'est clairement le cas *des langages de commandes des systèmes d'exploitation qui sont des formes primitives de langages de programmation de procédés.* Les commandes du système d'exploitation constituent alors les opérateurs élémentaires de l'environnement (les types activités), les fichiers de commandes des opérateurs plus complexes (les types de tâches) et les fichiers, gérés par le système de gestion de fichier, les opérandes. Le système de gestion de fichiers joue alors le rôle du système de gestion d'objets et l'interpréteur de commandes le rôle de l'interpréteur de de procédés. Ainsi, un fichier de commandes est un procédé métier (de fabrication) primitif et le langage de commandes utilisé pour son écriture un langage primitif de description de procédés métiers. Remarquons également que si un fichier de commandes peut inclure un autre fichier de commande, une tâche peut inclure une sous-tâche, on a une description hiérarchique d'un procédé.

Soulignons aussi que l'interprétation de ces procédés métiers primitifs gère la coordination entre les activités purement humaines et les activités complètement automatisées. En effet, d'une part, l'utilisateur utilise les capacités d'identification, de désignation ou de sélection fournies par le langage de commande pour identifier, participer ou sélectionner les activités incluses dans le procédé. D'autre part, si l'utilisateur demande l'interprétation de fichiers de commandes pour réaliser des séquences d'opérations répétitives, le programme s'exécutant peut être interrompu à différents moments pour rendre le contrôle du procédé à l'utilisateur. L'instant privilégié de dialogue avec l'utilisateur est en général la fin d'une opération. Remarquons aussi que les langages de commandes ont été introduits pour automatiser certaines tâches qui ne l'étaient pas auparavant et pour rendre cette automatisation plus flexible. Une autre faiblesse de ces langages

est la pauvreté de leurs structures de contrôle, celles-ci étant généralement limitées aux habituelles itérations et conditionnelles des langages de programmation classiques. Ces structures sont insuffisantes pour décrire les règles de coordination entre sous-procédés d'un même procédé [Guabtni and Charoy2004]. Afin d'aider au développement des procédés métiers à grande échelle, des modèles intégrant des structures de contrôle plus riches ont été développés, en particulier des modèles diagrammatiques. Ces modèles ont été enrichis par des structures décrivant les flux de données et la hiérarchisation des procédés. En général, un procédé y est représenté sous la forme d'une boîte, les liens de hiérarchie entre les procédés sous formes de symboles et de flèches. Mais une fois encore les limites de tels formalismes se sont traduites par la nécessité de spécialiser les informations portées sur les diagrammes. Ces informations se spécialisent par type de données, par type d'application, par type d'utilisateur, par famille ... et souvent par combinaison de différents points de vue. Les points de vues peuvent être si nombreux qu'il est alors nécessaire d'ajouter des couleurs (pour les différents états), des traits de forme différentes (pour les dépendances de contrôle, donné, transactionnel), des flèches de différentes formes, etc. Des exemples de telles représentations sous forme de diagrammes se trouvent dans de nombreuses spécifications, mais de toutes façons les relations potentielles entre les données du procédés, les acteurs et le contexte sont si nombreuses qu'il n'y a pas de solutions sans une représentation interne et automatisée de ces relations supportée par des outils capables de recréer les différents points de vue.

Dans tous les cas, la raison de l'échec est l'absence d'un langage opérationnel adapté à la description des procédés ; dans le premier cas, on trouve un langage opérationnel, mais largement incomplet, dans le second cas ; un langage plus spécialisé, mais moins précis et généralement peu opérationnel. Ainsi, les développeurs de spécification de procédé ont utilisé les langages de commandes des systèmes d'exploitations comme langages primitifs de programmation de procédés métiers ; mais ceux-ci ont montré leur limites en ce qui concerne l'écriture de procédés de grandes tailles et de grande complexité. D'autre part, les développeurs de méthodes ont tenté de modéliser des procédés plus larges et plus complexes sous forme de diagrammes, mais sans parvenir à décrire de façon suffisamment claire et précise les différents aspects d'un tel procédé. Par conséquent, la définition d'un langage de description de procédés métier est un problème ouvert. Dans la suite de cette section, nous allons détailler les éléments caractéristiques d'un langage de description de procédés. La définition de ces éléments explique leur considération au sein de l'initiative workflow. Il est ensuite nécessaire de contrôler les appels et l'enchaînement des appels à ces opérateurs dans le cadre d'une organisation hiérarchique des procédés. Il est clair qu'une description de procédés doit posséder au moins les structures de contrôle des langages de commandes ou de programmation classique. Tout d'abord, il est nécessaire d'introduire des structures de contrôle pour modéliser l'exécution concurrente des sous-procédés, leur débuts et leur synchronisations. En particulier, la cohérence des objets manipulés pendant l'exécution concurrent doit être satisfaite. Si certains procédés peuvent se décrire dans un style de programmation déclarative, ce n'est pas le cas des procédés complètement automatiques. Il est claire que c'est le cas des procédés les plus créatifs tels que les procédés décrivant les travaux coopératifs comme la conception coopérative d'objets conceptuels. En général, dans ce type de procédé si on est capable de spécifier l'objectif du procédé, on n'est pas capable de décrire entièrement la façon d'atteindre cet objectif. On est seulement capable de contraindre l'exécution du procédé en l'obligeant de valider certains points de contrôle, même si certains sous-procédés peuvent être complètement spécifiés sous une forme algorithmique. Il est à noter qu'il est possible de l'intégrer l'approche déclarative dans l'approche impérative dans la modélisation des procédés. Ceci est le cas des procédés flexibles.

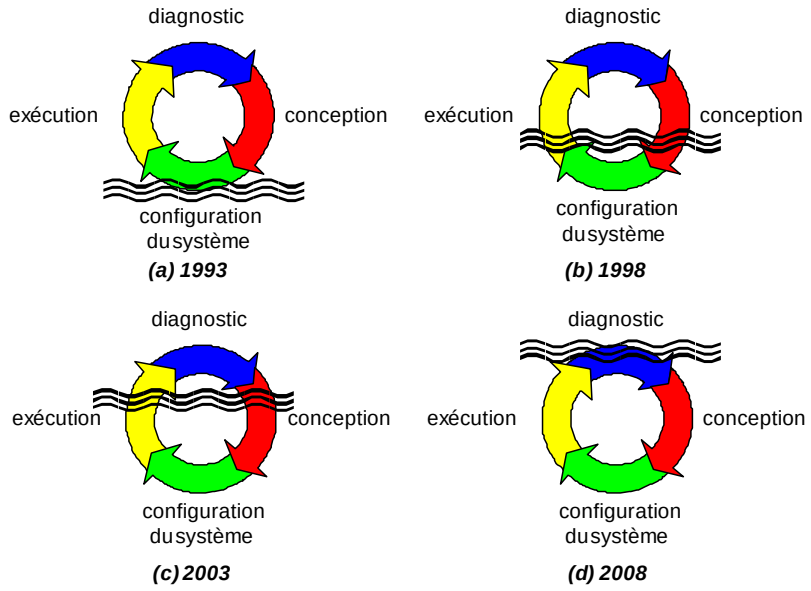


FIG. 3.7 – Le cycle de vie d'un procédé [van der Aalst2005]

De ces observations, nous pouvons définir la définition d'un procédé comme suit : *Un procédé est un ensemble d'activités ordonnées selon un ensemble de règles procédurales (impératif ou déclaratif) pour faire réaliser un objectif précis par un groupe d'acteurs tels que des machines et/ou de personnes. Il décrit la structure des acteurs tels que les rôles et les droits, les activités de ces acteurs et les critères de contrôle du déroulement de ces activités.*

L'objectif et surtout le cycle de vie d'un procédé ne sont pas limités à la modélisation ou à l'exécution des procédés. La figure 3.7 illustre les étapes d'un cycle de procédé d'après Van Der Aalst [van der Aalst2005]. Même si cette proposition appartient à l'auteur, elle contient les étapes essentielles par rapport aux approches de la littérature. L'illustration inclut le focus des activités de recherche qui porte sur les aspects des procédés dans le temps comme la conception, l'exécution et le diagnostic. Notre travail s'intéresse à la modélisation (conception) et l'exécution des procédés.

3.4.2 L'initiative Workflow

Dans la section précédente, nous avons présenté l'évolution historique de la modélisation des procédés métiers. Dans cette section, nous nous intéressons à la considération de l'approche workflow pour ce propos.

Un groupe de chercheurs considèrent un workflow comme une spécification exécutable de procédé. Dans cette approche, les aspects qui ne peuvent pas être spécifiés dans une spécification exécutable sont intégrés dans celle-ci avec des spécifications déclaratives. Comme nous l'avons mentionné ci-dessus, cela constitue les workflows flexibles. Une deuxième approche considère un workflow comme une spécification conforme aux normes définies par la Workflow Management

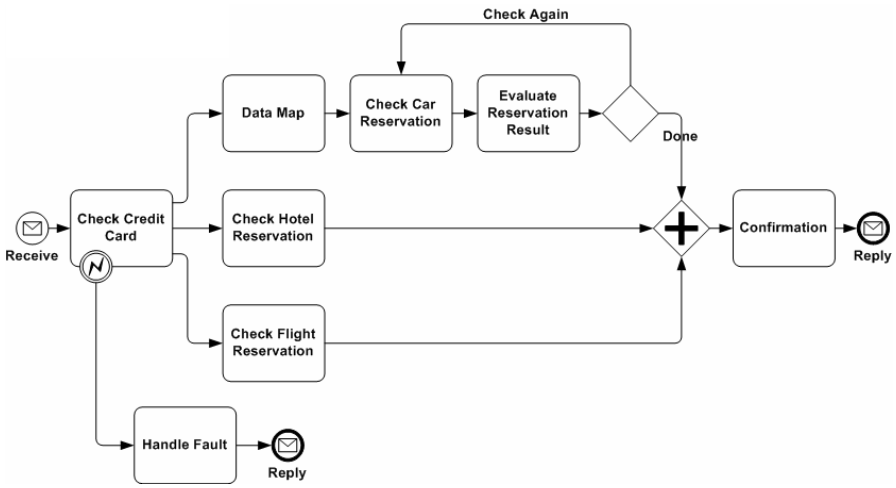


FIG. 3.8 – Un exemple de workflow en BPMN

Coalition [WfMC1996]. Le point commun de deux approches est une représentation similaire par une graphe composée des activités en forme de boîte et des dépendances en forme d'arcs. La figure 3.8 illustre un workflow spécifié en utilisant la notation Business Process Modeling Language (BPML) [BPMI.org2001]. Les activités du procédé correspondent aux tâches que les acteurs humains ou les machines doivent réaliser. Les losanges caractérisent les dépendances du flux de contrôle. Le procédé a un point de démarrage et un ou plusieurs points de terminaison. Dans cette notation, on peut également caractériser les dépendances de données des activités. Une dépendance de données exprime la relation d'entrée/sortie de deux activités.

Un système de gestion de workflow (Workflow Management System, WfMS) est un système informatique qui transforme la représentation formelle du workflow spécifié en un format interne et exécutable dans un environnement opérationnel. La fonctionnalité essentielle d'un workflow est d'exécuter le procédé. En outre, il peut avoir des fonctionnalités additionnelles comme par exemple la vérification du modèle par rapport à un ensemble de critère de cohérence [Sadiq and Orlowska1999] ou la surveillance pour faire des analyses a posteriori [van der Aalst2005]. Le but ultime d'un système de gestion de workflow est d'éclairer la sémantique d'un procédé au sein d'une organisation.

Pour ce faire, l'approche workflow fait plusieurs hypothèses :

- Les procédés sont relativement rigides et sont adaptés aux expressions en utilisant des notations similaire aux graphes,
- La coordination des participants peut se faire par une entité centralisée (celle qui détient le système de gestion de workflow),

Ces deux hypothèses sont à l'origine des limitations de l'approche workflow classique. La première hypothèse limite l'utilisation des workflows pour supporter les activités qui nécessitent une certaine flexibilité. L'exemple par excellence de ces types de procédés qui ne sont pas ou bien qui sont très difficilement supportés avec des workflow sont les procédés de co-conception, d'édition coopérative ou de co-développement. Cette limitation bien connue de l'approche workflow[van der Aalst and Berens est à la base des recherches sur les modèles de procédé flexible et sur les systèmes de gestion de workflow décentralisés. C'est le cœur des travaux liés de nos contributions.

A l'origine, les systèmes de gestion de workflow sont conçus pour supporter des procédés de

production répétitifs et ne supportent pas le changement des procédés dans le temps. Une telle vision n'est pas suffisante dans le contexte des organisations qui doivent être toujours flexibles et ouvertes aux changements. Un grand nombre de travaux qui portent sur différent aspects de la culture de workflow. Compte tenu de la diversité de ces travaux, il n'est pas possible de les détailler. Nous faisons une taxonomie non-exclusive qui résume les travaux divers qui portent sur l'approche workflow :

- *La gestion dynamique des changements du modèle de procédé* : Ce groupe de travaux vise à adapter le changement dynamique du modèle de procédé et les instances actives de celle-ci [Rinderle *et al.*2005] [Ly *et al.*2008].
- *Les modèles de workflows flexibles* : ceux-ci visent la modélisation des procédés qui ne peuvent pas être exprimés avec l'approche rigide du workflow classique [Deng *et al.*2006] [Andonoff *et al.*2005].
- *La modélisation transactionnelle* : ces travaux ont pour but d'intégrer les propriétés transactionnelles des bases des données à la modélisation des procédés métiers [Hagen and Alonso2000]. Le but de cette approche est de fournir des environnements robustes comme dans les systèmes de gestion de base de données. Le modèle transactionnel peut être intégré au niveau du modèle de workflow[Rusinkiewicz and Sheth1995] ou des données gérées pas le workflow[Schuldt *et al.*1999].
- *Séparations des préoccupations* : les systèmes de gestion de workflow doivent gérer plusieurs dimensions des procédés métiers qu'ils modélisent. La séparation des préoccupations vise à séparer l'aspect concerné et le modèle de procédés pour faciliter la complexité de la gestion. Les exemples incluent la séparation des devoirs [Bertino *et al.*1999] ou les sphères de comportement [Guabtni *et al.*2006].
- *La vérification des modèles de workflow* : les travaux de vérification ont pour but de vérifier la cohérence des modèles de procédé comme c'est le cas dans les langages de programmation traditionnels [Sadiq and Orlowska1999]. Il s'agit de trouver les deadlocks, les chemins non exécutables et les blockages.
- *Les patrons de workflow* : la modélisation des workflows par patrons facilite leur compréhension et traitement. Cette approche est similaire à l'utilisation des patrons dans l'ingénierie des logiciels comme les patrons de conception. Les patrons de workflow incluent la considération des patrons pour abstraire les dépendances de contrôle [van der Aalst *et al.*2000] et les patrons des modalités de communication [Barros *et al.*2005].
- *La transformation des workflows* : la transformation d'un modèle fait l'analyse du workflow à transformer en vue de le transformer en un autre modèle. Le modèle obtenu peut être équivalent ou pas. Un exemple de la problématique de la transformation est la production de modèles de workflow structurés à partir d'une spécification aléatoire [Kiepuszewski *et al.*2000] [Liu and Kumar2005].

3.4.3 Worklow centralisé versus décentralisé

L'approche workflow et les outils de gestion de workflow, bien qu'au cœur des organisations, souffrent des limitations des systèmes centralisés. Les limitations précises de l'approche centralisée du workflow sont résumées dans les sections précédentes. La gestion de workflow décentralisée est une approche qui n'est pas nouvelle mais qui est toujours un sujet d'actualité. Elle a pour but de maîtriser les limitations de l'approche classique du workflow. L'idée de base de cette approche qui change de formes dans chaque contribution différente est de décentraliser la modélisation et l'exécution d'un procédé. Ces recherches furent stimulées par le contexte d'une course aux systèmes à grande échelle qui s'intensifie pendant les années 90 durant lesquelles la popularité

d'Internet s'accroît. Sans rentrer dans les détailles de la comparaison des systèmes centralisés et décentralisés, qui sont largement détaillés dans les sections précédentes, nous faisons une étude des propositions de systèmes de gestion de workflow décentralisés. Il est à noter que chaqu'un de ces travaux a été introduit comme un alternative aux systèmes centralisés. Certains d'entre eux peuvent engendrer des composants centralisés, pourtant ils sont confrontés au défi des systèmes de gestion de workflow centralisés en général.

Exotica/FMQM [Alonso *et al.*1996] Le système de gestion de workflow Exotica/FMQM est développé au centre de recherche de IBM Almaden. Le but de ce projet est d'incorporer la gestion des transactions avancées dans un système de gestion de workflow dans le cadre des produits d'IBM. Le cœur essentiel de Exotica/FMQM est FlowMark qui est un produit antérieur d'IBM. Avec ce dernier, les plate-formes spéciales pour l'échange de messages et de quieng sont utilisées. Le but ultime de Exotica/FMQM est de diminuer le besoin d'une entité centralisée qui peut devenir le goulot d'étranglement pendant l'exécution du procédé. Le focus du système est la distribution du procédé, les aspects liés au passage à l'échelle et la tolérance aux pannes. Dans ce système, l'entité centralisée qui gère les données du procédé, est décentralisée avec des échanges de messages persistants entre les nœuds. Chaque nœud dispose d'un tableau de procédé qui contient tout l'information statique pour les instances actives. Pendant l'exécution, les informations contenues dans le tableau de procédé sont traitées par les processus des activités. Les données de procédé sont passés d'un "output container" d'un nœud vers l' "input container" d'un autre nœud récepteur. Le prototype Exotica/FMQM a permis d'étudier l'impact de la décentralisation et la faisabilité d'une telle approche. Plusieurs aspects indispensables d'un procédé, comme le contrôle du flux d'information, qui doivent être étudiés dans le contexte de la décentralisation ne sont pas résolus.

ADEPT [Reichert and Dadam1998] Le projet ADEPT (Application Development on Encapsulated pre-modeled Process Templates) concerne le développement d'un système de gestion de workflow pour l'exécution des procédés inter-organisationnels à large-échelle. L'idée de base d'ADEPT est d'empêcher la surcharge d'un moteur d'exécution centralisé. Afin de diminuer la surcharge d'une entité centralisée, ADEPT transporte les sous-instance d'un procédé sur plusieurs moteur d'exécution en fonction de leur disponibilité. La disponibilité du serveur n'est pas la seule critère pour la sélection et la migration des parties des instances. La complexité de la spécification du procédé, les dépendances entre les instances partagées sont également considérées. La sélection dynamique des moteurs d'exécution réduit considérablement le temps d'exécution des instances malgré l'augmentation de la charge liée à la migration des instances.

DartFlow [Cai *et al.*1996] DartFlow est un système de gestion de workflow orienté agent. Le système développé à l'université de Dartmouth, utilise un navigateur Web qui inclut des applets Java pour les utilisateurs finaux du système. L'utilisation des concepts agents font le système hautement flexible lorsque l'utilisation des applets le font fiable et indépendant des plate-formes. La motivation de DartFlow, qui date bien avant des technologies de services Web, est de maîtriser les difficultés inhérentes des systèmes de gestion de workflow centralisés. Plus précisément, chaque instance de workflow est modélisée comme un agent mobile qui porte les données nécessaire pour son exécution. L'agent qui concrétise l'instance est transporté d'un nœud à un autre en fonction de l'état de l'instance. Il est à noter quelle transport de l'agent peut être fait par les nœud ainsi que l'agent lui-même. En conséquence, l'agent contient des informations additionnelles aux données de l'instance de procédé. Une entité centralisée n'est pas exigée pour

l'exécution d'une instance.

METUflow [Gokkoca *et al.*1997] METUFlow est un système de gestion de workflow développé à L'Université de Middle East Technical. Le but de ce projet est de fournir un système de gestion de workflow décentralisé dans lequel une instance d'un procédé est exécutée par un ensemble de nœuds distribués dans un réseau. Chaque nœud exécute une partie de l'instance qu'il reçoit. Ainsi, l'exécution est distribuée. Cela augmente la tolérance aux pannes de l'exécution de l'instance du procédé. L'approche du METUflow est basée sur l'observation de l'exécution des instances en traçant les événements produits par ces derniers. En conséquence, chaque nœud doit décider quand il doit exécuter une activité qui appartient à son instance. L'activation, la fin ou l'échec des activités génèrent des événements à propager vers les nœud correspondants. L'occurrence d'un événement est possible si ses gardes sont satisfaites. Les définitions des événements et leur gardes sont similaires au traitement complexe des événement (Complexe Event Processing) [Adi and Etzion2004]. Au niveau de l'implémentation, chaque nœud est un objet CORBA avec une interface qui inclut les interactions correspondantes.

SELF-SERV [Benatallah *et al.*2002] SELF-SERV est un des premiers exemples de système de gestion de workflow décentralisés visant les architectures orientées services. Il a été développé à l'Universite de New South Wales avant les efforts de standardisation des procédés composants les services Web. Comme de nombreux systèmes existants, le but de SELF-SERV est de diminuer le rôle d'une entité centralisée dans la composition dynamique d'un ensemble de services. La composition d'un service Web est exprimée comme une machine à états. Des mécanismes basées sur les critères de disponibilités sont proposées pour le transfert de contrôle d'un nœud à un autre.

OSIRIS [Schuler *et al.*2004] Ce système récent est proposé à ETH Zurich. Le but de OSIRIS est étudier la décentralisation des procédés métiers. Le système est composé de composant centralisés ainsi que d'agents implémentés sur des nœuds décentralisés. Les composants centralisés assurent les services d'annuaire, le support transactionnel, et les services qui ne peuvent pas être fournis d'une manière décentralisée. Les composants centralisés sont répliqués pour augmenter la tolérance aux pannes et la disponibilité. L'exécution décentralisée est faite entre les nœuds en coordination avec les composants centralisés. La cohérence et la synchronisation des composants répliqués ne sont pas explicitement définies. Le projet a pour but de diminuer la surcharge d'un composant centralisé pendant l'exécution d'un procédé en routant les instances vers les nœuds plus disponibles.

PeCo [Coon2002] Peco qui est l'abréviation de *Peer Collaboration*. C'est un projet réalisé au sein de l'organisation Proteus Technologies. Le but de cette organisation est de fournir des plate-formes de coopération pour les domaines des services d'intelligence, de transportation et le secteur médical. Plus précisément, la plate-forme Peco est un système de gestion de workflow réalisé selon une approche P2P. Dans la plate-forme, les agents sont responsables pour l'exécution de certaines tâches comme la coordination d'un group d'agents, les services de base, le déploiement des tâches ou l'administration. L'infrastructure Jini est utilisée pour supporter la découverte, la tolérance aux pannes et la conscience de disponibilité des agents impliqués. En général, les agents contactent le coordinateur d'un groupe pour coopérer avec les autres agents.

WWPD et WWP [Fakas and Karakostas2004] Ce système de workflow décentralisé est composé essentiellement de deux composants WWPD (Web Workflow Peers Directory) et WWP (Web Workflow Peer). Il a été proposé par les checheurs de l'Université de Metropolitan de Manchester. Le WWPD est un composant centralisé qui fournit le service d'annuaire. Les agents qui implémente le composant WWP sont enregistrés au près de WWPD pour utiliser les services des autres agents et fournir leurs services à la communauté. Pendant l'exécution des instances de workflow, un système de notification est utilisé pour coordonner les agents impliqués dans une instance de workflow. L'architecture décentralisée du workflow a pour objectif de diminuer la charge d'un composant centralisé en décentralisant la coordination.

RainMan [Paul *et al.*1997] RainMan est un système de gestion de worfklow décentralisé. C'est un système orienté-objet développé au sein de IBM. Rainman est conçu pour fonctionner sur l'Internet afin d'exploiter l'intéropératibilité des composants accessible sur l'Internet. C'est un système composé des agents faiblement couplés. Le système RainMan est basé sur l'API Rainmaker qui fournit une squelette pour la conception et l'exécution des workflow. Rainmaker identifie quatre abstractions importantes pour le domaine workflow : *Sources, Activités, Acteurs*, et *Tâches*. Dans ce système, la gestion du procédé, la distribution des activités, le service d'annuaire, et la liste des tâches actives sont traités par des composants indépendants qui fournissent le service de workflow aux utilisateurs de l'Internet. Outre les éléments de base d'un système de gestion procédé, des fonctionnalités additionnelles comme l'analyse de performance, la gestion des exceptions, et les aspects de sécurité sont étudiées dans le cadre du système RainMan. Apparue avec l'Internet, RainMan est un des premiers systèmes de gestion de workflow fonctionnant sur l'Internet.

Serendipity [Grundy and Hosking1998] Serendipity est un environnement de gestion décentralisée développé à l'Université de Waikato et l'Université de Auckland. Le but de l'environnement est de fournir un support pour la conception et l'exécution coopérative des projets de développement logiciel. Afin de fournir un support robuste, efficace et décentralisé, l'environnement Serendipity est basée sur une architecture décentralisée qui utilise les communications multicasts point à point. Chaque utilisateur de l'environnement doit disposer de ses propres composants qui doivent lui permettre de communiquer avec les autres utilisateurs de l'environnement. En employant les composants de la plate-forme, les utilisateurs peuvent éditer les spécifications des procédés. Afin d'exécuter le procédé, chaque utilisateur a un environnement d'exécution de procédé capable d'exécuter les activités locales. L'exécution est conduite par la gestion complexe des événements qui sont propagés vers les utilisateurs par ceux qui les produisent. Outre les éléments caractéristiques d'un environnement de gestion de procédés, des fonctionnalités comme la trace de l'exécution et des outils de conscience de groupe sont fournis.

Matrix [SDSC1999] Le projet matrix est développé au sein des laboratoires de San Diego Supercomputer Center. Le projet Matrix porte sur la recherche et le développement des plateformes des protocols de workflow conçus pour le gride. Les workflow grids sont utilisé pour le traitement des procédés orientés donnée. Ces derniers concernent la manipulation des données de grande quantité venant par exemple des satellites artificielles. Actuellement, le middleware Matrix fournir un support pour que les services Web puissent être intégrés dans le gride de Matrix. Les fonctionnalités de notification et de requête sont offertes d'une manière dynamique pour coordonner les participants de gride. Outre les services caractéristiques d'un workflow et d'une plate-forme de gride, un service de brokering est fourni pour des analyses de performance.

Les protocoles pour la coordination et la confluence des brokers sont conçus.

Mentor [Wodtke *et al.*1997] Ce système de gestion de workflow est peut être le plus proche au modèle proposé dans cette thèse. Il consiste en l'étude de la décentralisation du point de vue de la production de fragments de worfklow coopérants. Le système proposé considère également des composant centralisés pour satisfaire certaines fonctionnalités telles que la synchronisation des branches concurrentes. Le modèle de procédé est basé sur une description UML composée de deux parties indépendantes, notamment le flux de données et le flux d'activité. La considération des composants centralisés tels que TP, est évidemment la limitation de cette approche. Toutefois, la relation entre le flux de contrôle et le flux de donnée n'est pas traitée du point de vue de l'implémentation des systèmes de gestion de workflow. Néanmoins, nous avons utilisé une approche similaire à celle de MENTOR pour prouver l'exactitude des algorithmes que nous avons employés.

Une analyse comparative des solutions de workflow décentralisé ci-dessus est présentée dans le tableau 3.1 à la fin de ce chapitre.

3.4.4 Autres travaux liées

Depuis 2004 il y a un certain nombre de travaux indépendants qui sont vraiment proches à notre approche. Dans [Nanda *et al.*2004], la traduction d'une spécification centralisée BPEL en un ensemble de procédés coopérants et distribués est étudiée. Dans [Chafle *et al.*2005] [Nanda and Karnik2004] [Chafle *et al.*2004], le même travail est étendu pour discuter différents aspect de décentralisation comme les problèmes de synchronisation dans le cas de l'exécution décentralisée, ou bien l'implémentation des restrictions de flux de données entre les fragments traduits. Le travail original est basé sur le partition d'un code impératif en analysant le flux de contrôle et de données. La contribution essentielle vise à diminuer les communications entre les fragments décentralisés en gardant une analogie avec les problèmes de partition d'un code tout en utilisant les méthodes similaires. Même si ce travail constitue une avancée importante dans le domaine, il présente de nombreuses lacunes. Comme la fondation essentielle est motivé par le code impératif, les aspects du workflow dont les compositions sont basées ne sont pas traités. Les exemples incluent l'élimination des chemins morts (Dead Path Elimination) [Leymann and Roller2000], les patrons de workflow et les compositions conversationnelles.

En se basant sur les mêmes principes, [Sadiq *et al.*2006] étudie la traduction du contrôle de flux d'une spécification workflow mais ne se penche jamais sur les mêmes problèmes. [Baresi *et al.*2007] utilise une technique de translation des procédés BPEL en utilisant les techniques développées dans la réécriture de graphes. Il s'agit de la réécriture du flux de contrôle dans un contexte mobile. Un autre travail qui précède tous les travaux ci-dessus est présenté dans [Atluri *et al.*2001]. Le problème de décentralisation est étudié dans un contexte de sécurité. Même si les contributions précédent l'approche service et sont largement intuitives, le travail est très innovateur. Les hypothèses de ce travail restent irréalistes dans le contexte service. Parce que l'opération de translation assume la présence d'un mécanisme implémenté par les participants de workflow. Comme nous avons déjà souligné, cette hypothèse est forte dans le contexte des architectures orientées services et conduit à une solution intuitive. La différence de ces travaux, par rapport aux travaux antérieurs qui portent sur la gestion décentralisée, est qu'ils attaquent le problème de la gestion décentralisée comme un problème de traduction de spécification et pas une question de support architectural.

3.4.5 Synthèse sur le workflow décentralisé

Dans cette section, nous faisons une synthèse sur les travaux liées concernant les systèmes de gestion de procédés. Nous complétons cette section avec un tableau qui synthétise en montrant les points faibles forts et les points faibles de chacune des propositions essentielles.

Depuis les années 1990, l'évolution de l'informatique et les technologies de communication suivent des voies qui font des workflows l'un des outils essentiels pour la conception et l'implémentation des procédés métiers intra et inter-organisationnels. La mise en œuvre des aspects essentiels des procédés métiers comme la modélisation, l'automatisation, l'exécution, et le surveillance trouvent dans les workflows l'approche qui semble parfaitement adapté. Ce mouvement s'est structuré dans le contexte des procédés métiers puis dans le contexte des architectures orientées services. Une génération de contributions ayant débuté au début de ces années, nommés système de gestion de workflow décentralisés, sont en mesure de participer pleinement à la définition de nouveaux objectifs. Les succès et les premières retombées impliquent même une réflexion sur le renouvellement des approches. Ce renouvellement, toujours difficile, est facilité par le fait que les *anciens* sont encore relativement jeunes. Dans ce jeu subtil, un système de gestion de workflow décentralisé doit intégrer des aspects complémentaires : ceux de l'infrastructure de communication sous-jacente et les aspects applicatifs de la gestion décentralisée. L'approche workflow décentralisé maîtrise les limitations essentielles de l'approche classique du workflow centralisé comme elle permet essentiellement de configurer les interactions des participants par rapport à un ensemble de critères. Le problème essentiel des systèmes de gestion de workflow décentralisés est qu'ils ne sont pas adaptés aux nouvelles exigences des architectures orientées services parce qu'ils se basent sur des hypothèse qui peuvent être irréaliste dans ce nouveau contexte.

Un autre ensemble de travaux portent sur la production de fragments de procédés coopérants à partir d'une spécification centralisée similaire aux contributions de notre thèse. Essentiellement, ils proposent des structures de données et des algorithmes qui concrétisent l'opération de transformation. Formulation élégante, positive, mais qui pourrait paraître un peu ambiguë. À vrai dire, les propositions actuelles portant sur la partition des spécification semblent beaucoup plus se préoccuper de l'opération de partition et tentent de l'adapter sans fournir une solution complète et extensible aux exigences diverses. Par exemple, [Nanda *et al.*2004] se penche sur la distribution des activités d'initialisation des variables dans des fragments de procédés coopérants lorsque [Baresi *et al.*2007] se penche sur la décentralisation du flux de contrôle.

L'exécution décentralisée des procédés avec des fragments dérivés d'une spécification centralisée est une approche très nouvelle qui touche des domaines de plus en plus larges. Les points forts de cette approche tels qu'ils sont définis dans les travaux liés sont nombreux. Alors que la standardisation et la gestion des procédés par l'approche workflow ont tendance à converger, celle-ci est une approche prometteuse.

3.5 Synthèse et situation par rapport à d'autres approches de la problématique

Dans cette section nous faisons une synthèse du contexte et des travaux relatifs. Il s'agit d'une méta-synthèse des synthèses antérieures.

Au cours de ces 20 dernières années qui ont vu les Sciences et technologies de l'information et de la communication (STIC) révolutionner la société, l'évolution des composants, des langages et des infrastructures de réseau engagent informatique et télécommunications dans une convergence

	Architecture sophistiquée	Protocoles sophistiqués	P2P	Migration	Configuration avancées des interactions P2P
ADEPT [Reichert and Dadam1998]	√	√			√
Exotica/FMQM [Alonso *et al.*1996]	√	√	√		
SELF-SERV [Benatallah *et al.*2002]	√	√	√		√
OSIRIS [Schuler *et al.*2004]	√	√	√	√	
WWPD [Fakas and Karakostas2004]	√	√	√		
PeCo [Coon2002]	√	√	√		√
METUFlow [Gokkoca *et al.*1997]	√	√	√		
MENTOR [Wodtke *et al.*1997]		√	√		
Matrix [SDSC1999]	√	√	√		
Serendipity [Grundy and Hosking1998]	√	√	√		
RainMan [Paul *et al.*1997]	√	√	√		

TAB. 3.1 – L'analyse comparative de workflows décentralisés

qui ne relève plus d'une mise en commun d'éléments ou de potentiels mais touche chacun des deux domaines au cœur de sa logique de construction et de fonctionnement des entreprises virtuelles. Les conséquences organisationnelles de cette mutation, qu'elles concernent les organisations ou les institutions publiques, ne sont cependant guère prises en compte. Non par *aveuglement* mais parce que les trajectoires de reconfiguration restent difficiles à prévoir. Évolution de la demande des consommateurs, restructuration des organisations, modalités et périmètre de l'intervention publique : de nombreux éléments sont très incertains, ne permettent pas d'établir des stratégies opérationnelles à long terme. Car si la convergence semble inéluctable, il reste à en préciser le rythme et les modalités. Si les problèmes actuels des systèmes d'information et de l'entreprise virtuelle sont en général très divers, ils sont aussi très nombreux. Après une longue phase de recherche diversifiée à la mesure du foisonnement actuel du domaine, les efforts de standardisation ont permis l'émergence des services Web comme support concret de développement des applications accessibles par l'Internet. Discipline jeune, basée sur les efforts antérieurs de l'industrie, les architectures orientées services ont trouvé au sein des procédés métiers le champion qui a su les acclimater aux problématiques de la recherche. Dans un contexte organisationnel certes toujours complexe mais récemment stabilisé, les architectures orientées services sont en mesure d'établir leur propre stratégie de conception et d'implémentation. Alors qu'auparavant l'industrie semblait construire des paradigmes d'architectures orientées services en fonction de leur besoins, c'est maintenant les architectures orientées services qui déterminent les éléments majeurs de leur problématique. Par *architecture*, il ne faut pas entendre des implémentations ou de technologie pures mais l'affirmation d'une culture *service* structurée originellement autour des notions d'autonomie, de couplage faible, de dynamisme et de nombreuses autres.

Certes, durant les développements des concepts et des technologies des services Web, l'entreprise virtuelle qui les emploient dut affronter les limitations des environnements d'exécution centralisés. L'introduction de nombreuses applications de partage de fichiers multimédias P2P attire l'attention des acteurs de la communauté de procédés métiers. Les avantages des conceptions et les architectures P2P expliquent pourquoi les scientifiques et les industriels voient en ce modèle une véritable alternative aux réalisations actuelles. Même si de bonnes propriétés motivent l'utilisation des systèmes P2P au sein de l'entreprises virtuelles, à cause de leurs limitations, les organisations sont loins de les considérer comme un sanctuaire pour implémenter leur procédés métiers.

Comme nous avons détaillé dans ce chapitre, les efforts de standardisations considèrent l'utilisation de l'approche workflow pour la conception et l'implémentation des architectures orientées services qui visent la composition. L'émergence de l'approche workflow dans l'entreprise virtuelle est restée un phénomène relativement discret au cours des années 1990. Dans les années qui suivent, l'évolution de l'approche workflow suit des voies qui font de la décentralisation l'un des aspects essentiels pour son développement. Ces travaux qui précédent la vision service, sont motivés par l'évolution du support réseau, l'espoir immense suscité par le potentiel *illimité* de la décentralisation qui incite en effet à leur utilisation. La motivation essentielle des travaux qui portent sur l'exécution décentralisée de workflow porte sur la répartition des moyens dans un système distribué comme par exemple le déploiement des instances vers des machines plus disponibles. Cela constitue, il est vrai, un défi important en termes de complexité dans un domaine nouveau. Pourtant elle ne prend pas un essor nouveau dans le contexte des architectures orientées service. Il est clair que les environnements de gestion de procédé décentralisés actuels ne remplissent pas les nouvelles exigences de l'approche service. Certains d'entre eux ne sont que des applications légères qui implémentent les protocoles de communication auprès des participants mais ne fournissent qu'un support très relatif aux nouvelles exigences des systèmes structurés autour des architectures orientées services comme par exemple l'autonomie. Parce

que tout simplement les organisations qui sont accessibles en tant que services ne peuvent pas avoir l'intention d'implémenter les protocoles ou les applications exigés par l'environnement de décentralisation. De ces observations, on peut retirer deux conclusions essentielles. La première limitation est que les systèmes de gestion de workflow ne considèrent pas les critères divers qui peuvent exister dans un procédé inter-organisationnel et se penchent seulement sur les avantages liés à la performance de la décentralisation. La deuxième est leur problème d'adéquation aux architectures orientées services. Intuitivement, les protocoles ou les scripts qu'ils exigent peuvent être contradictoires avec l'autonomie des services qui sont censés être dynamiques et ouverts à l'évolution interne. Quoi qu'il en soit, l'utilisation des systèmes de gestion de workflow décentralisés devient de plus en plus acceptable.

De l'autre côté, les efforts de standardisation ont permis l'émergence de plusieurs aspects nouveaux qui n'étaient pas présents dans le contexte de l'entreprise virtuelle. Un de ces aspects est la mobilité des spécifications basées sur XML entre les organisations et même les outils. Par exemple, un procédé spécifié ou instancié en WS-BPEL par une organisation peut être facilement reçu et exécuté par une autre organisation. Cet environnement d'exécution pour le procédé reçu est très raisonnable puisque ce dernier est une spécification standardisée. La mobilité des spécifications commence à attirer l'attention des chercheurs de communautés diverses. Mais aucune contribution n'est établie. Chaque contribution se penche sur un aspect indépendant de la mobilité ou de la décentralisation sans se focaliser sur les motivations et les défis sous-jacents. On peut également ajouter la réalisation concrète de la mobilité des spécifications aux manques soulignés ci-dessus. Nous pouvons résumer la synthèse des travaux liés et l'analyse que nous avons fait en quelques lignes comme suit : *Une nouvelle forme de conception et d'implémentation des procédés métiers structurée autour des services s'installe dans la communauté aussi bien scientifique qu'industrielle. Certes, cette nouvelle approche a plusieurs avantages. L'exécution décentralisée des procédés métiers est étudiée largement dans les travaux qui précédent l'approche service mais les contributions ne sont pas en adéquation avec les nouvelles exigences de cette dernière. Les apports secondaires de la standardisation ont des relations orthogonales avec les exigences des procédés métiers en termes de décentralisation. Ces relations ne sont pas claires et sont peu étudiées dans la littérature.*

Chapitre 4

La modélisation d'un procédé et sa décentralisation *par défaut*

Sommaire

4.1 Introduction

Dans ce chapitre, nous présentons notre première contribution qui concerne la définition d'un procédé métier qui caractérise la composition d'un ensemble de services. Ce premier point est essentiel pour aborder de façon cohérente et réaliste l'exécution décentralisée d'une composition. Par la suite, nous présentons notre technique qui décentralise un procédé sans se baser sur un critère précis de conception comme peuvent l'être les politiques de flux d'information.

Le choix d'un modèle de spécification semble faire partie des problèmes sujets à polémiques, tant il est intimement lié à la culture scientifique des concepteurs de procédé, au domaine d'application, et aussi aux implémentations visées. Le modèle présenté dans ce chapitre est un modèle théorique qui a pour avantage d'être simple et précis. Mais dans le chapitre sur l'implémentation, nous justifions la mise en œuvre avec un langage de spécification concret comme WS-BPEL. Par la suite de ce chapitre, nous allons utiliser l'exemple de procédé illustré dans la sous-section 4.2.2. Cet exemple est une version étendue de l'exemple de motivation que nous avons utilisé dans le chapitre de problématique. Si l'exemple est simple, nous attirons l'attention sur le fait qu'il est

suffisamment général pour exprimer les différents types de concepts qui nous intéressent. Les algorithmes sont d'abord décrits d'une manière abstraite et puis illustrés par des exemples.

4.2 Le modèle du procédé

La modélisation des procédés n'est pas nouvelle, puisque l'on s'y intéresse depuis de nombreuses années, d'abord des procédés industriels très automatisés, puis des procédés plus créatifs impliquant des interactions de personnes à personnes, comme les procédés logiciels [Godart1993]. Comme le montre bien l'étude bibliographique, il n'existe pas une solution universelle aux problèmes que posent la spécification d'un procédé qui compose un ensemble de services. Tout au plus, une proposition peut cibler au mieux une classe de problèmes à résoudre. Nous pensons que plutôt que de définir un langage, il s'agit de réutiliser des formalismes existants. Le formalisme que nous utilisons est composé des concepts empruntés à des domaines variés : workflow [Leymann and Roller2000], théorie des graphes [Bondy and Murty1976], systèmes d'information [Fischer2007] ... Il circonscrit le type de concepts que l'on veut exprimer et le type de problèmes que l'on peut espérer résoudre.

4.2.1 Éléments caractéristiques du modèle proposé et hypothèses préliminaires

Un procédé métier est la production d'un objet ou la réalisation d'une tâche. Le modèle d'un procédé métier décrit un moyen systématique pour produire un objet ou réaliser une tâche d'un certain type, d'une certaine classe. Une étude rapide des langages de programmation met en évidence différents paradigmes de modélisation des procédés (procédurale, logique, objet, fonctionnelle, impérative, déclarative, séquentielle, parallèle ...). Chaque style de programmation semble mieux correspondre à certains domaines d'application que d'autres [Osterweil1997]. Ceci est valable pour la spécification des procédés. Comme détaillé dans l'état de l'art, un modèle de procédé exécutable motive plus son utilisation qu'un modèle déclaratif ou partiellement exécutable ne l'est pas. Cette motivation conduira à une meilleure spécification de composition et en conséquence à une meilleur exécution et gestion. Comme nous l'avons déjà souligné, un procédé métier qui spécifie la composition d'un ensemble de service consiste essentiellement à définir la relation d'ordre qui doit être maintenue entre les invocations des services composés [Weerawarana *et al.*2005]. Il s'agit du flux de contrôle du procédé. Un modèle de procédé définit également les dépendances de données entre les différents invocations. Ce dernier constitue le flux de donnée du procédé [Leymann and Roller2000]. La structure du flux de contrôle peut engendrer différent types de dépendances de contrôle entre les éléments du procédé. Ces dépendances peuvent être exprimées par des patrons de contrôle (AND-join, AND-split, OR-join ...) d'une manière abstraite. Il n'existe pas actuellement de consensus sur la façon de représenter le flux de données ou d'autres types de flux comme le flux transactionnel [Hagen and Alonso2000]. Dans la modélisation des procédés, nous étudions trois points principaux. Le premier point concerne le flux de contrôle du procédé. Le deuxième est une généralisation des dépendances de données. Finalement, le troisième point concerne les aspects conversationnels des services composés.

Hypothèses concernant le modèle proposé : Si certains procédés peuvent se décrire dans un style de programmation impérative comme des workflows, les descriptions arbitraires peuvent être difficiles à analyser et à manipuler [Ouyang *et al.*2006]. Pourtant, il est souvent possible de passer d'un modèle arbitraire à un modèle équivalent plus structuré toute en gardant la sémantique. Le concept de *procédé structuré* est proposé dans [Kiepuszewski *et al.*2000]. On appelle un procédé structuré, une spécification de procédé qui est restreinte par un certain nombre de

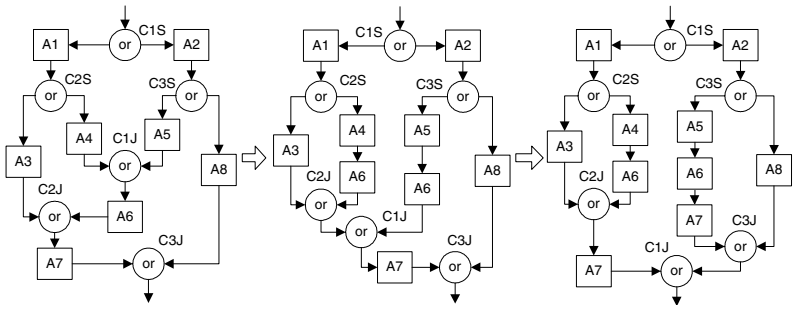


FIG. 4.1 – La transformation d'un modèle de procédé arbitraire en un modèle structuré en deux étapes [Liu and Kumar2005]

contraintes syntaxiques au niveau du flux de contrôle. Dans un procédé structuré, à chaque élément de disjonction (AND-split, OR-split ...) doit correspondre un élément de jonction de même type (AND-join, OR-join ...). Ainsi, un procédé structuré peut être construit d'une manière récursive en imbriquant les fragments par des paires appropriés. Le figure 4.2.1 illustre un exemple d'un procédé arbitraire et sa transformation en un procédé structuré équivalent en utilisant les techniques présentées dans [Liu and Kumar2005]. Il est clair que nous avons dû faire cette hypothèse simplificatrice pour nous concentrer plus sur les difficultés de notre approche. Nous pourrions travailler avec des procédés arbitraires mais au coût de faire des analyses extrêmement sophistiquées. Nous pouvons constater que cette difficulté peut être limitée en assumant que les procédés arbitraires sont transformés en leur équivalent structuré dans un étape qui précède la décentralisation. En vue de la quantité des travaux et des outils portant sur les opérations de transformation [Liu and Kumar2005] [Sadiq and Orlowska2000b] [Sadiq and Orlowska2000a], cette hypothèse semble raisonnable. Plus formellement, cette hypothèse est exprimée par Hypothèse 1.1.

Hypothèse 1 ***(Procédé Structuré)***
Un procédé, P, qui spécifie une composition de services est un *procédé structuré*.

Un des problèmes des procédés est la gestion de la concurrence entre ses activités [Schuldt *et al.*1999]. La concurrence concerne la manipulation des variables globales du procédé par des activités exécutées en parallèles. Alors que la concurrence est considérée comme une situation problématique qui rend l'exécution difficile, certains techniques nouvelles ont prouvé leur intérêt dans la phase d'exécution [Guabtni *et al.*2006]. Comme nous nous intéressons aux échanges directs entre des services composés au sein d'un procédé, la concurrence n'est pas un aspect souhaité de notre mode d'exécution où il ne peut pas avoir des données partagées. En conséquence, nous assumons que la concurrence n'existe pas dans les modèles de procédé que nous considérons. Il est à noter que la concurrence peut être détectée pendant les phases de conception du procédé en employant les différentes techniques comme [Marjanovic2000]. Nous pouvons concrétiser cette hypothèse par Hypothèse 1.2.

Hypothèse 2 ***(Concurrence)***
Un procédé, P, qui spécifie la composition de services ne contient pas des activités concurrentes qui peuvent accéder aux mêmes données de procédé en même temps.

La présence de l'hypothèse 1.2 est équivalente à l'hypothèse suivante (Hypothèse 1.3). Car, dans une modélisation abstraite la présence de la concurrence est caractérisée par l'échange de données entre les activités qui ne sont pas sur le même chemin.

Hypothèse 3 ***(Flux de contrôle)***
Un procédé, P, qui spécifie la composition de services ne contient que des dépendances de données entre les activités qui sont sur le même chemin de contrôle.

Nous avons fait également des hypothèses concernant la modalité de communication entre les services qui exécuteront les procédés. Nous détaillons ces hypothèses au long de ce chapitre.

4.2.2 Exemple illustratif

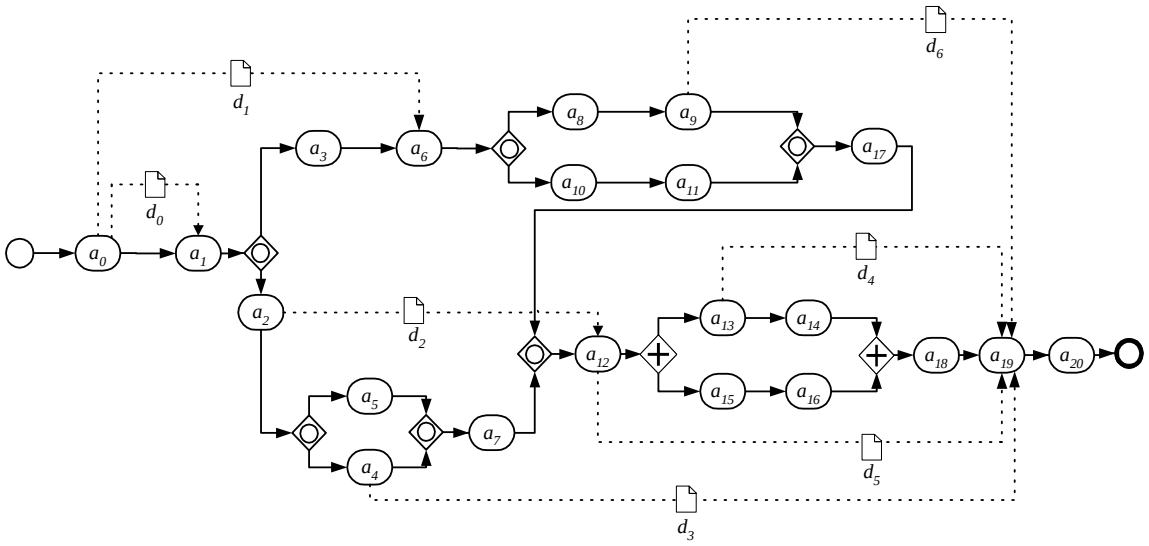


FIG. 4.2 – Un exemple de procédé

Dans cette sous-section nous présentons un procédé qui compose un certain nombre de service. Dans le reste de ce chapitre, nous allons l'utiliser comme exemple de motivation. Son objectif est d'illustrer simplement le fait que les dépendances de contrôle et de donnés peuvent co-exister sous différentes formes. A ce niveau, nous préférons utiliser cet exemple abstrait au lieu d'un procédé réel en vue de faciliter les notations, parce qu'il est plus simple d'illustrer des aspects conversationnels sur le même exemple en changeant la configuration de ce dernier. Le procédé contient 21 activités : a_0, a_1, ... , a_{20}. Les activités sont liées les unes les autres par les dépendances de contrôle et de donnée. Les dépendances de contrôle caractérisent l'ordre

partiel d'exécution des activités. Les dépendances de données caractérisent les relations entre les entrées et les sorties des activités du procédé. Les dépendances de donnée sont : d_0, ..., d_6. Les différentes conditions de transition liées à la combinaison des dépendances de contrôle sont représentées par deux types de patrons : AND-split/AND-join et OR-split/OR-join. Le patron AND-split définit un point de procédé où le flux de contrôle se divise en plusieurs flux de contrôle en parallèle, permettant une exécution concurrente. Respectivement, le patron AND-join définit un point de procédé où plusieurs flux de contrôle concurrents convergent et se synchronisent en un seul flux de contrôle. Le patron OR-split définit un point de procédé où, selon des conditions de transition, un sous ensemble de flux de contrôle est activé. Respectivement, le patron OR-join définit un point de procédé où un sous ensemble du flux de contrôle convergent et se synchronisent, s'ils sont déjà activés. Une question naturelle se pose : les activités correspondent à quoi ? Conformément à ce qui a été dit dans les sections précédentes, les activités d'un procédé qui composent un certain nombre de services sont essentiellement des invocations des opérations qui appartiennent aux services. L'invocation d'un service peut se faire avec des interactions *uni* ou *bi*-directionnelles. Une interaction uni-directionnelle peut évoquer un service ou bien recevoir une invocation. Une interaction bi-directionnelle fait une invocation à un service externe et attend en retour une invocation de la part du dernier. On peut la considérer comme deux interactions uni-directionnelles consécutives. Il est à noter que pour tester les conditions de transitions pour un OR-split, un certain nombre de variables de procédé peuvent être testées. Pour ce faire, les activités locales doivent être exécutées. Les activités locales d'un procédé ne font pas appel aux services externes. Chaque activité peut avoir un certain nombre d'entrées ainsi que de sorties. Les activités locales ont seulement des entrées comme elle ne fournissent rien. Les activités uni-directionnelles peuvent avoir des données de sortis si elles invoquent des services externes ou bien elles peuvent avoir des entrées si elles reçoivent des invocations venant des services externes. Un procédé a une unique activité de *start* et une unique activité *finale*.

4.2.3 Définition formelle

Après une étude générale des procédés qui caractérisent le modèle considéré, nous donnons une présentation formelle de ce dernier.

Définition 4 ***(Procédé)***
Un procédé, P, est un tuple ($\mathcal{A}$, $\mathcal{D}$, $\mathcal{E}_c$, $\mathcal{E}_d$, $\mathcal{S}$) où :

1. $\mathcal{A}$ est un ensemble d'activités,
2. $\mathcal{D}$ est un ensemble de données de procédé,
3. $\mathcal{E}_c \subseteq \mathcal{A} \times \mathcal{A} \times \mathcal{C}onds(\mathcal{D})$ est un ensemble de dépendances de contrôle avec des conditions de transition définies sur les éléments de $\mathcal{D}$,
4. $\mathcal{E}_d \subseteq \mathcal{A} \times \mathcal{A} \times \mathcal{D}$ est un ensemble de dépendances de données,
5. $\mathcal{S}$ est un ensemble de service composés.

Les activités ont des identités comme a_i. Si une activité invoque un service, elle la référence (voir Définition 4.2.3). Les activités conversationnelles qui référencent le même service s_i sont regroupées dans un ensemble $\mathcal{A}_{s_i}$. Une dépendance de contrôle $e \in \mathcal{E}_c$entre deux activités a_i et

a_j est notée $a_i \xrightarrow{c} a_j$. De même, la dépendance de donnée de deux activités a_i et a_j est notée $a_i \xrightarrow{d} a_j$. Une autre notation concerne un chemin de contrôle qui relie deux activités. Celle-ci est notée comme un ordre partiel $a_i < a_j$.

Définition 5 ***(Activité)***
Une activité, a_i, P, est un tuple ($\mathcal{I}n_i$, $\mathcal{O}ut_i$, s, *type*) où :

1. $\mathcal{I}n_i \subseteq \mathcal{D}$ est l'ensemble des entrées de a_i,
2. $\mathcal{O}ut_i \subseteq \mathcal{D}$ est l'ensemble des sorties de a_i,
3. $s \in \mathcal{S}$ est le service invoqué par l'exécution de a_i,
4. *type* $\in \{write, read, local\}$ est le type de a_i,

Une activité peut avoir un certain nombre d'entrées et de sorties. L'activité réfère un service qu'elle invoque pour son exécution par un moteur de procédé. L'interaction d'un moteur de procédé avec un service peut se faire de deux manières : la réception ou l'émission d'un message. Dans le premier cas, on parle d'une activité de type *read* alors que dans le deuxième cas on parle d'une activité de type *write*. Elles sont respectivement notées comme a^r et a^w. Un procédé peut aussi inclure des activités qui n'ont pas d'interaction avec un service particulier. C'est le cas des activités qui manipulent les variables ou les données locales au moteur d'exécution. Par exemple, une activité qui crée une commande pour déclencher un processus de production n'a souvent pas besoin d'entrées à part un message de contrôle. Dans notre modélisation, ce sont les activités de type *local*. Elles sont notées comme a^l.

Les activités ont des dépendances entrantes et sortantes. En conséquence nous proposons deux ensembles qui les regroupent.

Définition 6 ***(Postset)***
Le postset d'une activité a_i est notée $a_i\bullet$. Ce dernier contient les activités dont a_i est la source de leur dépendances rentrantes. Plus formellement, $a_i\bullet = \{a_j \in \mathcal{A} |\ a_i \xrightarrow{c} a_j \vee a_i \xrightarrow{d} a_j\}$.

Le *postset* d'une activité est l'ensemble des activités que ses dépendances sortantes visent. Ces dépendances peuvent être des dépendances de données et de contrôles.

Exemple 1 (Postset) *La figure 5.3 illustre un exemple de procédé. Prenons l'activité a_2. a_2 a deux activités qui la suivent par le flux de contrôle : a_4 et a_5. Une troisième activité, a_{12}, a une dépendance de donnée avec a_2. Ainsi le postset de a_2, $a_2\bullet = \{a_4, a_5, a_{12}\}$. L'activité a_4 a une activité qui la suit par le flux de contrôle. C'est l'activité a_7. a_4 a une dépendance de donnée avec l'activité a_{19}. En conséquence, $a_4\bullet = \{a_7, a_{19}\}$.*

Comme nous avons regroupé les activités qui sont visées par les dépendances sortantes, nous regroupons les activités qui sont à la source des dépendances rentrantes dans un ensemble qu'on appelle *preset*.

Définition 7 ***(Preset)***
Le *preset* d'une activité a_i est notée $\bullet a_i$. Ce dernier contient les activités dont a_i est la cible des dépendances sortantes. Plus formellement, $\bullet a_i = \{a_j \in \mathcal{A} |\ a_j \xrightarrow{c} a_i \vee a_j \xrightarrow{d} a_i\}$

Exemple 2 (Preset) *Prenons l'activité a_{19} du procédé illustré dans la figure. L'activité a_{19} a quatre dépendances de donnée entrantes d_3, d_4, d_5 et d_6. Elles correspondent aux activités a_4, a_{12}, a_{13} et a_9. L'activité a_{19} a une dépendance de contrôle entrante venant de l'activité a_{18}. Ainsi, le preset de l'activité a_{19} est notée $\bullet a_{19} = \{a_4, a_9, a_{12}, a_{13}, a_{18}\}$.*

Jusqu'ici nous avons présenté les éléments de base d'un procédé inter-organisationnel qui caractérise la composition d'un ensemble de services. Nous allons étendre ces définitions avec des concepts additionnels. Avant de présenter les autres définitions, nous présentons les principes de base de la décentralisation qui va traiter ces spécifications.

4.3 Principes de décentralisation et l'originalité de l'approche

Comme nous avons souligné plusieurs fois dans les chapitres précédents, notre approche est basée sur l'exécution de procédé coopérants dérivés à partir d'une spécification centralisée de telle sorte que les dépendances du procédé centralisé soient re-établies avec des interactions égal-à-égal entre les services sous-jacents. La production des procédés coopérants à partir de la spécification centralisée est un tâche difficile.

Dans cette section, nous présentons les cas problématiques en général et nous cherchons à expliquer comment nous les avons résolus d'une manière systématique. Par la suite, nous rappelons certains éléments de fond qui servent de briques de base comme la sémantique des procédés distribués et la cohérence des procédés coopérants. Le premier point concerne l'étude des procédés décentralisés pour à la fois guider et discipliner le raisonnement dans l'opération de décentralisation. Le second point concerne la définition des règles de transformation et de construction mécanisables que nous proposons pour dériver les procédés coopérant à partir de la spécification centralisée. Dans les deux étapes, nous expliquons certaines originalités comme :

- Un procédé peut contenir plusieurs types de dépendances. Nous ne nous limitons pas seulement à l'analyse du flux contrôle contrairement aux nombreuses travaux liés comme [van der Aalst and Weske2001] [Sadiq *et al.*2006] [Atluri *et al.*2001]. Nous considérons le flux de données en même temps que le flux de contrôle. Nous étudions les différents types de dépendances liées à la combinaison de ces deux formes.
- En dehors de sa nature formelle, la décentralisation nécessite la considération de l'intergiciel sous-jacent. Pour cela nous nous intéressons à la sémantique des procédés coopérants [Bultan *et al.*2006]. Cela inclut la communication synchrone, la communication asynchrone, les modes de lecture/écriture dans les connexions (channels) de communication etc. En conséquence, notre opération de décentralisation considère les aspects fonctionnels ainsi que les aspects formels.
- Quand un procédé est décentralisé, une exigence naturelle est la préservation de son comportement centralisé. Cette exigence nécessite une preuve claire et précise des opérations employées. L'équivalence des comportements est une première caractéristique importante que nous sommes obligés de vérifier avant d'évaluer l'aptitude d'une approche donnée à répondre aux différents besoins.

Avant de passer aux étapes suivantes, il est souhaitable de rappeler quelques éléments de base que nous avons utilisé pour concrétiser nos idées. Les services Web, les procédés qui composent les services comme des workflows, et les systèmes communicants en général sont basés sur des simples principes théoriques.

La sémantique des procédés distribués et coopérants

Les procédés coopérants et distribués sont un sujet d'étude depuis de nombreuses années [Kahn1974]. Les notions théoriques bien établies comme la communication, la concurrence et l'équivalence sont concrétisées par les algorithmiques parallèles, les worfklows, les grids et dernièrement les services Web dans le domaine informatique. La sémantique de la communication sous-jacente choisie influence directement le raisonnement aux niveaux d'abstraction plus hauts. La sémantique de la communication sous-jacente concerne la modélisation de la structure de communication qui relie deux éléments communicants comme par exemple la taille de celle-ci, les modalités d'écriture et de lecture concernées, ou la fiabilité. La sémantique des procédés communicants peut être vue comme l'équivalence des traces ou l'analyse du comportement des éléments communicants. Nous limitons notre discussion seulement à la structure des communications qui concerne la suite de notre contribution. Une structure de communication, souvent appelé un *channel* en anglais, relie deux entités qui communiquent. Dans les travaux théoriques, elle est modélisé comme illustré dans la figure 4.3. Dans des applications plus concrètes, il s'agit d'une espace mémoire partagée par les deux entités. Il est à noter qu'un troisième entité peut l'implémenter si les entités communicantes ne la partagent pas. Pendant la communication, une entité communicante envoie des données dans cette structure et l'autre reçoit les données de la même manière. Pour communiquer de cette manière, l'entité expéditeur exécute une activité qui écrit la donnée dans le channel et l'entité récepteur exécute une activité qui lit la donnée à partir du channel. Dans notre spécification, ce sont les activités de type *writer* et *read*. Comme deux entités communicantes sont indépendantes, elles peuvent exécuter ces activités également indépendamment l'une de l'autre. Cela peut poser un problème de synchronisation. Quel doit être le comportement de l'entité expéditrice si l'entité réceptrice n'est pas en un état d'accepter la donnée envoyée ? De même, qu'est-ce qu'il se passe si l'entité réceptrice est dans l'état réception mais la donnée n'existe pas dans le channel ? Doit-elle continuer son exécution ou attendre la donnée ? Dans le cas où les deux entités échangent plusieurs messages quelle est la politique de lecture à partir du channel ? Les réponses à ces questions simples définissent la sémantique des procédés communicants. Si les concepts de lecture/écriture et de channel peuvent être décrites de manière abstraite, leur implémentation concrète se fait dans le contexte des services Web [Barros *et al.*2005].

Les procédés communicants considèrent deux modes de communication essentiels : *Synchrone* et *Asynchrone*. La première modalité nécessite une synchronisation forte entre deux entités qui échangent des données de telle sorte que les deux exécutent respectivement une activité qui envoie la donnée et une activité qui reçoit la donnée envoyée. Si une donnée envoyée n'est pas reçue par son destinataire ou bien n'est pas envoyée par son expéditeur, les procédés exécutés par les entités restent bloqués. La deuxième modalité est plus flexible. Le channel qui relie les deux entités qui communiquent peut avoir une taille illimitée en théorie et suffisamment grande en réalité. Comme la taille du channel est illimitée, le procédé qui envoie des données dans le channel n'est pas bloqué même si le récepteur ne reçoit pas les données en même temps. Pourtant, le procédé qui doit consommer les données reste bloqué tant qu'il n'y a pas de donnée dans le channel. Pour le procédé, on peut définir des politiques de lectures comme FIFO, LIFO ou bien la lecture arbitraire. Pourtant cela ne change pas les lectures bloquantes à partir des channels.

La communication asynchrone est la modalité utilisée sur l'internet ainsi que par les services Web [Kazhamiakin *et al.*2006]. La modalité synchrone peut être vue comme un cas restreint de la modalité asynchrone qui utilise un channel de taille limitée à une seule donnée.

Dans la suite, nous allons utiliser la modalité asynchrone parce qu'elle est la plus répandue dans les réalisations concrètes. La modalité de lecture à partir des channels est la lecture arbitraire. Nous décrivons cette hypothèse plus formellement dans l'hypothèse 8.

Hypothèse 8 ***(Lecture arbitraire)***
La lecture à partir des channels de communication se fait d'une manière arbitraire. N'importe quelle donnée présente peut être consommée par le procédé récepteur.

Soulignons aussi que la fiabilité de communication est très importante dans un contexte où les messages envoyés sur les channels peuvent être perdus qui réduisent sérieusement la capacité d'application des autres restrictions citées ci-dessus. Nous faisons l'hypothèse de channels fiables en vue d'éviter les ambiguïtés concernant les sémantiques des procédés coopérants. Plus formellement, celle-ci est décrite dans l'hypothèse 9.

Hypothèse 9 ***(Communication parfaite)***
Les channels de communications entre les procédés sont parfaits. Aucune donnée envoyée dans un channel n'est perdue ni dupliquée.

Cette hypothèse peut être facilement satisfaite en utilisant les technologies récentes comme WS-ReliableMessaging.

Un Critère de décentralisation : Global Soundness

Par rapport à la sémantique des procédés coopérants et aux notions d'équivalence connues dans la littérature [Milner1999] [Krzysztof Apt1997], en 1999, Van Der Aalst a formulé un ensemble de critères connu comme *Global Soundness* [van der Aalst1999]. Il s'agit d'une extension de la notion de *Soundness* proposée par le même auteur dans ses travaux antérieurs [van der Aalst1998]. La notion de *Soundness* est un critère de correction pour un workflow centralisé qui est exécuté par une seule entité. Il peut être vérifié en utilisant les propriétés de *liveness* et *boundedness* des réseaux de Petri. Il est à noter que la modélisation des procédés avec les réseaux de Petri est une hypothèse de ce travail. La mise en place de son approche consiste à vérifier que le procédé se termine, et à sa terminaison il n'y a qu'un seul jeton dans sa place terminale. Quant à la propriété du *Global Soundness*, en résumé, elle s'intéresse à la vérification des procédés coopérants (ou bien inter-organisationnels) avec des connections (channels) synchrones et asynchrones. On peut considérer le global soundness comme un ensemble de critères minimaux qu'un procédé inter-organisationnel doit satisfaire. Dans un premier temps les communications prévues des procédés et les chemins qu'ils peuvent prendre pendant leur exécution sont examinés. Ensuite, une technique de vérification, qui est toujours basée sur les propriétés des réseaux de Petri, est introduite. Avec cette technique, on peut vérifier s'il existe des situations bloquantes dans les chemins possibles. Ensuite, la propriété du Global Soundness introduit des critères similaires à ceux de la propriété de soundness : chaque procédé doit se terminer et à sa terminaison, il ne doit comporter qu'un seul jeton dans sa place de terminaison.

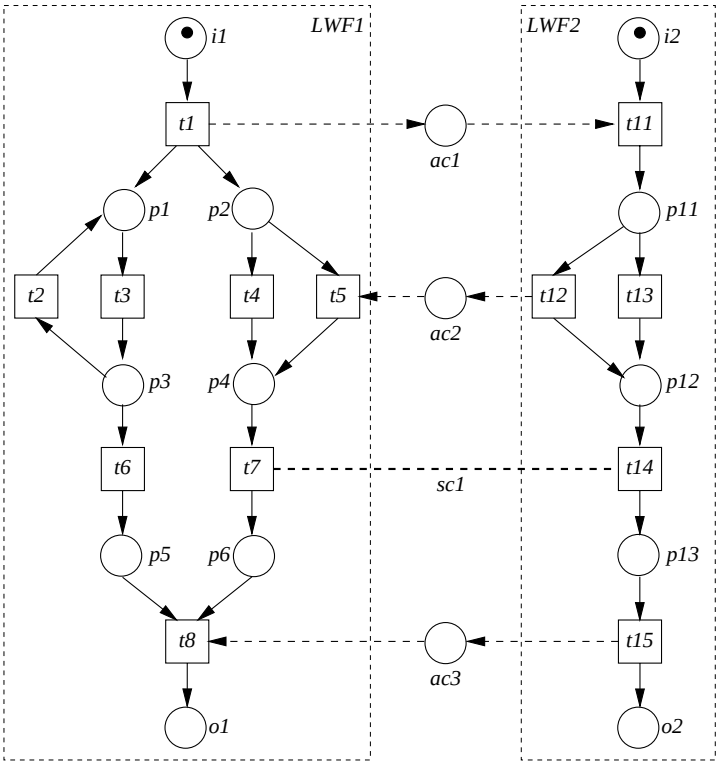


FIG. 4.3 – L'exemple de *Global Soundness* [van der Aalst1999]

Cela veut dire que tous les messages envoyés doivent être consommés par leurs destinataires. Le dernier critère n'est pas une situation bloquante qui empêche l'exécution, mais il n'est pas souhaitable pour la terminaison appropriée des procédés. Certains travaux pertinents [Wombacher2006] [Kindler *et al.*2000] étudient le global soundness dans un contexte décentralisé tout en respectant le travail original. Le problème essentiel qu'ils essaient de résoudre est la vérification de global soundness sans avoir une vue globale des procédés coopérants.

Exemple 3 (Global Soundness) *La figure 4.3 illustre un exemple de procédés inter-organisationnel pris du papier original de Van Der Aalst[van der Aalst1999]. Il s'agit de deux procédés coopérants, LWF1 et LWF2, modélisés avec des réseaux de Petri. En conséquence, seulement le flux de contrôle est considéré. Il y a quatre éléments de communications, ac1, ac2, sc1 et ac4. Ils décrivent les communications des activités correspondantes. Par exemple, l'activité t11 dans LWF2 doit attendre la complétion de l'activité t1 dans LWF1. L'élément synchrone sc1 force des activités t7 et t14. Considérons que LWF1 exécute l'activité t4 et LWF2 exécute t12. Dans ce cas, le message qui est associé à ac2 n'est pas traité comme il faut. Parce que le message envoyé par LWF2 par ne sera jamais consommé par LWF1 parce qu'il ne prend pas ce chemin. En conséquence, même si LWF2 n'est pas bloqué, il ne peut pas terminer correctement par rapport aux critères de global soundness. Considérons maintenant, les activités t5 et t12. Si LWF1 exécute t5, il va attendre le message que LWF2 doit fournir à travers ac2. Plus précisément, ce message est fourni par t12 que LWF2 doit exécuter. Pourtant, si LWF2 prend le chemin de l'activité t13, l'activité t12 n'est jamais exécutée. En conséquence, LWF1 reste bloqué.*

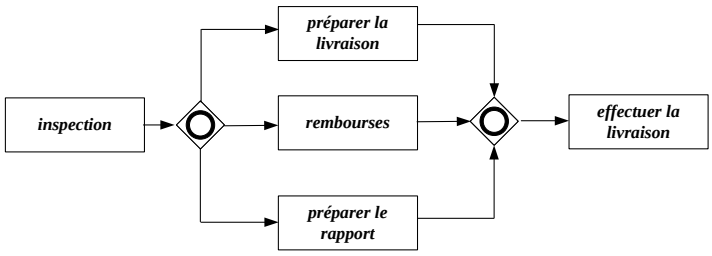


FIG. 4.4 – L'élimination des chemins morts dans une exécution centralisée

Comme l'exemple 3 l'illustre, les critères de cohérence des procédés coopérants sont simple à comprendre mais difficile à vérifier. Dans un premier temps les techniques de vérification nécessitent la spécification des procédés locaux présents dans le procédé inter-organisationnel. Dans le cas contraire, les approches alternatives peuvent être utilisées. Dans notre travail, nous ne cherchons pas à vérifier la cohérence des procédés locaux existants. Contrairement à la majorité des travaux existants, notre but n'est pas la vérification de la cohérence des procédés coopérants. Nous produisons des procédés coopérants à partir d'une spécification complète centralisée. En conséquence, le critère peut être vérifié à la fin de l'opération. La conformité de la réalisation pour chaque niveau de la conception en commençant par les parties les plus élémentaires puis en remontant progressivement jusquau procédé entier. Pour cela, en correspondance à chaque phase de la transformation est associée une phase de test et d'évaluation de la conformité. Ainsi, les deux démarches descendante et ascendante sont complémentaires.

L'Élimination des chemins morts (Dead Path Elimination) : Conformément à ce qui a été dit précédemment, notre approche met en œuvre la sémantique de la spécification centralisée dans une exécution décentralisée. L'élimination des chemins morts (DPE) est un processus particulier que les moteurs d'exécution de procédé supportent. Il s'agit des spécifications dans lesquelles un point de jointure peut causer un problème de synchronisation. L'implémentation de ce processus dans un moteur d'exécution centralisé n'est pas difficile. Mais il est à noter que certains moteurs d'exécution ne le supportent pas [Leymann and Roller2000] à cause des difficultés liées à l'implémentation. Ce processus a un rôle important dans une exécution décentralisée où les dépendances de données et de contrôles peuvent co-exister. Nous allons utiliser les principes de DPE pour concrétiser nos algorithmes. Avant de passer à la suite, nous rappelons brièvement ce principe.

Exemple 4 (L'élimination des chemins morts) *Nous considérons le fragment de procédé illustré dans la figure 4.4. La première activité est l'activité* ***inspection****. Après l'exécution de cette activité, la réclamation peut être remboursée ou refusée. Si la réclamation est remboursée, l'activité* ***rembourser*** *est exécutée sinon cette activité n'est pas exécutée. Les activités* ***préparer le rapport*** *et* ***préparer la livraison*** *sont exécutées dans les deux cas. L'activité* ***effectuer la livraison*** *est exécutée après la terminaison de toutes les activités précédentes activées. Dans cet exemple l'activité* ***rembourser*** *peut être suspendue. La problématique est la terminaison des activités* ***préparer la livraison*** *et* ***préparer le rapport*** *n'est pas suffisante pour l'activation de l'activité* ***effectuer la livraison*** *: il faut savoir si l'activité* ***rembourser*** *est activée ou pas. La suspension de l'activité* ***rembourser*** *doit être considérée par le système de gestion de procédé pour évaluer les conditions d'activation de* ***effectuer la livraison****. Cette opération s'appelle l'élimination des chemins morts (Dead Path Elimination, DPE en anglais).*

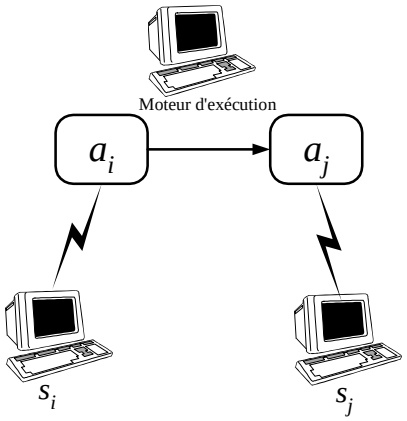


FIG. 4.5 – Une dépendance de contrôle entre deux activités qui évoquent des services différents

4.3.1 La décentralisation d'un procédé centralisé

Dans cette section, nous présentons notre technique de transformation qui dérive les procédés coopérants à partir d'une spécification centralisée. Nous commençons par expliquer l'opération qui dérive les procédés coopérants par des exemples intuitifs puis nous détaillons comment nous maîtrisons les cas essentiels ainsi que les cas problématiques.

Prenons deux activités a_i et a_j dépendantes d'une spécification centralisée par une dépendance de contrôle. Admettons que ces deux activités correspondent aux interactions entre deux services différents respectivement s_i et s_j comme illustrée dans la figure 4.5. Quand un moteur d'exécution exécute la spécification centralisée, d'abord a_i est exécutée et suite à la terminaison de son exécution a_j est exécutée. Pendant ce processus, les deux services sous-jacents s_i et s_j n'ont pas d'interaction directes. Ils sont évoqués par le moteur d'exécution qui exécute a_i et a_j. Comme nous l'avons déjà souligné, nous voulons que s_i et s_j aient des interactions directs quand ils sont choisis pour cette composition. Pour ce faire, nous allons fournir deux procédés qui seront dynamiquement associés aux services s_i et s_j de telle sorte que cette dépendance soit une interaction d'égal-à-égal entre s_i et s_j. Nous appelons ces deux procédés P_{s_i} et P_{s_j}. La première question de la transformation est : *Quelles sont les activités de ces procédés ?* La seconde question est : *Comment sont elles structurées ?* Dans le cas ci-dessus, P_{s_i} doit informer P_{s_j} à propos de la terminaison de son opération. Dans ce cas a_i doit être exécutée par P_{s_i} et a_j par P_{s_j}. Cela évite des messages de contrôle inutiles entre s_i, s_j et le moteur d'exécution qui devait exécuter la spécification centralisée. Il est à noter que le contenu de la donnée échangée lors d'un échange directe de contrôle de deux fragments n'est pas le même qu'un échange directe de données. La figure 4.14(a) illustre les deux procédés P_{s_i} et P_{s_j} dérivés à partir de la spécification de la figure 4.5. Si cet exemple est simple, il explique bien le principe de base de l'opération de décentralisation qui dérive les procédés coopérants. En partant de ce principe simple nous présentons d'autre type de fragments de procédé et nous essayons d'expliquer la généralisation de cette opération.

Exemple 5 (Le cas d'une dépendance de donnée) *La figure 4.6 illustre un fragment d'un procédé centralisé dont nous voulons dériver les procédés coopérants. Par rapport à l'exemple précédent, ce fragment a une dépendance de donnée qui relie deux activités qui sont sur le même flot de contrôle illustré par un nuage. Quand l'activité source et l'activité cible sont placées dans les fragments coopérants, il est préférable que cette dépendance de donnée soit implémentée*

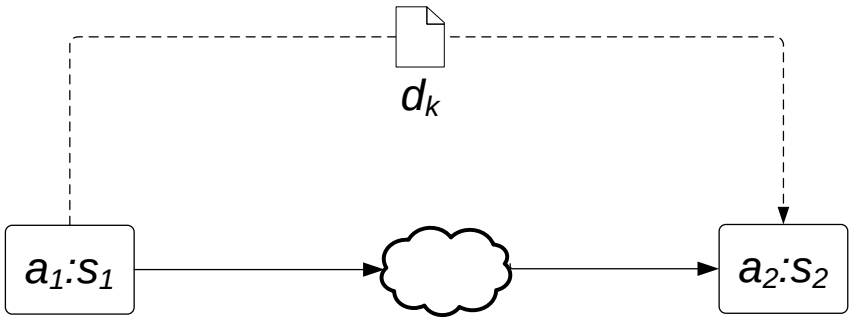


FIG. 4.6 – L'exemple de dépendance de donnée sur un chemin de contrôle

comme une interaction d'égal-à-égal entre les deux procédés fournis. Comme l'activité source et l'activité visée n'ont pas la possibilité d'être suspendues, cette dépendance peut être implémentée en plaçant une activité d'écriture dans le fragment qui contiendra l'activité source et une activité de lecture dans le fragment récepteur. Pourtant une différence importante par rapport au cas précédent qu'il ne faut pas oublier est que le contenu du message échangé est la donnée concernée et pas un message de contrôle.

Exemple 6 (Le cas d'une dépendance de donnée depuis un chemin non-exécutable) *La figure 4.7 illustre un fragment d'un procédé centralisé. Il s'agit de deux activités a_1 et a_2 qui évoquent respectivement deux services s_1 et s_2. Dans une exécution centralisée si l'activité a_1 est exécutée, une de ses sorties est utilisée comme une des entrées de a_2. Si l'activité a_1 n'est pas exécutée, l'entrée de a_2 ne sera pas initialisée avec la sortie de l'activité a_1.*

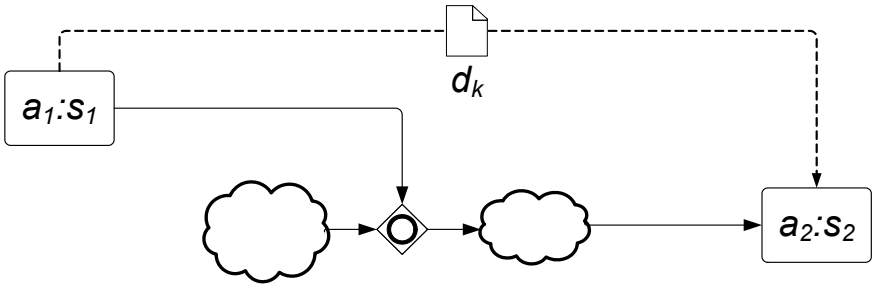


FIG. 4.7 – Un exemple de fragment où deux activités sont reliées par une dépendance de donnée

Quand on dérive deux fragments coopérants pour implémenter cette relation en vue de faire interagir deux services pour l'échange de la donnée d_k, on doit considérer explicitement le fait que l'activité source puisse être suspendue. Parce que si l'activité source est suspendue, elle ne peut pas fournir une donnée à envoyer à a_2 qui sera dans le fragment P_{s_2}. Dans ce cas, l'activité qui doit recevoir la donnée attendue est bloquée jusqu'à ce que cette donnée soit fournie. Comme cette donnée n'est jamais envoyée de P_{s_1} à P_{s_2}, P_{s_2} sera bloqué. Ceci est le résultat de la sémantique des procédés coopérants que nous avons présentés dans les sections précédentes où une activité de lecture reste bloquée tant qu'il n'y a pas de donnée à lire.

L'exemple 6 illustre une situation dans laquelle la démarche intuitive de l'opération qui fournit

les procédés coopérants cause un problème de blocage. En conséquence, l'opération doit être reformulée pour traiter ce cas.

Exemple 7 (Le cas d'une dépendance de donnée vers un chemin non-exécutable) *La figure 4.8 illustre un fragment d'un procédé centralisé. Nous nous intéressons à une dépendance entre deux activités a_1 et a_2 qui évoquent respectivement s_1 et s_2. Contrairement à l'exemple précédent, l'activité cible de la dépendance de donnée d_k peut être suspendue dans une exécution centralisée. Étant donné que les exécutions décentralisées et centralisées sont équivalentes, l'activité visée qui sera placée dans P_{s_2}. La suspension de l'activité a_2 ne cause pas une situation de blocage pour P_{s_1} comme les activités d'écriture ne sont pas bloquantes. Cela veut dire que P_{s_1} ne sera pas bloqué après avoir envoyé la donnée de sortie à P_{s_2}.*

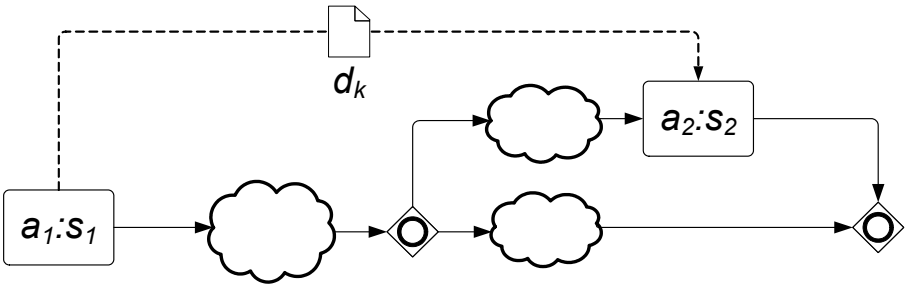


FIG. 4.8 – Un exemple de fragment où deux activités sont reliées par une dépendance de donnée

Pourtant si la donnée n'est pas reçue par un procédé, P_{s_1} ne peut pas terminer correctement puisqu'il va avoir une donnée non-consommée dans un de ses canaux de communication.

Encore une fois, l'opération intuitive présentée pour l'implémentation décentralisée d'une dépendance de contrôle ou de donnée ne satisfait pas les critères que les procédés coopérants doivent vérifier.

Comment implémenter l'élimination des chemins morts ?

Après la présentation des exemples qui détaillent les cas spécifiques que l'opération de décentralisation doit gérer, nous détaillons notre approche. Cette dernière est basée sur l'extension du processus DPE (Dead Path Elimination) classique. Dans un premier temps, cette extension traite la spécification centralisée à décentralisée puis les résultats de ce traitement sont utilisés pour l'implémentation de l'opération elle-même.

Intuitivement, nous pouvons expliquer cette approche en utilisant l'exemple de motivation du chapitre présent (cf. la figure 5.3). Le procédé illustré inclut des points OR-split depuis lesquels un DPE qui prend un ou plusieurs des chemins sortants peut être lancé. Un DPE lancé est la raison de la suspension des activités qui sont sur le chemins pris par le DPE. Ainsi, les activités qui sont visées par les dépendances de donnée venant des chemins pris sont influencées indirectement.

Exemple 8 (L'exemple de motivation de l'approche) *La figure 4.9 illustre un procédé. Nous supposons qu'au point OR-split qui suit l'activité a_6, DPE est lancé. Nous supposons que DPE prend le chemin qui inclut a_8 et a_9. Il est à noter que ce chemin se termine au point OR-join qui précède l'activité a_{17}. Les activités influencées par ce DPE sont essentiellement a_8 et*

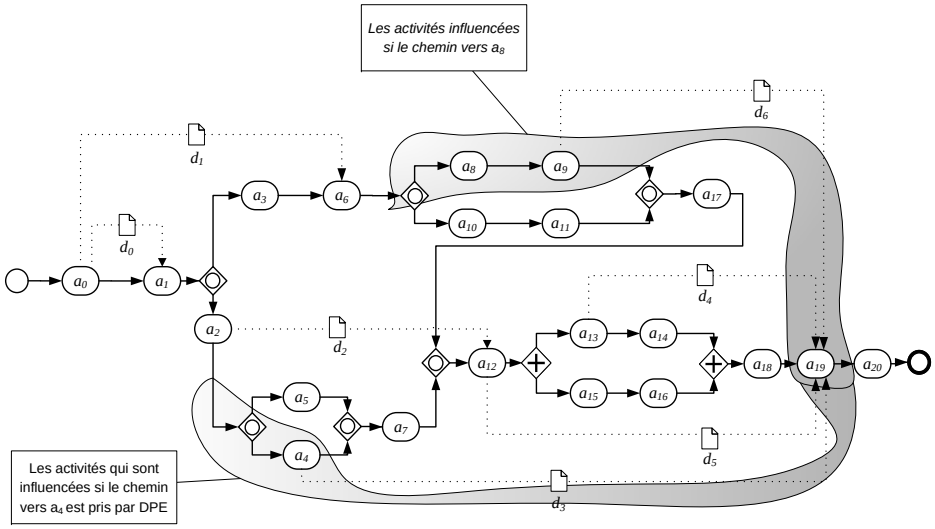


FIG. 4.9 – Les exemples d'activités influencées par le DPE

a_9. *Mais* a_9 *fournit une sortie que* a_{19} *doit utiliser comme une de ses entrées. En conséquence, cette activité est aussi influencée de ce DPE.*

Si on prend le point OR-split qui suit l'activité a_2*, un DPE qui prend le chemin de l'activité* a_4 *ou* a_5 *peut être lancé. Si l'activité* a_4 *est prise par le DPE, l'activité* a_{19} *sera influencée puisqu'elle attend une des entrées de* a_4*.*

L'exemple ci-dessus montre les conséquences de DPE quand on veut dériver les fragments coopérants d'un procédé centralisé. En vue de généraliser les dépendances que nous avons présenté, nous utilisons deux définitions qui étendent les définitions mentionnées dans les section précédentes. La première définition est le *postset étendu*. Un postset étendu est associé aux activités à partir des quelles un DPE peut être lancé. Le postset étendu contient des activités qui peuvent être influencées par le lancement du DPE. Comme nous l'avons expliqué, ce ne sont pas seulement les activités qui sont sur le chemin de contrôle pris par le DPE mais aussi les activités qui ont des dépendances de données rentrantes venant de ces derniers. Plus formellement, nous pouvons définir le postset étendu comme ci-dessous.

Définition 10 ***(Postset étendu)***
Le *postset étendue*, $\overline{a_i\bullet}$, d'une activité a_i est composé par trois sous-ensembles d'activités notés $\overline{a_i\bullet} = \overline{a_i^{F}\bullet} \cup \overline{a_i^{D,F}\bullet} \cup \overline{a_i^{D,T}\bullet}$ où

1. $\overline{a_i^{F}\bullet}$ est l'ensemble des activités qui sont sur les chemins pris par le DPE lancé après l'exécution de l'activité a_i. Plus formellement, $\overline{a_i^{F}\bullet} = \bigcup_{(a_j \in a_i\bullet) \wedge a_i \xrightarrow{c} a_j} a_i^{F,a_j}\bullet$ où
   $a_i^{F,a_j}\bullet = \{a_z \in \mathcal{A} | (a_j < a_z \wedge a_z < \widehat{a_i})\}$

2. $\overline{a_i^{D,F}\bullet}$ est l'ensemble des activités qui ont des dépendances de donnée avec des activités qui sont sur les chemins pris par le DPE lancé suite à l'exécution de l'activité a_i. Plus formelle,ent, $\overline{a_i^{D,F}\bullet} = \bigcup_{a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j} a_i^{D,F,a_j}\bullet$ où
   $a_i^{D,F,a_j}\bullet = \{a_k \in \mathcal{A} | \exists a_l \in \mathcal{A}, a_j < a_l \wedge a_l < \widehat{a_i}, a_l \xrightarrow{d} a_z\}$
3. $\overline{a_i^{D,T}\bullet}$ est l'ensemble des activités qui ont des dépendances de données avec des activités qui ne sont pas prises par le DPE lancé suite à l'exécution de l'activité a_i. Le chemin de a_i ne contient pas d'autre points de OR-split jusqu'à l'activité en question. Plus formellement, $\overline{a_i^{D,T}\bullet} = \bigcup_{(a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j)} a_i^{D,T,a_j}\bullet$ où
   $a_i^{D,T,a_j}\bullet = \{a_k \in \mathcal{A} | \exists a_l \in \mathcal{A}, \nexists a_m \in \mathcal{A}, (a_j < a_l \wedge a_l < \widehat{a_i}), a_l \xrightarrow{d} a_k,$ a_m est une activité OR-split, $(a_j < a_m \wedge a_m < a_k)$ $\}$

Exemple 9 (Postset étendu) *La figure 5.3 illustre un exemple de procédé que nous avons présenté en détail dans la sous-section 4.2.2. Prenons l'activité* a_1 *et identifions son postset étendu. L'activité* a_1 *est une activité de OR-split où le DPE prend le chemin qui commence avec l'activité* a_2 *ou celui qui commence avec l'activité* a_3*. Le postset de l'activité* a_1, $a_1\bullet$ *inclut deux activités liées à* a_1 *par le flux de contrôle. Plus précisément,* $a_1\bullet = \{a_2, a_3\}$*. Le postset étendu,* $\overline{a_i\bullet}$*, de l'activité* a_i *est composé par trois sous-ensembles notés* $\overline{a_i\bullet} = \overline{a_i^{F}\bullet} \cup \overline{a_i^{D,F}\bullet} \cup \overline{a_i^{D,T}\bullet}$*. Le premier sous-ensemble* $\overline{a_1^{F}\bullet}$ *est composé comme suit.* $\overline{a_1^{F}\bullet} = a_1^{F,a_2}\bullet \cup a_1^{F,a_3}\bullet$ *comme ces deux activités sont dépendantes par rapport au flux de contrôle.* $a_1^{F,a_2}\bullet$ *inclut les activités qui doivent être suspendue si le chemin vers* a_2 *est pris par DPE jusqu'au point de OR-join correspondant à* a_{12}*. Ainsi,* $a_1^{F,a_2}\bullet = \{a_4, a_5, a_7\}$ *et* $a_1^{F,a_3}\bullet = \{a_6, a_8, a_9, a_{10}, a_{11}, a_{17}\}$ *(voir. la figure 4.10). Si les activité de* $\overline{a_1^{F}\bullet}$ *sont prises par DPE et ne sont pas exécutées alors les autres activités qui doivent utiliser les sorties de ces activités doivent être informées. Ainsi,* $\overline{a_1^{D,F}\bullet}$ *inclut deux sous-ensembles* $a_1^{D,F,a_2}\bullet$ *et* $a_1^{D,F,a_3}\bullet$*.* $a_1^{D,F,a_2}\bullet$ *inclut les activité qui seront influencées si le chemin de l'activité* a_2 *est pris par DPE lancé suite à l'exécution de* a_1*.* $a_1^{D,F,a_2}\bullet = \{a_{12}, a_{17}\}$ *(voir la figure 4.11) parce que,* a_2 *et* a_{12}*, et* a_4 *et* a_{17} *ont des dépendances de données. De même,* $a_1^{D,F,a_3}\bullet = \{a_{19}\}$*. Le dernier sous-ensemble de* $\overline{a_1\bullet}$ *est* $\overline{a_1^{D,T}\bullet}$ *composé lui-même de deux sous-ensembles* $a_1^{D,T,a_2}\bullet$ *et* $a_1^{D,T,a_3}\bullet$*.* $a_1^{D,T,a_2}\bullet$ *inclut les activités qui doivent être informées si le chemin de l'activité* a_2 *n'est pas pris par DPE.* $a_1^{D,T,a_2}\bullet = \{a_{12}\}$*. Il est à noter que l'activité* a_4 *peut être prise par DPE lancé suite à l'exécution de l'activité* a_2*. Pour cette raison, elle n'est pas dans* $a_1^{D,T,a_2}\bullet$*.* $a_1^{D,T,a_3}\bullet = \emptyset$*.*

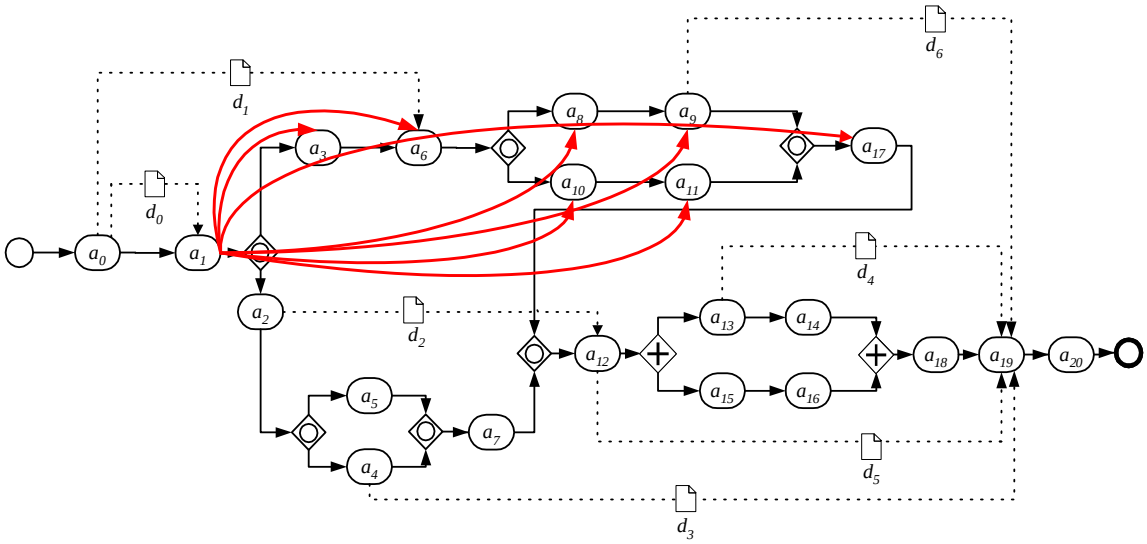


FIG. 4.10 – Les dépendances qui appartiennent à $a_1^{F,a_3}\bullet$

Les éléments de postset étendu sont regroupés en fonction des dépendances sous-jacentes de donnée ou de contrôle. Ceci facilite leur traitement dans les étapes suivantes.

Le postset étendu est le raisonnement sur les dépendances du point de vue de l'activité qui lance le DPE. Nous devons éventuellement modéliser les dépendances liées à l'extension de DPE de la part des activités qui sont visées par les dépendances étendues. Pour ce faire, nous proposons d'associer un *préset étendu* à chaque activité qui est susceptible d'être visé par une dépendance étendue venant d'une activité qui précède un point OR-split. Plus formellement, nous définissons un preset étendu comme ci-dessous.

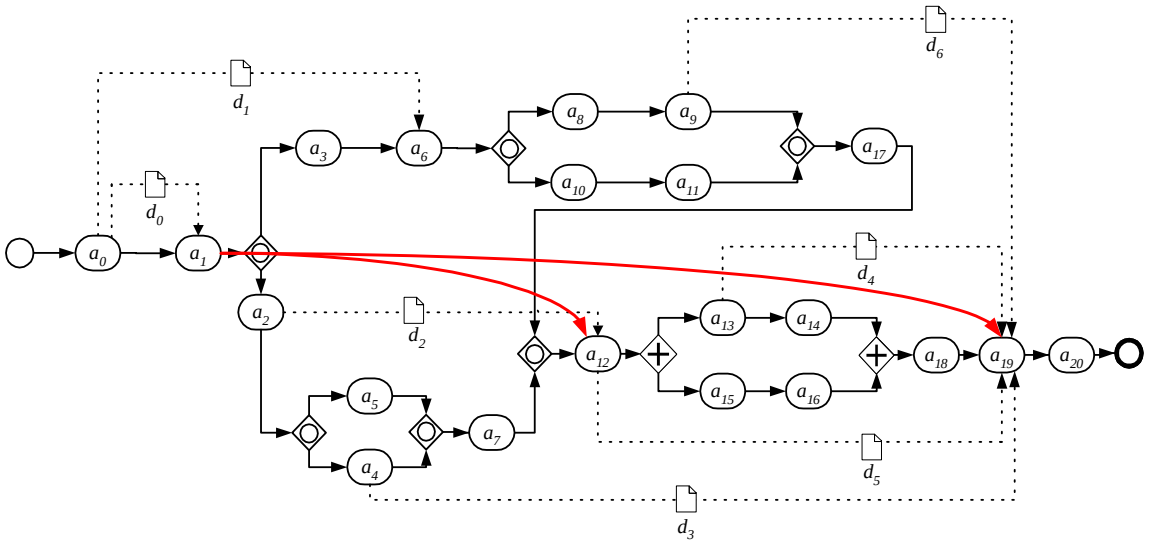


FIG. 4.11 – Les dépendances qui appartiennent à $a_1^{D,F,a_2}\bullet$

Définition 11 ***(Preset étendu)***
Le *preset étendue*, $\overline{\bullet a_i}$, d'une activité a_i est composé par trois sous-ensembles d'activités notés $\overline{\bullet a_i} = \bullet a_i^F \cup \overline{\bullet a_i^{D,F}} \cup \overline{\bullet a_i^{D,T}}$ où

1. $\bullet a_i^F$ est l'ensemble des activités à partir desquelles DPE peut être lancé et prend le chemin de l'activité a_i. Plus formellement, $\bullet a_i^F = \{a_k \in \mathcal{A} |$ a_z est une activité de OR-split, $a_z < a_i \wedge a_i < \widehat{a_z}\}$
2. $\overline{\bullet a_i^{D,F}}$ est l'ensemble des activités à partir de desquelles DPE peut être lancés vers les activités sources des dépendances de données entrantes avec a_i. Plus formellement, $\overline{\bullet a_i^{D,F}} = \bigcup_{(a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i)} \bullet a_i^{D,F,a_j}$ où
   $\bullet a_i^{F,a_j} = \bullet a_j^F$
3. $\overline{\bullet a_i^{D,T}}$ est l'ensemble des activités OR-split qui précédent les activités sources des dépendances de données entrantes de a_i. $\overline{\bullet a_i^{D,T}} = \bigcup_{(a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i)} \bullet a_i^{D,T,a_j}$ où
   $\bullet a_i^{D,T,a_j} = \{a_k \in \mathcal{A} | / \exists a_l \in \mathcal{A}, a_k, a_l$ sont des activités OR-split, $(a_k < a_j \wedge a_j < \widehat{a_k}) \wedge (a_k < a_l \wedge a_l < a_j \wedge a_j < \widehat{a_l})\}$

Exemple 10 (Preset étendu) *La figure 5.3 illustre un procédé. Prenons l'activité a_8 pour identifier son préset étendu. Le preset étendue, $\overline{\bullet a_8}$, de l'activité a_8 est composé par trois sous-ensembles. Formellement, $\overline{\bullet a_8} = \bullet a_8^F \cup \overline{\bullet a_8^{D,F}} \cup \overline{\bullet a_8^{D,T}}$. L'activité a_8 peut être prise par deux DPE différents lancés respectivement à partir des activités a_1 et a_6. Ainsi, $\bullet a_8^F = \{a_1, a_6\}$ (voir la figure 4.12). Comme l'activité a_8 n'a pas de dépendance de donnée venant d'une autre activité*

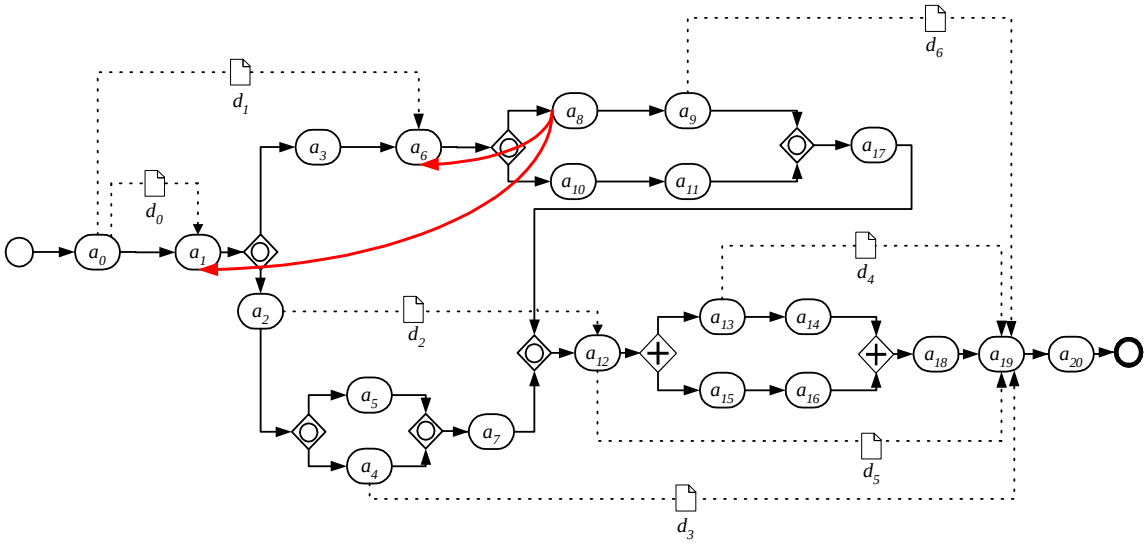


FIG. 4.12 – Les dépendances qui appartiennent $\bullet a_8^F$

$\overline{a_8^{D,F}}$ et $\overline{a_8^{D,T}}$ sont des ensembles vides. Considérons l'activité a_{19} qui a quatre dépendances de donnée rentrantes et une seule dépendance de contrôle rentrante. Le preset de a_{19} est $\bullet a_{19} = \{a_4, a_9, a_{12}, a_{13}, a_{18}\}$. De même, le preset étendu de l'activité a_{19} est composé de trois sous-ensembles $\overline{\bullet a_{19}} = \bullet a_{19}^F \cup \overline{\bullet a_{19}^{D,T}} \cup \overline{\bullet a_{19}^{D,F}}$. L'activité a_{19} n'est pas sur un chemin qui peut être pris par DPE. Pour cette raison $\bullet a_{19}^F$ est un ensemble vide. Pourtant, les activités dont a_{19} a des dépendances de donnée rentrantes peuvent être prises par DPE. $\overline{\bullet a_{19}^{D,F}}$ (voir la figure 4.13) est composé de quatre sous-ensembles. $\overline{\bullet a_{19}^{D,F}} = \bullet a_{19}^{D,F,a_4} \cup \bullet a_{19}^{D,F,a_9} \cup \bullet a_{19}^{D,F,a_{12}} \cup \bullet a_{19}^{D,F,a_{13}}$. Entre eux, $\bullet a_{19}^{D,F,a_{12}}$ et $\bullet a_{19}^{D,F,a_{13}}$ sont des ensembles vides comme a_{12} et a_{13} ne sont pas sur les chemins pris par DPE. $\bullet a_{19}^{D,F,a_4}$ inclut les activités à partir de la quelle DPE qui prend le chemin de l'activité a_4 peut être lancé. $\bullet a_{19}^{D,F,a_4} = \{a_1, a_2\}$ et $\bullet a_{19}^{D,F,a_9} = \{a_1, a_6\}$. $\overline{\bullet a_{19}^{D,T}} = \bullet a_{19}^{D,T,a_4} \cup \bullet a_{19}^{D,T,a_9}$. $\bullet a_{19}^{D,T,a_4}$ est un singleton qui inclut la dernière activité OR-split à partir de la quelle DPE qui prend a_4 peut être lancé. $\bullet a_{19}^{D,T,a_4} = \{a_2\}$. Pareillement, $\bullet a_{19}^{D,T,a_9} = \{a_6\}$.

4.4 Les algorithmes pour la production des fragments coopérants

Dans les sections précédentes, nous avons présenté notre approche en général avec les structures de données qui concrétisent les principes de l'approche. Dans cette section, nous présentons les algorithmes détaillés qui traitent ces structures de données en vue de produire les procédés coopérants pour une exécution décentralisée. Nous commençons sur les expliquer par des exemples intuitifs avant de donner leur spécification formelle.

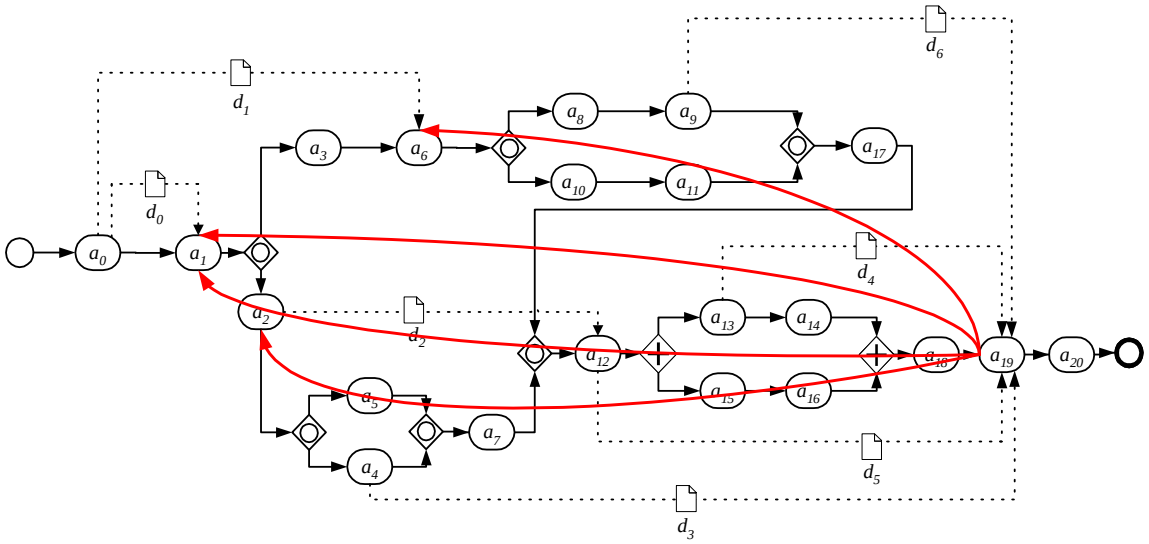


FIG. 4.13 – Les dépendances qui appartiennent à $\bullet\overline{a_{19}^{D,F}}$

4.4.1 Vue d'ensemble et présentation intuitive des algorithmes

Comme nous avons mentionné ci-dessus, le but essentiel des algorithmes est de systématiser le processus de production des procédés coopérants. Dans un premier temps, on distingue deux opérations essentielles que les algorithmes doivent traiter. La première opération est l'interconnexion des activités originales placées dans les procédés coopérants les unes avec les autres par des activités de lecture et d'écriture. La deuxième opération essentielle est la structuration des activités connectées par rapport à leur nature conversationnelle. Ceci a pour but de respecter l'ordre partiel et en conséquence la sémantique de l'exécution. En faisant ces opérations, nous cherchons à répondre aux question suivantes :

- quelles sont les activités de lecture et d'écriture qui permettent l'interconnexion des activités structurées dans les procédés fournis ?
- quelles sont les activités conversationnelles structurées connectées dans les procédés ?
- quels sont les contenus des messages échangés entre les procédés coopérants ?

La modélisation des procédés tient compte de plusieurs aspects modélisés indépendamment du flux de contrôle. C'est par exemple le cas du flux transactionnel du procédé ou bien de la gestion des droits des accès. Si la dimension transactionnelle est essentielle pour une exécution fiable, nous ne nous intéressons pas à la modélisation et la décentralisation du flux transactionnel. Mais nous expliquons brièvement comment l'approche présentée peut être étendue vers la modélisation des aspects transactionnels.

La figure 4.14 illustre comment les fragments coopérants sont produits à partir des concepts présentés dans la section précédente. Comme nous l'avons déjà précisé, l'approche présentée est intuitive. La figure 4.14(a) présente un fragment de procédé qui inclut deux activités a_1 et a_2 qui évoquent respectivement deux services s_1 et s_2. La partie gauche de la même figure illustre les fragments coopérants qui implémentent la même dépendance de contrôle avec deux procédés exécutés par s_1 et s_2. Ces procédés sont notés P_{s_1} et P_{s_2}. Le message échangé entre les fragments P_{s_1} et P_{s_2} consiste en un message de contrôle qui caractérise la terminaison correcte de a_1 dans P_{s_1} en vue d'activer a_2 dans P_{s_2}. La figure 4.14(b) illustre une dépendance de donnée et son implémentation d'égal-à-égal entre deux fragments de procédés exécutés par les services

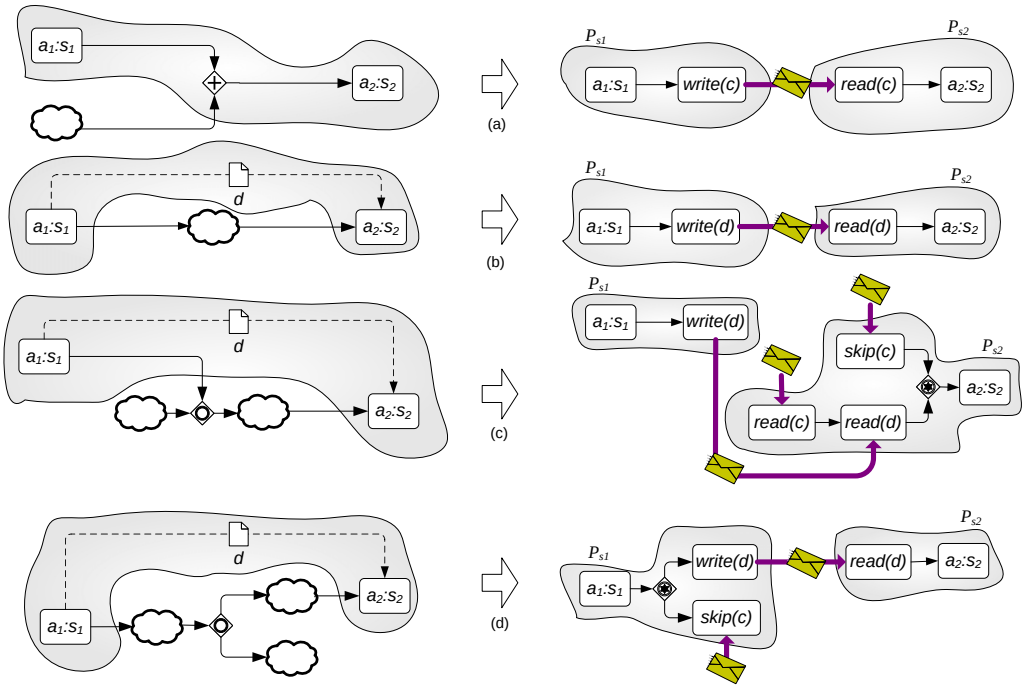


FIG. 4.14 – Les cas principaux qui doivent être considérés en fournissant les fragments de procédés

évoqués. Il est à noter que, le contenu du message échangé est une donnée de procédé et pas un message de contrôle qui active l'activité a_2 dans P_{s_2}. Cet interaction peut être implémentée de la même manière pour chaque dépendance de donnée qui connecte deux activités dans la spécification centralisée. La figure 4.14(c) et la figure 4.14(d) illustrent deux cas plus particuliers qui nécessitent un traitement spécial pour fournir les procédés coopérants. Les situations problématiques de ces deux cas sont présentées précédemment. Ces figures n'illustrent que d'une manière superficielle les procédés dérivés des spécifications centralisées. Dans la figure 4.14(c), le deuxième procédé qui doit recevoir la sortie de l'activité a_1 reçoit l'état d'exécution de cette activité avant d'exécuter l'activité qui recevra sa sortie. Dans ces illustrations, nous ne sommes pas concernés par l'origine des messages reçus qui sont expliqués dans les exemples précédents. Dans la figure 4.14(d), la terminaison appropriée du premier procédé est assurée par un message d'annulation qui permet à ce dernier de vider ses canaux de communications. Il est à noter que ces illustrations ne contiennent pas la totalité des activités qui permettent la communication avec des autres procédés. Dans les sous-sections suivantes, nous présentons la spécification précise pour la structuration des activités d'interconnexion. Le mécanisme de production des procédés, d'un point de vue algorithmique, traite 3 cas principaux. Le premier cas est la structuration des activités qui permettent l'interconnexion d'une activité avec les autres activités suite à son exécution dans le fragment associé. Cette partie est expliquée dans la section *Interconnexion avec les activités de postset et de postset étendue*. Dans cette partie, nous détaillons comment un procédé doit se comporter après l'exécution d'une activité qui appartient à la spécification centralisée et qui est placée dans ce procédé. Le deuxième cas que nous étudions est le comportement d'un procédé avant d'exécuter une activité. Il s'agit de l'*Interconnexion avec les activités de preset et de preset étendue*. Le troisième cas porte sur la structuration des activités connectées au sein

des procédés coopérants pour préserver leur ordre partiel dans la spécification centralisée. Dans les sections suivantes nous donnons une présentation intuitive des algorithmes que nous avons développées pour formaliser ces trois cas. Nous donnons également le résultat des algorithmes correspondantes appliqués à nos exemples de motivation. La sous-section suivante présente les notations que nous avons utilisées pour décrire les algorithmes.

4.4.2 Notations

Dans les notations, les activités de connexion qui envoient et reçoivent les messages sont notées respectivement comme a^w et a^r pour *write* et *read* en anglais. Comme les activités normales, elles évoquent des services. Les messages échangés par l'exécution des activités de connexions sont les messages de contrôle qui peuvent suspendre l'exécution d'une activité ou bien une autre activité de connexion. En outre, ils peuvent échanger des données. Nous proposons le format suivant :

Définition 12 ***(Message)***
Un message est un tuple de la forme $<activity, commande, data, activity\_source>$ où $activity \in \mathcal{A}$, $commande \in \{skip, exec\}$, $data \in \mathcal{D} \cup \emptyset$ et $activity\_source \in \mathcal{A} \cup \emptyset$.

Dans ce format de message, un message de type $< a_i, exec >$ ou $< a_i, skip >$ veut dire que l'activité a_i doit être respectivement exécutée ou suspendue. Si l'activité a_i est précédée par plusieurs activités dans la spécification centralisée, tous les messages correspondant de $< a_i, exec >$ doivent être reçus. Mais si l'activité est sur un chemin qui pourrait être pris par un DPE, il suffit qu'un seul message de $< a_i, skip >$ soit reçu. Il est à noter que dans ce type de message le troisième et le quatrième éléments sont *null* et n'ont pas d'effet. Si le message échangé est une donnée de procédé le message a pour contenu $< a_i, d, a_j >$ ce qui veut dire que le message contient la donnée d que a_i va utiliser et que a_j a déjà fournie. Si le message vise la suspension d'une activité qui est censée recevoir un message de contrôle, le message a le contenu suivant $< a_i, skip, d, a_j >$. Le dernier veut dire que l'activité qui doit recevoir la donnée d que a_i doit utiliser et que a_j a fournie, doit être suspendu parce que l'activité a_j n'est pas exécutée. Il est à noter que l'identité du service qui émet le message n'est pas inclut dans le message. Celle-ci est dans la description de l'activité qui va recevoir le message émis. Si on considère les analyses que nous avons faites dans le chapitre précédent, le contenu des messages que les services composés vont échanger lors de l'exécution des procédés est relativement simple et ne nécessite pas d'un traitement spécial qu'on ne peut pas implémenter dans la logique d'un procédé.

Les activités qui permettent la connexion des activités avec les autres activités dépendantes sont ajoutées dans les procédés coopératifs de manière itérative comme par exemple pour chaque élément d'un ensemble de preset ou d'un preset étendu. Nous utilisons les opérations d'ajout aux fragments qui composent les procédés coopérants fournis. Ces opérations sont notées $\overset{||}{\leftarrow}$, $\overset{\oplus}{\leftarrow}$ et $\overset{+}{\leftarrow}$. Elles caractérisent la structuration de leurs opérandes dans le fragment fourni. Par exemple, le premier opérateur consiste à ajouter ses opérandes dans un fragment AND-split/AND-join. Un fragment qui précède une activité a_i est caractérisé par $\bullet a_i$ et un fragment qui succède une activité a_i est caractérisé par $\widetilde{a_i\bullet}$. Si on considère les opérations qui ajoutent des activités de connexions dans ces fragments, on peut donner comme exemples $\widetilde{a_i\bullet} \overset{||}{\leftarrow} a_k^w$ et $\widetilde{a_i\bullet} \overset{||}{\leftarrow} a_l^w$. La dernière veut dire que les activités $a_k^w + a_l^w$ sont structurées d'une manière concurrente dans le

fragment $\widetilde{a_i\bullet}$. L'exemple suivant $\widetilde{a_i\bullet} \overset{||}{\leftarrow} a_k^w \oplus a_l^w$ et $\widetilde{a_i\bullet} \overset{||}{\leftarrow} a_m^w \oplus a_n^w$ décrit l'addition concurrente de quatre activités dans $\widetilde{a_i\bullet}$ qui sont exclusive entre elles.

4.4.3 Interconnexion avec les activités de postset et de postset étendue

Après avoir exécuté une de ses activités, le comportement d'un procédé coopérant est naturellement d'informer les procédés qui contiennent les activités qui suivent cette activité dans la spécification centralisée. Les sorties de l'activité doivent être transférées vers les activités qui doivent les utiliser. Si l'activité exécutée est une activité de type OR-split, le processus de l'élimination des chemins morts doit être lancé après l'évaluation des conditions de transfert. En conséquence, on peut résumer les activités d'interconnexion d'un procédé coopérant qui doivent suivre une activité a_i appartenant à la spécification centralisée :

- Les activités qui envoient les messages de contrôle aux activités qui la suivent par les dépendances de contrôle,
- Les activités qui envoient les messages de contrôle aux activités qui sont sur les chemins qui seront éliminés en fonction de l'évaluation des conditions de transition,
- Les activités qui envoient les messages de données qui contiennent les sorties de l'activité exécutée,

L'algorithme 1 détaille la structuration de ces activités dans le procédé après une activité a_i de la spécification centralisée.

L'algorithme 1 traite les dépendances en différent groupes pour structurer les activités de connexion. Informellement nous résumons comme suit. Comme nous l'avons souligné, la sortie de cet algorithme est le fragment qui doit être exécuté après a_i. Ce dernier est nommé $\widetilde{a_i\bullet}$. Le fragment $\widetilde{a_i\bullet}$ est composé de plusieurs fragments. Le fragment $\widetilde{a_i^{T,a_j}}$ contient les activités de connexions qui envoient des messages de contrôles pour l'exécution des activités suivantes dépendantes par rapport au flux de contrôle. Le fragment $\widetilde{a_i^{T,a_j}\bullet}$ inclut les activités qui viennent après l'activité a_j. Comme les messages de contrôle d'exécution sont envoyés seulement aux activités du postset dépendantes par rapport au flux de contrôle, le fragment $\widetilde{a_i^{T,a_j}\bullet}$ contient une activité de connexion qui envoie le message de contrôle au procédé de l'activité a_j. Si l'activité pour laquelle l'algorithme 1 est exécuté, la condition de transition vers ses activités de postset peut devenir $fausse$. Les activités de connexion qui envoient des messages de contrôle de suspension concernent toutes les activités qui sont prises par le DPE. Le fragment $\widetilde{a_i^{F,a_j}\bullet}$ inclut les activités de connexion qui envoient des messages de contrôle de suspension qui suivent l'activité a_j jusqu'au point de jointure. Il est clair que les fragments $\widetilde{a_i^{T,a_j}\bullet}$ et $\widetilde{a_i^{F,a_j}\bullet}$ sont structurés exclusivement. Les fragments $\widetilde{a_i^{D,F,a_j}\bullet}$ et $\widetilde{a_i^{D,T,a_j}\bullet}$, qui sont ajoutés respectivement aux $\widetilde{a_i^{F,a_j}\bullet}$ et $\widetilde{a_i^{T,a_j}\bullet}$ respectivement, considèrent les activités cibles des dépendances de données dont les activités sources sont prises par le DPE vers l'activité a_j. Les activités de connexion de $\widetilde{a_i^{F,a_j}\bullet}$ envoient des messages de suspension aux procédés des activités cibles de dépendances. De même, les activités de $\widetilde{a_i^{T,a_j}\bullet}$ envoient des messages d'exécution aux procédés des activités sources. Le dernier fragment ajouté à $\widetilde{a_i\bullet}$ est le fragment qui contient les activités de connexion qui envoient les sorties de l'activité a_i. Ce fragment est nommé $\widetilde{a_i^D\bullet}$.

Pour expliquer la structuration de l'interconnexion avec les éléments du postset nous utilisons l'exemple suivant.

Input : La spécification centralisée de la composition
Output: $\forall a_i \in \mathcal{A}$, $\widetilde{a_i\bullet}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j$ **do**
      $s_j = service(a_j)$;
      $\widetilde{a_i^{T,a_j}\bullet} \leftarrow a^w(s_j, < a_j, exec >)$
    end
    forall $a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j$ **do**
      forall $a_k \in a_i^{F,a_j}\bullet$ *où* $a_i^{F,a_j}\bullet \subseteq \overline{a_i^F\bullet}$ **do**
        $s_k = service(a_k)$;
        $\widetilde{a_i^{F,a_j}\bullet} \overset{||}{\leftarrow} a^w(s_k, < a_j, skip >)$
      end
      forall $a_k \in a_i^{D,F,a_j}\bullet$ *où* $a_i^{D,F,a_j}\bullet \subset \overline{a_i^{D,F}\bullet}$ **do**
        $s_k = service(a_k)$;
        $\widetilde{a_i^{D,F,a_j}\bullet} \overset{||}{\leftarrow} a^w(s_k, < a_i, skip, d, a_j >)$
      end
      forall $a_k \in a_i^{D,T,a_j}\bullet$ *où* $a_i^{D,T,a_j}\bullet \subset \overline{a_i^{D,T}\bullet}$ **do**
        $s_k = service(a_k)$;
        $\widetilde{a_i^{D,T,a_j}\bullet} \overset{||}{\leftarrow} a^w(s_k, < a_i, exec, d, a_j >)$
      end
    end
    forall $a_j \in a_i\bullet \wedge a_i \xrightarrow{d} a_j$ **do**
      $s_j = service(a_j)$;
      $\widetilde{a_i^D} \overset{||}{\leftarrow} a^w(s_j, < a_j, d, a_i >)$
    end
    forall $a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j$ **do**
      $\widetilde{a_i\bullet} \overset{||}{\leftarrow} \widetilde{a_i^{T,a_j}\bullet} \oplus \widetilde{a_i^{F,a_j}\bullet}$
    end
    $\widetilde{a_i\bullet} \leftarrow \widetilde{a_i^D\bullet}$
  end
end

Algorithm 1: Interconnexion avec les activités de postset et de postset étendue

Exemple 11 (Interconnexion avec les activités de postset et de postset étendue) *Nous considérons l'exemple de motivation illustré dans la figure 5.3. Nous prenons l'activité* a_1 *afin de la structurer dans un procédé. Dans son postset* $a_1\bullet$ *il y a deux activités,* $a_1\bullet = \{a_2, a_3\}$*. Dans son postset étendue,* $\overline{a_1^F\bullet}$ *on trouve* $a_1^{F,a_2}\bullet$ *et* $a_1^{F,a_3}\bullet$*. Ces ensembles sont respectivement* $a_1^{F,a_2}\bullet = \{a_4, a_5, a_7\}$ *et* $a_1^{F,a_3}\bullet = \{a_6, a_8, a_9, a_{10}, a_{11}, a_{17}\}$*. Dans* $\overline{a_1^{D,F}\bullet}$*, on trouve* $a_1^{D,F,a_2}\bullet$ *et* $a_1^{D,F,a_3}\bullet$*. Les éléments de ces ensembles sont respectivement* $a_1^{D,F,a_2}\bullet = \{a_{12}, a_{19}\}$ *et* $a_1^{D,F,a_3}\bullet = \{a_{19}\}$*. Concernant l'ensemble* $a_1^{D,T}\bullet$*, on en trouve un seul élément* $a_1^{D,T,a_2} = \{a_{12}\}$*. Par rapport à ces*

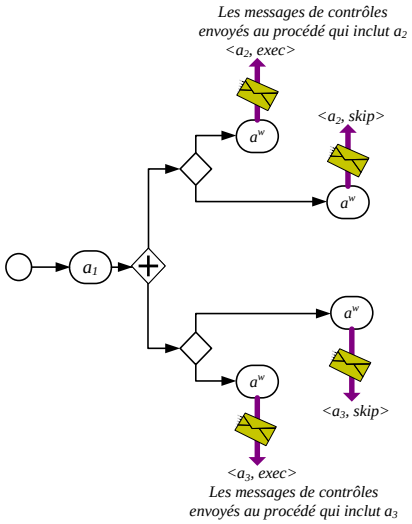


FIG. 4.15 – La structuration des activités de connexions concernant les messages de contrôles à envoyer vers les procédés qui exécutent a_2 et a_3

éléments, le fragment qui doit être exécuté après a_1 peut être fourni comme suit : Les conditions de transitions vers a_2 et/ou a_3 peuvent devenir vrai ou faux. Donc, $\widetilde{a_1\bullet}$ inclut dans un premier temps les messages de contrôle qui doivent être envoyés aux procédés qui exécutent a_2 et a_3. Les activités qui envoient les messages de contrôle sont structurées d'une manière concurrente mais exclusivement entre elles.

La figure 4.15 illustre les activités de connexion avec le contenu des messages qu'elles envoient vers les procédés qui incluent les activités concernées. Les activités qui envoient les messages de contrôles skip aux activités sont suivies par les activités qui envoient des messages de skip aux activités qui sont sur les chemins pris par l'élimination des chemins morts. En conséquence, les activités de connection qui envoient des messages de skip aux procédés qui incluent $a_6, a_8, a_9, a_{10}, a_{11}, a_{17}$ sont structurées d'une manière concurrente après l'activité de connexion qui envoie le message skip à a_3. De même, les activités de connexions qui envoient les messages de skip aux procédés qui incluent a_4, a_5, a_7 sont structurées après l'activité de connexion qui envoie le message skip au procédé qui inclut a_2. Ces parties du fragment sont illustrées dans la figure 4.16. Ces activités qui permettent l'interaction avec les procédés qui incluent les activités qui sont sur les chemins pris par l'élimination des chemins morts, sont suivies par les activités de connexions qui propagent l'état des mêmes activités vers les procédés qui doivent recevoir leurs sorties. La figure 4.17 illustre la structuration de ces activités après les activités de connexions. Celles-ci sont notamment les éléments de $a_1^{D,F,a_2}\bullet$ et $a_1^{D,F,a_3}\bullet$. La figure 4.18 illustre le dernière élément des interactions concernant les messages de contrôle.

La dernière activité de connexion qu'on peut ajouter au fragment concernant $\widetilde{a_1\bullet}$, est notamment l'exécution de l'activité a_2 comme a_1 est la première activité qui succède à a_2 et peut éliminer le chemin qui la prend. Le fragment $\widetilde{a_1\bullet}$ finit par une activité de terminaison qui caractérise la fin des activités de connexion associées avec a_1. Cette activité de terminaison est une activité intermédiaire que nous utilisons pour connecter le fragment produit aux autres fragments qui correspondent aux activités succesives.

L'exemple ci-dessus présente les activités qui doivent être exécutées pour connecter une activité de la spécification centralisée avec les activités qui seront dépendantes après son exécution.

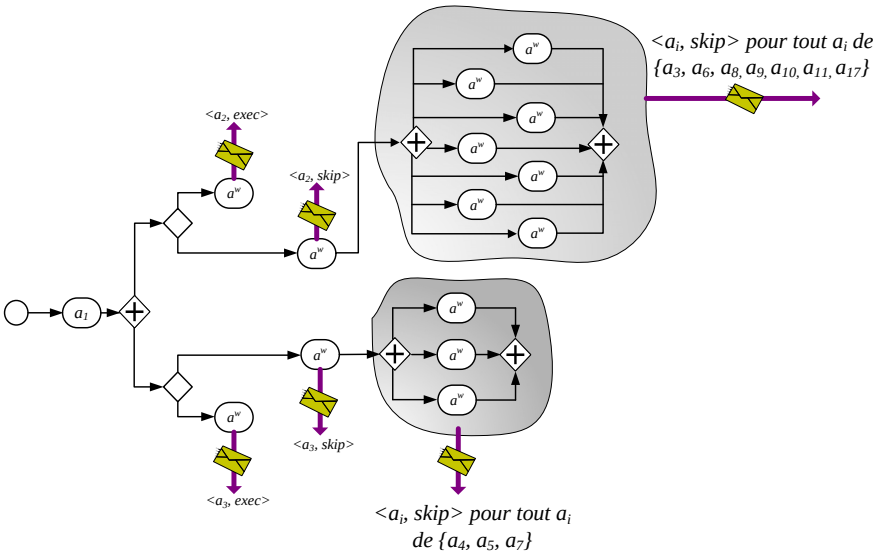


FIG. 4.16 – La structuration des activités de connexions concernant les messages de contrôles pour satisfaire l'élimination des chemins morts

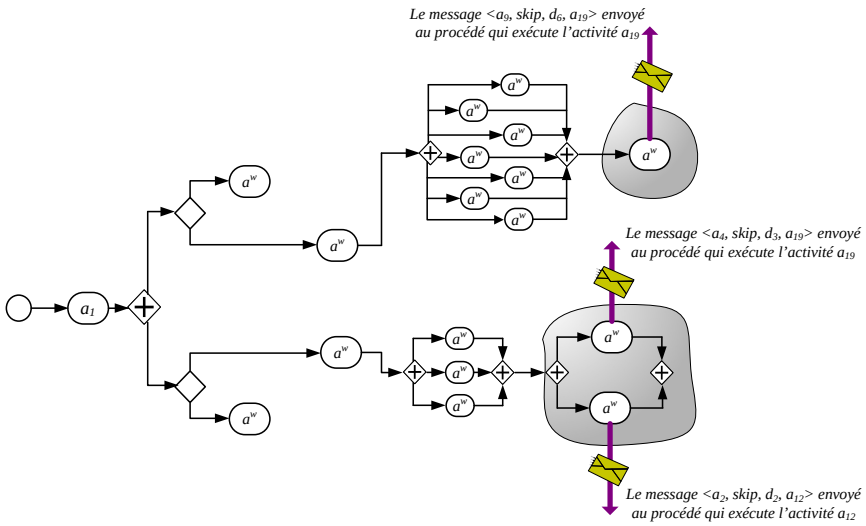


FIG. 4.17 – La structuration des activités de connexions concernant les messages de contrôles pour informer les procédés qui sont susceptibles d'exécuter des activités de connexions pour recevoir des données envoyées par les autres procédés

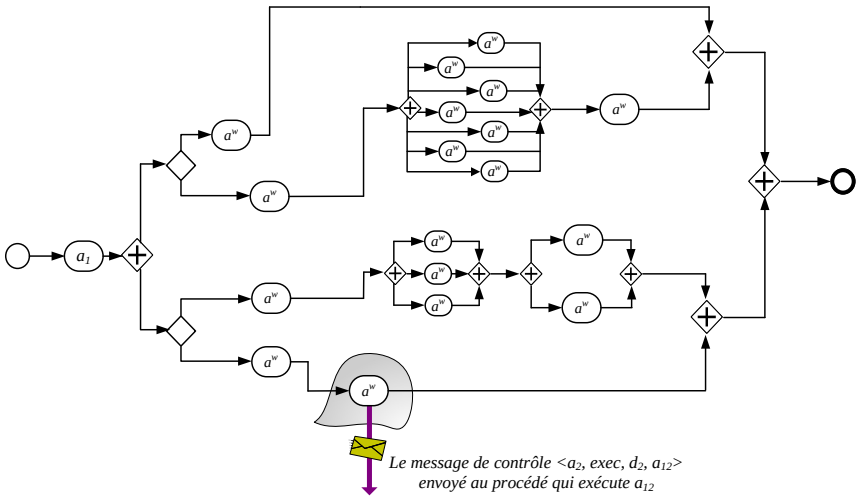


FIG. 4.18 – La structuration des activités de connexions concernant les messages de contrôles pour informer les procédés qui incluent les activités qui doivent utiliser les sorties des activités que l'élimination des chemins morts n'influence pas

L'exemple illustre une exécution de l'algorithme 1 pour l'activité a_1. Dans la section suivant, nous présentons l'algorithme qui permet la connexion d'une activité avec celles qui la précèdent dans la spécification centralisée.

4.4.4 Interconnexion avec les activités de preset et de preset étendue

Avant d'exécuter une activité a_i, un procédé doit être informé a propos de la terminaison des activités qui la précèdent dans la spécification centralisée. Les entrées de l'activité à exécuter doivent être reçues également des activités qui les fournissent. Il est à noter qu'une activité peut être suspendue si elle est prise par un DPE. L'élimination d'un chemin mort nécessite la suspension de l'activité a_i et ignore la réception des messages qui fournissent ses entrées. L'algorithme 2 détaille la structuration des activités de connexion qui permettent la réception des messages de contrôle et de données concernant l'exécution des activités.

Informellement nous pouvons expliquer l'algorithme 2 comme suit. La sortie de l'algorithme est le fragment $\widetilde{\bullet a_i}$ qui précède l'exécution de l'activité a_i dans son procédé coopérant. Comme nous l'avons fait pour la structuration des activités de connexion de postset, ce fragment est composé d'autres fragments. Le fragment $\widetilde{\bullet a_i^T}$ inclut les activités de connexion qui reçoivent des messages de contrôle des procédés qui exécutent les activités de preset de l'activité a_i. Elles sont structurées d'une manière concurrente, car la réception de tous les messages de contrôle est nécessaire pour l'exécution de a_i. Si l'activité est sur un chemin qui peut être pris par un DPE elle peut être suspendue par un message venant d'un procédé qui peut lancer le DPE. Le fragment $\widetilde{\bullet a_i^F}$ est composé des activités de connexion qui reçoivent les messages de suspension des procédés qui exécutent les activités à partir desquelles un DPE qui prend a_i peut être lancé. Il est à noter que les éléments de $\widetilde{\bullet a_i^F}$ sont structurés exclusivement puisq'une activité ne peut pas être prise pas plusieurs DPE. Si l'activité est exécutée, parce qu'elle n'est pas prise par un DPE et toutes les activités qui la précèdent sont exécutées, ses entrées de l'activité ont été reçues. Cette dernière considération est valable seulement pour les activités sources de dépendances de

Input : La spécification centralisée de la composition
Output: $\forall a_i \in \mathcal{A}$, $\widetilde{\bullet a_i}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_j \in \bullet a_i \wedge a_j \xrightarrow{c} a_i$ **do**
      $s_j = service(a_j)$;
      $\widetilde{\bullet a_i^T} \overset{||}{\leftarrow} a^r(s_j, < a_i, exec >)$
    end
    forall $a_j \in \bullet a_i^F$ **do**
      $s_j = service(a_j)$;
      $\widetilde{\bullet a_i^F} \overset{\oplus}{\leftarrow} a^r(s_j, < a_i, skip >)$
    end
    $\widetilde{\bullet a_i} \leftarrow \widetilde{\bullet a_i^T} \oplus \widetilde{\bullet a_i^F}$
    forall $a_k \in \bullet a_i^{D,F,a_j}$ *où* $a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i$ **do**
      $s_k = service(a_k)$;
      $s_j = service(a_j)$;
      $\widetilde{\bullet a_i^{D,T,a_j}} \leftarrow a^r(s_k, < a_i, exec, d, a_j >) + a^r(s_j, < a_i, d, a_j >)$
      forall $a_k \in \bullet a_j^F$ **do**
        $s_k = service(a_k)$;
        $\widetilde{\bullet a_i^{D,F,a_j}} \overset{\oplus}{\leftarrow} a^r(s_k, < a_i, skip, d, a_j >)$
      end
      $\widetilde{\bullet a_i^D} \overset{||}{\leftarrow} \widetilde{\bullet a_i^{D,T,a_j}} \oplus \widetilde{\bullet a_i^{D,F,a_j}}$
    end
    forall $a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i$ **do**
      $s_j = service(a_j)$;
      $\widetilde{\bullet a_i^{D,T}} \overset{||}{\leftarrow} a^r(s_j, < a_i, d, a_j >)$
    end
    $\widetilde{\bullet a_i} \leftarrow \widetilde{\bullet a_i^D}$ $\widetilde{\bullet a_i} \leftarrow \widetilde{\bullet a_i^{D,T}}$
  end
end

Algorithm 2: Interconnexion avec les activités de preset et de preset étendue

données entrantes qui sont exécutées. Pour ce faire, l'algorithme produit un fragment, nommé $\widetilde{\bullet a_i^{D,F}}$, qui inclut les fragments exclusifs pour chaque activité source qui est susceptibles de ne pas être exécutée. Ces éléments sont nommés $\widetilde{\bullet a_i^{D,T,a_j}}$ et $\widetilde{\bullet a_i^{D,T,a_j}}$ respectivement. Ces fragments sont produits pour chaque activité source qui fait partie du preset étendu, et, qui est susceptible d'être suspendue. Le fragment $\widetilde{\bullet a_i^{D,T,a_j}}$ contient deux activités de connexion séquentielles pour recevoir respectivement le message de contrôle d'exécution de l'activité source et la donnée échangée. Le fragment $\widetilde{\bullet a_i^{D,F,a_j}}$ contient des activités de connexion qui reçoivent les messages de suspension exclusivement pour la même activité source. Le dernier fragment à exécuter consiste en activités de connexions qui reçoivent les données venant des procédés qui exécutent les activités sources

des dépendances de données entrantes. Ce fragment est nommé $\widetilde{\bullet a_i^{D,T}}$. La différence entre les activités de ce procédé et les deux autres est que les activités de connexions de $\widetilde{\bullet a_i^{D,T}}$ reçoivent des données venant de procédés qui exécutent les activités sources des dépendances de données entrantes qui ne sont pas sur un chemin qui peut être pris par le DPE.

Nous utilisons l'exemple suivant pour décrire des exécutions de l'algorithme pour des activités différentes.

Exemple 12 (Interconnexion avec les activités de preset et de preset étendu) *Nous considérons l'exemple de motivation illustré dans la figure 5.3. Nous utilisons deux exemples. D'abord nous prenons l'activité a_9 pour détailler sa structuration avec les activités de connexions dans un fragment. Les ensembles de preset et postset sont respectivement $\bullet a_9 = \{a_8\}$ et $a_9\bullet = \{a_{17}, a_{19}\}$. L'activité a_9 n'a pas de dépendances de données entrantes. Par conséquent, $\overline{\bullet a_9^{D,F}} = \emptyset$ et $\overline{\bullet a_9^{D,T}} = \emptyset$. Le chemin de contrôle de l'activité a_9 peut être pris par deux DPE différents lancés soit à partir de l'activité a_1 ou a_6. Ces activités sont les éléments de l'ensemble $\bullet a_9^F = \{a_1, a_6\}$. Ainsi, un message de contrôle $< a_9, skip >$ venant du fragment de l'activité a_1 ou a_6 peut suspendre l'exécution de a_9 dans son procédé. Pour que l'activité puisse être exécutée dans le procédé qui l'inclut, il faut que le procédé qui inclut a_8 envoie un message $< a_9, exec >$ si aucun DPE qui la prend n'est lancé. En conséquence, ces trois messages peuvent être reçus exclusivement. La figure 4.19 illustre la structuration des activités de connexion qui précèdent a_9 dans son procédé.*

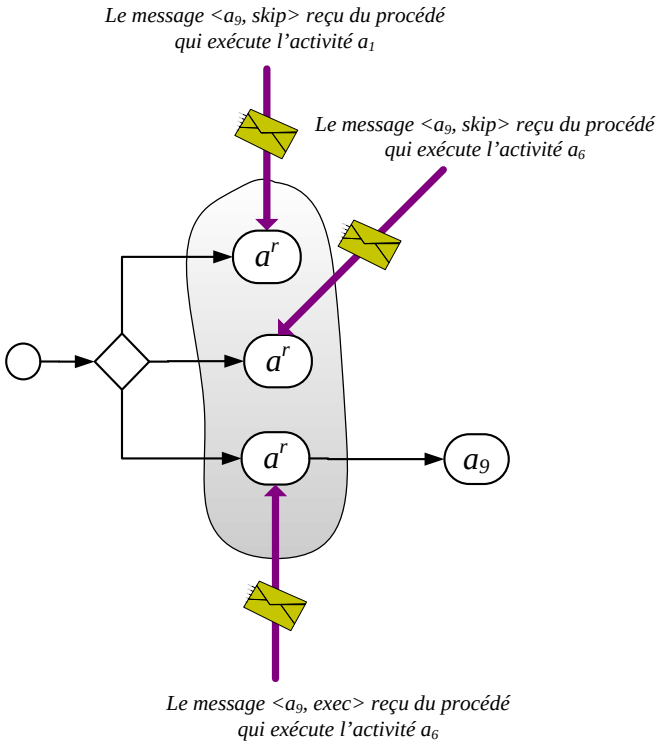


FIG. 4.19 – La structuration des activités de connexions concernant la réception des messages de contrôles pour la suspension ou l'activation de l'activité a_9

Comme l'exécution de l'activité a_9 ne requiert pas la réception d'une sortie venant d'autres

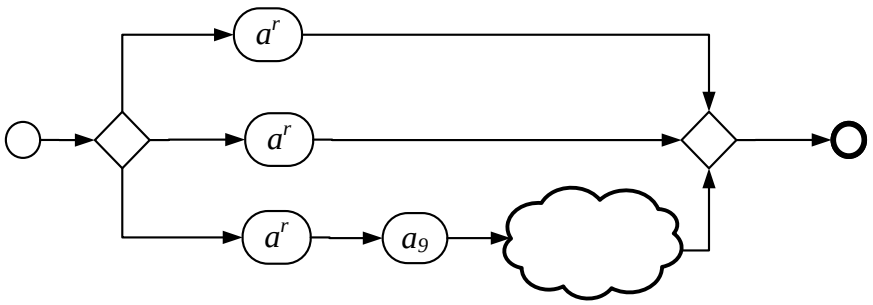


FIG. 4.20 – La structuration des activités de la figure4.19 par rapport à l'activé terminale du fragment $\widetilde{a_9\bullet}$

procédés, la réception du message de contrôle venant du procédé de l'activité a_8 est suivie par l'exécution de l'activité a_9 elle-même. Pourtant, la réception des messages de suspension ne sont pas suivis par l'exécution de l'activité a_9. Les activités de connexion de ces messages sont suivies par l'activité terminale du postset de a_8. Ainsi, la suspension de l'activité est suivie par le début des autres activités conversationnelles qui appartiennent au même procédé. Il est à noter que la structuration des activités de connexion qui reçoivent les messages de suspension est exclusive.

Le deuxième exemple d'activité est l'activité a_{19} qui a plusieurs dépendances entrantes. Le preset de l'activité a_{19} est $\bullet a_{19} = \{a_4, a_9, a_{12}, a_{13}, a_{18}\}$. Le fragment qui correspond à l'exécution des activités de connexions de l'activité a_{19} est $\widetilde{\bullet a_{19}}$. Ce fragment commence avec la réception du message de contrôle venant du procédé qui inclut l'activité a_8. Concernant les éléments de son preset étendu, l'activité a_{19} n'est pas sur un chemin qui peut être pris par un DPE. En conséquence, l'ensemble $\bullet a_{19}^F = \emptyset$. Les activités sources des dépendances de données entrantes peuvent être prises par plusieurs DPE. Les activités à partir desquelles un DPE peut être lancé sont les sous-ensembles des ensembles $\overline{\bullet a_{19}^{D,F}}$ et $\overline{\bullet a_{19}^{D,T}}$. Les activités a_{12} et a_{13} ne sont pas sur les chemins qui peuvent être pris par un DPE. Les ensembles $\bullet a_{19}^{D,F,12}$ et $\bullet a_{19}^{D,F,13}$ sont vides. Pour cela, le procédé qui exécute a_{19} ne doit pas tester l'état d'exécution de ces activités avant d'exécuter les activités de connexions qui reçoivent leur sorties. Au contraire, les activités a_4 et a_9 de preset qui fournissent des sorties qui seront utilisées comme les entrées de a_{19} doivent être considérées. L'activité a_4 peut être suspendue par deux DPE différents qui peuvent être lancés par deux différents points qui sont a_1 et a_2 respectivement. Ainsi, $\bullet a_{19}^{D,F,a_4} = \{a_1, a_2\}$. De même, le chemin de l'activité a_9 peut être pris par un DPE lancé par les procédés qui inclut les activités a_1 et a_6. Ainsi, $\bullet a_{19}^{D,F,a_9} = \{a_1, a_6\}$. Les activités de connexion qui correspondent à la réception des messages de suspension venant des procédés qui incluent ces activités sont structurées exclusivement. La figure 4.21 illustre la structuration des activités de connexion qui reçoivent les messages de contrôle pour tester l'état d'exécution des activités sources des dépendances de données entrantes. En outre, cette figure illustre le message de contrôle venant du procédé de l'activité a_8.

Les dernières activités de OR-split qui précèdent les activités sources des dépendances de données, et, à partie desquelles un DPE qui prend l'activité source n'est pas lancé, sont groupées dans $\overline{\bullet a_{19}^{D,T}}$. Ce dernier est composé de deux ensembles $\bullet a_{19}^{D,T,a_4}$ et $\bullet a_{19}^{D,T,a_9}$ où $\bullet a_{19}^{D,T,a_4} = \{a_2\}$ et $\bullet a_{19}^{D,T,a_9} = \{a_6\}$. Si les procédés de ces activités informent le procédé de l'activité a_{19} à propos de l'exécution des activités sources, les activités de connexions qui reçoivent les données de procédé peuvent être exécutées. Il est à noter que les activités de connexions qui reçoivent les sorties

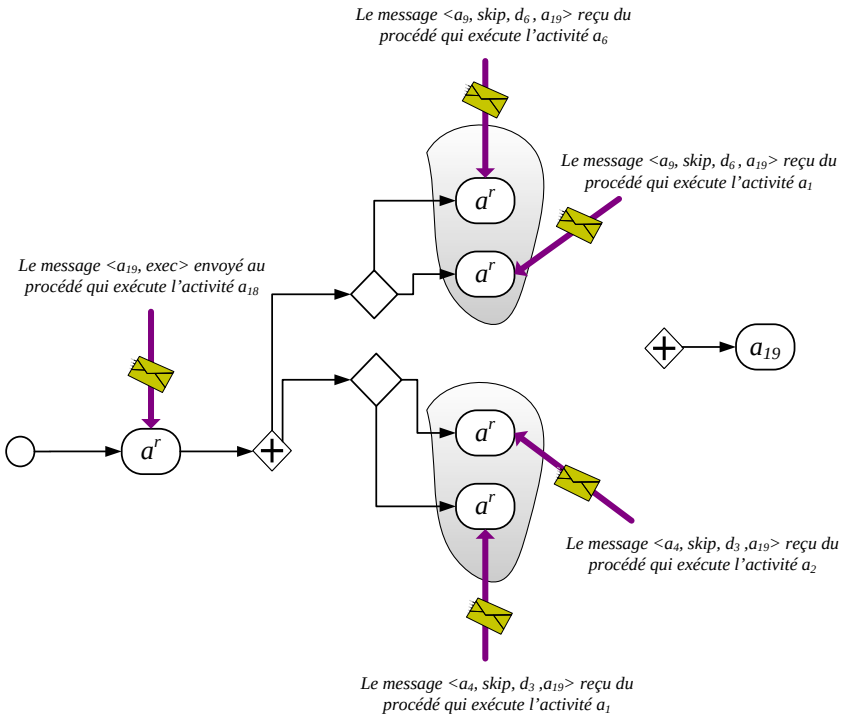


FIG. 4.21 – La structuration des activités de connexion qui reçoivent les messages de suspensions des activités sources des dépendances de données de l'activité a_{19}

des activités a_{12} et a_{13} ne nécessitent pas d'un test de l'état d'exécution des activités sources comme celles-ci ne sont pas sur les chemins qui peuvent être pris par le DPE. L'activité a_{19} a une seule activité qui la précède avec une dépendance de contrôle. Pour que l'activité a_{19} puisse être exécutée dans son procédé, le procédé de l'activité a_{18} doit envoyer un message de contrôle pour l'informer de la terminaison de l'activité a_{18}. La figure 4.22 illustre la structuration de l'activité de connexion qui concerne la réception du message de contrôle venant du procédé de l'activité a_{18}.

Implémentation de la terminaison correcte

Les algorithmes que nous avons présentés concernent les 3 cas principaux illustrés dans la figure 1.14. Nous avons essentiellement cherché à empêcher les situations de blocage des procédés qui attendent des messages de données de la part d'activités suspendues. Le quatrième cas consiste à implémenter la terminaison appropriée d'un procédé coopérant. Le critère de "global soundness", spécifie qu'un procédé ne doit pas contenir de données dans ses canaux de communication quand son activité finale est exécutée. Ceci est appelé la terminaison correcte. Pour implémenter ce principe, chaque procédé doit recevoir un ou des messages en vue de vider les données restantes des canaux de communication. Par rapport aux situations de blocages qui empêchent l'exécution décentralisée de la composition, les données stockées dans les channels de communication sont moins graves. Une désavantage de telles données qui restent dans les channels est certainement l'augmentation de l'espace consommé pour l'instance du procédé. Intuitivement, il est souhaitable que l'espace consommé ne soit pas important par rapport à la quantité totale d'espace disponible. Nous considérons deux stratégies pour gérer les données qui

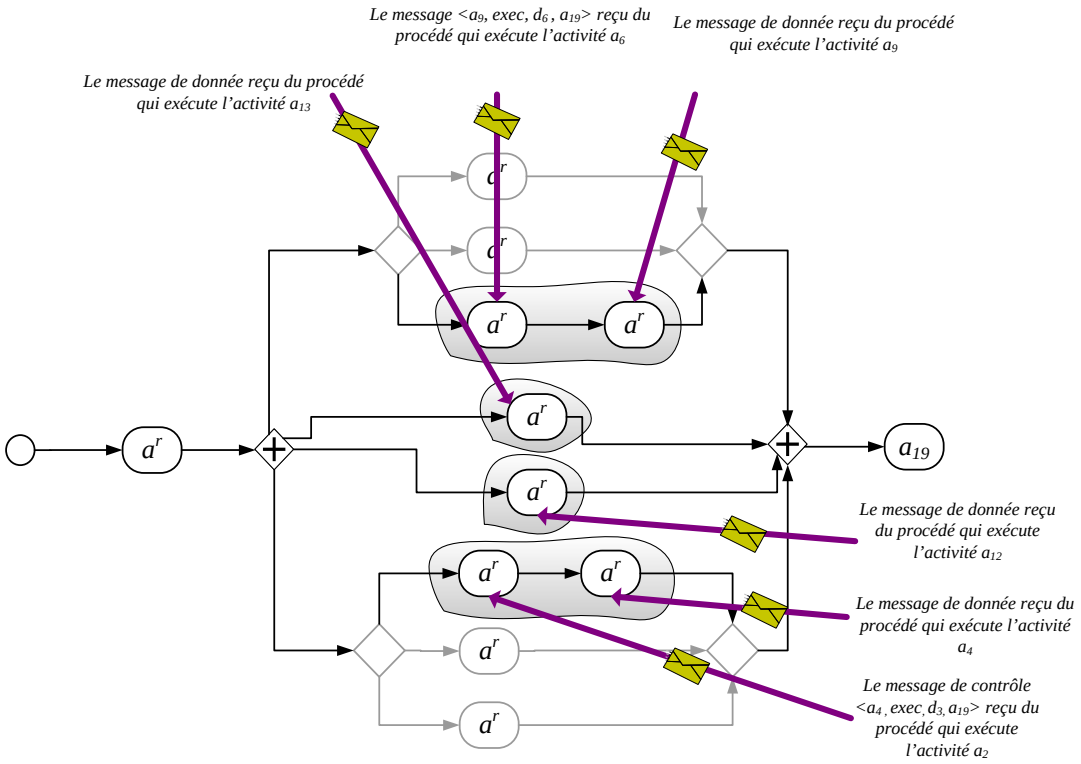


FIG. 4.22 – La structuration des activités de connexion qui reçoivent les messages d'exécution pour les activités sources des dépendances de données de l'activité a_{19} et les activités de connexion qui reçoivent les données de procédés des activités a_{12} et a_{13}

ne seront pas consommées :

1. L'annulation (*vidange des canaux*) des données envoyées et non consommées pendant l'exécution du procédé dynamiquement au moment de la suspension de l'activité source,
2. L'annulation des données envoyées et non consommées à la fin de l'exécution globale explicitement par un message de terminaison,

Nous avons supposé que la lecture à partir des canaux de communication est arbitraire. Cela signifie que la présence d'une donnée dans un canal n'empêche pas la lecture d'une autre donnée qui est ajoutée plus tard. La première méthode de gestion des données restant dans les canaux est certainement efficace mais n'empêche pas l'exécution correcte globalement. La deuxième méthode est moins efficace d'un point de vue de performance comme les données sont dans les canaux jusqu'à la fin de l'instance. Cette dernière nécessite un procédé, et aussi un service qui l'exécute, pour envoyer les messages de terminaison à tous les procédés coopérants. Si on considère des restrictions du flux d'information, les interactions pour la terminaison des procédés doivent être cohérentes avec ces restrictions. Une deuxième implémentation possible de la terminaison explicite peut être indépendante d'un procédé tiers. L'implémentation de la première méthodologie de terminaison est relativement compliquée. Elle consiste à spécifier un procédé coopérant avec des activités de connexions qui envoient des messages de contrôles aux procédés coopérants concernés pour annuler les données qui ne seront pas consommées. Plus précisément, il faut définir une autre type de dépendance qui caractérisera l'annulation d'une donnée après l'exécution d'une autre activité qui sera la source de cette nouvelle dépendance. Celle-ci peut être une activité qui est exécutée après la suspension de l'activité cible de la dépendance de donnée.

La modalité que nous avons implémentée est la deuxième. Les raisons de ce choix sont multiples. Notre but ultime était de fournir un méthodologie pour les services puissent établir des interactions P2P, et ceci, avec une cohérence dans laquelle la sémantique est préservée. L'utilisation performante des canaux de communication ne fait pas la problématique de notre travail. Ceci est valable aussi pour la proposition des méthodologies pour la gestion efficace de l'espace disponible.

En conséquence, chaque procédé coopérant a une activité de type *read* qui est placée avant l'activité terminale. Le procédé coopérant qui exécute la dernière activité de la spécification centralisée a une activité de connexion de type *write* qui envoie un message de contrôle qui sera reçu par les activités de connexion des autres procédés coopérants. Les instances des procédés coopérants sont enlevées par les services quand ce message de contrôle est reçu. La consommation de mémoire associée aux canaux est proportionnelle aux instances des procédés coopérants actifs.

4.5 Synthèse

Dans cette section, nous résumons les contributions de ce chapitre et notre positionnement par rapport aux travaux existants. Nous résumons également l'intérêt d'une telle contribution.

Au cours de ce chapitre, nous avons proposé une approche générale pour construire les procédés coopérants qui mettent en œuvre une exécution décentralisée d'un procédé workflow centralisé qui spécifie la coordination d'un ensemble de service. Le but d'une telle approche est d'établir des interactions d'égal-à-égal entre les services composés tout en respectant la sémantique de la spécification centralisée. Dans un contexte workflow, l'approche nécessite d'une part une spécification centralisée du procédé et d'autre part un ensemble de propriétés de communication sous-jacentes liés à l'environnement d'exécution des procédés. Étant conscient de ces restrictions, nous avons précisé un ensemble d'hypothèses nécessaires avant d'implémenter les

structures de données et les algorithmes qui permettent la production des procédés coopérants. Essentiellement, nous avons considéré *les procédés structurés* afin de diminuer la complexité de traitement des algorithmes. Ceci est une hypothèse raisonnable qui ne diminue pas seulement la complexité du traitement mais aussi facilite la compréhension des structures de données utilisées. Une autre considération importante a porté sur un ensemble de modalité de communication qui nécessite une configuration dans un système de gestion de procédé. Les modalités que nous avons choisies sont basées sur les principes bien connus des procédés coopérants qui sont actuellement implémentés dans la plupart des systèmes de gestion de procédés. La considération des modalités de communication, et notamment, les critères d'exactitude, permet de faire des vérifications formelles sur les structures de données et les algorithmes proposés.

La modélisation que nous avons adoptée ne nécessite pas une connaissance particulière à part des paradigmes caractéristique de l'approche workflow qu'on peut trouver dans de nombreuses travaux liés et outils existants. Les qualités reconnues des paradigmes sont la simplicité des concepts et la grande lisibilité des résultats, grâce à l'utilisation de graphismes et à large une large documentation. Il faut également souligner la démarche induite par ces méthodes qui permet la distinction de différents niveaux d'abstraction tels que le flux de contrôle et le flux de données. Nous avons choisi un raisonnement indépendant d'une spécification concrète telle que WS-BPEL pour être le plus générique possible. Celle-ci est une différence principale de notre contribution par rapport à d'autres travaux tels que [Baresi *et al.*2007] [Nanda *et al.*2004] [Khalaf and Leymann2006]. Contrairement aux travaux similaires, nous ne nous contentons pas d'une présentation intuitive d'une technique de décentralisation d'une spécification centralisée tels que [Atluri *et al.*2001] [Sadiq *et al.*2006] [Baresi *et al.*2007]. En partant de cas de figures simples, nous avons proposé une méthode générique qui peut traiter tous type de spécifications qui respectent nos hypothèses. Il est à noter que notre proposition vise la décentralisation des procédés qui spécifient la composition de services. La partition des programmes impératifs ou des graphes est assez loin de nos préoccupations même si on peut rencontrer l'utilisation des termes similaires.

Nous traitons ensemble les flux de contrôle, de données et les aspects conversationnels tout en respectant leur relations. Cette approche est extensible aux autres aspects qui ne sont pas étudiés dans le cadre de cette thèse. Un de ces aspects, peut être le plus populaire, est la gestion transactionnelle ou la gestion des exceptions du procédé qui spécifie une composition. Il consiste à spécifier un autre type de flux similaire à celui du contrôle en considérant les dépendances des activités qui doivent être suspendues ou reprises en cas d'échec d'une activité dépendante [Yang and Liu2006] [Curbera *et al.*2003]. Le flux transactionnel spécifie un autre type de dépendance, et naturellement, un autre type de message et une autre méthode de structuration des activités de connexion s'il est décentralisé. Par conséquent, il nous semble que notre approche peut être étendue pour formaliser d'autre cas similaire en suivant la même approche de production de procédés coopérants. Rappelons cependant que notre objectif n'est pas de définir un méta-méthode intégrant toutes les caractéristiques d'un procédé ou d'une composition, mais plutôt de fournir une base qu'on peut adapter facilement à des besoins divers.

Avec notre approche, chaque dépendance est traitée pour être implémentée comme une interaction égal-à-égal entre les services correspondants. Une des critiques que l'on peut lui faire est sans doute sa trop grande nature décentralisée. En effet une exécution complètement décentralisée d'un procédé n'est pas toujours souhaitable. Cet aspect est mentionné dans le chapitre problématique et va être détaillé dans le chapitre suivant sous la forme d'implémentation de politiques de flux d'information.

En vertu des besoins pour la décentralisation qui sont définis précédemment, et, l'acceptation des standards de procédé basés sur XML, l'exécution décentralisée d'une composition peut être

faisable en implémentant des paradigmes de la mobilité (ou de l'échangabilité) [Yildiz and Godart2007f] [Yildiz and Godart2007a] des fragments qui correspondent à l'exécution décentralisée de la spécification centralisée entre les services composés. Les contributions de ce chapitre montrent comment les procédés coopérants, qui constituent la problématique essentielle de cette approche, peuvent être produits et exécutés.

Chapitre 5

Contrôle du flux d'information : *Vers un modèle décentralisé efficace*

Sommaire

5.1 Introduction

Le paradigme de décentralisation est structuré autour de l'établissement des interactions d'égal-à-égal entre les services composés d'un procédé. L'une des tâches essentielles d'un système de décentralisation est de modéliser, de structurer et de contrôler les restrictions qui gouvernent ces interactions. Par nature, ces restrictions sont souvent complexes : complexes par leurs structures (une restriction peut être définie comme étant composée d'autres restrictions statiques ou dynamiques) mais aussi par leur validité (une restriction peut appartenir à un service ou être définie par le concepteur d'une composition à décentraliser). Elles sont également de granularité très variable, allant de la restriction d'une simple interaction à la spécification d'un chemin sophistiqué qu'une donnée doit suivre à travers les services. De plus, elles sont fortement interreliées. Les interactions peuvent être issues des dépendances de la spécification de la composition ainsi que des besoins du contexte de composition.

Dans le chapitre précédent, nous avons détaillé comment les interactions peuvent être établies par rapport à la spécification de la composition sans tenir compte de restrictions particulières.

Dans ce chapitre, nous présentons nos contributions d'adaptation de notre approche pour la prise en compte des restrictions particulières pour le contrôle du flux d'information. Le contrôle du flux d'information, qui est largement détaillé dans les sections suivantes, intègre plusieurs points de vue concernant les restrictions et les différents types de critères présents dans le contexte d'une composition. Les contributions présentées dans le chapitre précédent et celles qui seront présentées dans ce chapitre ne sont pas indépendantes. Elles montrent comment la mobilité d'un procédé peut être employée pour fournir des exécutions décentralisées en permettant de renforcer le volet applicatif d'une composition où les restrictions jouent un rôle important.

Le reste de ce chapitre est organisé de la manière suivante. Dans la prochaine section, nous présentons le contrôle du flux d'information dans sa généralité ; le contrôle du flux d'information dans les systèmes orientés objet et sa relation avec les applications inter-organisationnelles. Par la suite, nous donnons une vue d'ensemble de notre démarche pour contrôler le flux d'information dans les procédés inter-organisationnels selon l'approche présentée dans le chapitre précédent. Cette section est suivie par des considérations plus formelles sur le flux d'information et les algorithmes correspondants y sont présentés. Les résultats obtenus dans cette section sont utilisés pour concevoir des procédés coopérants qui implémentent les restrictions considérées.

5.2 Contrôle du flux d'information

Le contrôle du flux d'information est un aspect incontournable des compositions efficaces où les services composés ont des interactions restreintes par les politiques de sécurité, d'intimité ou de confiance. Dans ce chapitre, nous nous intéressons à la modélisation et au contrôle du flux d'information dans les procédés métiers basés sur la composition des services. Nous présentons une solution qui suit l'approche de gestion décentralisée de procédé comme présentée dans le chapitre précèdent. Notre solution s'appuie sur des techniques de décomposition d'une spécification de composition centralisée en un ensemble de procédés métiers coopérants qui établissent des interactions de services tout en respectant les restrictions du flux d'information.

Le contrôle du flux d'information consiste à spécifier et à configurer les interactions d'un ensemble de composants logiciels en considérant la propagation de l'information dans une composition qui les utilise. Historiquement, le terme *flux d'information* est apparu dans le domaine des langages orientées objets [Myers and Liskov1997]. Dans son contexte original, le contrôle de flux d'information a pour objectif d'analyser les dépendances entre classes concernant la propagation de l'information à travers celles-ci. Même si les contributions dans ce contexte original restent élémentaires par rapport au contrôle du flux d'information dans un contexte inter-organisationnel, elles y sont utiles. Les problèmes liés au contrôle du flux d'information dans une composition de service nécessitent la compréhension et l'intégration d'aspects qui sont souvent étudiés séparément comme le contrôle d'accès [Mecella *et al.*2006], la gestion de la confiance [Malik and Bouguettaya2007] ou la vérification d'une description de chorégraphie [Kazhamiakin and Pistore2006]. Le contrôle d'accès s'intéresse à l'accès à l'information du point de vue du service mais il n'amène aucune restriction à la propagation de l'information dans la composition après un accès. Une description de chorégraphie est efficace pour vérifier la cohérence syntaxique des interactions d'égal-à-égal (P2P) de services mais elle ne fournit aucun support pour la configuration des interactions avancées. Les exemples incluent l'établissement d'interactions P2P pour certaines interactions et exécuter d'autres interactions par un coordinateur centralisé pour la surveillance. Il est clair qu'un environnement de composition peut devenir chaotique face à l'augmentation du nombre du types de restrictions.

Comme nous l'avons souligné dans les chapitres précédents (voir le chapitre 2, section 2.3.1),

les restrictions qui peuvent co-exister dans un contexte de composition sont multiples. Elles sont souvent engendrées par les politiques des organisations qui implémentent les services composés. Ces politiques peuvent être entre autres, les droits d'accès, les relations de confiance, les relations d'intimité. En outre, les différents concepteurs peuvent définir les restrictions pour exprimer leur politiques diversifiées. Par exemple, un concepteur peut souhaiter isoler deux organisations partenaires dans le cadre d'une collaboration inter-organisationnelle, et il est sans doute important de considérer ce type de restrictions. Dans un modèle de composition traditionnel, une implémentation cohérente qui peut satisfaire l'ensemble des politiques n'est pas possible. En considérant les approches principales de composition de services, orchestration et chorégraphie, à ce stade de la réflexion, aucune solution alternative n'émerge. Car dans une composition orientée orchestration, les services sont isolés les uns des autres et il est difficile d'établir des interactions mutuelles entre eux pour satisfaire de telles restrictions. Dans une composition orientée chorégraphie dans laquelle les services établissent des interactions indépendamment d'une entité centralisée, les interactions mutuelles peuvent subir des restrictions comme par exemple l'empêchement d'interaction mutuelle entre deux services particulières. De même, les relations de confiance sont si nombreuses qu'elles ne peuvent pas être gérées par une seule entité. Par exemple, un service A peut être évoqué par un service B même si la sortie de A doit être utilisée par un service C et B peut router la sortie de A vers C qui ne peut pas évoquer A directement. Dans ce cas, les relations de confiance des services doivent être traitées pour tester la validité des *chemins* entre les services. Nous pouvons multiplier les exemples de motivation pour expliquer la propagation de l'information dans une composition. Le flux d'information contrôlé peut être issu des relations entre les services et de leur besoin d'intimité et de sécurité. Par ailleurs, la spécification du flux d'information peut être une chorégraphie sophistiquée pour laquelle le concepteur cherche des services qui la respectent. On peut aussi voir le contrôle du flux d'information d'une composition comme un moyen pour exploiter les restrictions présentes dans un contexte de composition, et qui peuvent être mutuellement contradictoires.

5.3 Vue d'ensemble et présentation intuitive de la démarche adoptée

Dans cette section, nous présentons la vue d'ensemble du contrôle du flux d'information et nous donnons une description intuitive de notre démarche pour l'implémenter. Dans le chapitre précédent, nous avons démontré comment les dépendances de la spécification centralisée sont implémentées sans prendre en compte des restrictions particulières. Nous avons montré comment les aspects liés à la décentralisation du flux de contrôle et du flux de donnée sont traités. Maintenant, nous révisons cette approche pour intégrer des restrictions du flux d'information. Cet objectif nécessite une considération formelle des restrictions considérées. En outre, il est nécessaire d'avoir un système de raisonnement pour les traiter. La nature de ce dernier peut être différent par rapport à la spécification et l'emplacement des restrictions.

Dans la spécification des restrictions des politiques, l'aspect critique que nous cherchons à résoudre concerne la propagation de l'information dans la composition. La propagation de l'information est sans doute un aspect important dans un contexte d'exécution décentralisée dans lequel les interactions des services sont essentielles. Le formalisme que nous avons choisi peut être relié aux concepts de confiance [Malik and Bouguettaya2007] [Jøsang *et al.*2006], de gestion des droits d'accès [Mecella *et al.*2006], de droit d'accès orientés propos [Bertino2004] et de métas-donnée ajoutée à une information principale [Srivastava and Velegrakis2007]. Nous ne nous intéressons pas au contexte dans lequel les restrictions sont définies, nous sommes essen-

tiellement concernés par leur satisfaction dans le cadre de l'approche présentée dans le chapitre précèdent. Les algorithmes que nous avons développés transforment les dépendances de la spécification centralisée en interactions égal-à-égal. En partant de ce principe, on peut considérer que chaque dépendance est potentiellement une interaction entre les services sous-jacents évoqués par l'activité source et l'activité cible de la dépendance. Par rapport aux restrictions venant du contexte de composition, toute interaction doit être en adéquation avec ces dernières. Si une interaction n'est pas en adéquation avec une restriction ; une possibilité est d'annuler la restrictions qui empêche l'interaction. La deuxième possibilité est de trouver un ensemble d'interactions qui satisfait toutes les restrictions en routant l'information à travers les services disponibles. Cette approche générale cache difficilement la difficulté du traitement des restrictions et les problèmes d'implémentation des interactions correspondantes. Parmi les solutions d'implémentation des interactions, la production des procédés coopérants est une nouvelle voie que nous avons exploré. Les questions principales de cette approche sont bien sûr multiples : *Comment les restrictions sont-elles modélisées ? Comment les restrictions sont-elles traitées ? Comment les interactions possibles sont implémentées comme des procédés coopérants ?* La figure 5.1 donne une vue d'ensemble de la méthodologie adoptée.

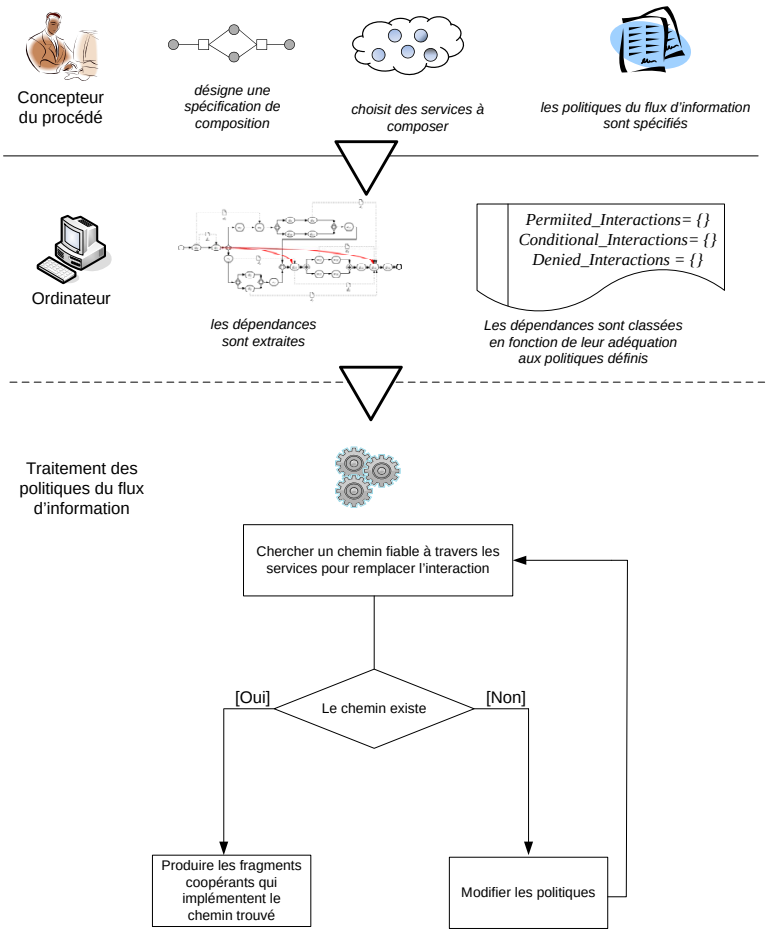


FIG. 5.1 – Vue d'ensemble de la méthodologie du contrôle du flux d'information

Le choix de la méthodologie est en partie intuitif : il est le fruit d'une réflexion en spirale sur la modélisation et l'exécution des procédés. On peut le justifier en considérant les éléments suivants :

- la présence d'un nombre important de restrictions qui doivent être considérées dans un procédé inter-organisationnel et leur diversité,
- la volonté de définir certaines restrictions indépendamment d'une spécification particulière,
- la volonté d'utiliser le même formalisme pour décrire le plus grand nombre d'aspects d'un procédé afin de simplifier la mise en œuvre du modèle,
- la possibilité d'implémenter des interactions avancées d'une manière systématique au moment de la production des procédés coopérants.

Les restrictions du flux d'information sont sensibles au temps et à l'espace. Le lecteur peut juger les restrictions présentées ci-après comme élémentaires par rapport à la définition des restrictions similaires tels que les droits d'accès dans des systèmes de gestion de base de données [Joshi *et al.*2005] [Bertino *et al.*2005] ou dans les systèmes de gestion de workflow [Bertino *et al.*2004] [Mecella *et al.*2006]. Nous attirons à nouveau l'attention sur le fait que les restrictions que nous avons considérées ne sont qu'un sous-ensemble de restrictions de base qui peuvent être considérées pour exprimer des restrictions plus sophistiquées. En outre, le problème important que nous cherchons à résoudre est l'implémentation d'une exécution décentralisée dans le contexte de telles restrictions.

5.4 Représentations formelles

Dans cette section, nous présentons quelques considérations formelles concernant les politiques de flux d'information. Comme nous l'avons déjà souligné, ces restrictions peuvent être issues des divers éléments impliqués dans la composition. Nous considérons trois groupes de restrictions.

1. Les politiques du flux d'information d'un service,
2. Les politiques du flux d'information des concepteurs,
3. Les politiques du flux d'information contextuelles (routage).

Chaque groupe de restrictions est détaillé dans les sous-sections suivantes. Comme nous allons l'argumenter dans la suite de ce chapitre, il n'est pas possible de proposer une spécification complète de la modélisation des restrictions qui peuvent exister dans un contexte inter-organisationnel riche en termes d'aspects de sécurité, de confiance, de contrainte temporelle etc. Nos considérations portent sur la modélisation de restrictions qui peuvent servir comme des constructeurs atomiques pour la modélisation d'un grand nombre de restrictions potentielles.

5.4.1 Politiques du flux d'information d'un service

Dans cette section, nous présentons les politiques du flux d'information qu'un service peut exposer dans une composition dont il en fait partie. La définition et l'implémentation de ces politiques sont similaires aux restrictions de contrôle d'accès. Dans le chapitre précédent nous avons vu que les activités originales d'une spécification centralisée sont structurées dans un procédé coopérant qui est exécuté par le service et que les activités de connexion portent les interactions respectives. Les flux d'information qui régularisent les données entrantes et sortantes d'un service doivent, intuitivement, spécifier les politiques sur ces interactions. Le tableau 5.1 résume les politiques qu'un service peut exposer. Les restrictions de base spécifient les services qui peuvent fournir les entrées d'une opération et recevoir les sorties d'une opération à un

service. De même que nous avons fait une distinction entre les données d'une opération et son invocation, un autre type de restriction spécifie les restrictions concernant l'invocation d'une opération indépendamment de ses données d'entrée et de sortie. L'invocation est faite par les messages de contrôle venant de la part d'autres procédés exécutés par d'autres services. Comme un service peut restreindre les invocations entrantes, il peut restreindre les messages de contrôle emis vers les autres services. Le dernier type de politique de flux d'information concerne les messages de suspension qui constituent un autre type de message de contrôle.

`Prédicat`	`Opérandes`	`Les types des opérandes`	`Exemple`
`can_in`$[d_c]$	3	$\mathcal{D}$, $\mathcal{OP}$, $\mathcal{S}$	si `can_in`(in_i, op_i, s_j)$\wedge[d_c]$ est vraie, l'entrée in_i de l'opération op_i du service s_i peut être fournie au service s_j
`can_out`$[d_c]$	3	$\mathcal{D}$, $\mathcal{OP}$, $\mathcal{S}$	si `can_out`(out_i, op_i, s_j)$\wedge[d_c]$ est vraie, la sortie out_i de l'opération op_i du service s_i peut être fournir au service s_j
`can_act_from`$[d_c]$	2	$\mathcal{OP}$, $\mathcal{S}$	si `can_act_from`(op_i, s_j)$\wedge[d_c]$ est vraie, l'opération op_i du service s_i peut être évoquée par le service s_j
`can_act_to`$[d_c]$	2	$\mathcal{OP}$, $\mathcal{S}$	si `can_act_to`(op_i, s_j)$\wedge[d_c]$ est vraie, le message de contrôle issue de la terminaison de l'opération op_i peut être envoyé au service s_i
`can_abt`$[d_c]$	3	$\mathcal{OP}$, $\mathcal{S}$, $\mathcal{S}$	si `can_abt`(op_j, s_k, s_j)$\wedge[d_c]$ est vraie, l'activité de connexion qui fournit l'entrée de l'opération op_j qui évoque le service s_k peut être suspendue avec le message de contrôle venant du service s_j. Il est à noter que si le service s_k est *null*, l'invocation de l'opération op_j est suspendue selon le message venant du service s_j.

TAB. 5.1 – Les politiques du flux d'information appartenant à un service

Le tableau 5.1 donne une représentation formelle des politiques du flux d'information d'in-

formation sous forme de prédicats. Les politiques sont définies au niveau de l'identité du service, de l'identité de l'opération et de l'identité de la donnée. Mais elles peuvent être facilement étendues à un groupement. Par exemple, une politique définie pour un service peut être exprimée de telle sorte qu'elle soit valable pour un groupe de services. Ce type de vision est le même pour les données d'un procédé ou un type d'opération. Comme cette considération ne change pas le raisonnement dans ses grandes lignes, nous préférons limiter notre travail aux les identités des entités décrites dans les tableaux de politiques (voir les tableaux 5.1, 5.2 5.3).

Exemple 13 (Politique du flux d'information d'un service) *Cconsidérons l'exemple présenté dans le chapitre 2. Si nous voulons spécifier les politiques considérés, nous pouvons utiliser les prédicats suivants. Admettons que l'opération de l'*Inspecteur *qui est invoquée dans le cadre de cette composition soit nommée* op_{ins}*. Cette opération a deux données d'entrée qui sont nommées :* in_{ins1} *et* in_{ins2}*. Ce service spécifie que ces données peuvent être fournies de la part du service* Hôpital *et du service* Police*. Avec le formalisme que nous avons développé, ces restrictions sont spécifiées comme :* **can_in**(in_{ins1}, op_{ins}, Hôpital), **can_in**(in_{ins2}, op_{ins}, Hôpital), **can_in**(in_{ins1}, op_{ins}, Police), **can_in**(in_{ins2}, op_{ins}, Police)*. Nous pouvons également imaginer la définition d'attributs conditionnels pour ces restrictions. Un attribut est la valeur de la donnée échangée. Admettons que le rapport venant du service* Hôpital *puisse être accepté s'il appartient au client "Dupont". Ainsi cette condition peut être spécifiée par un attribut "ins2.owner==Dupont" dans un prédicat comme* **can_in**(in_{ins1}, op_{ins}, Hôpital)*[ins1.owner==Dupont].*

5.4.2 Politiques du flux d'information d'un concepteur

Le concepteur d'un procédé qui compose un ensemble de services peut définir la spécification de plusieurs manières. Les spécifications de composition que nous utilisons pour dériver les procédés coopérants expriment les dépendances de contrôle et de données des appels aux services. Un grand nombre de travaux récents portent sur la définition automatique de ces spécifications en traitant les dépendances des services atomiques [Berardi *et al.*2005] [Betin-Can *et al.*2005] [Fu *et al.*2005] [Fu *et al.*2005] [Bultan *et al.*2006]. Sans rentrer dans les détails de ces travaux, on peut les résumer comme suit. Analyser les dépendances de composants autonomes consiste à calculer des combinaisons par des analyses formelles de *model cheking* [Marconi *et al.*2006] ou de *theorem prover* [Rao *et al.*2006]. Comme dans les systèmes de vérifications traditionnels, l'idée de base est de définir une *configuration but* et d'essayer de calculer les interactions des composants de telle sorte que la configuration but soit atteinte. Cette dernière peut prendre la forme d'une planification comme dans les systèmes de raisonnements d'intelligence artificielle [Pistore *et al.*2005] ou de calcul de chemins dans un graphe [Dahl *et al.*]. Il est à noter que des analyses formelles similaires peuvent être utilisées pour vérifier les caractéristiques secondaires des spécifications de composition telle que la cohérence transactionnelle [Rusinkiewicz and Georgakopoulos1994]. Un concepteur peut également définir des politiques du flux d'information. Les motivations sont nombreuses. Une motivation essentielle est le contexte organisationnel dans lequel un concepteur peut vouloir isoler des services les uns des autres ou bien les faire interagir par rapport à un ensemble de restrictions. Le tableau 5.2 résume les politiques qu'un concepteur peut implémenter dans une composition. Comme nous l'avons mentionné ci-dessus, la plus simple de ces restrictions est celle qui spécifie la possibilité d'interaction entre deux services. Par la suite, les autres restrictions spécifient les interactions qui concernent l'invocation d'une opération et les données entrantes et sortantes d'une opération. Ces restrictions sont similaires aux restrictions définies par un service et peuvent être éventuellement contradictoires avec ces dernières. Une restriction porte sur l'implication d'un tiers service dans une interaction entre deux services. Un

concepteur peut vouloir s'informer ou bien informer un autre service à propos de l'interaction de deux services. C'est souvent le cas des compositions orientées chorégraphie dans lesquelles les services interagissent les uns avec les autres d'une manière décentralisée.

Prédicat	Opérandes	Les types des opérandes	Exemple
`can_see`	2	$\mathcal{S}$, $\mathcal{S}$	si `can_see`(s_i, s_j) est vraie, les services s_i et s_j peuvent avoir des interactions directes
`must_in`$[d_c]$	4	$\mathcal{OP}$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{S}$	si `must_in`$(op_i, in_i, s_i, s_j) \wedge [d_c]$ est vraie, l'entrée in_i de l'opération op_i du service s_i doit être fournie par le service s_j
`must_out`$[d_c]$	4	$\mathcal{OP}$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{S}$	si `must_out`$(op_i, out_i, s_i, s_j) \wedge [d_c]$ est vraie, la sortie out_i de l'opération op_i du service s_i doit être fournie au service s_j
`must_act_from`$[d_c]$	3	$\mathcal{OP}$, $\mathcal{S}$, $\mathcal{S}$	si `must_act_from`$(op_j, s_j, s_i) \wedge [d_c]$ est vraie, l'opération op_j du service s_j doit être évoquée par le service s_i
`must_inv_to`$[d_c]$	3	$\mathcal{OP}$, $\mathcal{S}$, $\mathcal{S}$	si `must_inv_to`$(op_i, s_i, s_j) \wedge [d_c]$ est vraie, le message de contrôle issu de la terminaison de l'opération op_i du service s_i doit être envoyé au service s_j
`must_copy`$[d_c]$	5	$\mathcal{OP}$, $\mathcal{D}$, $\mathcal{D}$, $\mathcal{S}$, $\mathcal{S}$	si `must_copy`$(op_i, in_i, out_i, s_i, s_j) \wedge [d_c]$ est vraie, les informations d'activation et de terminaison de l'opération op_i du service s_i, l'entrée in_i et la sortie out_i doivent être reçues par le service s_j. Il est à noter que in_i et/ou out_i peut être nuls.

TAB. 5.2 – Description des politiques du flux d'information d'un concepteur de procédé

Les restrictions peuvent être également étendues vers la définition de types de services, d'opérations ou de données comme nous l'avons détaillé dans la section précédente. Les services sont

toujours considérés comme des entités autonomes qu'on ne peut pas modifier. Les restrictions d'un concepteur, même si elles spécifient les interactions de services, peuvent être modifiées par le concepteur.

Exemple 14 (Politiques du flux d'information de concepteur) *De nouveau, nous considérons l'exemple de motivation présenté dans le chapitre 2. Les dépendances de la spécification centralisée définissent les interactions futures de l'exécution décentralisée. En partant de ces dépendances, un concepteur peut identifier des politiques qui spécifient les interactions correspondantes. Admettons que le concepteur veuille empêcher l'invocation du service* Livraison *de la part d'un service tiers. Dans la spécification centralisée, l'activité* a_4 *qui invoque le service* Inspecteur *a une dépendance de contrôle avec l'activité* a_5 *qui invoque le service de* Livraison*. En outre, la dernière activité a deux dépendances de donnée entrantes. Ces dépendances de données ne concernent pas l'invocation de l'activité. L'absence d'une politique du type* `can_see`(Inspecteur, Livraison *est suffisant pour empêcher une interaction de ce type. Si cette politique existe, le concepteur peut utiliser une politique de type* `must_act_from`*. Admettons que l'opération du service* Livraison *invoquée par l'activité* a_5 *soit* op_{livrer}*. Si le concepteur ne veut pas que cette opération soit invoquée par le service* Inspecteur*, il peut spécifier* `must_act_from`(op_{livrer}, Livraison, Assurance*). Ainsi, le service est invoqué de la part du service qui aurait été le coordinateur centralisé de l'exécution centralisé. De même, on peut ajouter des attributs conditionnels à cette politique.*

5.4.3 Politiques du flux d'information contextuelles

Dans cette section, nous présentons un autre type de restrictions nommées politiques contextuelles ou politiques de routage. Ces politiques peuvent appartenir à la fois aux services composés et aux concepteurs. Le contrôle du flux d'information consiste à définir le chemin qu'une donnée doit suivre dans la composition. Ainsi, une donnée peut être routée à travers les services tout en respectant les restrictions. Dans ce contexte, il est sans doute important de discuter l'adéquation d'un service pour faire partie d'un chemin. Évidemment, l'adéquation d'un service à une l'opération de routage peut être définie par un concepteur, les informations contextuelles ou le service qui fournit la donnée. Les politiques d'adéquation peuvent être définies en traitant plusieurs types de connaissances. Dans un contexte décentralisé, la propagation de l'information est gouvernée par les relations liées à la sécurité telles que la confiance. Notre compréhension de l'adéquation d'un service par rapport à un autre est la mise en œuvre des relations de sécurité liées à la circulation de données entre ceux-ci. Un service peut être considéré comme adéquate pour une opération de routage s'il respecte les règles confiances liées aux exigences de sécurité ou d'intimité. Les politiques du flux d'information explicitent les règles de sécurité pour décider de l'adéquation d'un service à remplacer une interaction qui n'est pas conforme aux restrictions dans un chemin particulier. Elles se manifestent de trois façons. Le premier type définit l'adéquation d'un service à la construction d'un chemin via d'autres services. Deux autres politiques permettent la spécification de l'adéquation par rapport à la source et à la destination du message routé. Les trois types de restrictions peuvent également contenir des attributs conditionnels permettant d'exprimer les situations conditionnelles sur les données échangées. Il faut tout de même dire que les politiques qui peuvent être considérées ne caractérisent pas la réalité du contexte à partir duquel elles explicitent les critères de choix de services pour les opérations de routage. En particulier, il n'est pas toujours aussi évident de donner un énoncé initial d'un problème de définition de ce type de restrictions. D'autre part, les restrictions contextuelles présentées peuvent servir comme des restrictions atomiques à combiner pour l'expression de restrictions

plus sophistiquées. Le tableau 5.3 formalise les principales restrictions.

Prédicat	Opérandes	Les types des opérandes	Exemple
`can_route`$[d_c]$	2	$\mathcal{M}$, $\mathcal{S}$	si `can_route`$(m, s_i) \wedge [d_c]$ est vraie, le message m peut être routé à travers le service s_i
`can_route_to`$[d_c]$	3	$\mathcal{M}$, $\mathcal{S}$, $\mathcal{S}$	si `can_route_to`$(m, s_i, s_j) \wedge [d_c]$ est vraie, le service s_i peut router le message m au service s_j. Ce politique caractérise la relation de confiance du service s_i pour le service s_j
`can_route_from`$[d_c]$	3	$\mathcal{M}$, $\mathcal{S}$, $\mathcal{S}$	si `can_route_from`$(m, s_i, s_j) \wedge [d_c]$ est vraie, le service s_j peut router le message m reçu du service s_i. Ce politique caractérise la confiance du service s_j pour le service s_i

TAB. 5.3 – Les politiques du flux d'information contextuelles

Exemple 15 (Politiques du flux d'information contextuelles (routage)) *Dans l'exemple de motivation du chapitre 2, nous avons discuté la possibilité de routage de la sortie du service Hôpital vers Inspecteur à travers le service Police. À ce point, il faut certainement définir l'adéquation du service Police. L'adéquation peut être définie par le service Police ou par le concepteur. La première politique du flux d'information contextuelle qui doit exister dans ce scénario est la politique* `can_route`*(d_1, Police) qui spécifie l'adéquation de Police pour le routage de la donnée de procédé. Les deux autres restrictions ne sont pas importantes pour ce scénario. Parce que le service Police est l'unique intermédiaire entre Hôpital et Inspecteur. En conséquence, il est mentionné dans les politiques du flux d'information exposés des services. Les deux dernières politiques du flux d'information sont appropriées pour l'expression des relations entre plusieurs services intermédiaires.*

Les attributs conditionnels

Les politiques du flux d'information sont associées à des attributs conditionnels qui spécifient leur validité. Dans cette section, nous présentons les attributs conditionnels. La spécification de ceux-ci est importante pour la traitement des politiques d'une manière statique. Comme nous l'avons déjà précisé, les politiques de flux d'information sont traitées seulement avant l'exécution. L'implémentation des attributs conditionnels fournit une certaine flexibilité à la nature statique des restrictions considérées. Le tableau 5.4 illustre les prédicats conditionnels que nous avons

utilisés pour l'expression des situations conditionnelles. Les conditions expriment des contraintes définies pour les valeurs de données, les identités des services et leur états d'exécution possibles. Il est à noter que les politiques sont identifiées par un service s_i et les restrictions sont exprimées pour les éléments de ce service.

Prédicat	Opérandes	Les types des opérandes	Exemple
$<, \leq, =, \geq, >$	1	$\mathcal{D}$	Ils permettent la spécification des restrictions sur les valeurs des données de procédés
`invoked`	2	$\mathcal{OP}$, $\mathcal{S}$	Si `invoked`(op_i, s_j) est vraie, l'opération op_i du service s_i est activée par le service s_j
`aborted`	2	$\mathcal{OP}$, $\mathcal{S}$	Si `aborted`(op_i, s_j) est vraie, l'opération op_i du service s_i est suspendue par le message de contrôle envoyé par le service s_j
`out`	3	$\mathcal{D}$, $\mathcal{S}$, $\mathcal{S}$	Si `out`(out_i, op_i, s_j) est vraie, la sortie out_i de l'opération op_i du service s_i est fournie au service s_j
`in`	3	$\mathcal{D}$, $\mathcal{S}$, $\mathcal{S}$	Si `in`(in_i, op_i, s_j) est vraie, l'entrée in_i de l'opération op_i du service s_i est fournie par le service s_i

Tab. 5.4 – Les attributs conditionnels

Comme les politiques du flux d'information, les attributs conditionnels n'expriment pas un ensemble important des situations conditionnelles telles que les situations temporelles ou spatiales. En outre, la spécification des attributs conditionnels peut se faire à un niveau plus dynamique, parce que les assertions portent sur des situations qu'on peut prédire avant l'exécution. En considérant ces simples attributs conditionnels, notre but est de démontrer l'implémentation des interactions des procédés coopérants produits dans des situations conditionnelles.

5.5 Traitement des politiques du flux d'information

Dans cette section, nous présentons notre approche du traitement des politiques du flux d'information présentées dans la section précédente. Le but ultime du traitement des politiques est de trouver des chemins fiables dans la composition à travers les services composés. Nous soulignons que notre travail ne porte pas sur la recherche des chemins dans les graphes ou de résolution des contraintes. Notre but est d'intégrer l'implémentation des restrictions existantes dans la production des procédés coopérants. Ceci est certainement lié à la nature des restrictions considérées ainsi qu'à leur traitement. Nous pouvons illustrer notre choix par un contre-exemple. Supposons que les restrictions du flux d'information soient définies par une seule entité et que la même entité soit impliquée dans l'exécution globale. Grâce à l'implication continue de cette entité dans l'exécution, les restrictions peuvent être définies comme des règles et un moteur d'inférence peut les traiter dynamiquement. L'exemple par excellence pour ce type de raisonnement est

la modélisation et le raisonnement sur les politiques d'accès dans les systèmes de gestion de workflow centralisés [Bertino *et al.*1999]. Un autre exemple où une entité centralisée est employée est le système OSIRIS pour le choix dynamique des services en fonction de leur disponibilités [Schuler *et al.*2004].

5.5.1 Vérification des politiques

Dans cette sous-section, nous détaillons la vérification des politiques du flux d'information comme la première étape de l'établissement des chemin fiables entre les services. Comme nous l'avons antérieurement souligné, chaque dépendance de contrôle ($\mathcal{E}_c$), de donnée ($\mathcal{E}_d$), et d'étendue ($\mathcal{E}_{ext}$) correspond à une interaction entre le service source et le service cible de la dépendance. Dans les grandes lignes, la vérification porte sur le test d'adéquation entre les dépendances et les politiques de flux d'information. La vérification des politiques concerne la production de trois ensembles qui regroupent les dépendances du procédé par rapport à leur validité avec les politiques de flux d'information. Les ensembles sont : $Denied\_Interactions$, $Permitted\_Interactions$ et $Conditional\_Interactions$. Les éléments de $Denied\_Interactions$ et $Permitted\_Interactions$ contiennent respectivement les dépendances du procédé pour lesquelles une interaction mutuelle entre les services sous-jacents est respectivement, impossible et possible. Les éléments de l'ensemble $Conditional\_Interactions$ concernent les dépendances du procédé qui peuvent être implémentées comme des interactions directes entre les services pour la partie vraie ou fausse de la condition. En conséquence, une partie de l'interaction est considérée comme une interaction permise et l'autre partie est considérée comme une interaction qui doit être implémentée par un autre ensemble d'interactions. La vérification ne peut pas se faire par des tests simples. Car, les attributs conditionnels nécessitent un traitement orienté planification. Ceci est lié à des attributs conditionnels qui expriment des restrictions sur l'histoire de l'exécution telles que l'état d'exécution des activité. Par rapport aux spécifications des situations conditionnelles, les interactions exigées peuvent être nombreuses. Nous formalisons les interactions conditionnelles dans plusieurs sous-ensembles. Le premier sous-ensemble correspond aux situations conditionnelles qui peuvent être implémentée comme une interaction directe entre deux services. Cette partie de l'ensemble est nommée $Conditional\_Interactions^{+}$. Elle contient un élément qui caractérise l'interaction possible. Les interactions qui doivent être remplacées par des chemins fiables sont regroupées dans l'ensemble $Conditional\_Interactions^{-}$. Par rapport au nombre de situations conditionnelles, les éléments de l'ensemble $Conditional\_Interactions^{-}$ sont nommés $Conditional\_Interactions^{-}_{1}$ ... $Conditional\_Interactions^{-}_{n}$. Chaque élément $Conditional\_Interactions^{-}_{i}$ de l'ensemble $Conditional\_Interactions^{-}$ est censé être remplacé par un chemin fiable. La figure 5.2 illustre la vue d'ensemble du processus de vérification des politiques et les interactions correspondantes. Dans le processus de vérification, un premier test porte sur la vérification des politiques en vue du groupement des interactions dans les ensembles $Permitted\_Interactions$ et $Denied\_Interactions$. Cette vérification est en partie intuitive. Les politiques du flux d'information des services composés et les politiques spécifiées par les concepteurs sont au cœur de cette opération de vérification. Par la suite, les politiques concernant les attributs conditionnels sont traités. Ces politiques peuvent porter sur les situations conditionnelles dépendant des valeurs des données du procédé ou bien de l'histoire d'exécution du procédé. Pour les restrictions qui portent sur les conditions exprimées avec les données du procédé, un regroupement initial est fait dans les deux ensembles $Conditional\_Interactions^{+}$ et $Conditional\_Interactions^{-}$. Ensuite, la vérification porte sur les éléments des interactions regroupés dans $Permitted\_Interactions$ et $Conditional\_Interactions^{+}$. Car ces deux ensembles contiennent les interactions probables exprimées dans certaines politiques du flux d'information.

Le traitement des éléments de ces deux ensembles produit les éléments de $Conditional\_Interactions^{-}$.

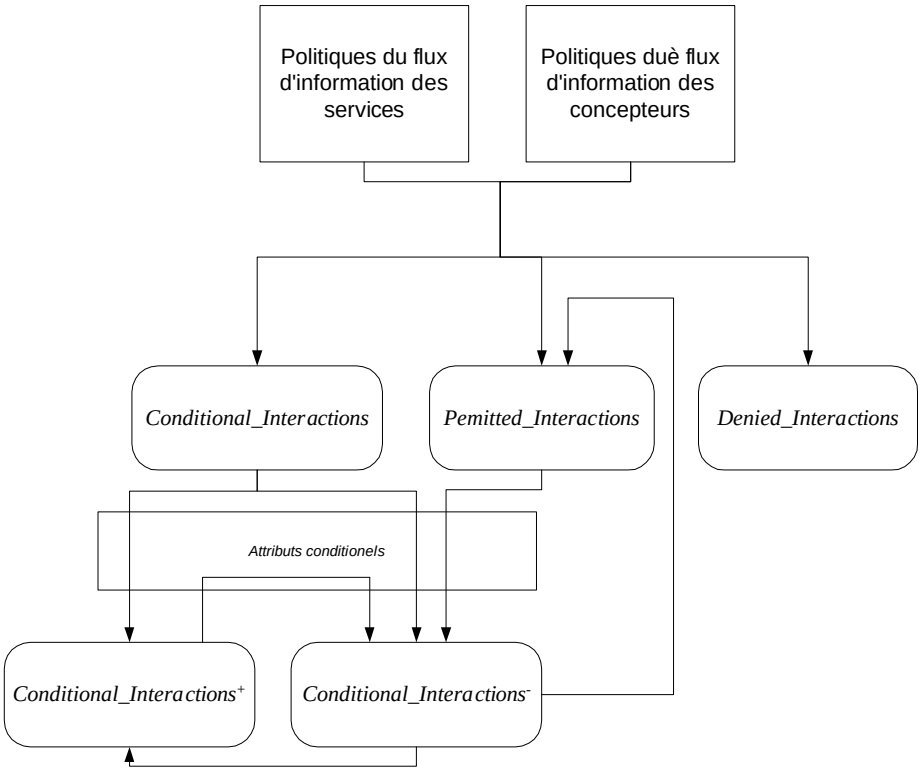


FIG. 5.2 – Vue d'ensemble du traitement des politiques pour la vérification de l'adéquation des politiques et des interactions

Les aspects formels de la vérification des politiques du flux d'information sont illustrés dans les deux algorithmes suivants. L'algorithme 3 illustre la vue d'ensemble de la vérification appliquée à une dépendance de donnée. La première partie de vérification consiste à classer les dépendances en fonction de leur adéquations dans trois ensembles. L'algorithme est détaillé pour une dépendance de donnée qui relie deux activités. L'algorithme peut être appliqué à une dépendance de contrôle ou à une dépendance d'étendue. Il est à noter que la deuxième partie de l'algorithme utilise les résultats de la première partie en vue d'identifier les interactions conditionnelles qui dépendent des états d'interaction des autres activités.

Exemple 16 (Vérification des politiques du flux d'information) *Nous considérons le procédé illustré en figure 5.3 et les politiques présentées dans le tableau 5.5 pour expliquer la vérification des politiques par rapport au procédé. Les interactions vérifiées concernent les dépendances de donnée entre les activités a_{12} et a_{19}, et, a_{13} et a_{19}. Nous assumons que les activités a_{12} et a_{13} sont exécutées par le service Hôpital et l'activité Inspecteur. Par rapport aux politiques du tableau les interactions de ces services doivent être remplacées, parce que l'interaction qui correspond à la dépendance de donnée d_5 n'est pas possible à cause de la politique du service Inspecteur qui limite l'entrée in_1 de son opération op_{19} au service Police. En conséquence cette interaction est placée dans l'ensemble de Denied_Interactions. La deuxième interaction qui correspond à la dépendance d_5, est une interaction conditionnelle, car la politique définit une condition sur*

Hôpital	can_out(out_1 (d_4), op_{13}, Inspecteur)
Inspecteur	can_in($in_1(d_4), op_{19}$, Police) can_in($in_2(d_5), op_{19}$, Hôpital)[$in_2.att_1$=="Dupont"]

TAB. 5.5 – Exemples de politiques du flux d'information

must_out(op_2, $out_1(d_2)$, Police, Assurance)
must_in(op_{12}, $in_1(d_2)$, Livraison, Hôpital)

TAB. 5.6 – Exemples de politiques du flux d'information

la politique exposée. L'échange peut se faire entre Hôpital et Inspecteur si l'attribut att_1 de la donnée échangée est "Dupont". Pendant l'exécution, cette donnée peut prendre cette valeur ou pas. En conséquence, il y a deux possibilité. Pour cette valeur de la donnée, il y a deux possibilités d'interaction dont l'une est permise. La partie permise de l'interaction est placée dans l'ensemble $Conditional\_Interactions^+$. L'autre partie de l'interaction fait partie de l'ensemble $Conditional\_Interactions^-$. Il est à noter que le concepteur doit supporter ces interactions en définissant la politique can_see(Hôpital, Inspecteur). Considérons maintenant les politiques illustrées dans le tableau 5.6. Ces politiques sont des restrictions définies par le concepteur de procédé. Elles illustrent les interactions requises par le concepteur en vue de cette composition. Ces restrictions concernent la sortie de l'activité a_2 et l'entrée de l'activité a_{12} liée par la dépendance de donnée d_2. Nous assumons que l'activité source est exécutée par le service Police et l'activité cible est exécutée par le service Hôpital. La première politique du tableau 5.6 spécifie que la sortie de l'activité a_2 qui invoque op_2 de Police doit être routé vers le service Assurrance par le service Police. La deuxième politique spécifie que cette entrée qui doit être utilisée comme l'entrée de l'opération op_{12} invoquée par l'activité a_{12}, doit être reçue par le service Hôpital depuis service Livraison. En conséquence, cette dépendance est placée dans l'ensemble Denied_Interactions. Les interactions qui sont regroupées dans ces ensembles constituent les entrées du mécanisme qui remplace les interactions non-conformes par des chemins fiables et du mécanisme qui produit les procédés coopérants correspondants.

Les éléments principaux que nous traitons sont le modèle de procédé à décentraliser sont les dépendances de données, les dépendances de contrôles et les dépendances étendues. Chaque dépendance est vérifiée par rapport aux politiques pour décider de la possibilité de son implémentation comme une interaction d'égal-à-égal entre les services sous-jacents. Les politiques sont vérifiés en plusieurs étapes. La première vérification porte sur la vérification des politiques du service source et celles du service cible de la dépendance. La deuxième vérification porte sur le traitement des politiques des concepteurs. Les politiques des concepteur concernant la possibilité des interactions entre les services sont vérifiées. Les dépendances qui sont conformes à la fois aux politiques des services et aussi aux politiques des concepteurs sont placées dans l'ensemble *Permitted_Interaction*. Si une dépendance n'est pas conforme à une politique, elle est placée dans l'ensemble *Denied_Interaction*. Les dépendances qui font objets des attributs conditionnels sont placées dans l'ensemble *Conditional_Interactions*. Une partie d'une dépendance peut être en adéquation avec la politique alors que le complément ne l'est pas. Dans ce cas, la partie faisable peut être implémentée comme une interaction directe entre les services. La partie qui n'est pas en adéquation doit être traitée comme une dépendance qui est dans l'ensemble *Denied_Interaction*. Les attributs conditionnels qui expriment les conditions sur les données du

Input : 1) A process, P
2) Information flow policies of composed services, $\mathcal{I}_{\mathcal{S}}$
3) Information flow policies of process designers, $\mathcal{I}_{PD}$
Output: 1) $Denied\_Interactions(P)$
2) $Conditional\_Interactions(P)$
3) $Permitted\_Interactions(P)$
forall $e \in \mathcal{E}_d$ **do**
$s_i = source_{ser}(e)$;
$s_j = target_{ser}(e)$;
$a_i = source_{act}(e)$;
$a_j = target_{act}(e)$;
$op_i = operation(a_i)$ où out_i est la sortie de op_i ;
$op_j = operation(a_j)$ où in_j est l'entrée de op_j ;
switch e **do**
case $\texttt{can\_see}(s_i, s_j) \wedge \texttt{can\_in}(in_j, op_j, s_i) \wedge \texttt{can\_out}(out_i, op_i, s_i)$
$Permitted\_Interactions(P) \leftarrow e$;
end
case $\neg\texttt{can\_see}(s_i, s_j) \vee \neg\texttt{can\_in}(in_j, op_j, s_i) \vee \neg\texttt{can\_out}(out_i, op_i, s_i)$
$Denied\_Interactions(P) \leftarrow e$;
end
case $(\texttt{can\_see}(s_i, s_j)[d_a] \vee \texttt{can\_in}(in_j, op_j, s_i)[d_b] \vee \texttt{can\_out}(out_i, op_i, s_i)[d_c]) \wedge (d_b \neq d_c \vee d_b \neq d_a \vee d_a \neq d_c)$
$Conditional\_Interactions(P) \leftarrow e$;
end
end
end
forall $e \in Conditional\_Interactions$ *où subit une condition dépendante de l'historique de l'exécution avec* d_a *la condition considérée* **do**
forall $e' \in Permitted\_Interactions \cup Conditional\_Interactions^{+}$ **do**
if d_a *est vraie pour l'interaction* e' **then**
$Denied\_Interactions \leftarrow e$
end
$Permitted\_Interactions \leftarrow e$
end
end

Algorithm 3: La vérification de l'adéquation des politiques et des interactions pour une dépendance de donnée

procédé peuvent être vérifiés directement. Les conditions qui expriment l'état d'exécution des activités exigent la vérification de la possibilité d'exécution de l'exécution passée. Les dépendances restreintes par les attributs conditionnels des autres politiques sont placées dans un ensemble intermédiaire pour être vérifiées. L'ensemble $Permitted\_Interaction$ inclut deux sous ensembles nommés $Permitted\_Interaction^{+}$ et $Permitted\_Interaction^{-}$ pour le traitement intermédiaire. Aussi, les interactions sont regroupées dans trois ensembles : $Permitted\_Interactions$, $Denied\_Interactions$, $Conditional\_Interactions$. Le dernier est composé de deux sous-ensembles comme nous l'avons illustré dans la figure 5.2.

5.5.2 Établissement des chemins fiables entre les services

La vérification classe les interactions en trois groupes. Comme nous l'avons souligné ci-dessus, les interactions qui ne sont pas en adéquation avec les politiques doivent être remplacées par des interactions alternatives qui respectent les politiques de chacun des services composés. Les interactions alternatives sont des chemins produits à partir des politiques du flux d'information. Ces chemins fiables sont constitués d'interactions mutuellement en adéquation. Dans cette section, nous expliquons les principes de l'établissement des chemins fiables et certaines définitions nécessaires pour l'évolution des algorithmes qui produisent les procédés coopérants. Comme nous l'avons souligné ci-dessus, le problème d'implémentation des chemins fiables est un problème de recherche de chemin par rapport aux politiques du flux d'information. Il est à noter que nous ne cherchons pas à implémenter une nouvelle méthode de recherche de chemin pour ce propos. Pour ce faire, nous avons utilisé l'algorithme de Dijkstra [Dijkstra1959] qui est largement utilisé dans la modélisation de recherche de chemin dans les systèmes d'information. L'algorithme de Dijkstra consiste à trouver le plus court chemin entre deux points. En conséquence, les concepts que nous avons utilisés doivent être adaptés pour ces algorithmes considérés. Ces mesures peuvent être essentiellement définies par les niveaux de confiance, le coût d'utilisation du chemin ou bien des critères similaires que l'on peut traduire en valeurs numériques. Dans l'implémentation du prototype, nous avons utilisé ces deux métriques ci-dessus. Nous n'étendrons pas la discussion sur ce point. Parce que cela ne change pas la démarche globale de l'approche. Par rapport à l'existence des chemins fiables, nous définissons les concepts suivants. Ils correspondent aux types de chemins définis pour chaque type d'interactions définies ci-dessus.

Définition 1 ***(Chemin fiable)***
$Uncheminfiable\text{rp}^i$ qui remplace une interaction $e \in \mathcal{E}_c \cup \mathcal{E}_d \cup \mathcal{E}_{ext}$ est une série de services $rp_i = <s_m, s_{m+2}, ..., s_{n-1}, s_n>$ où $\forall s_i \in rp_i$, il y a une adéquation du flux d'information entre s_i et s_{i+1}.
- Si $e \in Denied\_Interactions$ alors rp_i est unique.
- Si $e \in Conditional\_Interactions$, $\forall e_j \in Conditional\_Interactions^-$, le chemin rp_i inclut un sous-chemin rp_i^j.

Comme nous l'avons présentée dans la définition 1, un chemin fiable est constitué d'une série de services où les services sont mutuellement en adéquation avec le flux d'information. Pour les services qui sont les éléments intermédiaires d'un chemin, les politiques contextuelles et les politiques des concepteurs sont traitées. Les services qui sont au début et à la fin d'un chemin fiable concernent les politiques du flux d'information exposées par ces services et ainsi que celles de concepteurs de procédé. Les détails de l'algorithme Dijkstra implémenté peut être trouvé dans l'annexe B. Il est à noter que la problématique principale de notre travail ne porte pas sur la proposition d'un algorithme de recherche de chemin. Nous avons utilisé un algorithme approprié pour formaliser ce problème.

Exemple 17 (Détail de l'algorithme simplifié) *L'algorithme 4 résume une partie de la démarche de l'algorithme complet détaillé dans l'annexe B. Il illustre la recherche de chemin pour une interaction correspondant à une dépendance de contrôle et à une dépendance de donnée. En effet, les politiques traitées pour la recherche de chemin sont plus nombreuses que celles qui sont présentés dans cet algorithme réduit.*

Entrée : 1) La spécification du procédé, P
2) Les politiques du flux d'information,
Sortie : 1) FALSE, si un chemin fiable n'est pas trouvé pour une interaction
2) Les chemins pour toutes interactions qui doivent être remplacées, $\mathcal{R}_P = [rp_0, ..., rp_n]$;

1. **for all** $e \in \mathcal{E}_c \wedge$ *Denied_Interactions* de P tels que $\Theta(e) = 0$, a_i est l'activité source et a_j l'activité cible de e, s_i et s_j sont invoqués respectivement a_i et a_j, op_i et op_j sont des opérations de s_i et s_j invoquées par les activités a_i et a_j ;
Trouver un chemin fiable rp_i tels que $rp_i = <s_m, s_{m+1}, ..., s_{n-1}, s_n>$ où `can_route`(* ,s_m, m) $\wedge$ `can_route`(s_i, s_{m+1}, m) $\wedge$ `can_route`(s_m, s_{m+2}, m) $\wedge$ ... $\wedge$ `can_route`(s_{n-2}, s_n, m) $\wedge$ `can_act_from`(s_n, op_j)
end for
ajouter rp_i à $\mathcal{R}_P$
2. **for all** $e \in \mathcal{E}_d \wedge$ *Denied_Interactions* de P tels que $\Theta(e) = 0$, a_i est l'activité source et a_j l'activité cible de e, s_i et s_j sont respectivement par a_i et a_j, op_i et op_j sont des opérations de s_i et s_j sont invoqués respectivement par les activités a_i et a_j ;
Trouver un chemin fiable rp_i tels que $rp_i = <s_m, s_{m+1}, ..., s_{n-1}, s_n>$ où `can_out`(s_m, op_i, e) $\wedge$ `can_route`(s_i, s_{m+1}, m) $\wedge$ `can_route`(s_m, s_{m+2}, m) $\wedge$ ... $\wedge$ `can_route`(s_{n-2}, s_n, m) $\wedge$ `can_in`(s_n, op_j, e)
end for
ajouter rp_i à $\mathcal{R}_P$
3. **If** $\exists e \in \mathcal{E}_c \cup \mathcal{E}_d$ de P avec $\Theta(e)$=0 et il n'y a pas de rp_i ; **alors** return FALSE ;

Algorithm 4: L'exemple de recherche de *chemins fiables*

5.6 Production des procédés coopérants

Dans la section précédente, nous avons détaillé la vérification des politiques de flux d'information et la production des chemins fiables à travers les services. Les chemins fiables constituent l'entrée du système de production des procédés coopérants que nous détaillons dans cette section. Selon notre approche, un service a deux rôles principaux. Il établit des interactions d'égal-à-égal avec les autres services de la composition en exécutant un procédé coopérant. Ainsi, il peut router ses sorties directement vers les services qui doivent les utiliser et recevoir les données qu'il doit utiliser directement de la part des services qui les fournissent. Le deuxième rôle d'un service est de contribuer à l'ensemble de l'exécution en faisant partie des chemins établis entre les services qui ne peuvent pas avoir d'interaction directe. Ce deuxième rôle consiste à router les données venant d'un service vers un autre service qui fait aussi partie du même chemin. D'un point de vue de la sémantique des procédés coopérants, ces deux rôles sont indépendants. Par conséquent, leur implémentation au sein d'un unique procédé coopérant que le service exécute est relativement difficile, et de plus cela n'est pas nécessaire d'un point de vue technique. Nous préférons produire des procédés coopérants séparément pour implémenter ces deux rôles. Ainsi, quand un service est impliqué dans une composition, il exécute un procédé principal qui permet d'établir des interactions pour l'invocation de ses propres opérations. En outre, il exécute, autant de fragments que de chemins sur lesquels il se trouve. Il est à noter que ces derniers sont moins sophistiqués que le procédé principal.

Dans cette section, nous présentons les algorithmes que nous avons utilisés pour produire les

procédés coopérants selon les principes ci-dessus. Quand un service s_i est choisi pour être utilisé dans une composition, il exécute un ensemble de fragments de procédé pour cette composition. Cet ensemble de procédé est nommé $\mathcal{P}_{s_i}$ avec $\mathcal{P}_{s_i} = \{P^*_{s_i}, P^1_{s_i}, P^2_{s_i}, ..., P^n_{s_i}\}$ où la variable n est le nombre des chemins fiables auquel le service s_i contribue. Le premier procédé de cet ensemble, nommé $P^*_{s_i}$, est le procédé principal que s_i exécute pour contribuer à l'exécution décentralisée. Les autres fragments incluent les activités de routages des données d'un service et à router vers un autre. Comme présenté dans le chapitre précédent, nous examinons la production des procédés coopérants en deux parties. La première consiste à étudier la production des procédés coopérants vis-à-vis des interactions sortantes. La deuxième étude porte sur la production des procédés coopérants vis-à-vis des interactions entrantes. Par rapport aux algorithmes présentés dans le chapitre précèdent, la différence essentielle du procédé principal $P^*_{s_i}$ est la restructuration des activités de connexion et leurs instanciations vis-à-vis des services évoqués. L'établissement des chemins alternatifs entre les services nécessite la configuration des identités des services composés. Par exemple, si un service A évoque un service B pour implémenter une restriction spécifiée de la part du service de A et/ou de B, le procédé coopérant exécuté par le service A doit référencer le service B et réciproquement le procédé exécuté par le service B doit référencer le service A. Dans l'implémentation des situations conditionnelles qui nécessitent le routage des données ou l'invocation des services par rapport aux conditions satisfaites, les procédés coopérants ne peuvent pas être structurés d'une manière séquentielle dans lesquels les activités de connexion sont placées sans tenir compte des situations conditionnelles imposées par les politiques du flux d'information.

Dans la suite, nous présentons les algorithmes pour la production de procédés coopérants qui permettent les interactions des éléments principaux du service avec les autres procédés coopérants. Les algorithmes 5, 6 et 7 détaillent l'implémentation de l'interconnexion avec les éléments des postsets. Dans ces algorithmes, la structuration des activités de connexion est examinée en trois étapes. Dans un premier temps, les dépendances de contrôle correspondantes sont traitées par rapport à leur situations dans la classification des interactions. Si la dépendance traitée fait partie de l'ensemble $Permitted\_Interactions$, dans ce cas, cette dépendance est implémentée comme nous l'avons expliquée dans le chapitre précédent. On peut dire qu'il n' y a pas de différence d'implémentation entre une dépendance de l'ensemble de $Permitted\_Interactions$ et une implémentation normale du chapitre précédent. Si la dépendance de contrôle fait partie de l'ensemble $Denied\_Interactions$, l'implémentation se fait par l'implémentation d'un chemin fiable à travers les services. Dans ce cas, l'activité de connexion correspondante est initialisée avec l'identité du premier service qui est dans le chemin qui remplace la dépendance. Ainsi, le message de contrôle qui caractérise la terminaison d'une activité, et, qui sera utilisé comme le message d'activation de l'activité dépendante par rapport au flux de contrôle est routé vers le service qui est le suivant dans le chemin fiable construit. La structuration des activités de connexion qui implémentent les situations conditionnelles est relativement plus compliquée. Les interactions conditionnelles peuvent être implémentées de deux manières différentes. Si la restriction conditionnelle peut être examinée comme une partie en adéquation avec les restrictions et une autre partie qui sera traitée comme une dépendance dans $Denied\_Interactions$. Le deuxième cas est l'existence de plusieurs situations conditionnelles dont chacune est remplacée par un autre chemin. Dans ce cas, le procédé coopérant qui exécute l'activité de connexion doit considérer non seulement une activité de connexion mais un ensemble d'activités de connexion pour chacun des services qui font partie de ces chemins. Les activités de connexion qui permettent les interactions avec les premiers services des chemins fiables doivent être structurés d'une manière exclusive comme les conditions sont traitées exclusivement en vue de l'établissement des chemins correspondants.

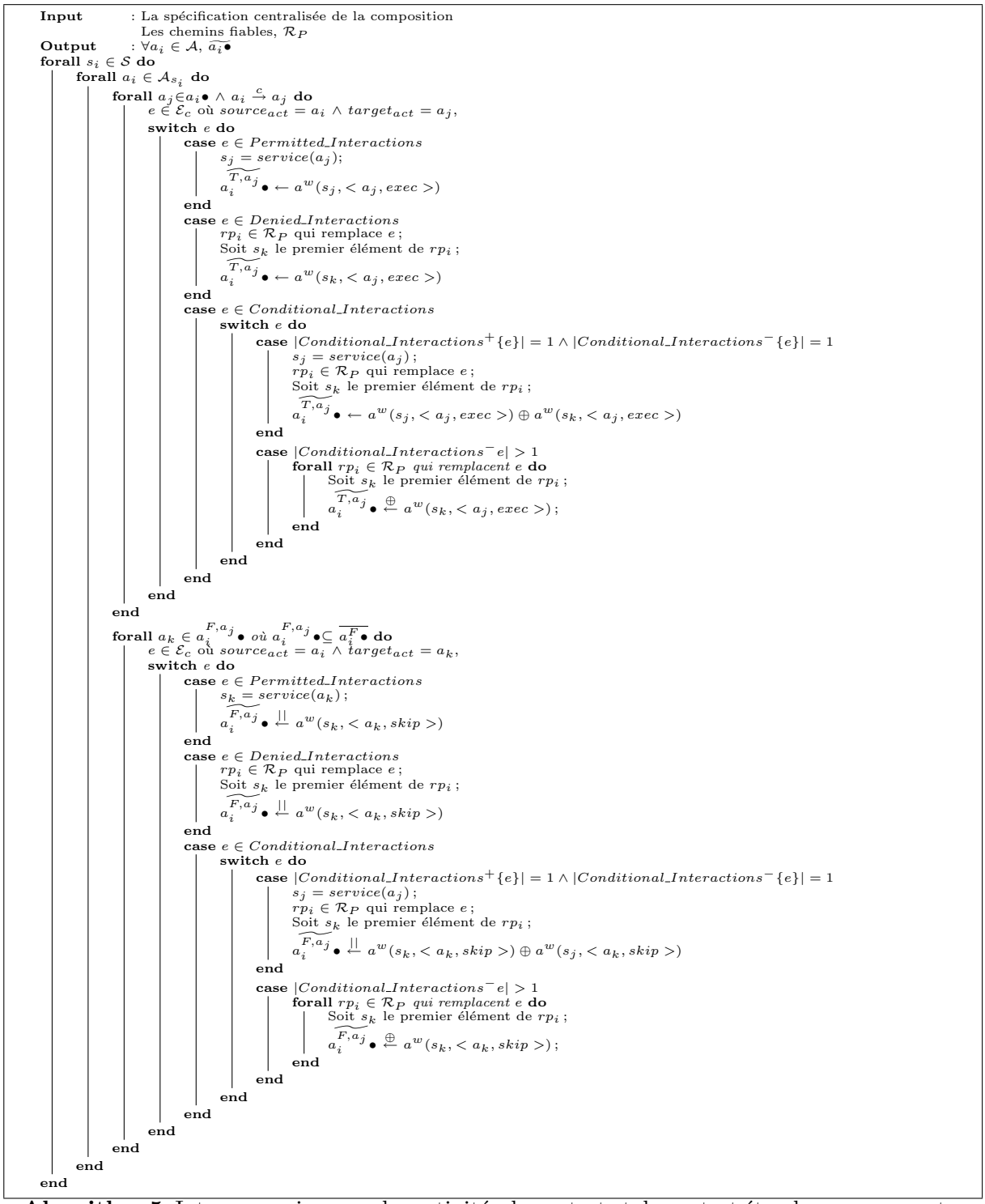
La structuration des activités de connexion qui correspondent aux dépendances étendues de l'ensemble $\mathcal{E}_{ext}$ et les dépendances de données de l'ensemble $\mathcal{E}_d$ sont traitées de la même manière. Il est à noter que la structuration exclusive des activités de connexion nécessite l'évaluation des données de procédé qui permettent le choix des chemins appropriés. Les conditions de transition évaluées ne peuvent pas être évaluée au près d'un service qui n'exécute pas les activités de connexion. En conséquence, les restrictions du flux d'information doivent spécifier des contraintes sur les données qui peuvent être traitées seulement par le procédé coopérant courant de l'activité de connexion envisagée. Dans les cas où les conditions spécifient des restrictions sur des données fournies par d'autre services, les instances de ces données doivent être transmises au procédé qui va les évaluer pour initialiser les activités de connexion.

La production des procédés pour la structuration des activités de connexion est présentée dans les algorithmes 8, 9 et 10 en trois parties. La démarche adoptée est similaire à celle de l'interconnection des éléments des presets. Des différences mineures existent entre les algorithmes. La différence majeure porte sur la réception des données de procédés et sur les messages de contrôle correspondants. Cela veut dire qu'un procédé coopérant peut envoyer un message de suspension au procédé qui reçoit la sortie d'une activité qui sera envoyée par un procédé coopérant. Cette situation est formalisée avec des activités de connexion exclusives qui sont ajoutées dans le fragment qui suit l'activité originale de la spécification centralisée.

Comme nous l'avons déjà mentionné, la production des procédés coopérants qui implémentent le routage des données à travers des chemins fiables est relativement plus simple par rapport à la production des procédés coopérants principaux. L'algorithme 11 illustre la formalisation de cette production. La production des chemins fiables est expliquée dans les sections précédentes. Dans cette section, nous présentons seulement la production des procédés en utilisant les chemins produits. Intuitivement, les algorithmes peuvent être expliqués comme ci-dessous. Le rôle secondaire de chaque service est concrétisé par la réception d'un message d'un autre service et l'envoi de ce même message au service qui le suit sur le chemin fiable. Bien évidemment, la réception ou l'envoi du message routé peut être conditionnée. La réception est implémentée par une activité de connexion de lecture lorsque l'envoi est implémenté par une activité de connexion d'écriture. Les situations conditionnelles sont implémentées par les activités de connexion structurées d'une manière exclusive. La dernière est similaire à l'implémentation des situations conditionnelles au sein du procédé coopérant principal. L'algorithme 11 traite la production des fragments de routages en trois parties. Si le service qui fait partie du chemin produit est un élément intermédiaire du chemin traité, le procédé de routage $P^i_{s_i}$ est formé de deux activités de connexion qui sont initialisées avec l'identité des services qui précède et succède ce service dans le chemin produit. Si le service constitue le premier ou le dernier élément du chemin, son identité est intégrée dans le procédé coopérant principal que le service exécute.

L'exemple suivant illustre l'utilisation des algorithmes pour un procédé et un ensemble de politique du flux d'information.

Exemple 18 (Production des fragments coopérants) *La figure 5.3 illustre un exemple de procédé que nous avons déjà utilisé dans dans le chapitre problématique et le chapitre précédent. Il s'agit d'une spécification centralisée d'un procédé à dériver en procédés coopérants. Nous allons étudier deux versions différentes du même exemple avec des restrictions différentes. Le procédé illustré dans la figure 5.3 compose plusieurs services. Les services que nous utilisons pour cet exemple sont* Police(P), Banque1(B1), Banque2(B2), Hôpital(H), Inspecteur1(I1), Inspecteur2(I2), Livraison(L). *Les activités associées avec ces services sont :* $\mathcal{A}_P = \{a_3, a_8\}$, $\mathcal{A}_{B1} = \{a_{13}, a_{14}\}$, $\mathcal{A}_{B2} = \{a_{15}, a_{16}\}$, $\mathcal{A}_H = \{a_{12}\}$, $\mathcal{A}_{I1} = \{a_{19}\}$ *et* $\mathcal{A}_{I2} = \{a_{18}\}$. *Nous ne nous intéressons pas aux autres activités et aux détails de la logique du procédé. Pour faciliter la notation, nous supposons que*



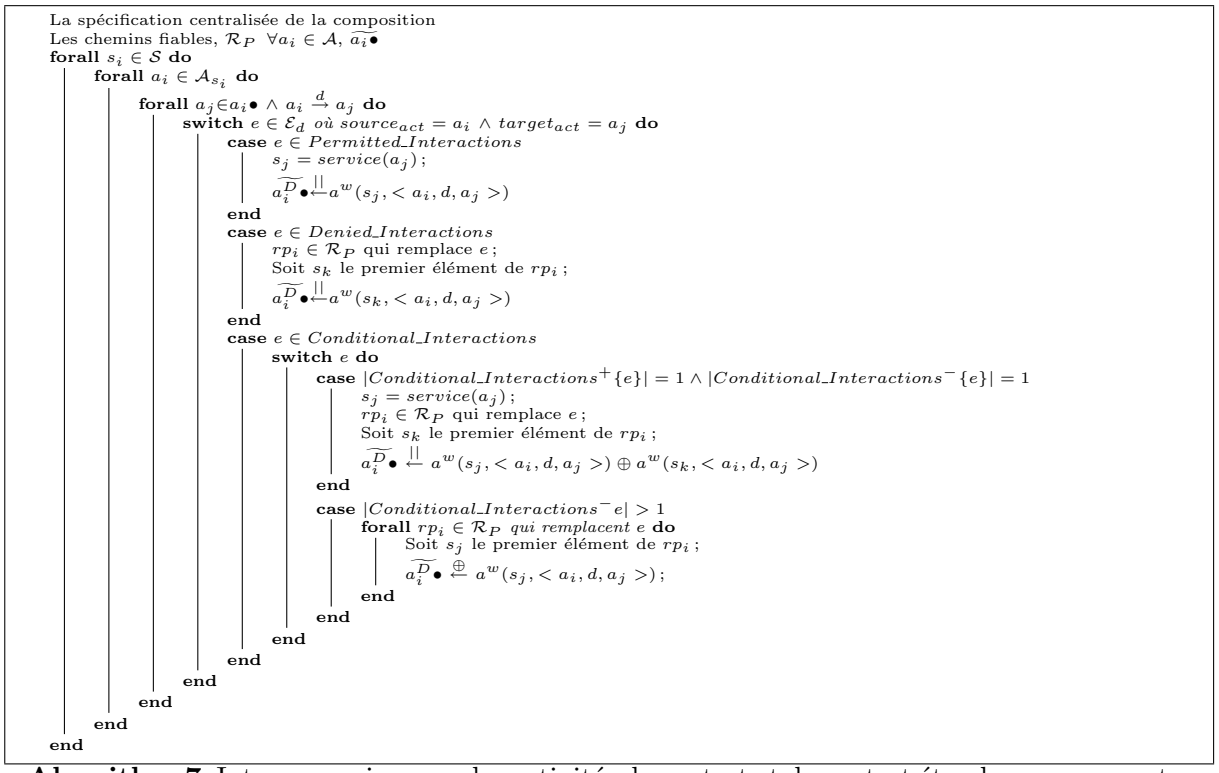
Input : La spécification centralisée de la composition
Les chemins fiables, $\mathcal{R}_P$
Output : $\forall a_i \in \mathcal{A}$, $\widetilde{a_i \bullet}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_j \in a_i\bullet \wedge a_i \xrightarrow{c} a_j$ **do**
      $e \in \mathcal{E}_c$ où $source_{act} = a_i \wedge target_{act} = a_j$,
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_j = service(a_j)$;
          $a_i^{\widetilde{T,a_j}} \bullet \leftarrow a^w(s_j, < a_j, exec >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_k le premier élément de rp_i ;
          $a_i^{\widetilde{T,a_j}} \bullet \leftarrow a^w(s_k, < a_j, exec >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
              $s_j = service(a_j)$;
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              Soit s_k le premier élément de rp_i ;
              $a_i^{\widetilde{T,a_j}} \bullet \leftarrow a^w(s_j, < a_j, exec >) \oplus a^w(s_k, < a_j, exec >)$
            end
            case $|Conditional\_Interactions^- e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                Soit s_k le premier élément de rp_i ;
                $a_i^{\widetilde{T,a_j}} \bullet \overset{\oplus}{\leftarrow} a^w(s_k, < a_j, exec >)$;
              end
            end
          end
        end
      end
    end
    forall $a_k \in a_i^{F,a_j}\bullet$ où $a_i^{F,a_j}\bullet \subseteq \overline{a_i^F \bullet}$ **do**
      $e \in \mathcal{E}_c$ où $source_{act} = a_i \wedge target_{act} = a_k$,
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_k = service(a_k)$;
          $a_i^{\widetilde{F,a_j}} \bullet \overset{||}{\leftarrow} a^w(s_k, < a_k, skip >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_k le premier élément de rp_i ;
          $a_i^{\widetilde{F,a_j}} \bullet \overset{||}{\leftarrow} a^w(s_k, < a_k, skip >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
              $s_j = service(a_j)$;
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              Soit s_k le premier élément de rp_i ;
              $a_i^{\widetilde{F,a_j}} \bullet \overset{||}{\leftarrow} a^w(s_k, < a_k, skip >) \oplus a^w(s_j, < a_k, skip >)$
            end
            case $|Conditional\_Interactions^- e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                Soit s_k le premier élément de rp_i ;
                $a_i^{\widetilde{F,a_j}} \bullet \overset{\oplus}{\leftarrow} a^w(s_k, < a_k, skip >)$;
              end
            end
          end
        end
      end
    end
  end
end

Algorithm 5: Interconnexion avec les activités de postset et de postset étendue par rapport aux politiques du flux d'information (Première partie)

les opérations des services ont les mêmes identités que les activités de la spécification centralisée. Par exemple, les opérations du service B1 sont nommées op_{13} et op_{14} et les opérations du service I1 sont nommées op_{19}. Nous supposons que les politiques du flux d'information sont illustrées

Input : La spécification centralisée de la composition
Les chemins fiables, $\mathcal{R}_P$
Output : $\forall a_i \in \mathcal{A}$, $\widetilde{a_i \bullet}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_k \in a_i^{D,F,a_j}\bullet$ *où* $a_i^{D,F,a_j}\bullet \subset \overline{a_i^{D,F}\bullet}$ **do**
      $e \in \mathcal{E}_{ext}$ où $source_{act} = a_i \wedge target_{act} = a_k$,
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_k = service(a_k)$;
          $a_i^{\widetilde{D,F,a_j}}\bullet \overset{||}{\leftarrow} a^w(s_k, < a_i, skip, d, a_j >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_l le premier élément de rp_i ;
          $a_i^{\widetilde{D,F,a_j}}\bullet \overset{||}{\leftarrow} a^w(s_l, < a_i, skip, d, a_j >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
              $s_j = service(a_j)$;
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              Soit s_l le premier élément de rp_i ;
              $a_i^{\widetilde{D,F,a_j}}\bullet \overset{||}{\leftarrow} a^w(s_k, < a_i, skip, d, a_j >) \oplus a^w(s_l, < a_i, skip, d, a_j >)$
            end
            case $|Conditional\_Interactions^-e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                Soit s_l le premier élément de rp_i ;
                $a_i^{\widetilde{D,F,a_j}}\bullet \overset{\oplus}{\leftarrow} a^w(s_l, < a_i, skip, d, a_j >)$;
              end
            end
          end
        end
      end
    end
    forall $a_k \in a_i^{D,T,a_j}\bullet$ *où* $a_i^{D,T,a_j}\bullet \subset \overline{a_i^{D,T}\bullet}$ **do**
      $e \in \mathcal{E}_{ext}$ où $source_{act} = a_i \wedge target_{act} = a_k$,
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_k = service(a_k)$;
          $a_i^{\widetilde{D,T,a_j}}\bullet \overset{||}{\leftarrow} a^w(s_k, < a_i, exec, d, a_j >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_l le premier élément de rp_i ;
          $a_i^{\widetilde{D,T,a_j}}\bullet \overset{||}{\leftarrow} a^w(s_l, < a_i, exec, d, a_j >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
              $s_k = service(a_k)$;
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              Soit s_l le premier élément de rp_i ;
              $a_i^{\widetilde{D,T,a_j}}\bullet \leftarrow a^w(s_l, < a_i, exec, d, a_j >) \oplus a^w(s_k, < a_i, exec, d, a_j >)$
            end
            case $|Conditional\_Interactions^-e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                Soit s_l le premier élément de rp_i ;
                $a_i^{\widetilde{D,T,a_j}}\bullet \overset{\oplus}{\leftarrow} a^w(s_l, < a_i, exec, d, a_j >)$;
              end
            end
          end
        end
      end
    end
  end
end

Algorithm 6: Interconnexion avec les activités de postset et de postset étendue par rapport aux politiques du flux d'information (Deuxième partie)



Algorithm 7: Interconnexion avec les activités de postset et de postset étendue par rapport aux politiques du flux d'information (Troisième partie)

dans le tableau 5.7. Il est à noter que ces politiques ne sont pas exclusives. Les politiques considérées sont les politiques exposées par les services composés et les politiques contextuelles qui spécifient les relations d'adéquation entre les services pour les données routées. Les entrées et les sorties des activités sont notées avec les indices $i = 1..n$. La notation des variables d'entrées et de sorties avec les activités auxquelles appartiennent.

Un traitement initial de ces politiques donne les interactions qui doivent être remplacées par des chemins fiables entre les services composés. La dépendance de donnée entre l'activité a_{12} et l'activité a_{19} n'est pas en adéquation avec les politiques exposées par le service Hôpital et le service Inspecteur1 par rapport aux politiques du tableau 5.7. Si les politiques contextuelles du tableau 5.8 sont traitées, un chemin Hôpital, Banque2, Banque1, Inspecteur1 est trouvé pour la donnée de sortie de l'activité a_{12}. Ce chemin est nommé rp_1 pour cet exemple. Les services intermédiaires de ce chemin sont Banque1 et Banque2. En conséquence, ils doivent exécuter des fragments pour pouvoir router les données de l'Hôpital vers le service Inspecteur1. La figure 5.4 illustre les fragments de procédés exécutés respectivement par Banque1 et Banque2.

Le premier fragment nommé P^1_{B2} inclut deux activités (voir la figure 5.4). La première reçoit la donnée à router du service Hôpital (H) et la deuxième activité envoie la donnée reçue au service Banque1 (B1). Le procédé coopérant qui implémente le chemin au sein du service Banque1 est concrétisé par deux activités. La première reçoit la donnée routée de la part du service Banque2 qui constitue l'élément précédent sur ce chemin et la route vers le service Inspecteur1 (I1). Pour ce faire, l'identité du service Banque1 est utilisée pour l'instantiation de l'activité de connexion dans le procédé P^1_{B2} et respectivement l'identité du service Banque2 est utilisée pour l'instantiation de l'activité de connexion du procédé P^1_{B1}. La figure 5.6 illustre partiellement les procédés principaux exécutés par les services Hôpital et Inspecteur2. Dans ces procédés, nous attirons l'attention sur

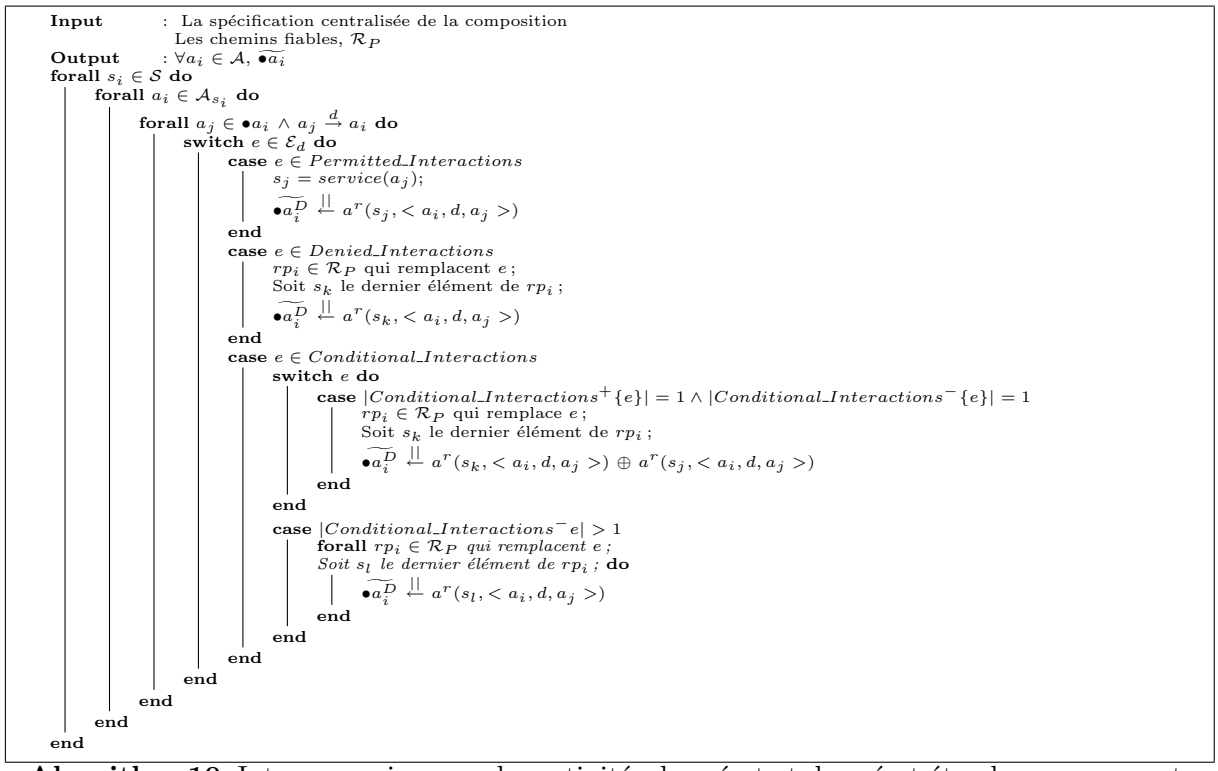
Input : La spécification centralisée de la composition
Les chemins fiables, $\mathcal{R}_P$
Output : $\forall a_i \in \mathcal{A}$, $\widetilde{\bullet a_i}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_j \in \bullet a_i \wedge a_j \xrightarrow{c} a_i$ **do**
      $e \in \mathcal{E}_c$ où $source_{act} = \{a_j\} \wedge target_{act} = \{a_i\}$;
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_j = service(a_j)$;
          $\widetilde{\bullet a_i^T} \overset{||}{\leftarrow} a^r(s_j, < a_i, exec >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_k le dernier élément de rp_i ;
          $\widetilde{\bullet a_i^T} \overset{||}{\leftarrow} a^r(s_k, < a_i, exec >)$
        end
      end
      case $e \in Conditional\_Interactions$
        switch e **do**
          case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
            $s_j = service(a_j)$;
            $rp_i \in \mathcal{R}_P$ qui remplace e ;
            Soit s_k le dernier élément de rp_i ;
            $\widetilde{\bullet a_i^T} \overset{||}{\leftarrow} a^r(s_j, < a_i, exec >) \oplus a^r(s_k, < a_i, exec >)$
          end
          case $|Conditional\_Interactions^- e| > 1$
            forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
              Soit s_k le dernier élément de rp_i ;
              $\widetilde{\bullet a_i^T} \overset{\oplus}{\leftarrow} a^r(s_k, < a_i, exec >)$;
            end
          end
        end
      end
    end
    forall $a_j \in \bullet a_i^F$ **do**
      $e \in \mathcal{E}_{ext}$ où $source_{act} = \{a_j\} \wedge target_{act} = \{a_i\}$;
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_j = service(a_j)$;
          $\widetilde{\bullet a_i^F} \overset{\oplus}{\leftarrow} a^r(s_j, < a_i, skip >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          Soit s_k le dernier élément de rp_i ;
          $\widetilde{\bullet a_i^F} \overset{\oplus}{\leftarrow} a^r(s_j, < a_i, skip >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^+\{e\}| = 1 \wedge |Conditional\_Interactions^-\{e\}| = 1$
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              Soit s_k le dernier élément de rp_i ;
              $\widetilde{\bullet a_i^F} \overset{\oplus}{\leftarrow} a^r(s_k, < a_i, skip >) \oplus a^r(s_j, < a_i, skip >)$
            end
            case $|Conditional\_Interactions^- e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                Soit s_k le dernier élément de rp_i ;
                $\widetilde{\bullet a_i^F} \overset{\oplus}{\leftarrow} a^r(s_k, < a_i, skip >)$
              end
            end
          end
        end
      end
    end
  end
end

Algorithm 8: Interconnexion avec les activités de préset et de préset étendue par rapport aux politiques du flux d'information (Première partie)

l'initialisation des activités de connexion qui permettent les interactions avec les services qui sont spécifiés dans les politiques du flux d'information. Il est à noter que les procédés coopérants exécutés respectivement par les services Banque2 *et* Banque1 *incluent les dépendances de données entre les activités de connexion qui caractérisent les données routées. Les activités de connexion échangent ces données comme la sortie et l'entrée de ces deux activités.*

Input : La spécification centralisée de la composition
Les chemins fiables, $\mathcal{R}_P$
Output : $\forall a_i \in \mathcal{A}$, $\widetilde{\bullet a_i}$
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    forall $a_k \in \bullet a_i^{D,F,a_j}$ *où* $a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i$ **do**
      forall $a_k \in \bullet a_j^{F}$ **do**
        $e \in \mathcal{E}_{ext}$ où $source_{act}(e) = a_k \wedge target_{act}(e) = a_i$;
        switch e **do**
          case $e \in Permitted\_Interactions$
            $s_k = service(a_k)$;
            $\widetilde{\bullet a_i^{D,F,a_j}} \overleftarrow{\oplus} a^r(s_k, < a_i, skip, d, a_j >)$
          end
          case $e \in Denied\_Interactions$
            $rp_i \in \mathcal{R}_P$ qui remplace e ;
            Soit s_k le dernier élément de rp_i ;
            $\widetilde{\bullet a_i^{D,F,a_j}} \overleftarrow{\oplus} a^r(s_k, < a_i, skip, d, a_j >)$
          end
          case $e \in Conditional\_Interactions$
            switch e **do**
              case $|Conditional\_Interactions^{+}\{e\}| = 1 \wedge |Conditional\_Interactions^{-}\{e\}| = 1$
                $rp_i \in \mathcal{R}_P$ qui remplace e ;
                Soit s_l le dernier élément de rp_i ;
                $s_k = service(a_k)$;
                $\widetilde{\bullet a_i^{D,F,a_j}} \overleftarrow{\oplus} a^r(s_k, < a_i, skip, d, a_j >) \oplus a^r(s_l, < a_i, skip, d, a_j >)$
              end
              case $|Conditional\_Interactions^{-}e| > 1$
                forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e **do**
                  Soit s_l le dernier élément de rp_i ;
                  $\widetilde{\bullet a_i^{D,F,a_j}} \overleftarrow{\oplus} a^r(s_l, < a_i, skip, d, a_j >)$
                end
              end
            end
          end
        end
      end
    end
    forall $a_k \in \bullet a_i^{D,T,a_j}$ *où* $a_j \in \bullet a_i \wedge a_j \xrightarrow{d} a_i$ **do**
      $e \in \mathcal{E}_{ext}$ où $source_{act}(e) = a_k \wedge target_{act}(e) = a_i$;
      switch e **do**
        case $e \in Permitted\_Interactions$
          $s_k = service(a_k)$;
          $s_j = service(a_j)$;
          $\widetilde{\bullet a_i^{D,T,a_j}} \overleftarrow{||} a^r(s_k, < a_i, exec, d, a_j >) + a^r(s_j, < a_i, d, a_j >)$
        end
        case $e \in Denied\_Interactions$
          $rp_i \in \mathcal{R}_P$ qui remplace e ;
          $rp_j \in \mathcal{R}_P$ qui remplace d si la dépendance de donnée est remplacée par un chemin ;
          Soit s_l le dernier élément de rp_i ;
          Soit s_m le dernier élément de rp_j ;
          $\widetilde{\bullet a_i^{D,T,a_j}} \overleftarrow{||} a^r(s_l, < a_i, exec, d, a_j >) + a^r(s_m, < a_i, d, a_j >)$
        end
        case $e \in Conditional\_Interactions$
          switch e **do**
            case $|Conditional\_Interactions^{+}\{e\}| = 1 \wedge |Conditional\_Interactions^{-}\{e\}| = 1$
              $rp_i \in \mathcal{R}_P$ qui remplace e ;
              $rp_j \in \mathcal{R}_P$ qui remplace d si la dépendance de donnée est remplacée par un chemin ;
              Soit s_l le dernier élément de rp_i ;
              Soit s_m le dernier élément de rp_j ;
              $\widetilde{\bullet a_i^{D,T,a_j}} \overleftarrow{||} (a^r(s_l, < a_i, exec, d, a_j >) + a^r(s_m, < a_i, d, a_j >)) \oplus (a^r(s_k, < a_i, exec, d, a_j >) + a^r(s_j, < a_i, d, a_j >))$
            end
            case $|Conditional\_Interactions^{-}e| > 1$
              forall $rp_i \in \mathcal{R}_P$ *qui remplacent* e *;*
              $rp_j \in \mathcal{R}_P$ *qui remplace* d *si la dépendance de donnée est remplacée par un chemin ;*
              do
                Soit s_l le dernier élément de rp_i ;
                Soit s_m le dernier élément de rp_j ;
                $\widetilde{\bullet a_i^{D,T,a_j}} \overleftarrow{||} (a^r(s_l, < a_i, exec, d, a_j >) + a^r(s_m, < a_i, d, a_j >))$
              end
            end
          end
        end
      end
    end
  end
end

Algorithm 9: Interconnexion avec les activités de préset et de préset étendue par rapport aux politiques du flux d'information (Deuxième partie)



Algorithm 10: Interconnexion avec les activités de préset et de préset étendue par rapport aux politiques du flux d'information (Troisième partie)

Input : $\mathcal{RP}$
Output: $\forall rp_j \in \mathcal{RP}$, $P^i_{s_i}$
forall s_i *de la composition* **do**
  forall $rp_j \in \mathcal{RP}$ **do**
    if $rp_j = < ..., s_{i-1}, s_i, s_{i+1}, ... >$ **then** $P^j_{s_i} \leftarrow a^r(s_{i-1}$, le message routé) + $a^w(s_{i+1}$, le message routé) ;
    else if $rp_j = < s_i, s_{i+1}, ... >$ **then**
      Intégration s_{i+1} dans $P^*_{s_i}$
    else if $rp_j = < ..., s_{i-1}, s_i >$ **then**
      Intégration s_{i-1} dans $P^*_{s_i}$
    end
  end
end

Algorithm 11: Production des fragments pour le routage des données à travers

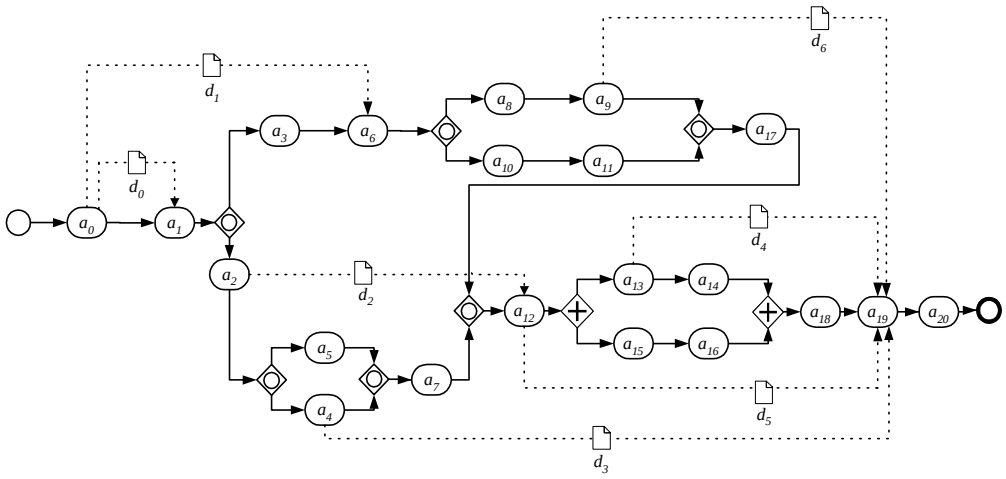


FIG. 5.3 – Exemple de motivation pour le contrôle du flux d'information

Police	can_out(out_1, op_9, I1)
Banque1	can_act_to(op_{14}, I2)
Banque2	can_act_to(op_{14}, B1)
Hôpital	can_out(out_1 (d_5), op_{12}, B2) can_out(out_1 (d_5), op_{12}, P) can_act_to(op_{12}, B1) can_act_to(op_{12}, B2)
Inspecteur1	can_in(in_1 (d_4), op_{19}, B1) can_in(in_2 (d_5), op_{19}, B1) can_act_from(op_{19}, I2)
Inspecteur2	can_act_to(op_{18}, I2) can_act_from(op_{18}, B1)

TAB. 5.7 – Politiques du flux d'information de l'exemple

can_route(d_5, Banque1)
can_route(d_5, Banque2)
can_route_to(d_5, Banque2, Banque1)
can_route_from(d_5, Banque1, Banque2)
can_route_to(d_5, Banque1, Inspecteur1)

TAB. 5.8 – Les politiques du flux d'information contextuelles de l'exemple

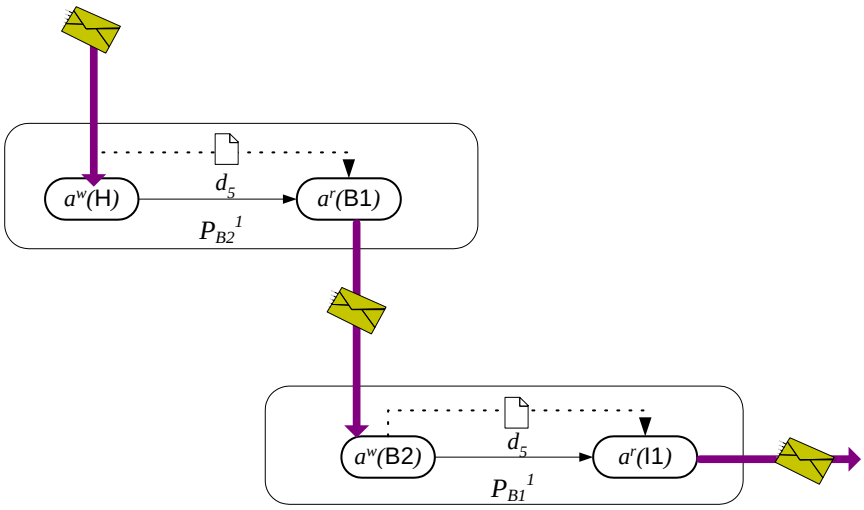


FIG. 5.4 – Les fragments coopérants qui permettent le routage de la sortie (P^1_{B1} et P^1_{B2})

La deuxième interaction qui ne peut pas être implémentée par une interaction directe entre les services sous-jacents est la dépendance de contrôle entre l'activité a_{16} du service Banque2 et l'activité a_{18} du service Inspecteur1. Cette dépendance peut être remplacée par un chemin passant par le service Banque1. Nous appelons ce chemin rp_2. Il constitue le deuxième chemin fiable de cette composition pour remplacer une dépendance. Il est à noter que dans ce cas, le message routé est un message de contrôle qui caractérise la terminaison de l'activé a_{14}. La figure 5.5 illustre les procédés exécutés par les services Banque2 et le service Banque1. Pour le service Banque2, la figure illustre le procédé coopérant principal $P^*_{Banque2}$ qui envoie l'information de terminaison de l'activité a_{16} au procédé $P^2_{Banque1}$ qui route ensuite le même message vers le procédé $P^2_{Inspecteur2}$.

5.7 Validation de la production des procédés coopérants

Dans cette section, nous nous penchons sur un aspect important de la décentralisation qui est la vérification de l'équivalence entre la spécification centralisée et la spécification décentralisée. La composition de services dans un procédé à partir de laquelle nous dérivons des procédés coopérants, est elle-même une question de vérification. La composition consiste à définir un modèle exécutable en combinant les interactions de composants indépendants. Les travaux liés qui portent sur la composition selon l'approche centralisée ainsi que l'approche décentralisée cherche à trouver une composition vérifiée : les interactions ne sont pas bloquées par rapport à l'histoire d'exécution de la composition ou encore il n'y a pas de boucle infinie etc [Pistore *et al.*2005] [Berardi *et al.*2005]. En effet, la dimension fonctionnelle de la composition porte essentiellement sur la vérification des interactions des composants. Les critères vérifiés sont typiquement ceux du *Global Soundness* que nous avons présenté dans le chapitre précédent. Notre travail ne concerne pas le problème de la composition d'un point de vue classique. Nous cherchons à créer des procédés qui peuvent coopérer en respectant les critères de global soundness. Comme nous l'avons déjà mentionné, le notion d'exactitude de notre approche nécessite une vérification.

Une solution au problème de vérification des procédés coopérants est la vérification a posteriori. Ce type de vérification est similaire à répéter le problème de composition en vue d'identifier

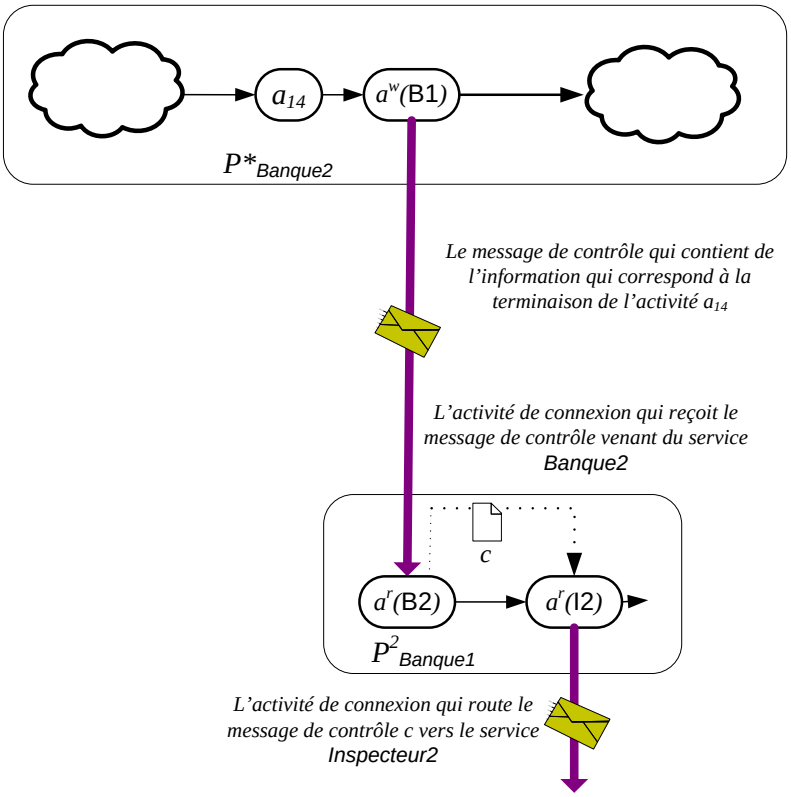


FIG. 5.5 – Les fragments coopérants $P^*_{Banque2}$ et $P^2_{Banque1}$

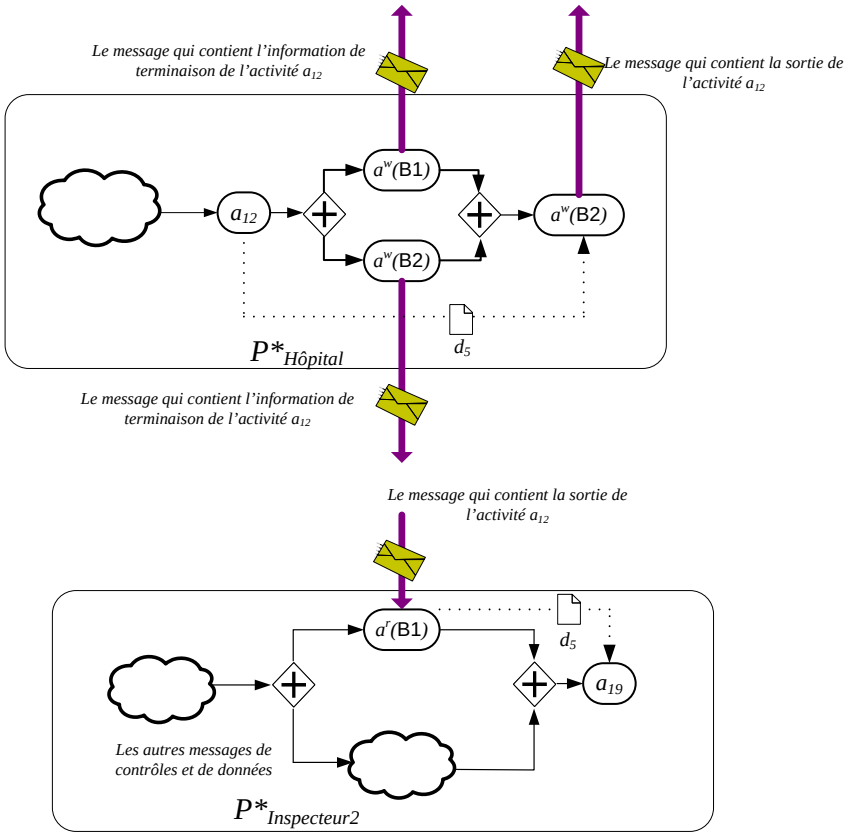


FIG. 5.6 – Les fragments coopérants des services Hôpital et Inspecteur2, ($P^*_{\text{Hôpital}}$ et $P^*_{\text{Inspecteur2}}$)

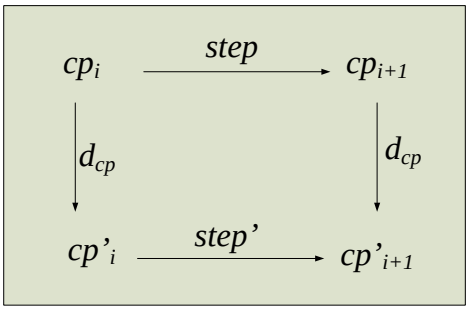


FIG. 5.7 – Les structures algèbriques et leur dépendances

les interactions qui peuvent causer des situations qui violent les critères d'exactitude. C'est une approche raisonnable qui peut être utilisée pour la vérification de notre méthodologie qui dérive les procédés coopérants. Pourtant, c'est une tâche qui est difficile. Parce que la même opération qui produit les procédés coopérants est effectuée pour chaque procédé de composition. La vérification de la cohérence des procédés coopérants doit être considérée pour l'opération qui les dérive. Ainsi, la vérification, même si elle est difficile, est effectuée une seule fois. Pour ce faire, nous faisons une vérification formelle des algorithmes développés en vue de la détermination de la relation d'équivalence entre la spécification centralisée et les procédés coopérants dérivés. Pour cela, nous nous penchons sur les définitions formelles et les algorithmes présentés dans le chapitre précédent. Dans la littérature, il y a plusieurs travaux qui portent sur la vérification de l'équivalence engendrée par un système de transformation [Wodtke and Weikum1997] [Harel and Naamad1996] [Emerson1990] [Leung and Reghbati1987]. Notre technique est inspirée de ceux-ci. Mais, elle est adaptée au formalisme que nous avons proposé. Premièrement, nous définissons une variable qui décrit le contexte d'exécution d'un procédé. L'ensemble des contextes d'exécution d'un procédé est nommé $\mathcal{CP}$. Chaque contexte du procédé $cp \in \mathcal{CP}$ décrit les valeurs des variables du procédé, l'état d'exécution des activités, les conditions de transitions évaluées. Pour le contexte du procédé, nous définissions une opération *step* qui transforme un contexte cp_i du procédé en son contexte suivant cp_{i+1}. Il est à noter qu'il peut y avoir plusieurs contextes suivants. Le notion de contexte existe aussi pour les procédés coopérants. Pourtant, dans ce deuxième cas, le contexte est dispersé entre les procédés coopérants au lieu d'appartenir à un seul procédé. L'ensemble des contextes des procédés coopérants est nommé $\mathcal{CP}'$. Chaque élément de $\mathcal{CP}'$ est nommé cp'. Nous utilisons une opération *step'* qui transforme un contexte $cp_i^{'}$ de $\mathcal{CP}'$ dans les contextes suivants $cp_{i+1}^{'}$. Les deux structures qui constituent respectivement l'entrée et la sortie de mécanisme proposé sont considérées comme des structures algébriques où le contexte est l'ensemble et l'opération *step* est une loi de composition associée. Entre les deux algèbres qui correspondent respectivement à la spécification centralisée et à ses procédés coopérants qui simulent son comportement, il est possible de trouver une dépendance d_{cp}. La relation d_{cp} est une fonction qui associe chaque contexte de la spécification centralisée à un contexte des procédés coopérants. Plus formellement, $d_{cp} : \mathcal{CP} \rightarrow \mathcal{CP}'$. L'association de fonction d_{cp} des contextes consiste à coupler les éléments respectifs des contextes.

La figure 5.7 illustre les relations entre les contextes, l'opération de transformation et la fonction qui associe les contextes. La première ligne de cette figure constitue l'algèbre qui correspond à la spécification centralisée alors que la deuxième ligne correspond à l'algèbre des procédés coopérants. Si les dépendances de la figure 5.7 sont commutatives alors la deuxième algèbre est l'homomorphisme de la première algèbre [Wodtke and Weikum1997] [Muth *et al.*1998]. Par

conséquent, nous pouvons dire que le comportement des procédés coopérants et le comportement de la spécification centralisée sont équivalents. Cela veut dire que quand la spécification centralisée dans un contexte, le même contexte est mise en œuvre par les procédés coopérants. On y trouve les mêmes variables, les mêmes conditions de transition, et les mêmes états d'exécution d'activités etc. Plus formellement, ce théorème est décrit ci-dessous.

Théorème 2 ***(L'homomorphisme entre la spécification centralisée et les procédés coopérants)***
La fonction d_{cp} qui associe chaque contexte cp_i de la spécification centralisée à un contexte cp'_i des procédés coopérants est un homomorphisme si la relation de commutativité suivante est vraie :

$$d_{cp}(step(cp_i)) = step'(d_{cp}(cp_i))$$

La preuve de ce théorème est dans l'annexe C.

5.8 Synthèse

Dans cette section, nous résumons les contributions de ce chapitre et nous présentons notre positionnement par rapport aux travaux liés.

Dans ce chapitre, nous avons présenté une approche de contrôle de flux d'information dans une composition de service. Le problème du contrôle du flux d'information est réduit à celui de la production de procédés coopérants qui permettent des interactions conformes aux politiques du flux d'information. Pour ce faire, l'approche présentée dans le chapitre précédent est adaptée aux exigences de la production des procédés coopérants qui doivent implémenter le flux d'information. En vue de la modélisation des politiques de flux d'information, nous avons proposé une spécification compréhensible qui regroupe les politiques qui peuvent exister dans un contexte de composition. Il faut tout de même dire que les politiques considérés ne reflètent pas la réalité du contexte à partir de laquelle ils explicitent les interactions qui doivent être implémentées. En particulier, il n'est pas toujours aussi évident de donner un énoncé initial d'un problème de définition de ce type de restrictions. À la mesure du foisonnement actuel de domaine, il n'est pas possible de modéliser tous les politiques qui existent dans un contexte de composition. Tout en étant complémentaires, chacun de ces types de politique peut devenir primordial en fonction de la classe du problème traité. L'exemple par excellence est la modélisation de la séparation des devoirs dans les systèmes de gestion de workflow. Ces restrictions peuvent consister à définir des simples restrictions qui spécifient l'association des acteurs aux activités tels que [Bertino *et al.*1999] [Atluri and Warner2005] et des restrictions qui consistent à spécifier le contexte temporel ou spatial de la séparation des devoirs [Joshi *et al.*2005] [Damiani *et al.*2007]. Même si les politiques que nous avons définies restent simples par rapport aux travaux relatifs, elles sont caractéristiques des interactions atomiques. D'autres types de politiques peuvent être exprimées en utilisant les politiques présentées dans ce chapitre. Enfin, le dernier mais non le moindre des aspects caractéristiques des modèles que nous avons abordés dans ce chapitre, concerne le fait qu'un modèle peut ou non supporter les politiques dynamiques. Les restrictions peuvent exprimer des cas de figures qui ne sont pas traitables avant l'exécution du procédé. La

gestion de restrictions peut être faite soit de façon statique à l'aide du traitement des politiques, soit de façon dynamique à l'aide d'une unité centralisée qui trace l'exécution des procédés. Nous avons seulement considéré les politiques qu'on peut traiter de façon statique et les politiques hybrides qui considèrent les attributs conditionnels. Ces derniers peuvent être planifiés de façon statique.

Du fait de l'enjeu économique que cela implique, le contrôle du flux d'information constitue un problème fondamental dans le cycle de vie d'un procédé métier qui spécifie la composition d'un ensemble de services. Ces problèmes se posent en général dès les premières investigations, mais peuvent aussi se poser dans des étapes très avancées du processus de conception du procédé, voire encore plus tard en phase d'exécution. Le traitement efficace des politiques permet de déterminer les meilleurs compromis pour trouver des exécutions faisables à partir de politiques qui peuvent être mutuellement contradictoires. Le contrôle du flux d'information est un sujet d'étude relativement nouveau dans le contexte des procédés métiers et la composition des services. Même si des exemples relatifs existent dans les pistes de recherche proches tels que la séparation des devoirs [Bertino *et al.*1999], les droits d'accès [Mecella *et al.*2006], les travaux qui portent sur le contrôle du flux d'information ne sont pas nombreux. Deux exemples principaux qui méritent d'être mentionnés sont [Atluri *et al.*2001] et [Chafle *et al.*2005]. Le premier consiste à implémenter des modules logiciels pour router l'instance du procédé d'un service à l'autre selon l'approche multi-agents. Le dernier qui est plus proche de l'approche présentée dans ce chapitre consiste à calculer tous les fragments coopérants d'un procédé et éliminer ceux qui ne respectent pas les restrictions exigées.

De par son caractère innovant, le contrôle du flux d'information selon cette approche permet une conception progressive d'une exécution décentralisée face aux restrictions principales qui gouvernent les interactions et facilite le passage d'une spécification à une exécution [Yildiz and Godart2007d] [Yildiz and Godart2007c] [Yildiz2008]. L'aide que nous apportons au concepteur et aux organisations qui implémentent les services n'est pas seulement de la séparation de la logique du procédé et les restrictions secondaires, mais aussi le fait de fournir une plate-forme dans laquelle ces restrictions sont traitées en vue de fournir une exécution faible.

Chapitre 6

L'implémentation du choix dynamique et décentralisé des services

Sommaire

6.1 Introduction

Notre vision de l'exécution décentralisée consiste à déployer des procédés coopérants vers les services composés afin d'établir des interactions d'égal-à-égal entre eux. La production des procédés coopérants est effectuée à partir d'une spécification centralisée de la composition. Il est à noter que le déploiement des procédés coopérants est fait par le service qui aurait été le coordinateur centralisé dans une exécution centralisée. Une des critiques que l'on peut faire à cette approche est la centralisation de l'opération de déploiement des procédés coopérants. Comme nous l'avons déjà souligné, un des critères d'efficacité permettant d'atteindre les objectifs d'une exécution décentralisée est également la décentralisation des fonctions telle que le déploiement des procédés coopérants. Permettre la décentralisation de l'opération de déploiement des procédés coopérants demande de répondre à plusieurs questions. Comment les services se choisissent-ils les uns les autres ? Comment les procédés coopérants sont déployés d'une manière décentralisée ? Quels sont les problèmes de synchronisation qui peuvent être rencontrés ?

Les contributions présentées dans les chapitres précédents portaient sur la production des procédés coopérants. Plus précisément, nous avons étudié les aspects liés à la structuration des activités de la spécification centralisée en procédés coopérant par l'implémentation des activités de connexion. Dans un premier temps, nous avons considéré la décentralisation d'un procédé centralisé sans tenir compte de restrictions particulières. Dans un deuxième temps, nous avons considéré les restrictions particulières telles que les politiques de flux d'information. La contribution de ce chapitre porte sur l'étude d'un autre aspect : nous ne nous intéressons pas à la production des procédés coopérants, mais nous étudions les problèmes liés à leur déploiement de façon dynamique. Le reste de ce chapitre est organisé de la manière suivante : la prochaine section détaille le notion du choix de dynamique et décentralisé et l'illustre à travers un exemple de motivation. Ensuite, nous discutons les problèmes liés à l'implémentation du choix dynamique et décentralisé des services tels que ceux liés à la synchronisation du choix concurrent. Nous développons une solution générique pour le choix dynamique et décentralisé de services. La contribution présentée dans ce chapitre n'est pas indépendante des contributions présentées dans les chapitres antérieurs. Elle considère une autre dimension de la décentralisation effectuée par des procédés coopérants.

6.2 Choix dynamique et décentralisé de services

Le choix dynamique des services est un avantage inhérent aux architectures orientées services où les concepteurs de procédés ont la possibilité de choisir un service lors de l'exécution de procédé. Au moment de la modélisation d'un procédé composé, on peut identifier les services à utiliser sans identifier l'instance précise du service. Dans ce cas, une instance de service qui va satisfaire la fonctionnalité doit être choisie au moment de l'invocation. Bien sûr, le choix dynamique nécessite des mécanismes qui maintiennent et mettent à jour un répertoire de services. Ainsi un service nécessaire peut être dynamiquement choisi à partir du répertoire qui contient la liste des services disponibles. L'implémentation du choix dynamique de services n'est pas difficile dans un système de gestion de procédé centralisé. Mais la complexité augmente considérablement dans un contexte décentralisé. Dans un contexte décentralisé où un service choisit dynamiquement un autre service et déploie le procédé coopérant correspondant, il est nécessaire de formaliser deux opérations pour empêcher les situations de blocage et de synchronisation que nous détaillons ci-dessous [Nanda and Karnik2004] [Yildiz and Godart2007e] [Yildiz and Godart2007b]. La complexité de la formalisation de choix dynamique est liée naturellement à la complexité de la spécification du procédé décentralisé lui même. Lors d'une exécution centralisée, le système de gestion de procédés est muni d'un module de choix dynamique de service qui effectue cette tâche en se basant sur plusieurs critères. L'approche est similaire à l'implémentation de la séparation des devoirs dans les systèmes de gestion de workflow [Bertino *et al.*1999] [Joshi *et al.*2005] [Damiani *et al.*2007]. Elle consiste à définir et à satisfaire des politiques qui spécifient les relations entre les utilisateurs et les activités d'un procédé. Ces relations peuvent être définies selon une approche dynamique. En outre, la dynamicité peut dépendre d'aspects temporels, conditionnels ou encore spatiaux. Dans un contexte décentralisé, nous considérons ce problème comme un problème de choix d'égal-à-égal des services. Ce point de vue est approprié au contexte considéré mais n'est pas traité dans les propositions existantes basées sur la séparation des devoirs. Il est à noter que nous ne discutons pas toutes les restrictions reliées à l'implémentation du choix dynamique et décentralisé des services. Nous limitons notre contribution aux cas principaux qui peuvent être reformulés pour les cas particuliers. Plus précisément, ces aspects, liés à l'implémentation des canaux de communications, sont discutés

dans la section synthèse.

6.2.1 Exemple de motivation et vue d'ensemble

Dans cette section, nous reprenons l'exemple de motivation du chapitre 2. Dans cet exemple, le service Assurance peut choisir les services composés statiquement avant de commencer l'exécution ou les choisir dynamiquement pendant l'exécution du procédé. Dans une exécution décentralisée implémentée avec des procédés coopérants, le même service choisit les services et déploie leur procédé respectif. Si le choix dynamique est exigé, le service Assurance doit effectuer les mêmes opérations d'une manière dynamique. Afin d'éviter le rôle centralisé du service Assurance, nous souhaitons implémenter le choix des autres services de façon dynamique et décentralisée. Par exemple, le service Police et/ou le service Hôpital peuvent être capable de choisir directement un service Inspecteur pour traiter les données qu'ils lui fournissent. De même, Inspecteur peut être capable de choisir un service Banque. Afin de permettre l'interaction du service choisi avec les autres services dépendants, l'opération de choix dynamique doit être suivie par le déploiement du procédé coopérant correspondant. La figure 6.1 illustre respectivement les dépendances de choix dynamique des services dans une exécution centralisée (a) et décentralisée (b). Les flèches vertes illustrent le service qui choisit et le service choisi.

Comme nous l'avons déjà souligné, l'implémentation décentralisée du choix dynamique nécessite un système de gestion pour traiter les dépendances de choix entre les services composés. Pour ce faire, nous analysons la spécification centralisée de composition afin d'en extraire les dépendances qui formalisent le choix dynamique. Il est à noter que le problème essentiel que nous cherchons à résoudre est l'identification des couples de services, constitués d'un service qui choisit l'autre. Nous abordons ce problème de façon générale sans nous baser sur un critère particulier, comme un aspect temporel ou un aspect conditionnel. Cette approche par défaut suit un principe simple qui consiste à choisir un service au moment de son invocation. Ainsi, un procédé de composition peut être défini avec des services génériques ne faisant pas référence à des instances précises de services. Lors de l'exécution, les services génériques sont remplacés par des instances invocables et les procédés coopérants sont déployés afin de permettre l'interaction du service choisi avec les autres services. Le choix d'un service au moment où il est utilisé est une approche par défaut. Un autre point de l'approche porte sur l'identification du service qui va choisir ce service. Intuitivement, on peut considérer que c'est le premier service qui s'exécute avant le service choisi qui est le service *sélecteur*. Il est important de souligner encore une fois que nous définissons les relations entre les services sans nous baser sur un critère précis qui gouverne l'opération de sélection. La figure 6.2 illustre les dépendances de sélection entre les services. Ces relations définissent un ordre partiel d'invocation des services sans prendre leur aspects conversationnels en compte. Cela veut dire que nous supposons que deux activités différentes n'invoquent pas le même service. Dans cette figure, nous soulignons deux aspects importants. Le premier est le cas des activités conversationnelles. Les activités conversationnelles peuvent nécessiter l'invocation du même service pour l'exécution d'activités différentes. Dans ce cas, si un service est choisi dynamiquement dans une conversation, il faut que la même instance du service choisit soit utilisé pour toutes ses activités conversationnelles. C'est le cas du service Livraison qui est le cible de deux dépendances. Le deuxième point important est le choix concurrent des services. Si l'ordre d'invocation des services est utilisé pour formaliser le choix dynamique, un service précédé par plusieurs services (par exemple le service Inspecteur) doit être choisi par l'ensemble des services qui exécutent ces activités dépendantes. C'est le cas du service Inspecteur qui est le cible de plusieurs dépendances de choix dynamique entrantes.

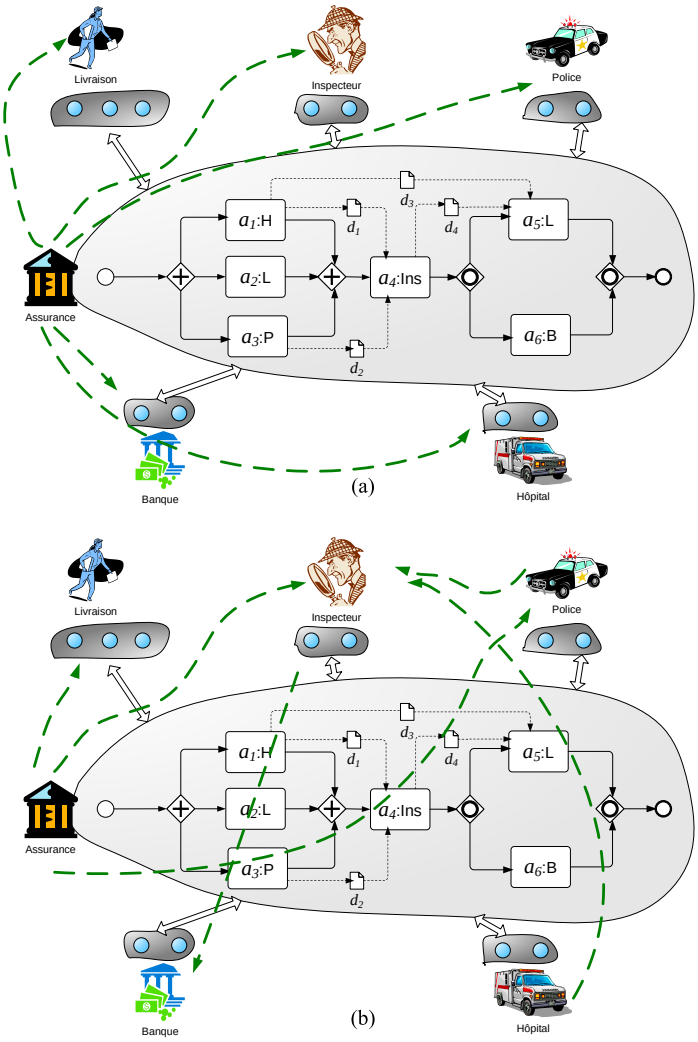


FIG. 6.1 – Le choix dynamique centralisé (a) et décentralisé (b)

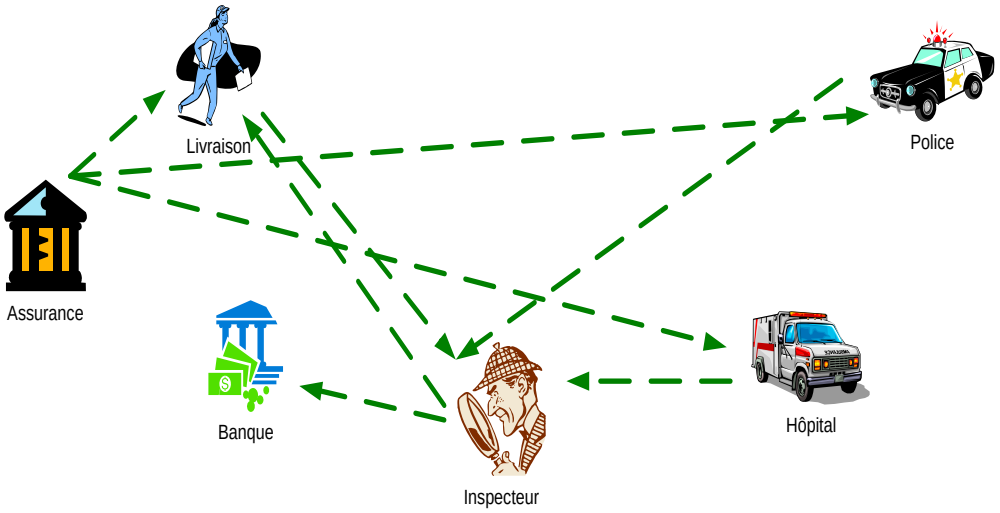


FIG. 6.2 – Les dépendances de choix dynamique et décentralisé

Nous avons associé l'opération du choix d'un service à celui qui le précède dans le flux de contrôle de la spécification centralisée. Dans une exécution décentralisée effectuée par des procédés coopérants, cet ordre de sélection doit être préservé. On peut facilement remarquer que la définition des dépendances de sélection entre les services peut causer des problèmes de choix concurrent des services. Cette situation de concurrence est liée à la complexité de la spécification centralisée du procédé de composition. Dans le reste du chapitre, nous étudions le choix dynamique et décentralisé des services en nous penchant sur les problèmes liés au choix concurrent de services. Notre approche formalise les concepts liés au choix dynamique et décentralisé de services et les solutions qui empêchent les situations de blocages. Nous terminons notre discussion par une analyse critique.

6.3 Aspects formels du choix dynamique et décentralisé

Dans cette section, nous formalisons le concept de choix dynamique et décentralisé des services d'une composition. Comme nous l'avons souligné, le choix dynamique et décentralisé définit une relation de dépendance entre deux services. Nous la formalisons ci-dessous.

Définition 1 ***(Dépendance de choix)***
Une dépendance de choix, d_{chs}, est un tuple $(s_i, s_j) \in \mathcal{S}\times\mathcal{S}$ où s_i est le service qui choisit l'instance du service s_j.

La définition ci-dessus considère les relations de choix entre les services. Pourtant, les services peuvent être invoqués par des activités. Ceci nécessite la considération des aspects conversationnels des activités pour l'identification des dépendances de choix. Pour ce faire, nous déléguons les dépendances de choix de services aux activités du procédé auquel elles appartiennent. Ci-dessous, nous donnons la définition formelle de choix dynamique projeté sur les activités.

Définition 2 ***(Dépendance de choix des activités)***
Une dépendance de choix d'activités, d_{cha}, est un tuple $(a_i, a_j) \in \mathcal{A} \times \mathcal{A}$ où $service(a_i) = s_i$ et $service(a_j) = s_j$ et s_i choisit l'instance du service s_j.

Les dépendances du procédé ne sont pas limitées à des dépendances de contrôle. Les dépendances de données sont également concernées par le choix dynamique et décentralisé de services. En effet, une activité qui fournit une donnée à une autre activité peut être exécutée par un service qui sera le premier dans l'ordre d'interaction avec le service invoqué par l'activité cible de la dépendance. Aussi un service peut faire l'objet du choix dynamique par n'importe quel service invoqué par les activités sources de ses dépendances entrantes. La multiplicité des dépendances entrantes est la raison essentielle du problème du choix concurrent de service.

6.3.1 Choix concurrent de services

Dans un contexte dans lequel les services choisissent dynamiquement d'autres services, le choix concurrent d'un service par plusieurs services risque de conduire à des instances différentes par les différent services. Dans une exécution décentralisée correcte, un service est choisi par un seul service et son identité propagée vers les autres services dépendants. La figure 6.2 illustre un exemple de choix concurrents où les services se choisissent les uns les autres suivant l'ordre d'invocation et un service qui est précédé par plusieurs services est choisi plusieurs fois. Si les instances choisies sont les mêmes, il n'y a pas de problème de synchronisation entre les services dépendants qui ont des interactions avec le service choisit. Pourtant, ceci n'est pas une hypothèse raisonnable dans le contexte de Web dans lequel on peut facilement trouver plusieurs services qui ont la même compétence. En conséquence, le choix de service doit être traité afin d'empêcher des situations de concurrence. La définition 3 formalise les services qui font l'objet du choix concurrent.

Définition 3 ***(Choix du service concurrent)***
Un service s_i fait l'objet du choix concurrent si $\exists\ a_i, a_j \in \mathcal{A}_{s_i}$ où $d_{cha}= (a_k, a_i)$ et $d'_{cha} = (a_l, a_j)$ avec $service(a_k) \neq service(a_l)$. De plus, a_i et a_j ne sont pas précédées par une autre activité conversationnelle. Plus formellement, $\nexists a_z \in \mathcal{A}_{s_i}$ tels que $a_z < a_i \wedge a_z < a_l$.

Informellement nous pouvons définir une situation de choix concurrent comme suit. La première activité conversationnelle qui invoque un service peut être précédée par plusieurs services. Les services qui sont susceptibles de choisir le service peuvent être dépendants de ce service par les dépendances de contrôle ou de donnée. Les services sélecteurs eux-mêmes peuvent être dépendants par le flux de contrôle ou de donnée. Afin d'empêcher les situations de concurrence dans lesquelles un service peut être choisi par plusieurs autres services, un unique service sélecteur doit être identifié pour effectuer le choix du service. La définition de ce service n'est pas une opération intuitive car celle-ci est liée à la complexité du procédé qui spécifie la composition. Dans le reste de cette section, nous développons deux solutions au choix concurrent des services. La première consiste à implémenter le choix dynamique et décentralisé d'une manière fiable où un service choisit un autre service et l'identité du service choisi est propagée vers les autres services dépendants pour permettre leurs interactions. La deuxième proposition porte sur la proposition

d'un mécanisme de choix dynamique qui assure la synchronisation avec le support du choix retardé de service. Avec cette approche, nous implémentons un système de synchronisation entre les services qui doivent choisir un service commun. Ainsi, un service peut être dynamiquement choisi avant son invocation.

6.3.2 Choix dynamique par défaut

Comme nous l'avons souligné ci-dessus, le choix dynamique par défaut consiste en l'identification d'un service de telle sorte que les autres services dépendants soient informés du choix effectué pour empêcher les problèmes liés à la synchronisation. Notre point de départ pour l'identification des services est naturellement la spécification centralisée du procédé. D'après la définition 3, les services invoqués par les activités qui ont plusieurs dépendances entrantes font l'objet du choix concurrent. Afin d'expliquer le choix concurrent, nous utilisons les sous-concepts suivants qui projettent les dépendances du procédé vers les dépendances du choix de service. Une dépendance de contrôle ou de donnée entre deux activités implique l'existence d'une dépendance de choix de service entre le service invoqué par l'activité cible et le service invoqué par l'activité source. Nous appelons cette dépendance *une dépendance minimale de choix* nommé $minbind(s_i, s_j)$. Afin d'identifier un service sélecteur s_i qui fera le choix d'un service s_j, nous traitons d'abord les activités conversationnelles. La première activité conversationnelle est traitée afin d'identifier si elle fait l'objet de choix concurrent ou pas. Si c'est le cas, nous traitons les activités qui cible celle-ci en considérant les services qui sont invoqués. Il est à noter qu'une unique activité conversationnelle qui précède toutes les autres activités dépendantes n'existe pas forcement. Si ce n'est pas le cas, un autre service, qui n'est pas forcement par une activité conversationnelle, peut être utilisé. Nous expliquons les cas principaux dans les exemples de motivations suivants.

Exemple 19 (Cas 1) *La figure 6.3 illustre trois activités qui invoquent trois services différents. Selon l'approche de l'exécution décentralisée, ces trois services exécutent trois procédés nommés P_{s_9}, $P_{s_{11}}$, $P_{s_{17}}$. Il est à noter que les activités originales de la spécification centralisée sont structurées dans ces procédés. Nous souhaitons implémenter le choix dynamique et décentralisé pour le service s_{17}. Cela veut dire que le service s_{17} doit être choisi au moment où il doit être utilisé. L'invocation du service s_{17} est faite après la terminaison des activités précédentes. Dans ce cas, les services qui exécutent les activités précédentes doivent choisir un service s_{17} et déployer $P_{s_{17}}$. Pourtant si s_9 et s_{11} choisissent un service en même temps, elles peuvent choisir un service différent. Dans ce cas, un seul service doit effectuer le choix du service s_{17}, de plus ce même service doit diffuser l'identité du service choisi vers les services dépendants.*

Exemple 20 (Cas 2) *Le figure 6.4 illustre trois activités qui invoquent trois services différents. Le service s_3 qui exécute a_6 ne peut pas être choisi avant les deux services qui exécutent respectivement a_1 et a_3. Même si ces services ne sont pas concurrents, la sélection de s_6 par les deux posent un problème de synchronisation. Par rapport au cas précédent, ce cas illustre le fait que les dépendances entrantes des activités séquentielles entre les activités peuvent causer une situation problématique. Cela est essentiellement lié à la présence des dépendances de donnée des activités précédentes.*

Exemple 21 (Cas 3) *La figure 6.5 illustre cinq activités qui invoquent quatre services différents. Nous supposons que les activités a_9 et a_{11} invoquent le même service s_3. Les activités conversationnelles qui invoquent le service s_3 sont concurrentes. Même si chacune de ces activités est précédée par une seule activité (a_8 et a_{10}), le service s_3 ne peut pas être choisi par les*

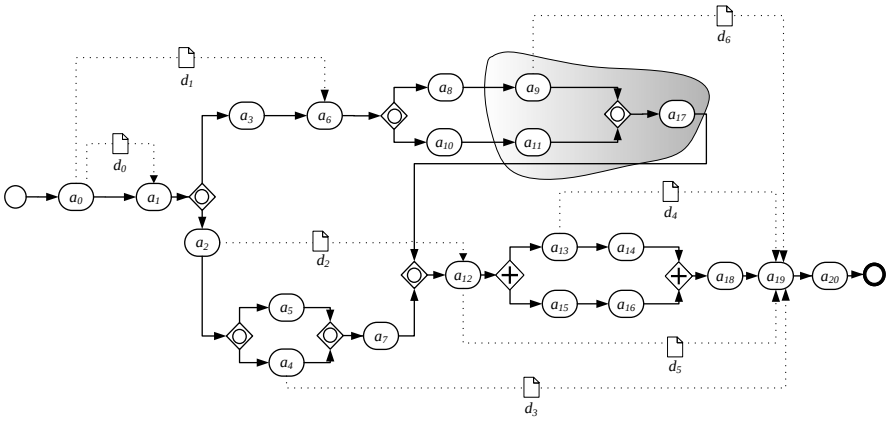


FIG. 6.3 – Cas 1 : La sélection du service s_{17} qui exécute a_{17}

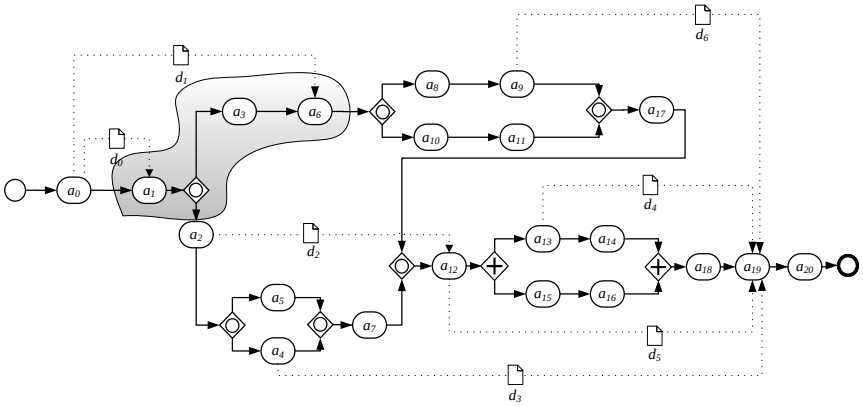


FIG. 6.4 – Cas 2 : La sélection du service s_3 qui exécute a_6

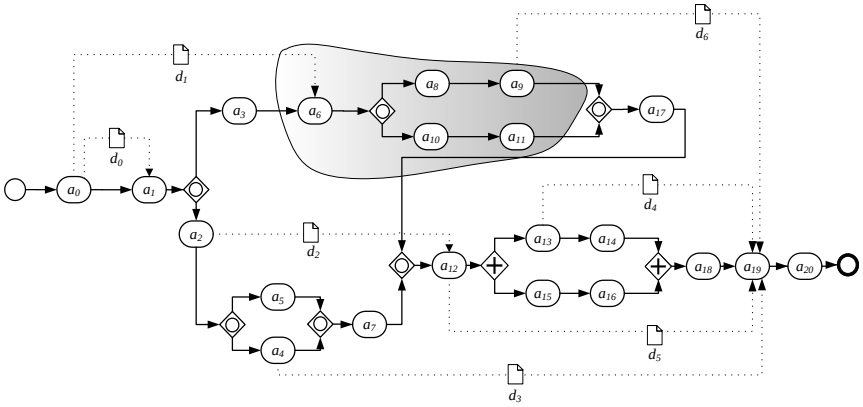


FIG. 6.5 – Cas 3 : La sélection du service s_3 invoqué par les activités a_9 et a_{11}

services qui exécutent ces deux activités en même temps car cela peut facilement causer un choix multiple.

Les trois exemples ci-dessus illustrent le choix dynamique et décentralisé par rapport à la spécification du procédé. Dans tous ces cas, la dépendance *minbind* doit être révisée avec les dépendances de choix appropriées afin d'empêcher les choix concurrents. Cette dépendance est caractérisée dans la spécification du procédé. Nous l'appelons *bind*. Ci-dessous nous discutons l'identification de celle-ci. La définition 4 donne la représentation formelle de la dépendance de type *bind*.

Définition 4 ***(La dépendance** bind**)***
Une dépendance de type *bind* est un uplet $(a_i, a_j) \in \mathcal{A}\times\mathcal{A}$ où $s_i = service(a_i)$ et $s_j = service(a_j)$. Le service s_i est l'unique service qui choisit le service s_j.

Comme la définition 4 le décrit, la dépendance *bind* exprime une relation entre deux activités. Le service invoqué par l'activité source de la dépendance est responsable du choix du service invoqué par l'activité cible. La question principale est l'identification des dépendances de choix à partir de la spécification centralisée. Il est à noter que le but ultime de ce traitement est l'identification d'un service unique pour le choix d'un service et la propagation de son identité vers les autres services dépendants.

L'algorithme 12 définit l'opération qui identifie les services sources des dépendances *bind*. Nous pouvons définir le choix du service source comme suit. Dans le cas des activités conversationnelles, le traitement est effectué pour la première activité des activités conversationnelles dans le flux de contrôle. Si une activité conversationnelle unique qui précède les autres n'existe pas ou elle existe mais fait l'objet d'un *minbind*, un service unique doit être sélectionné. Ce service est naturellement invoqué par l'activité qui précède l'activité(s) pour laquelle le service invoqué fait l'objet du choix concurrent. Le choix doit être effectué suffisamment tôt pour que l'identité du service puisse être propagé vers les services dépendants. Pour ce faire, nous avons développé un algorithme qui identifie le premier service qui précède les activités conversationnelles dans l'ordre partiel du procédé. La sélection des service et le déploiement des procédés coopérants sont effectués par ce service de telle sorte que les problèmes de synchronisation soient évités.

L'algorithme 12 définit le mécanisme qui identifie les couples de services sélecteur et choisi comme défini dans la définition 4. Il est à noter que nous avons imaginé les cas extrêmes dans lesquels chaque service composé est dynamiquement choisi.

Exemple 22 (Identification de service sélecteur) *La figure 6.6 illustre le procédé de l'exemple de motivation. Nous examinons le choix du service invoqué par l'exécution de l'activité a_{19}. Nous assumons que l'activité a_{19} est la première activité qui invoque le service concerné. Dans ce cas, le plus compliqué, chaque activité est exécutée par un service différent. Nous identifions les services avec les numéros que les activités qui les invoquent s_1, ..., s_{20}. Les relations de dépendance de choix existent entre les activités (a_4, a_{19}), (a_9, a_{19}), (a_{12}, a_{19}), (a_{13}, a_{19}) et (a_{18}, a_{19}). La structuration des activités sources des dépendances de choix correspond aux cas 3 que nous avons présenté ci-dessus. Dans ce cas, les activités cibles sont concurrentes. En conséquence, une activité qui précède ces activités doit être la source de la dépendance de choix. Par rapport à cette spécification, c'est l'activité a_1. C'est la première activité qui précède toutes les activités*

Input : La spécification du procédé
Output: Les relations *bind*
forall $s_i \in \mathcal{S}$ **do**
  forall $a_i \in \mathcal{A}_{s_i}$ **do**
    if $|\mathcal{A}_{s_i}|>1$ **then**
      if $\exists a_i, a_j \in \mathcal{A}_{s_i},\ a_i \not< a_j \wedge a_j \not< a_j$ **then**
        soit a_z la première activité qui précède $a_z<a_i \wedge a_z<a_j$
        $bind(a_z, a_i)$
        $bind(a_z, a_j)$
      else
        soit a_z, $minbind(a_z, a_i) \vee minbind(a_z, a_j)$
      end
    else
    end
  end
end

Algorithm 12: L'identification du service qui effectue la sélection qui fait l'objet de problème de synchronisation

sources des dépendances de choix. La figure 6.6 illustre la propagation de l'identité du service choisi vers les services concernés à partir du service qui fait la sélection. Bien évidemment, les dépendances liant les activités du procédé expriment les relations des services sous-jacents.

6.3.3 Déploiement des procédés coopérants à travers les dépendances de choix

Dans notre modèle d'exécution décentralisé, chaque service exécute un procédé coopérant. La composition d'un service au sein d'un procédé est suivie par le déploiement de ce procédé coopérant dans le procédé englobant. Dans les chapitres précédents, cette opération était effectuée par un service centralisé. Les procédés coopérants implémentaient les interactions d'égal-à-égal. Selon l'approche que nous présentons dans ce chapitre, cette opération est maintenant décen-

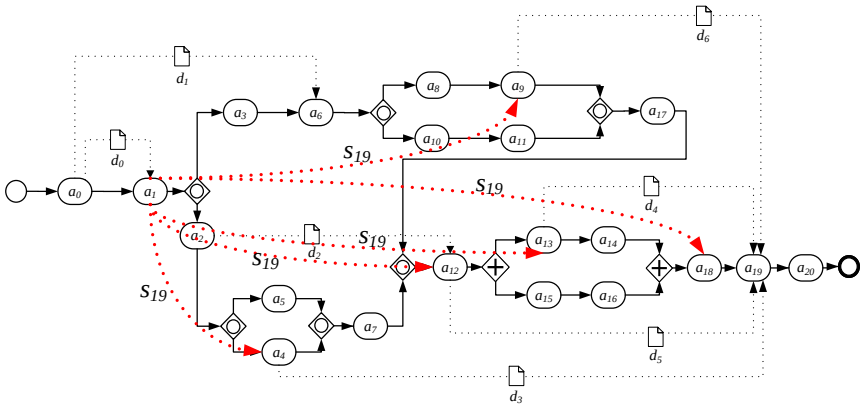


FIG. 6.6 – La choix du service s_{19} invoqué par l'activité a_{19}

tralisée. En conséquence, tous les procédés coopérants ne sont pas déployés par un seul service. Puisque le déploiement des procédés est codé au sein des procédés coopérants, un service doit être capable d'invoquer un autre service en déployant son procédé coopérant. Cette opération elle-même doit être inclue au procédé exécuté par le service sélecteur. On peut implémenter ce mécanisme en incluant un procédé coopérant dans un autre comme une variable de procédé. Ainsi,au moment du choix d'un service, son procédé coopérant peut être déployé par celui qui le choisit en exécutant l'opération de déploiement. Pour un service sélecteur, son procédé coopérant est en effet une série de procédés imbriqués qui constituent les procédés à déployer. La production des procédés coopérants n'est pas changée au niveau de la structuration des activités de connexion. Une opération qui trace les dépendances de déploiement est exécutée pour imbriquer les procédés coopérants. L'exemple 23 décrit une exécution de l'algorithme pour l'exemple de motivation introduit.

Exemple 23 (L'identification du service sélecteur et déploiement des procédés coopérants) *La figure 6.7 illustre un exemple de procédé. Dans ce procédé, nous nous intéressons au choix du service qui exécute l'activité a_{12}. Nous supposons que l'instance du service invoqué par cette activité n'est pas connue pendant la conception du procédé de composition. Nous appelons ce service s_x. Selon le principe de choix dynamique et décentralisé, ce service va être choisi par les autres services pendant l'exécution. Il est à noter que les procédés coopérants sont dérivés statiquement avant le début de l'exécution. La première approche de choix dynamique impose que le service s_x soit choisi avant son invocation de telle sorte que les problèmes de synchronisation soient évités. Par rapport à l'approche intuitive que nous avons résumée ci-dessus, le service s_x a des dépendances de choix avec d'autres services. Ces services sont associés avec les activités dépendantes de a_{12} par rapport aux dépendances entrantes. Nous les appelons respectivement s_2, s_7 et s_{17}. Il est à noter que ces services peuvent être dynamiquement choisis à leur tours. Les dépendances de choix sont notées $minbind(s_2, s_x)$, $minbind(s_7, s_x)$ et $minbind(s_{17}, s_x)$. Il est à noter que l'activité a_{12} est la première activité qui sera exécutée par le service s_x. En conséquence, une instance de service doit être choisie pour celui-ci. Les services s_2 et s_7 sont séquentiels. Donc si le choix est effectué par le service s_2, le service s_7 peut être informé de l'identité du service choisi pour s_x. Mais l'exécution de l'activité a_{17} par le service s_{17} est concurrente à ces deux services. L'algorithme 12 cherche un service commun à ces trois services concurrents pour identifier la source de la dépendance de choix dynamique. Le premier point de disjonction qui précède le service s_{17} est l'activité a_6. Mais, le service qui exécute cette activité ne précède pas les services qui exécutent les activités a_7 et a_2. Le service, nous l'appelons s_1, qui exécute l'activité a_1 précède les trois activités concurrentes. Cette activité est la dernière activité qui précède les activités par rapport au flux de contrôle (l'ordre partiel des activités).*

Le déploiement des procédés coopérants est effectué selon l'ordre d'implémentation du choix dynamique. Les procédés de cette composition sont nommé respectivement P_{s_1}, P_{s_2}, P_{s_3}, ..., $P_{s_{17}}$. Des dépendances de choix existent entre les activités a_1 et a_2, et, a_1 et a_3. Ceci veut dire que le service s_1 est responsable pour le choix de s_2 et s_3. Le déploiement des procédés coopérants P_{s_2} et P_{s_3} sont effectués par le service s_1 qui exécute P_{s_1}. En conséquence, le procédé P_{s_1} contient les procédés P_{s_2} et P_{s_3}. De même, le service s_1 est responsable pour le choix du service s_x qui est associé avec l'activité a_{17}. Cela veut dire que le procédé P_{s_x} est déployé par P_{s_1}. La figure 23 illustre les procédés coopérants et leur déploiement selon les dépendances du choix de services. La figure illustre également la propagation de l'identité du service choisi vers les services dépendant comme décrit dans l'exemple ci-dessus.

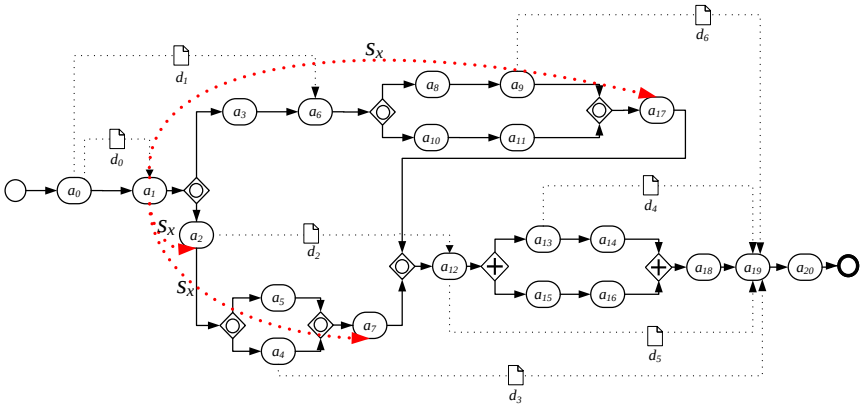
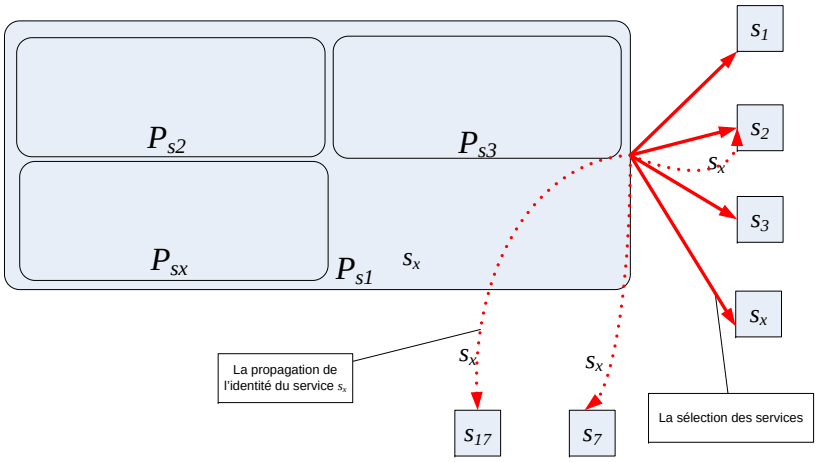


FIG. 6.7 – Le choix du service s_x qui exécute l'activité a_{12}



6.3.4 Extension de l'approche par des considérations d'implémentation

Le mécanisme de choix dynamique et décentralisé des services se penche sur un aspect particulier de l'exécution décentralisée d'un procédé. La technique que nous avons développée garantit la synchronisation entre les services dépendants par rapport à un service dynamiquement choisi. La limitation principale de cette approche est le choix de certains services bien avant leur invocation pour maîtriser les difficultés de synchronisation. Dans le reste de ce chapitre, nous développons une deuxième approche pour ce problème. Outre, cette limitation principale, plusieurs aspects du choix dynamique et décentralisé de services sont à considérer. L'implémentation d'une composition de service se base sur un ensemble de critères qui spécifient l'infrastructure sous-jacente. Dans le chapitre 4, nous avons judicieusement discuté ces critères. Typiquement, ils considèrent les canaux de communication implémentés par les services qui exécutent les activités de connexion qui reçoivent les données écrites dans les canaux de communication. Même si ce type d'implémentation est l'approche principale adoptée par les implémentations actuelles [Kazhamiakin *et al.*2006], elle n'est pas la seule. L'implémentation des canaux de communication peut se faire de plusieurs autres manières en révisant les critères d'exactitude d'une composition [Stricker *et al.*1995]. L'implémentation du canal de communication par le service qui émet les messages est éventuellement une approche à l'implémentation d'une composition. Dans ce cas, les données envoyées sont stockées par le service qui les envoie et le service qui doit les recevoir récupère de canal implémenté par le service expéditeur. Si la dernière est considérée pour l'implémentation du choix décentralisé de services comme nous l'avons présenté ci-dessus, le mécanisme n'est pas obligé de choisir un service qui précède tous les services dépendants. Une donnée à envoyer peut être stockée le canal implémenté par l'expéditeur et le service récepteur, quand il est choisi, peut venir chercher les données.

6.4 Choix dynamique et décentralisé retardé

Selon l'approche présentée dans la section précédente, un service est choisi dynamiquement avant son invocation. Il est à noter que la sélection du service est suivie par le déploiement du procédé coopérant que le service choisi doit exécuter. Cette approche a une limitation. Elle est liée à la solution adoptée pour empêcher les problèmes de synchronisation. En fait, un service n'est pas choisi juste avant son invocation, il est choisi à une étape de l'exécution qui permet de régler le problème de synchronisation. Afin de maîtriser cette difficulté, nous avons développé une deuxième approche pour le choix dynamique et décentralisé de services. Cette approche considère le choix retardé sans choisir le service dans une étape précédente. L'approche générale consiste à implémenter une technique de synchronisation explicite entre les services qui doivent être synchronisés pour le choix du service dépendant. La méthode implémentée assure la synchronisation des services dépendants par rapport au service choisi. Comme nous l'avons souligné ci-dessus, cette approche est basée sur une politique par défaut qui spécifie qu'un service doit être choisi avant son invocation. Dans des exécutions décentralisées plus sophistiquées, les concepteurs peuvent également définir des politiques additionnelles qui identifient des couples de services explicitement. Dans cette section, nous donnons les détails de l'approche du choix retardé des services. Dans ce cas encore, les services exécutent toujours des procédés coopérants déployés dynamiquement. Pourtant, une opération additionnelle qui permet l'échange des messages de synchronisation est implémentée au sein des procédés. Il est à noter que cette approche nécessite l'échange de données qui ne sont pas les données principales du procédé. Nous tenons à souligner que l'idée centrale est toujours basée sur les procédés coopérants qui implémentent ces échanges. Dans les travaux relatifs, les mêmes échanges ne sont pas implémentées au sein

des procédés mais plutôt par des protocoles explicites. L'exemple 24 suivant est similaire à ceux qui sont présentés ci-dessus. Nous l'utilisons pour donner une description informelle de la synchronisation des procédés coopérants par le choix retardé de services.

Exemple 24 (La synchronisation dans le choix retardé) *La figure 24 illustre un fragment qui contient six activités a_6, a_8, a_9, a_{10}, a_{11}, et a_{17} qui invoquent respectivement les services s_6, s_8, s_9, s_{10}, s_{11} et s_{17}. C'est un cas extrême dans lequel chaque activité est exécutée par un service différent. Nous considérons un autre cas extrême où tous ces services sont choisi dynamiquement. Dans ce cas, le service s_6 choisit s_8 et s_{10}, et par la suite, le service s_8 choisit s_9, et, le service s_{10} choisit le service s_{11}. Jusqu'ici le choix dynamique et décentralisé n'est pas changé. Mais, dans ce scénario, le service s_{17} doit être choisi dynamiquement, mais pas dans une étape antérieure. Comme la relation minbind existe entre s_{10} et s_{17}, et, s_{11} et s_{17}, si un de ces deux services choisit s_{17}, l'identité du service choisi doit être diffusé vers l'autre service dépendant. Le problème essentiel de cette approche est que les services qui doivent être synchronisés sont choisis dynamiquement. En plus, les services qui choisissent ces services sont aussi choisis dynamiquement. Dans ce cas, la synchronisation se complique considérablement. L'idée centrale de l'approche du choix retardé est la proposition d'une technique de synchronisation entre les services concernés.*

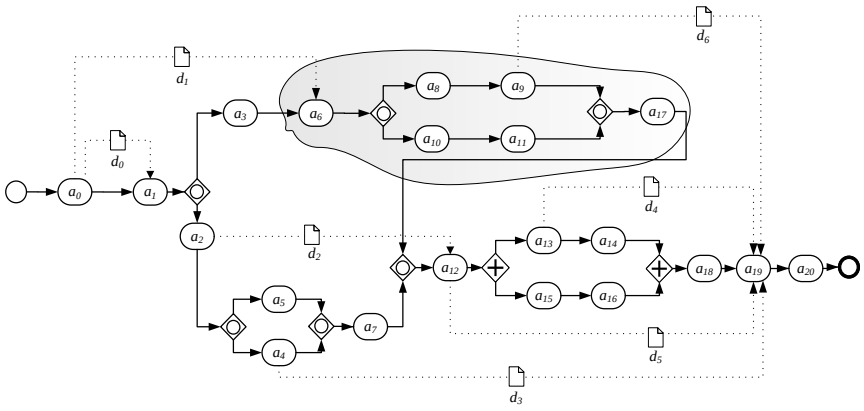


FIG. 6.8 – La synchronisation des services pour le choix dynamique des services

Pour formaliser le choix dynamique retardé, nous proposons les concepts suivants.

Définition 5 ***(Services concurrents)***
Un ensemble de services concurrents est un ensemble $\mathcal{E}_{\mathcal{S}} \subseteq \mathcal{S}$, $\mathcal{E}_{\mathcal{S}} = A \cup B$ t.q. $A = \{s_i \in \mathcal{S} |$ $a_i = activity(s_i)$, $\widehat{a_j} < a_i < a_j$ où a_j est l'activité invoquée par le service qui fait l'objet du choix concurrent$\}$ et $B = \{s_i \in \mathcal{S} |\ a_i = activity(s_i), \exists d \in \mathcal{E}_d, a_i = source(d) \wedge target(d) = a_j\}$.

Dans cette définition, nous rappelons qu'une activité nommée $\widehat{a_i}$ correspond au point de disjonction correspondant (resp. conjonction) d'un point de conjonction (resp. disjonction). En outre $\mathcal{E}_d$ est l'ensemble de dépendances de donnée. La définition 5 donne une caractérisation formelle des services qui sont concernés par le choix dynamique retardé d'un service qui fait l'objet d'un

choix concurrent. Ces services sont ceux qui exécutent les activités qui restent entre le point de conjonction qui précède l'activité en question et le point de disjonction (AND/OR-split) correspondant. Cet ensemble inclut également les services qui exécutent les activités qui sont les sources des dépendances de données entrantes. Le choix dynamique du service qui fait l'objet du choix concurrent ne nécessite pas une synchronisation complète entre toutes ces activités. Néanmoins, les services qui exécutent ces activités sont concernées par le processus de synchronisation des services qui ont des relations *minbind* avec ce dernier. La synchronisation se fait par l'échange de messages consécutifs entre les services qui appartiennent à l'ensemble de services concurrents. Il n'est pas souhaitable que tous les services de l'ensemble $\mathcal{E}_{\mathcal{S}}$ qui correspond au service choisis soient synchronisés ensemble. Pour chaque service qui fait partie de cet ensemble, nous avons identifié un ensemble de services qui sont les activités cibles des dépendances *minbind*. Notre technique consiste à échanger les identités des services choisis à travers les éléments de $\mathcal{E}_{\mathcal{S}}$. L'exemple 25 suivant détaille cette opération pour l'exemple que nous avons utilisé ci-dessus.

Exemple 25 (La synchronisation des services) *Le fragment présenté dans la figure 6.9 illustre la sélection dynamique des services et la diffusion d'identités par rapport à des dépendances de synchronisation. Les dépendances rayées en vert illustrent les dépendances de bind entre les services qui exécutent les activités sources et cibles. Quant aux dépendances rouges pointillées, elles illustrent comment les identités des services choisis sont propagées à travers les éléments de l'ensemble de services concurrents. Il est à noter que les dépendances qui relient les activités sont une projection des interactions des services sous-jacents invoqués avec l'exécution de celles-ci.*

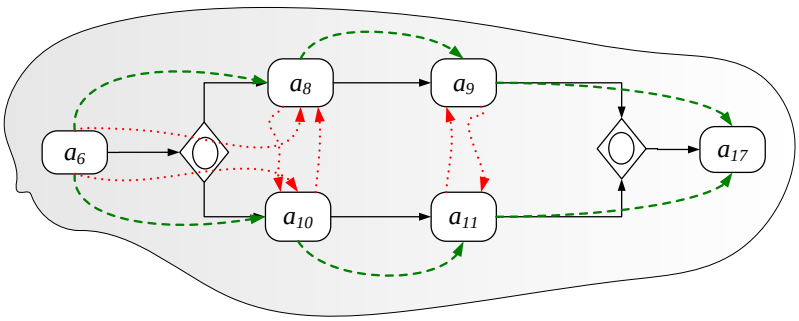


FIG. 6.9 – La synchronisation mutuelle des services dynamiquement choisis

Nous soulignons que les services qui exécutent ces activités ont les mêmes identités. Le service s_6 choisit les services s_8 et s_{10} pour l'exécution. En même temps, l'identité du service s_8 est transmise au service s_{10}, et, l'identité du service s_{10} est transmise vers le service s_8. Ainsi, les services s_8 et s_{10} peuvent se synchroniser dans l'étape suivante. À son tour, le service s_{10} choisit s_{11}, et, le service s_8 choisit s_9. Il est à noter que les services s_8 et s_{10} se "connaissent" mutuellement. Le processus de synchronisation continue avec les échanges d'identité des services suivants. Ainsi, l'identité du service s_{11} est diffusé vers s_8, et, l'identité du service s_9 est diffusé vers s_{10}. Par la suite, c'est le tour des services s_9 et s_{11} de se synchroniser.

L'exemple 25 a illustré les couples de services qui doivent se synchroniser pour la sélection du service qui exécutent l'activité a_{17}. Chaque service qui est impliqué dans la sélection du

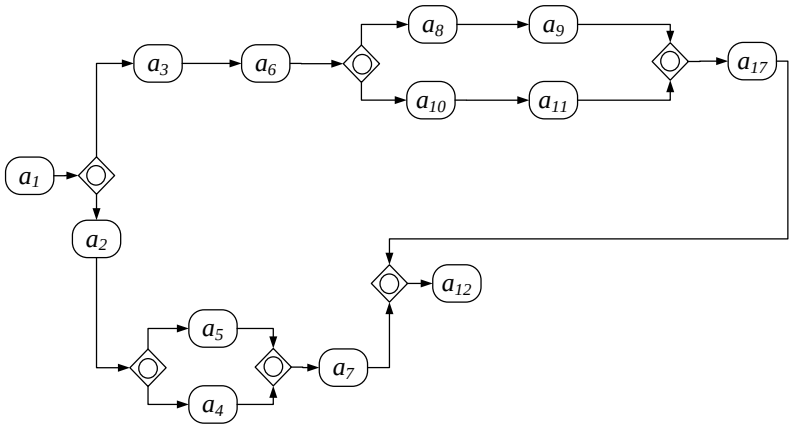


FIG. 6.10 – Le choix retardé du service s_{12} invoqué par l'activité a_{12}

service cible doit échanger les identités des services qu'il choisit avec les services concernés de l'ensemble des services concurrents. Outre les couples de services qui échangent des messages de synchronisation, les services qui choisissent le service qui exécute a_{17} doivent explicitement être synchronisés pour la sélection de ce dernier. En partant de ces principes, nous proposons les concepts suivants pour formaliser les interactions des services concernés.

Définition 6 ***(Services synchrones)***
Soit a_z une activité pour laquelle un service est dynamiquement choisi. Un ensemble de services $\mathcal{S}_s = \{s_i, ..., s_j\}$ est dit synchrone pour un ensemble d'activités $\mathcal{A}_s$ si $\forall a_i, a_j \in \mathcal{A}_s$, $a_i \not< a_j \wedge a_j \not< a_i$, $nb_{ci} = nb_{cj}$ où nb_{ci} et nb_{cj} sont les nombres de dépendances de contrôle depuis le point de disjonction qui précède les activités de $\mathcal{S}_s$. $\mathcal{S}_s$ inclut également les activités sources des dépendances de données qui ciblent l'activité du service dynamiquement choisi et les dernières activités qui précèdent a_z. Plus formellement, $\mathcal{S}_s^d = \{a_k \in \mathcal{A} |\ (\exists d \in \mathcal{E}_d,\ source(d) = a_k \wedge target(d) = a_z) \vee (\exists c \in \mathcal{E}_c,\ source(d) = a_k \wedge target(d) = a_z)\}$

La définition 6 illustre les tuples de services qui doivent échanger des messages de synchronisation qui incluent les identités des services qu'ils choisissent. Nous donnons un exemple de motivation pour les services synchrones dans l'exemple 26. Il est à noter que les services qui peuvent être choisis sans procéder à un processus de synchronisation sont ceux qui ne font pas l'objet de choix concurrent. L'identification des couples de services synchrones se fait statiquement pendant l'étape de conception du procédé pendant laquelle les concepteurs définissent également les politiques du flux d'information.

Exemple 26 (Identification et échanges des services synchrones) *La figure 6.10 illustre un fragment de procédé simplifié que nous avons précédemment utilisé. Il s'agit du fragment qui commence avec l'activité a_1 et se termine a_{12}.*

Nous avons exclu les dépendances de données. Le service qui doit être dynamiquement choisi est s_{12}. Nous allons l'utiliser pour montrer les échanges de messages entre les services. Nous imaginons le cas, le plus compliqué, dans lequel chaque activité invoque un service différent. Les

services invoqués ont les mêmes identités que les activités. En conséquence, les services sont nommés s_1, ... , s_{17}. *Le service* s_{12} *est celui qui doit être dynamiquement choisi par les services* s_7 *et* s_{17}. *Le point de disjonction correspondant qui précède* a_{12} *est l'activité* a_1. *Le service* s_1 *invoqué par* a_1 *choisit dynamiquement les services qui exécutent respectivement* s_2 *et* s_3 *qui invoquent respectivement* a_2 *et* a_3. *Les services* s_2 *et* s_3 *constituent le premier tuple de services synchrones, ce dernier nommé* $cs_1 = \{s_2, s_3\}$. *Les éléments de l'ensemble de servies synchrones doivent se connaître pour échanger les identités des services qu'ils choisissent. La figure 6.11 illustre cette opération de synchronisation. Le service* s_1 *passe l'identité* s_2 *au service* s_3, *et de même, passe l'identité du service* s_3 *au service* s_2. *Nous soulignons aussi que les interactions illustrées dans cette figure se réfèrent aux interactions des services sous-jacents invoqués par ces activités.*

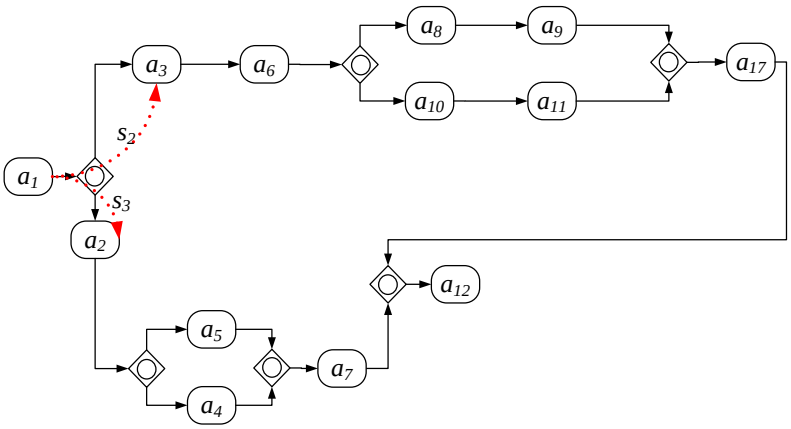


FIG. 6.11 – La synchronisation des services s_2 et s_3

Le service s_2 *choisi les services* s_4 *et* s_5. *De même, le service* s_3 *choisit* s_6. *Ces trois services* s_4, s_5 *et* s_6 *constituent le deuxième tuple de services synchrones,* $cp_2 = \{s_4, s_5, s_6\}$. *Afin de permettre la synchronisation des services choisis dans l'étape prochaine du processus de synchronisation, les services* s_2 *et* s_3 *échangent les identités des services choisis et propagent les identités vers les services concernés. La figure 6.12 illustre ces interactions sous forme de dépendances qui lient les activités. Les interactions sont examinées en deux étapes. La première interaction a lieu entre les services* s_2 *et* s_3 *pour échanger les identités des servies qu'ils choisissent. Dans la deuxième étape, les identités reçues sont propagées vers les services qui sont choisis par le service même.*

À leur tour, les services s_4 *et* s_5 *choisissent le service* s_7 *par le processus de synchronisation. Le service* s_6 *choisit les services* s_8 *et* s_{10} *par rapport à ses dépendances de contrôles sortantes. Ainsi, le troisième tuple de service synchrones* $cp_3 = \{s_7, s_8, s_{10}\}$. *Les échanges de synchronisation de ces servies sont illustrés dans la figure 6.13. Les services* s_4, s_5 *et* s_6 *qui connaissent les identités échangent les identités des services qu'ils choisissent. Par la suite, les identités échangées sont propagées vers les services choisis dans l'étape précédente. Ainsi, le service* s_6 *envoie l'identité de* s_8 *et* s_{10} *vers ses services synchrones qui sont* s_4 *et* s_5. *Ces derniers envoient l'identité du service* s_7 *vers* s_6. *Il est à noter que les services* s_4 *et* s_5 *choisissent le service* s_7. *En conséquence, c'est seulement le service* s_4 *qui envoie l'identité du service* s_7 *vers* s_6. *Par la suite, le service* s_8 *choisit le service* s_9 *et le service* s_{10} *choisit* s_{11}. *Ces deux services choisis constituent le tuple* cp_4 *avec le service* s_7, $cp_4 = \{s_7, s_9, s_{11}\}$. *La figure 6.14 illustre cette étape*

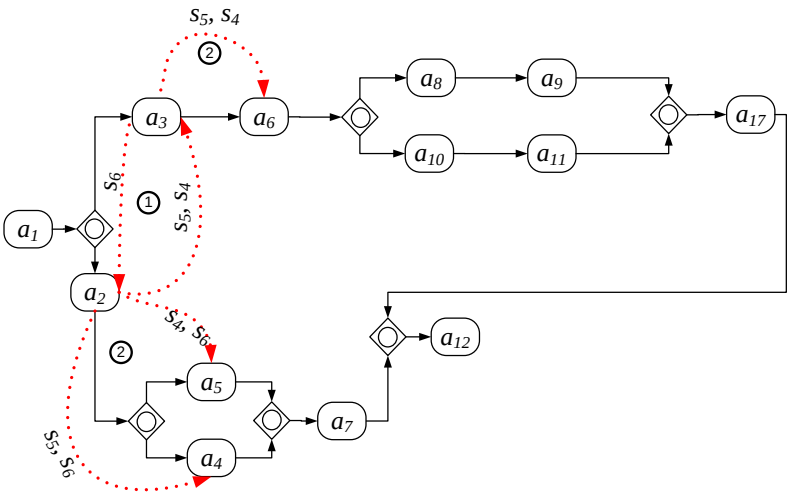


FIG. 6.12 – La synchronisation des services s_4, s_5 et s_6

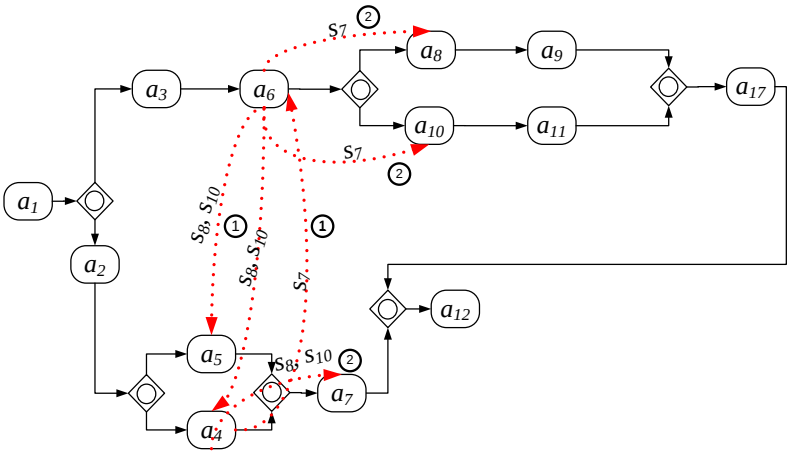


FIG. 6.13 – La synchronisation des services s_4, s_5 et s_6

du processus de synchronisation. Le service s_8 envoie l'identité du service s_9 vers le service s_7. De même, le service s_{10} envoie l'identité du service s_{11}. L'identité du service s_7 est transférée vers le service s_9 et s_{11}.

Le service s_7 est inclu dans cet ensemble comme il est un des services qui est responsable du choix du service s_{12}. Les services s_9 et s_{11} constituent un autre ensemble de services synchrones pour le choix du service s_{17}. Le dernier ensemble de services synchrones $cp_5 = \{s_7, s_{17}\}$. La figure 6.15 illustre les dépendances qui caractérisent les interactions des services s_7, s_9 et s_{11} pour échanger les identités du service s_7 avec le service s_{17} et l'identité du service s_{17} avec le service s_7.

Il est à noter que les services s_7 et s_{17} sont les services responsables pour le choix du service s_{12}. Le processus de synchronisation a pour but d'échanger les identités des services qui précédent un service par rapport aux dépendances de sélection. Les échanges intermédiaires ne concernent pas la sélection du service s_{12} directement. Leur but est de permettre l'échange des identités des

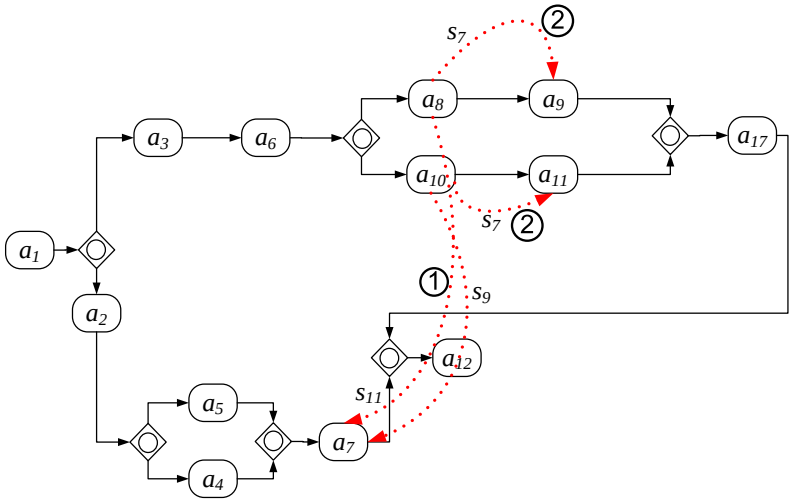


FIG. 6.14 – La synchronisation des services s_7, s_8 et s_{10}

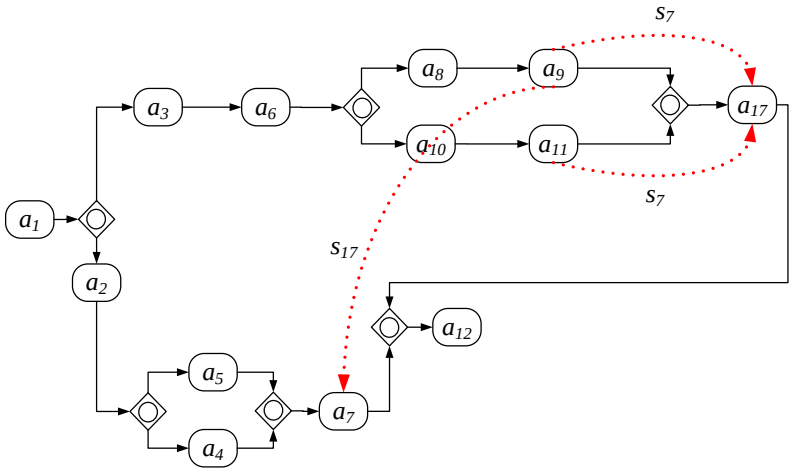


FIG. 6.15 – La synchronisation des services s_7, s_9 et s_{11}

services intermédiaires qui précédent s_{12}.

6.4.1 Synchronisation explicite de services

La deuxième approche qui porte sur le choix dynamique et décentralisé de services assure la *connaissance* des derniers services les uns avec les autres pour le choix du service concerné. Pourtant, le mécanisme n'impose pas une synchronisation explicite sur l'instance précise du service à choisir. Dans l'exemple 26, le processus de synchronisation suppose que les servies qui exécutent les activités a_7, a_9 et a_{11} aient les identités des uns des autres pour pouvoir faire les échanges nécessaires pour le choix du service s_{12} qui exécutera l'activité a_{12}. À ce point de l'exécution, la synchronisation de ces trois services n'est pas une opération intuitive. Ce problème peut être résolu de plusieurs autres manières. Nous avons implémenté une solution simple pour maîtriser ce problème. Le service qui a la plus petite identité est responsable pour la sélection

et la propagation de l'identité du service concerné aux autres services synchrones. Ce service est également responsable pour le déploiement du procédé coopérant du service choisi.

6.5 Implémentation du choix dynamique et décentralisé dans WS-BPEL

Dans les chapitres précédents, nous avons souligné que l'exécution décentralisée implémentée par les procédés coopérants était motivée essentiellement pas les efforts de standardisation qui permettaient l'échange des spécification standardisées telles que WS-BPEL. Une des raisons qui a motivé l'extension de la même approche vers le choix dynamique et décentralisé des services est certainement la facilité de l'implémentation de l'approche en utilisant WS-BPEL [Schuler *et al.*2004] [Pautasso and Alonso2005]. Ce détail technique est important pour expliquer l'intérêt du choix dynamique. La figure 6.16 illustre un exemple de code qui porte sur l'encodage de choix dynamique en WS-BPEL. L'exemple pris de [Pautasso and Alonso2005] illustre un exemple d'implémentation.

```
<process
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
name="AsyncEchoService" targetNamespace="urn:asyncEcho:Service"
xmlns:this="urn:asyncEcho:Service" suppressJoinFailure="no"
enableInstanceCompensation="no" abstractProcess="no">
<partnerLinks>
< name="caller" partnerLinkType="this:EchoPLT"
myRole="service" partnerRole="client"/>
</partnerLinks>
<variables>
< name="echoMessage" messageType="this:EchoMessage"/>
</variables>
<sequence>
<receive name="echoReceive" partnerLink="caller"
portType="this:EchoService" operation="echo"
variable="echoMessage" createInstance="yes"/>
<assign>
<copy>
<from variable="echoMessage" part="replyTo"/>
<to partnerLink="caller"/>
</copy>
</assign>
<invoke name="echoReply" partnerLink="caller"
portType="this:EchoClient" operation="echoCallback"
inputVariable="echoMessage"/>
</sequence>
```

Fig. 6.16 – Implémentation du choix dynamique dans WS-BPEL

En effet la dynamicité supportée par WS-BPEL consiste à manipuler les `end-points` des services en faisant un changement de variable classique. Les `end-points` peuvent être identifiés en utilisant le standard WS-Adressing. L'activité `receive` reçoit une donnée qui contient le `end-point` d'un service. Cette donnée remplace le end-point du service invoqué par l'activité `invoke` prochain. Le remplacement se fait par l'exécution de l'activité `assign`.

6.6 Synthèse

L'approche de l'exécution décentralisée présentée dans les deux chapitres précédents détaille la production des procédés coopérants à partir d'une spécification centralisée. Par rapport à

ces contributions qui considèrent le déploiement centralisé et statique des procédés, l'approche qui a été présentée dans ce chapitre permet l'exploitation de techniques de composition dynamique de services. L'expérimentation des contributions précédentes nous a permis de traiter des exemples qui n'auraient pas pu être directement implémentées avec l'approche initiale qui considère le déploiement centralisé. Cependant, les problèmes identifiés ne font pas appel à des techniques spécifiques à la production des procédés coopérants. On peut donc envisager d'étendre une telle démarche à d'autres dimensions. Dans ce chapitre, nous nous sommes attachés à identifier les besoins en termes de dynamicité de compositions décentralisées en nous positionnant par rapport au modèle de référence d'une part, et à l'étude de standards ou d'outils existants d'autre part. L'objectif de ce travail étant l'implémentation d'exécutions décentralisées destinées à l'évaluation d'une méthodologie ou à sa démonstration, nous avons proposé deux approches à la composition dynamique et décentralisée. Les méthodologies et les environnements de décentralisation ont pour objectif d'établir les interactions entre les services composés. Cependant, ces environnements se focalisent sur la description de solutions d'interactions et n'offrent pas de proposition satisfaisante du point de vue de la dynamicité. L'implémentation est donc réduite à tester la faisabilité d'interactions au moyen de méthodes non exhaustives basées sur des jeux de tests. L'implémentation d'une solution exprimée dans les formalismes habituellement proposés entraîne souvent des dérives importantes liées à une mauvaise interprétation des exigences initiales en termes de décentralisation. L'implémentation du choix dynamique et décentralisée présente une solution pratique dans le contexte AOS. De plus, cette technique permet d'utiliser la spécification produite dans la phase d'exécution. Cependant, leur complexité de mise en œuvre est un frein qui empêche actuellement leur utilisation de manière généralisée.

Notre approche considère deux méthodes de choix dynamique et décentralisé. En partant des exigences intuitives des procédés métiers, nous avons une première approche qui supporte l'exécution décentralisée du procédé par le choix dynamique et décentralisé de services. Dans ce mode d'exécution, un service est choisi pour le procédé avant sa première utilisation lors de l'exécution. Ce mode de choix engendre des problèmes de sélection concurrente et éventuellement des problèmes de synchronisation entre les éléments choisis d'une manière concurrente. Nous avons développé une méthodologie de sélection dynamique et décentralisée qui empêche les problèmes liés au choix concurrent en permettant à un service d'être choisi avant son exécution. Cette méthodologie consiste à identifier des couples de services dont l'un choisit l'autre. Une limitation de cette approche est que la sélection n'est pas toujours faite avant l'invocation du service afin d'empêcher les problèmes de synchronisation. Dans ce contexte, nous avons développé une deuxième approche qui maîtrise la limitation introduite par celle-ci. Avec la deuxième approche, un service est choisi juste avant son exécution grâce à l'échange des messages de synchronisation entre les services qui le précédent sur les chemins allant vers celui-ci. Les opérations de synchronisation entre les services sont implémentées comme des fragments de procédés concurrents aux procédés coopérants principaux exécutés par les services. Dans les deux cas, nous nous plaçons dans une optique de choix dynamique et décentralisé de services [Yildiz and Godart2007f] [Yildiz and Godart2007e] [Yildiz and Godart2007b]. La méthodologie de déploiement dynamique et décentralisé exploite les possibilités de l'approche "exécution décentralisée" par des procédés coopérants dans un contexte plus applicatif en maîtrisant les difficultés liées. Cependant, il existe un problème lié à l'approche dès qu'il s'agit de définir des restrictions de déploiement sophistiquées qui manipulent de gros procédés : complexité des modèles, structuration des mémoires de connexion, difficulté d'interprétation des restrictions etc.

Chapitre 7

Mise en œuvre

Sommaire

7.1 Introduction

Dans ce chapitre, nous présentons la plate-forme d'implémentation des contributions présentées dans les chapitres antérieurs. L'idée clé des exécutions décentralisées est la possibilité d'échangeabilité d'une spécification de procédé entre les différentes organisations. Dans un premier temps, cette approche nécessite d'une plate-forme qui permet l'échangeabilité des spécifications. Les structures de données et les algorithmes présentés précédemment précisent les détails de la production des procédés coopérants d'une manière générique. Dans un deuxième temps, cette partie de notre travail nécessite une implémentation concrète pour montrer la faisabilité de l'approche. En conséquence, la validation logicielle de nos contributions peut être examinée en deux partie indépendantes mais complémentaires.

7.2 L'implémentation de l'échangeabilité des procédés coopérants

Le déploiement d'un procédé vers un service par un autre service ou d'un client nécessite la présence d'un moteur d'exécution du procédé implémenté par le service qui reçoit et qui exécutera le procédé. Dans le contexte des applications modernes, la présence d'un moteur d'exécution implémenté au delà de l'interface du service est une hypothèse assez raisonnable. Le moteur d'exécution peut instancier un procédé standard tel que WS-BPEL ou bien le convertir en un autre type de procédé qu'il peut exécuter.

La figure 7.1 illustre une vue d'ensemble de l'implémentation d'un service qui peut supporter notre approche dans sa généralité. Ce module est nommé SDS (**S**ervice with **D**ecentralized **S**upport). Comme nous l'avons indiqué dans les chapitres précédents, un service est une application logicielle qui est accessible aux clients sur le Web par une interface. Le déploiement d'un

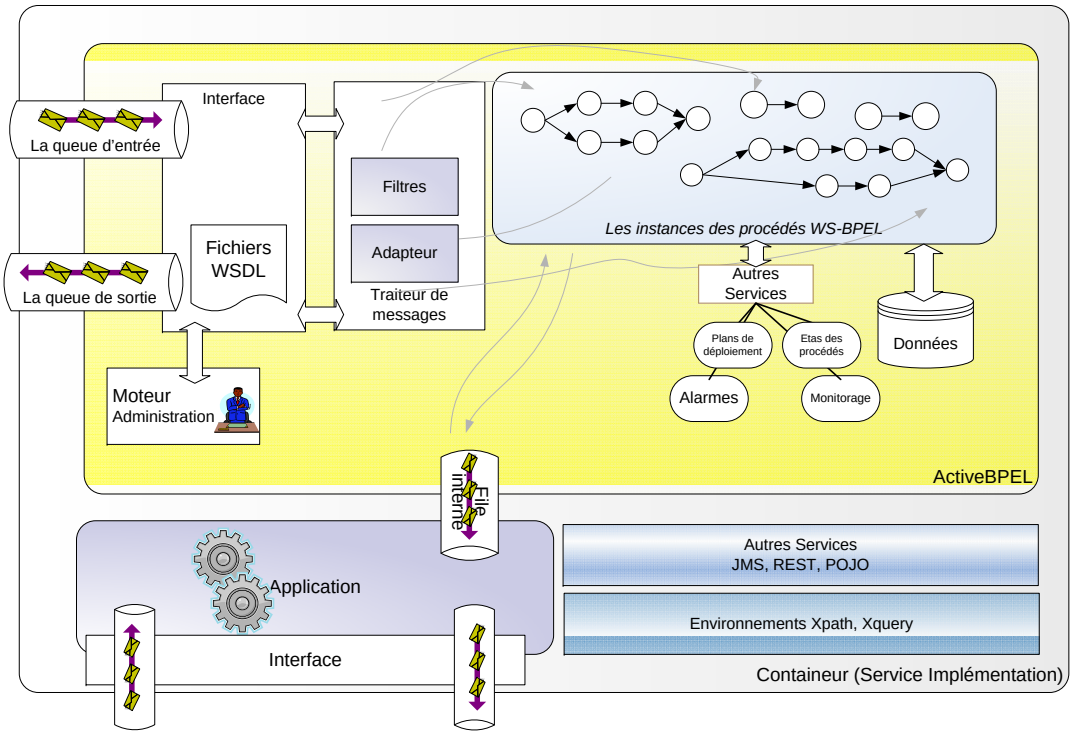


FIG. 7.1 – Vue d'ensemble de l'implémentation d'un service

procédé à exécuter se fait par des appels aux méthodes d'administration du moteur d'exécution qui font partie de l'interface de service. Les opérations d'administration d'un moteur d'exécution permettent la gestion des procédés telles que l'initialisation d'un modèle dans le moteur, la surveillance des instances ou leur suspensions. Les opérations d'administration peuvent être évoquées localement et elles ne sont pas accessibles par les clients via le Web. L'implémentation de ces opérations comme les opérations accessibles du service permet leur invocations par les clients via le Web. Cette implémentation est une solution simple et élégante à l'échangeabilité des procédés entre les services. Cette architecture de service considère un moteur d'exécution et l'accessibilité des opérations de l'administration du moteur comme des services Web. Il est à noter qu'on peut considérer l'implémentation d'autres modules logiciels secondaires pour supporter les deux premières fonctionnalités principales. L'implémentation logicielle que nous avons réalisée utilise le moteur ActiveBPEL pour exécuter les procédés. La raison de ce choix est essentiellement motivée par le fait que le moteur est une application open-source clairement documentée. Le moteur choisi fournit un certain nombre de fonctionnalités caractéristiques des moteurs de workflow telles que le déploiement des procédés WS-BPEL, le monitorage des procédés déployés ou leur visualisations. L'administration du moteur ActiveBPEL se fait par *ActiveBPEL Engine Administration Interface (API)*. La figure 7.2 illustre les composants principaux du moteur avec ceux que nous avons travaillés en vue d'implémenter les opérations d'administration. Dans nos premières expérimentations, nous avons réalisé une implémentation qui transforme les opérations d'administration de *ActiveBPEL*. Pendant ce temps, le producteur d'*ActiveBPEL* a introduit *ActiveBPEL Engine Administration Interface (API)*.

Le code ci-dessous illustre un fragment de WSDL qui contient l'implémentation des opéra-

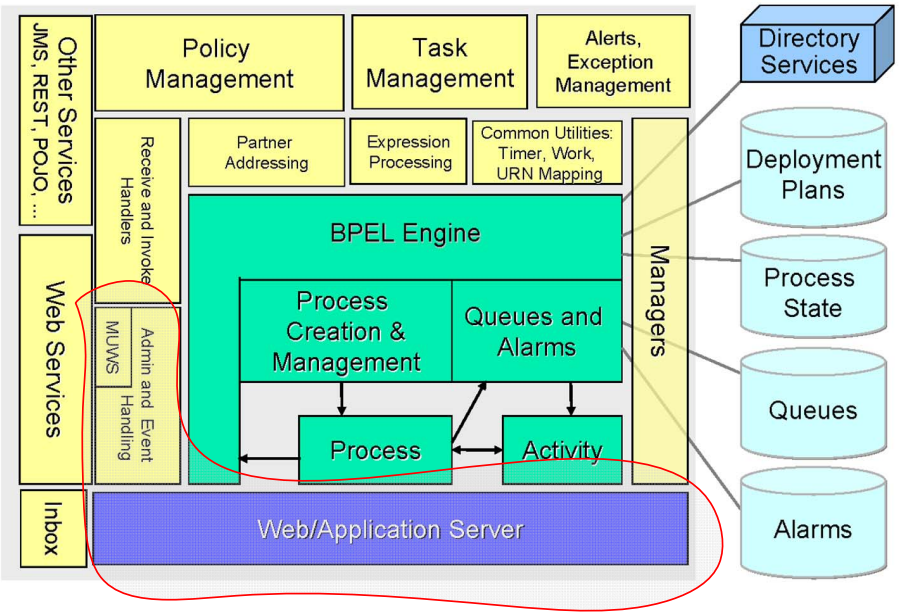


FIG. 7.2 – Vue d'nsemble de l'ActiveBPEL et les modules que nous avons étendus

tions d'administration dans l'interface de service. Nous attirons l'attention sur la définition des variables des opérations d'administration dans l'entête du document WSDL et leur définitions dans les opérations dans le reste du fichier WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:AeAdminServices"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:AeAdminServices"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://rt.activebpel.org"
xmlns:tns2="http://bpel.rt.activebpel.org"
xmlns:tns3="http://list.impl.bpel.rt.activebpel.org"
xmlns:tns4="http://server.rdebug.admin.server.bpel.rt.activebpel.org"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
<schema targetNamespace="http://rt.activebpel.org"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<complexType name="AeException">
<sequence>
<element name="info" nillable="true" type="xsd:string"/>
</sequence>
</complexType>
</schema>
<schema targetNamespace="http://bpel.rt.activebpel.org"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<import namespace="http://rt.activebpel.org"/>
<complexType name="AeBusinessProcessException">
<complexContent>
<extension base="tns1:AeException">
<sequence/>
</extension>
</complexContent>
</complexType>
```

```
</schema>
<schema targetNamespace="http://list.impl.bpel.rt.activebpel.org"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://list.impl.bpel.rt.activebpel.org">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<import namespace="http://bpel.rt.activebpel.org"/>
<import namespace="urn:AeAdminServices"/>
<complexType name="AeListingFilter">
<sequence>
<element name="listStart" type="xsd:int"/>
<element name="maxReturn" type="xsd:int"/>
</sequence>
</complexType>
<complexType name="AeProcessFilter">
<complexContent>
<extension base="tns:AeListingFilter">
<sequence>
<element name="processCompleteEnd" nillable="true"
type="xsd:dateTime"/>
<element name="processCompleteStart" nillable="true"
type="xsd:dateTime"/>
<element name="processCreateEnd" nillable="true"
type="xsd:dateTime"/>
<element name="processCreateStart" nillable="true"
type="xsd:dateTime"/>
<element name="processName" nillable="true"
type="xsd:QName"/>
<element name="processState" type="xsd:int"/>
</sequence>
</extension>
</complexContent>
</complexType>
<complexType name="AeProcessInstanceDetail">
<sequence>
<element name="ended" nillable="true" type="xsd:dateTime"/>
<element name="name" nillable="true" type="xsd:QName"/>
<element name="processId" type="xsd:long"/>
<element name="started" nillable="true"
type="xsd:dateTime"/>
<element name="state" type="xsd:int"/>
<element name="stateString" nillable="true"
type="xsd:string"/>
</sequence>
</complexType>
<complexType name="AeProcessListResult">
<sequence>
<element name="totalRowCount" type="xsd:int"/>
<element name="completeRowCount" type="xsd:boolean"/>
<element name="rowDetails" nillable="true"
type="impl:AeProcessInstanceDetailArray"/>
<element name="empty" type="xsd:boolean"/>
</sequence>
</complexType>
</schema>
<schema
targetNamespace="http://server.rdebug.admin.server.bpel.rt.acti
vebpel.org" xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<import namespace="http://list.impl.bpel.rt.activebpel.org"/>
<import namespace="urn:AeAdminServices"/>
<complexType name="AeBreakpointInstanceDetail">
<sequence>
<element name="processName" nillable="true"
type="xsd:QName"/>
<element name="nodePath" nillable="true" type="xsd:string"/>
</sequence>
</complexType>
<complexType name="AeBreakpointList">
```

```
<sequence>
.....
<wsdl:portType name="IAeBpelAdmin">
<wsdl:operation name="getConfiguration">
<wsdl:input message="impl:getConfigurationRequest"
name="getConfigurationRequest"/>
<wsdl:output message="impl:getConfigurationResponse"
name="getConfigurationResponse"/>
<wsdl:fault message="impl:AeException" name="AeException"/>
</wsdl:operation>
<wsdl:operation name="setConfiguration" parameterOrder="aXmlString">
<wsdl:input message="impl:setConfigurationRequest"
name="setConfigurationRequest"/>
<wsdl:output message="impl:setConfigurationResponse"
name="setConfigurationResponse"/>
<wsdl:fault message="impl:AeException" name="AeException"/>
</wsdl:operation>
<wsdl:operation name="suspendProcess" parameterOrder="aPid">
<wsdl:input message="impl:suspendProcessRequest"
name="suspendProcessRequest"/>
<wsdl:output message="impl:suspendProcessResponse"
name="suspendProcessResponse"/>
<wsdl:fault message="impl:AeBusinessProcessException"
name="AeBusinessProcessException"/>
</wsdl:operation>
<wsdl:operation name="resumeProcess" parameterOrder="aPid">
<wsdl:input message="impl:resumeProcessRequest"
name="resumeProcessRequest"/>
<wsdl:output message="impl:resumeProcessResponse"
name="resumeProcessResponse"/>
<wsdl:fault message="impl:AeBusinessProcessException"
name="AeBusinessProcessException"/>
</wsdl:operation>
<wsdl:operation name="resumeProcessObject" parameterOrder="aPid
aLocation">
<wsdl:input message="impl:resumeProcessObjectRequest"
name="resumeProcessObjectRequest"/>
<wsdl:output message="impl:resumeProcessObjectResponse"
name="resumeProcessObjectResponse"/>
<wsdl:fault message="impl:AeBusinessProcessException"
name="AeBusinessProcessException"/>
</wsdl:operation>
<wsdl:operation name="terminateProcess" parameterOrder="aPid">
<wsdl:input message="impl:terminateProcessRequest"
name="terminateProcessRequest"/>
<wsdl:output message="impl:terminateProcessResponse"
name="terminateProcessResponse"/>
<wsdl:fault message="impl:AeBusinessProcessException"
name="AeBusinessProcessException"/>
</wsdl:operation>
```

La spécification WSDL ci-dessus est l'élément principal de module SDS de l'implémentation. Nous avions basé notre travail sur la facilité de réalisation des exécutions décentralisées avec des procédés coopérants échangés entre les services composés. Cette réalisation elle-même est basée sur la possibilité de réalisation des services qui sont capables de recevoir et d'exécuter des procédés. Comme l'implémentation ci-dessus le démontre cette réalisation est relativement simple et consiste en une implémentation typique de service. Il est à noter que les procédés coopérants incluent des activités qui invoquent les opérations décrites dans cette interface.

Un procédé WS-BPEL est conçu avec un fichier WSDL qui reçoit les invocations entrantes. En des procédés coopérants, nous fournissons les fichiers WSDL correspondants. L'implémentation des opérations d'administration de ActiveBPEL considère également le déploiement des fichiers WSDL. Les fichiers déployés sont gérés dans un pool dynamique qui les maintient. La version

3.1 de ActiveBPEL supporte l'accès aux opérations d'administrations sur le Web. Pourtant le déploiement n'est pas une opération supportée. Outre les problèmes d'implémentation des opérations d'administration, une deuxième difficulté est liée à la gestion des messages entrants et sortants. L'implémentation des files consiste à faire des choix relatifs au support matériel sous-jacent. Nous avons implémenté un file commune à tous les fichiers WSDLs, la lecture y est arbitraire. Cette considération architecturale peut causer des problèmes de performance parce que la taille de la file est proportionnelle au nombre de procédés instanciés. Une autre détail de l'architecture logicielle porte sur l'interaction du moteur d'exécution avec l'application qui implémente le service. Les activités originales de la spécification sont structurées dans les procédés exécutés par le moteur implémenté pour le service auxquels elles se référe. Dans ce cas, les activités exécutées par le procédé du service se référent aux opérations du service même. Cette situation cause l'établissement d'une connexion vers l'extérieur qui peut surcharger les files de communications. Pour empêcher ces connexions non-souhaitables, le module de traitement des invocations reçues modifie le contenu des activités de telle sorte que l'activité évoque l'application interne qui implémente le service.

7.2.1 Implémentation des services

Dans cette section, nous détaillons certains aspects de l'implémentation des réalisations de services que nous avons utilisés pour nos expérimentations. Le moteur ActiveBPEL est le composant principal de l'implémentation. Le moteur est implémenté au dessus d'un serveur Web. L'implémentation de ce dernier considère un type de protocole qu'il utilise pour communiquer avec ses clients. Typiquement, on peut considérer deux types de protocoles SOAP-over-HTTP et SOAP-over-JMS pour ce propos. Pour procéder aux expérimentations, nous avons implémenté des services qui emploient ces deux types de protocoles. Le premier protocole, SOAP-over-HTTP, est un protocole synchrone alors que l'exécution décentralisée considère des communications asynchrones. Les messages uni-directionnel peuvent être échangés en envoyant des messages vides aux expéditeurs des messages. Quand un service Web est invoqué en utilisant HTTP, le service est implémenté comme un servlet contenu dans un container servlet. Les servlets sont conçus pour être implémentés avec des messages synchrones de type request-response. Le message de réponse ne peut pas être envoyé tant que le traitement de message request n'est pas fini. En conséquence, un client qui veut invoquer un service est bloqué lorsqu'il y a un autre client qui l'a déjà invoqué et n'a pas reçu de réponse. ActiveBPEL maintient un *pool* indépendant de servlet. ActiveBPEL envoie un message vide quand il reçoit un message et la demande reçue est mise dans le canal interne pour être transféré au servlet. Dans le deuxième type d'implémentation, on a considéré SOAP-over-JMS. Quand un service est invoqué en utilisant JMS (Java Messaging Service), le service est implémenté comme un MDB (Message Driven Bean). Ce dernier est inclu dans un EJB (Enterprise Java Bean). Les EJB ont un seul canal et également un seul processus qui traite ce canal. Le fonctionnement des EJB est quasiment semblable à l'implémentation d'un servlet avec un pool de taille variable. Même si les deux implémentations sont relativement similaires, les servlets analysent les messages reçus ce qui peut constituer un problème de performance si la taille et le nombre des messages deviennent important.

7.3 La production des procédés coopérants avec WS-BPEL

Dans cette section, nous discutons les détailes d'implémentation de la production des procédés coopérants en utilisant la spécification WS-BPEL. Cette partie de l'implémentation est nommée PPP (**P**eer **P**rocess **P**roducer). En langage WS-BPEL, un procédé peut être exprimé

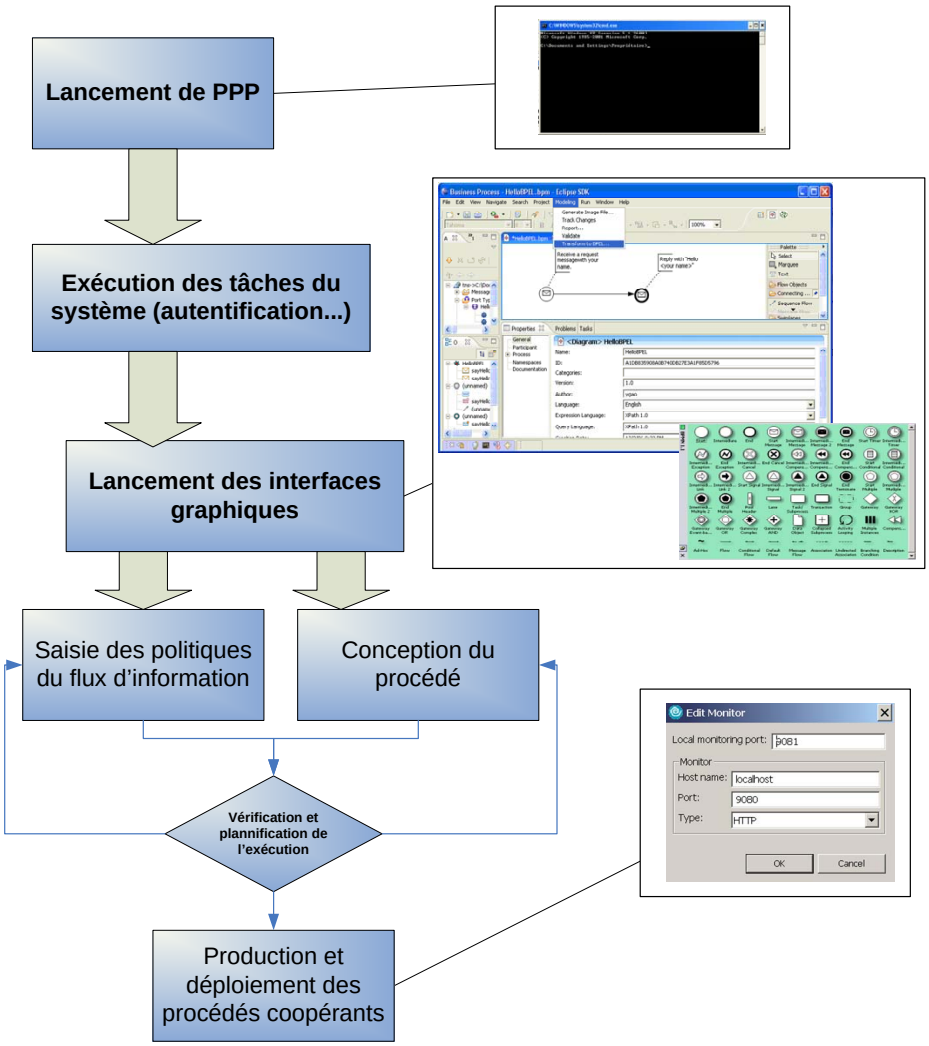


FIG. 7.3 – Vue d'ensemble du module PPP

sous des formes différentes. Ceci est lié à l'utilisation complémentaires des activités structurées (`flow, sequence, switch`) et `link` de la spécification WS-BPEL [Reichert *et al.*2004]. Dans [Wohed *et al.*2003] [Bergholtz *et al.*2004] présentent des frameworks pour passer d'une spécification WS-BPEL à une autre spécification plus conceptuelle telle que les patrons de workflow. Nous avons essentiellement utilisé les analyses présentées dans ces travaux.

La production des procédés coopérants se fait en plusieurs étapes comme expliqué dans les chapitres précédents. Ici nous allons présenter les éléments principaux de l'implémentation. La figure 7.3 illustre la vue d'ensemble de l'utilisation de l'initialisation du module PPP alors que la figure 7.3 illustre ses classes principales.

Les classes implémentées correspondent aux concepts utilisés dans les chapitres précédents. L'implémentation du module PPP se base sur l'utilisation de l'API JDOM de Java qui est conçu pour la manipulation des fichiers XML. Ce choix est en partie intuitif. Parce que la spécification d'un procédé WS-BPEL est analogue à une spécification XML. JDOM est une API du langage

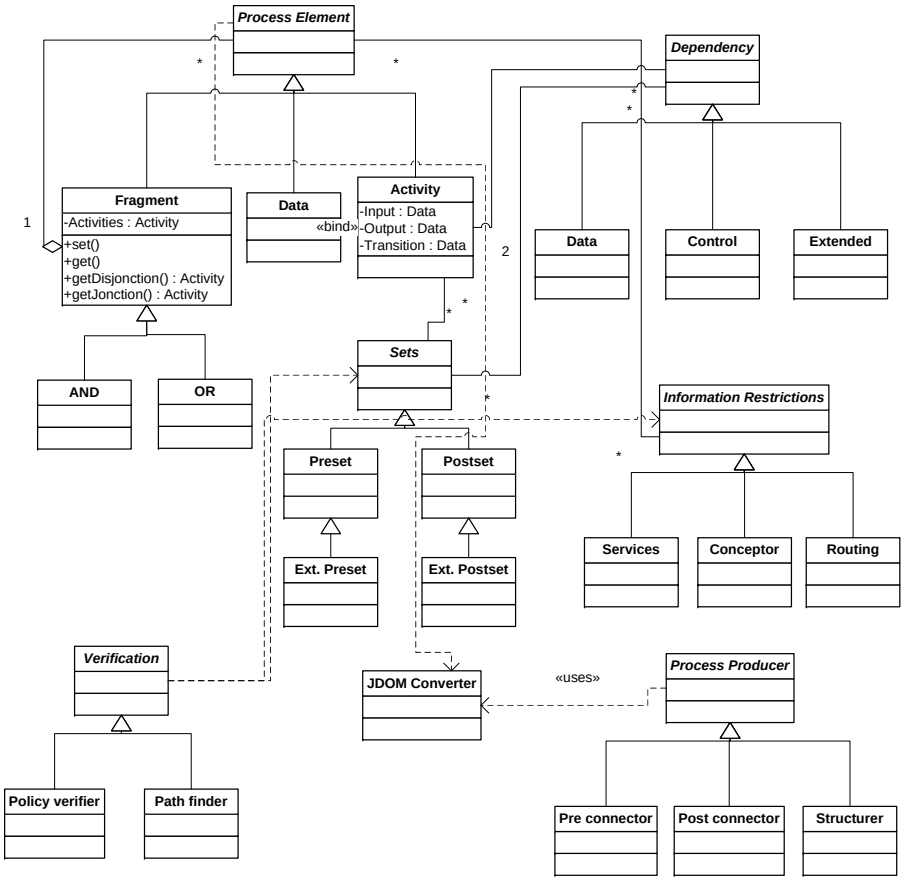


FIG. 7.4 – Les classes principales du module PPP

Java développée indépendamment de Sun Microsystems. Elle permet de manipuler des donnés 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 utilise DOM pour manipuler les éléments d'un Model Objet de Document spécifique crée grâce à un constructeur basé sur SAX. On peut donc construire des documents, naviguer dans leur structure, ajouter, modifier, ou supprimer soit des éléments soit du contenu. DOM est l'acronyme de *Document Object Model*. C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (recherche d'un élément précis, mise à jour d'élément trouvé...). À partir de cette représentation (le modèle), DOM propose de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspect les plus intéressant de DOM pour la production des procédés coopérants. DOM est défini pour être indépendant du langage dans lequel il sera implémenté. DOM n'est qu'une spécification qui pour être utilisée doit être implémentée par un éditeur tiers. DOM n'est donc pas spécifique à Java. SAX est l'acronyme de *Simple API* for XML. Ce type de parseur utilise des événements pour piloter le traitement d'un fichier XML. Un objet (nommé handler en anglais) doit implémenter des méthodes particulières définies dans une interface de l'API pour fournir les manipulations à implémenter : selon les événements, le parseur appelle ces méthodes. La première utilisation de JDOM porte sur l'extraction des dépendances nécessaires pour la production des procédés coopérants. La première opération de JDOM est de créer un objet de type `org.w3c.dom.Document` à partir de la spécification WS-BPEL à décentraliser. Tant que l'intégralité du document n'a pas été analysée et ajoutée dans la structure arborescente à ce niveau supérieur par rapport à l'élément racine du document XML, les données du fichier d'entrée ne se trouvent pas dans un état utilisable. En fait, comme le standard DOM ne spécifie pas de méthode pour obtenir l'objet `Document`, il existe plusieurs méthodes à cette fin. Les algorithmes principaux que nous avons implémentés sont présentés dans l'annexe A. Ce fichier qui contient les dépendances de contrôle, de donnée, et d'étendue ainsi que les ensembles qui correspondent aux activités dépendantes, est utilisé pour la production des procédés coopérants. Ci-dessous, nous donnons quelques lignes de code concernant la manipulation des fichier.

```
import org.apache.xerces.parsers.DOMParser;
public class WSBPELDOM
{
public static void main( String [] args ) throws Exception
{
WSBPELDOM parser = new WSBPELDOM();
parser.parse("ws-bpel-sample.xml");
Document document = parser.getDocument();
}
}
```

Pour extraire les dépendances nécessaires à la production des procédés, nous récupérons d'abord l'élément racine du document ws-bpel-sample.xml, via la méthode `getRootElement()` appliquée au noeud de type Document précédemment défini. Ensuite, nous définissons un noeud de type `Element`, équivalent à une collection Java, qui regroupe tous les éléments dont le type est titre, via la méthode getElementsByTagName("Activity"). Enfin, nous itérons sur cette liste afin de récupérer la valeur de chacun des noeuds de type `Element` : en effet la structure hiérarchique du DOM impose de devoir récupérer le contenu textuel d'un élément, non à partir de l'élément lui-même, mais à partir du noeud fils de l'élément. La première étape (récupération du noeud fils de l'élément <titre> se fait grâce à la méthode `getFirstChild()`, tandis que la récupération de la valeur textuelle se fait en utilisant la méthode `getNodeValue()`.Voici le code correspondant aux étapes que l'on vient de décrire :

```
...
Element catalogue = document.getRootElement();
NodeList titres = ws-bpel-sample.getElementsByTagName("Activity");
while(iter.hasNext()){
LinkedList <Element> = (iter.next()).getTextTrim()
}
```

Un autre aspect important est l'initialisation des contenus des messages échangés. Dans un premier temps, cet aspect concerne l'échange du contrôle entre deux activités liées par une dépendance de contrôle. Dans un deuxième temps ceci concerne une dépendance de donnée. Nous ne discutons pas les interactions qui implémentent les dépendances étendues car elles sont des combinaisons des interactions de contrôle et de donnée.

```
<scope name="a1a2control-scope">
<faultHandlers>
<catch faultName="jfns:joinFailure">
<invoke partnerLink="a1a2" portType="apnsf:op1"
operation="a1a2Link" inputVariable="falseAndCorrel"/>
</catch>
</faultHandlers>
<invoke name="a1a2control" suppressJoinFailure="no" partnerLink="a1a2"
portType="apnsf:op2" operation="a1a2Link"
inputVariable="trueAndCorrel">
<target linkName="c"/>
</invoke>
</scope>
```

La réception du message de contrôle dans le procédé coopérant qui exécute l'activité de connexion de lecture est illustré dans le code suivant.

```
<receive name="receive1" partnerLink="a1a2" portType="apnsf:op2"
operation="a1a2Link" variable="statusa1a2" createInstance="yes">
<correlations>
<correlation set="name" initiate="yes"/>
</correlations>
<source linkName="op1"
transitionCondition= "bpws:getVariableData('statusa1a2', 'status')"/>
</receive>
```

Dans l'exemple suivant, nous décrivons l'initialisation de deux activités de connexion qui échangent une donnée de procédé. Dans le code ci-dessus, on décrit l'implémentation d'une dépendance de donnée entre une activité de type `assign` et une de type `reply`. Nous les appelons les opérations *op*1 et *op*2. Il est à noter que ce code ne considère pas la dépendance de contrôle entre ces deux activités.

```
<scope>
<faultHandlers>
<catch faultName="jfns:joinFailure">
<invoke partnerLink="a1a3" portType="lns:op1"
operation="a3a1Data" inputVariable="falseAndCorrel" />
</catch>
</faultHandlers>
<sequence suppressJoinFailure="no">
<target linkName="data-assign-to-main"/>
<assign>
<copy> <from><status xmlns="" xsi:type="xsd:boolean">true</status></from>
<to variable="v1" part="status"/> </copy>
<copy> <from variable="trueAndCorrel" part="name"/>
<to variable="v1" part="name"/> </copy>
<copy> <from variable="v2" part="accept"/>
<to variable="v1" part="accept"/></copy>
</assign>
<invoke partnerLink="n1n3" portType="lns:op2"
operation="n3n1Data" inputVariable="v1" />
</sequence>
</scope>
```

Le code suivant décrit l'échange de données de la part du service qui exécute le procédé qui inclut l'activité `reply`. Nous soulignons que ce code inclut essentiellement une activité `receive` qui correspond à l'activité de type a^r dans les algorithmes génériques des chapitres précédents et l'activité `reply` placée dans ce code.

```
<receive name="n3n1data" partnerLink="n1n3"
portType="lns:op2" operation="n3n1Data"
variable="v1N3" createInstance="yes">
<correlations> <correlation set="name" initiate="yes"/> </correlations>
<source linkName="data-assign-to-reply" transitionCondition=
"bpws:getVariableData('v1N3','status')"/>
</receive>
<assign>
<copy><from variable="v1N3" part="accept"/><to
variable="v2" part="accept"/></copy>
<target linkName="data-assign-to-reply"/> <source linkName="n3data-to-reply"/>
</assign>
<reply name="mainreply" partnerLink="customer"
portType="apns:op1" operation="approve"
variable="v2"
joinCondition="(bpws:getLinkStatus('setMessage-to-reply') or bpws:getLinkStatus('op2-toreply'))
and (bpws:getLinkStatus('n3data-to-reply') or
not(bpws:getLinkStatus('n3data-to-reply'))) and (bpws:getLinkStatus('n2data-toreply')
or not(bpws:getLinkStatus('n2data-to-reply')))">
<target linkName="setMessage-to-reply"/><target linkName="op1"/>
<target linkName="n3data-to-reply"/> <target linkName="n2data-to-reply"/>
</reply>
```

7.4 Synthèse

Pour valider nos travaux, nous nous sommes attachés à les concrétiser par un produit logiciel. Cette approche pragmatique, reposant sur une réalisation d'outils, constitue, à notre sens, un moyen principal d'évaluer objectivement l'apport de notre contribution. Les travaux décrits dans ce document ont abouti à la réalisation de deux outils complémentaires : SDS/PPP (**S**ervice with **D**ecentralized **S**upport/ **P**eer **P**rocess **P**roducer). Dans ce chapitre, nous avons présenté les réalisations logicielles que nous avons développées pour tester la faisabilité de nos travaux. La réalisation considère un environnement distribué hétérogène. Les difficultés ont été nombreuses non seulement pour structurer un très gros logiciel (dont le développement atteint maintenant plusieurs milliers de lignes de code), mais aussi pour intégrer toutes les contraintes associées aux environnements distribués hétérogènes.

La première implémentation concerne la réalisation de la mobilité des spécifications de procédés entre les services. Cette partie de l'implémentation est basée sur l'extension de l'implémentation d'un système de gestion de workflow. L'environnement du système de gestion de workflow gère les problèmes de communication, concurrence et tolérance aux pannes entre les exécutions des procédés coopérants. La réalisation des services invoqués avec les procédés coopérants (SDS) n'était pas une tâche fastidieuse car nous avons conçu une interface de service qui implémente les opérations d'administration du moteur de workflow. Cette réalisation permet de réaliser l'exécution décentralisée de manière efficace par la mobilité des procédés coopérants. Intuitivement, cette implémentation qui considère les hypothèses architecturales, est plus simple à réaliser par rapport à ceux qui ont été proposées dans la littérature scientifique. Cette facilité de réalisation était un des objectifs de notre travail.

La deuxième implémentation (PPP) qui produit les procédés coopérants échangés porte sur la réalisation des algorithmes présentés dans les chapitres précédents. Nous avons utilisé WS-BPEL comme le langage de procédé. Ce choix est en partie intuitif comme la dernière est le *standard*

considéré par l'industrie. Notre approche était sur des hypothèses simplificatrices comme la décentralisation des procédés structurés. Ces hypothèses simplificatrices restent valables pour les procédés WS-BPEL traités. Contrairement à l'implémentation du module SDS, la réalisation du PPP a été généralement une tâche fastidieuse, nous sommes en train de développer une interface utilisateur proposant une forme graphique homogène pour la saisie des données du flux d'information, la gestion des menus ainsi que pour la visualisation des résultats. Cette normalisation permettra de réaliser, de manières efficace, des interfaces particularisables par le concepteur de procédé. Le concepteur du procédé n'aura alors plus à se préoccuper de la manipulation des données graphiques. L'implémentation actuelle est une application Java qui utilise principalement les APIs pour la manipulation des fichiers XML. Le module PPP peut être décomposé en couches qui s'empilent depuis le niveau le plus physique (celui des messages reçus) jusqu'au niveau le plus applicatif dans lequel on trouve les algorithmes implémentés et l'interface de concepteur. Pour ces dernières, l'implémentation apparaît bien sûr comme une seule entité, masquant la complexité de la réalisation avec des classes variées. Le but ultime de PPP est tester la faisabilité des algorithmes qui produisent les procédés coopérants. Nous avons constaté que WS-BPEL a des éléments que nous n'avons pas considérés dans le développement de notre approche tels que la gestion des exceptions et l'expression de la structure par des éléments différents. Il est à noter que notre travail ne portait pas sur un langage spécifique, mais le plus générique possible.

Chapitre 8

Conclusions et perspectives

Sommaire

8.1 Rappel du contexte et des objectifs de la thèse

Le paradigme Architectures Orientée Services (AOS) décrit l'utilisation de services pour supporter le développement d'applications distribuées, inter-opérables, et agiles. Les services sont vus comme des entités autonomes et sont indépendants de la plate-forme sur laquelle ils s'exécutent. La vision associée à AOS consiste à assembler différents services afin d'obtenir un ensemble de services faiblement couplés qui supportent de manière agile des procédés métiers composés de plusieurs entreprises et s'exécutant sur différentes plate-formes/outils. Cette vision va au-delà du simple échange d'information qui est actuellement le moyen privilégié pour que plusieurs entreprises coopèrent. Les standards de services Web sont actuellement la technologie et les intergiciels les plus prometteurs pour mettre en œuvre cette vision. Cela permet aux entreprises d'exposer leurs compétences à travers des services qui sont disponibles quelques soit la plate-forme (pc, téléphone, pda,...) et d'utiliser éventuellement les services exposés par les autres entreprises. En raison de la nature inter-organisationnelle des procédés métiers basés sur la composition des services maintenus par les différents partenaires, la gouvernance de leurs interactions mutuelles est un défit important. En effet, une composition de services utilisée dans un procédé métier inter-organisationnel ne pourra fonctionner correctement et efficacement que si les interactions des services sont gouvernées en accord avec d'une part les objectifs, et d'autre part avec les politiques définies par chaque partenaire et par le contexte. Ainsi, les services doivent respecter les objectifs fonctionnels, mais également les objectifs non-fonctionnels tels que les contraintes du flux d'information et le choix dynamique des services. Actuellement, les divers travaux sur la composition se sont bornés à étudier les aspects fonctionnels d'une composition par une approche centralisée telle que l'orchestration ou par une approche chorégraphie plus décentralisée. Cela n'est plus suffisant, avec la prise en compte de critères non-fonctionnels, en particulier pour améliorer la gouvernance des interactions des services composant un procédé d'une manière systématique. Un modèle d'exécution décentralisée dans lequel les services composés peuvent avoir des interactions restreintes par les contraintes de chaque composition est un besoin primordial

pour profiter des avantages respectifs des modalités d'exécution centralisées et décentralisées. Le modèle doit également permettre la spécification et le traitement des contraintes qui spécifient les interactions. Une autre caractéristique nécessaire à la re-modélisation de l'exécution des procédés métiers inter-organisationnels est la conformité de l'approche proposée aux efforts de standardisation. Conceptuellement, un modèle alternatif est nécessairement différent des modèles habituels par les hypothèses architecturales qu'il engendre. Le problème est d'autant plus complexe si on exige que ces hypothèses soient plus raisonnables par rapport aux implémentations actuelles.

Dans ce contexte, le but de notre travail était de proposer une nouvelle approche de modélisation et d'exécution des procédés métiers orientés services. La première caractéristique exigée d'une telle approche est sa conformité aux efforts de standardisation. Concernant la vision conceptuelle, la nouvelle approche doit traiter les difficultés respectives des approches principales afin de proposer une exécution qui exploite leurs avantages complémentaires. Dans cette nouvelle infrastructure d'exécution, les interactions des services composés doivent être en adéquation avec plusieurs types de restrictions imposées à la composition. Notre domaine de recherche est assez large puisqu'il couvre un ensemble de problèmes liés à la modélisation, l'analyse des procédés métiers et à la réalisation de systèmes distribués. Pour chacun de ces problèmes, il a été nécessaire de maîtriser et de développer les théories et les techniques associées.

8.2 Bilan des contributions

Le travail que nous avons présenté décrit une méthodologie et une architecture pour fournir un environnement d'exécution décentralisée dans lequel les services composés peuvent établir des interactions d'égal-à-égal. Comme nous l'avons souligné ci-dessus, l'implémentation des interactions est gouvernée par plusieurs type de restrictions impliquées à la composition. Nous avons défini une méthodologie de génération de procédés coopérants à partir d'une spécification centralisée de la composition. Ces procédés coopérants déployés vers les services composés permettent les interactions d'égal-à-égal des services. L'implémentation des interactions est gouvernée par les restrictions définies à la fois par les concepteurs de composition et à la fois par les éléments contextuels. Dans un premier temps, le déploiement est effectué selon une approche statique et dans un deuxième temps les aspects liés au déploiement dynamique des procédés sont étudiés. Ainsi, nos recherches ont intégrés différents aspects :

- Méthodologiques : nous avons défini et synthétisé les étapes de l'approche globale permettant de passer de la modélisation d'une composition à la réalisation de son exécution décentralisée. L'approche est motivée par les avancements des efforts de standardisation qui permettent l'échange de la spécification des procédés entre différents organisations et outils,
- Formels : nous avons intégré l'approche workflow pour la modélisation et développé une interprétation particulière du modèle, sur la base du calcul d'un ensemble de critères principaux tels que les aspects conversationnels, les flots de contrôle et de données. Nos considérations incluent une spécification pour l'expression de restrictions pour l'implémentation des interactions d'égal-à-égal. L'utilisation de paradigmes workflow facilitent la compréhension et la vérification des concepts de l'approche.
- Techniques : la maîtrise des caractéristiques et des contraintes du langage WS-BPEL a été nécessaire pour réaliser le prototype qui valide notre approche.

La méthodologie proposée a pour qualité d'être simple et complète, d'un pouvoir d'expression équivalent à celui des spécifications standardisée telles que WS-BPEL. Les concepts utilisés

sont plus nombreux et mieux formalisés que les méthodologies similaires de la littérature. Nous résumons les contributions principales de cette thèse comme ci-dessous :

- Nous avons introduit une technique pour décentraliser un procédé inter-organisationnel centralisé en respectant sa sémantique et en allant vers des interactions d'égal-à-égal entre les services composés par celui-ci. Cette contribution consiste en la définition de structures de données et d'algorithmes qui respectivement caractérisent et traitent les procédés métiers en vue de fournir les procédés coopérants que les services composés doivent mettre en œuvre.
- Nous avons introduit un langage de base pour exprimer les restrictions imposées aux interactions d'égal-à-égal des services composés. Nous avons présenté également les algorithmes qui traitent ce langage en vue de spécifier les interactions des services. Les techniques développées dans l'étape précédente sont adaptées à cette deuxième contribution en permettant la configuration des interactions des procédés coopérants fournis.
- La dernière contribution se penche sur la sélection dynamique et décentralisée de services lors de l'exécution avec les procédés coopérants. Nous avons proposé une technique qui intègre les problèmes de synchronisation développés précédemment.

Les contributions conceptuelles ci-dessus sont validées par les outils logiciels développés. La première validation logicielle porte sur la proposition d'un outil de traitement et de décentralisation de WS-BPEL. Nous avons également développé une architecture de référence pour les services qui peuvent supporter la mobilité des spécification de procédés.

8.3 Perspectives

Au delà des restrictions du flux d'information présentées, l'approche que nous avons menée autour de la mobilité des procédés dérivés à partir d'une spécification centralisée constitue une méthodologie nouvelle qui est, à nos yeux, extensible à diverses restrictions. La mise en œuvre a permis de dégager plusieurs points qui restent à approfondir. La version du prototype permettant de produire des procédés coopérants d'une spécification est quasiment opérationnelle. A partir de cette version, nous envisageons d'entamer une campagne d'implémentation des restrictions afin d'enrichir les politiques implémentées. Parallèlement, plusieurs modifications, tant d'un point de vue optimisation qu'extension, seront effectuées. Elles concerneront en particulier les points suivants :

- L'actuelle décentralisation d'une spécification centralisée en procédés coopérants repose sur des hypothèses encore restrictives. La caractérisation formelle de certaines propriétés des procédés telles que les spécifications structurées, permettra d'étendre la classe des spécifications centralisées *décentralisables*. Pour cela, il faudra également intégrer de manière formelle ces nouvelles fonctionnalités au prototype développé,
- Dans la perspective de produire des procédés coopérants, nous envisageons d'étendre les restrictions actuelles. Les problèmes liés au contrôle de flux d'information dans une composition de services nécessitent la compréhension et l'intégration d'aspects qui sont souvent étudiés séparément comme le contrôle d'accès, la gestion de la confiance ou la vérification d'une description de chorégraphie. Dans ce contexte, le traitement des politiques nécessite un modèle plus global.
- La méthodologie n'intégrant les spécificités du langage cible que dans les étapes ultimes, nous envisageons de l'appliquer à la génération d'autres langages cibles (BPMN, WS-CDL, ...).
- Concernant l'implémentation du choix dynamique de services, un placement avancé des

canaux de communication sur les différents sites de l'infrastructure de communication est également envisagé. Ceci est également lié au choix d'implémentation du choix dynamique.

Afin d'éprouver les limites de notre système de décentralisation, en particulier liés aux composants externes et à l'expression des restrictions et des spécifications centralisées, nous envisageons de traiter une application réelle. Ci-dessus nous avons résumé les perspectives immédiates de notre travail. Nous pouvons étendre ces perspectives dans des pistes différentes. Les perspectives à long terme de notre travail s'articulent sur trois point principaux :

- L'expérimentation de notre prototype nous a conduit à observer certaines limites liées à l'utilisation des spécifications orientées graphes. Par exemple, si celles-ci sont parfaitement adaptés à la spécification/vérification de comportements, elles sont moins appropriés à la description de comportements conversationnels. D'autres formalismes comme les spécifications algébriques, devraient, sur ce point, apporter des solutions satisfaisantes. De même, d'autres méthodes formelles (e.g. théorie des automates) permettraient d'étudier de nouveaux aspects de la dynamique des interactions entre les services.
- L'exploitation des propriétés issues du contrôle du flux d'information est un problème qui ouvre un champ de perspectives important. Lorsqu'une restriction intéressante par rapport à la sécurité est implémentée, il est souvent difficile de transférer une donnée sur un chemin dynamiquement défini. Ce point devient particulièrement délicat lorsque les techniques utilisées sont encapsulées par des formalismes de haut niveau traitées avant l'exécution où la réponse obtenue est encore du type *oui/non*. Étudier les mécanismes permettant de traduire ces restrictions, dans les termes du formalisme de haut niveau, en exécution décentralisée est une perspective nécessaire à la popularisation de l'approche globale de l'exécution décentralisée.
- L'implémentation des politiques issues du contrôle du flux d'information est pertinente pour exploiter l'architecture d'un système décentralisé et proposer des directives de placement afin d'en optimiser leur exécution. Ce point est en cours d'étude. Nous envisageons dans un premier temps de définir un lien entre les restrictions utilisées actuellement et les critères liés à la taille des données échangées. À terme, il sera peut-être possible de proposer des modèles plus proches de ceux des workflows scientifiques qui sont orientés données.

Un autre problème des méthodes de conception/développement est la prise en compte de l'existant. L'approche pour l'implémentation de la composition par des procédés coopérants offre une solution à leur prise en compte dans un nouveau mode d'exécution. Cependant, pour s'assurer de la conformité d'un composant logiciel à son abstraction, un retour aux techniques traditionnelles de tests est nécessaire. L'exploitation de cette abstraction pour générer automatiquement des séquences de tests de conformité et de robustesse est une perspective intéressante. Des travaux sur ce sujet ont été entamés. Ce point s'intéresse à l'utilisation des standards comme WS-Transactions, WS-Security et WS-ReliableMessaging.

Annexe A

Le traitement des procédés WS-BPEL

Extraction des dépendances à partir d'une spécification WS-BPEL

Dans cette section, nous donnons le listing du module de l'implémentation qui analyse les procédés WS-BPEL. Le but des analyses faites par cette classe est d'identifier les types de dépendances nécessaires à la production des procédés coopérants comme nous l'avons illustré dans le chapitre 4.

```
/**
 * @(#)ParcoursXPATH.java
 *
 *
 * @author YILDIZ Ustun
 * @version 1.00 2007/5/3
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.swing.JOptionPane;
import java.util.Iterator;
import java.util.List;
import java.util.LinkedList;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.xpath.XPath;

public class ParcoursXPATH {

    /*
     * Creer une nouvelle instance de ParcoursXPATH
     */
    public ParcoursXPATH() {
    }


/*
 * recupere un document WS-BPEL a partir du fichier '_FilePath',
 * appelle la methode choisie par l'utilisateur
 */
    public void parcours(File _FilePath) {
     String elem ;
        org.jdom.Document document = null ;
```

```
   try {
       /* On crée une instance de SAXBuilder */
       SAXBuilder sxb = new SAXBuilder();
       document = sxb.build(_FilePath);
   } catch (IOException e) {
       System.out.println("Erreur lors de la lecture du fichier " + e.getMessage() );
       e.printStackTrace();
   } catch (JDOMException e){
       System.out.println("Erreur lors de la construction du fichier JDOM " + e.getMessage() );
       e.printStackTrace();
   }

   try {
       /* On initialise un nouvel élément avec l'élément racine du document. */
       Element debRech = document.getRootElement() ;
    Object[] choixPossibles = {"Predecesseurs" ,
         "Successeurs" ,
         "Predecesseurs & Successeurs",
         "Successeurs jusqu'a conjonction",
         "Predecesseurs jusqu'a disjonction",
         "Elements n'etant pas sur le meme chemin",
         "Chemin passant par N points",
         } ;


    /* ici on initialise les variables de base de l'analyse*/
    Object choix = JOptionPane.showInputDialog(null,
       "Choisissez une des options suivantes:",
       "Operations a effectuer",
       JOptionPane.QUESTION_MESSAGE,
       null,
       choixPossibles,
       choixPossibles[2]
       ) ;

    Iterator<Element> iter = null ;
    LinkedList<Element> liste = null ;
    StringBuffer st = null ;

    if (choix.equals("Predecesseurs") || choix.equals("Predecesseurs & Successeurs")) {
elem = JOptionPane.showInputDialog(null,
       "Quel element souhaitez-vous chercher?",
   "Titre",
   JOptionPane.QUESTION_MESSAGE) ;
       liste = listPred(document.getRootElement() , elem) ;
       iter = liste.listIterator() ;
       st = new StringBuffer("Liste des predecesseurs de "+elem+" : ") ;
       while (iter.hasNext()){
        st.append((iter.next()).getTextTrim()+" ") ;
       }
       JOptionPane.showMessageDialog(null,
          st.toString(),
          "Resultat",
          JOptionPane.PLAIN_MESSAGE
          ) ;
    }

       if (choix.equals("Successeurs") || choix.equals("Predecesseurs & Successeurs")) {
elem = JOptionPane.showInputDialog(null,
       "Quel element souhaitez-vous chercher?",
   "Titre",
   JOptionPane.QUESTION_MESSAGE) ;
       liste = listSucc(document.getRootElement() , elem) ;
       iter = liste.listIterator() ;
       st = new StringBuffer("Liste des successeurs de "+elem+" : ") ;
       while (iter.hasNext()){
        st.append((iter.next()).getTextTrim()+" ") ;
       }
```

```
       JOptionPane.showMessageDialog(null,
          st.toString(),
          "Resultat",
          JOptionPane.PLAIN_MESSAGE
          ) ;
       }

       if (choix.equals("Successeurs jusqu'a conjonction")) {
elem = JOptionPane.showInputDialog(null,
       "Quel element souhaitez-vous chercher?",
   "Titre",
   JOptionPane.QUESTION_MESSAGE) ;
       liste = succAPointConj(document.getRootElement() , elem) ;
       iter = liste.listIterator() ;
       st = new StringBuffer("Liste des successeurs de "+elem+" jusqu'a conjonction: ") ;

       while (iter.hasNext()){
        st.append((iter.next()).getTextTrim()+" ") ;
       }
       JOptionPane.showMessageDialog(null,
          st.toString(),
          "Resultat",
          JOptionPane.PLAIN_MESSAGE
          ) ;
    }

       if (choix.equals("Predecesseurs jusqu'a disjonction")) {
elem = JOptionPane.showInputDialog(null,
       "Quel element souhaitez-vous chercher?",
   "Titre",
   JOptionPane.QUESTION_MESSAGE) ;
       liste = predAPointDisj(document.getRootElement() , elem) ;
       iter = liste.listIterator() ;
       st = new StringBuffer("Liste des predecesseurs de "+elem+" jusqu'a disjonction: ") ;
       while (iter.hasNext()){
        st.append((iter.next()).getTextTrim()+" ") ;
       }
       JOptionPane.showMessageDialog(null,
          st.toString(),
          "Resultat",
          JOptionPane.PLAIN_MESSAGE) ;
    }

    if (choix.equals("Elements n'etant pas sur le meme chemin")) {
elem = JOptionPane.showInputDialog(null,
       "Quel element souhaitez-vous chercher?",
   "Titre",
   JOptionPane.QUESTION_MESSAGE) ;
       liste = pasMemeChemin(document.getRootElement() , elem) ;
       iter = liste.listIterator() ;
       st = new StringBuffer("Liste des elements n'etant pas sur le meme chemin que "+elem+" : ") ;
       while (iter.hasNext()){
        st.append((iter.next()).getTextTrim()+" ") ;
       }
       JOptionPane.showMessageDialog(null,
          st.toString(),
          "Resultat",
          JOptionPane.PLAIN_MESSAGE
          ) ;
    }

    if (choix.equals("Chemin passant par N points")) {
    String s = JOptionPane.showInputDialog("Combien de points de passage souhaitez-vous definir?") ;
    int nbPoints = Integer.decode(s).intValue() ;
    String[] points = new String[nbPoints] ;
    for (int i=0 ; i<nbPoints ; i++) {
    points[i] = JOptionPane.showInputDialog("Point numero "+(i+1)) ;
    }
```

```
            liste = passN(document.getRootElement() , points) ;
            iter = liste.listIterator() ;
            st = new StringBuffer("Liste des elements figurants sur ce chemin : ") ;
            while (iter.hasNext()){
             st.append((iter.next()).getTextTrim()+" ") ;
            }
            JOptionPane.showMessageDialog(null,
               st.toString(),
               "Resultat",
               JOptionPane.PLAIN_MESSAGE
               ) ;
         }
        }
        catch (Exception e) {
System.out.println("Erreur dans parcours(File) : " + e.getMessage() );
            e.printStackTrace();
        }
    }// fin de parcours(File _FilePath)


    /*
     * recherche d'un element 'elem' en partant de debRech
     */
    public Element rechercheElement(Element debRech , String elem) {
     Element res = null ;
     try {

            if (debRech.getTextTrim().equals(elem)) return debRech ;

            /* On recherche 'elem' chez les enfants de debRech. */
            List<Element> l = debRech.getChildren() ;
            Iterator<Element> it = l.iterator() ;
            Element nc = null ;
            while(it.hasNext()) {
             nc = it.next() ;
             nc = rechercheElement(nc,elem) ;
             res = (res==null?nc:res) ;
            }

        }
        catch (Exception e) {
System.out.println("Erreur dans rechercheElement(Element,String) : " + e.getMessage() ) ;
            e.printStackTrace() ;
            }
        return res ;
    }// fin de rechercheElement(Element,String)


    /*
     * etablit la liste des predecesseurs de 'elem' en partant de 'racine'
     */
    public LinkedList listPred(Element racine , String elem) {
     try {
     if (racine.getTextTrim().equals(elem)){
     /* si la racine est l'element recherche */
     LinkedList liste = new LinkedList() ;
     return liste ;
     }
     else {
     if (racine.getChildren().size()!=0) {
     if (!racine.getTextTrim().equals("")) {
     LinkedList liste = listPred(racine.getChild("and"),elem) ;
     liste.addFirst(racine) ;
     return liste ;
     }
     else {
     /* il y a des 'or' */
     List<Element> l = racine.getChildren() ;
```

```
 LinkedList<Element> liste = new LinkedList() ;
 /* Parcours de chaque 'or' a fond jusqu'a trouver 'elem' */
 boolean trouve = false ;
 int i ;
 int j = -1 ;
 for(i=0 ; i<l.size() && !trouve ; i++) {
 if ((l.get(i)).getName().equals("link")) {
 trouve = contient(l.get(i),elem) ;
 }
 else {
 j = i ;
 }
 }
 if (!trouve){
 /* 'elem' n'a pas ete trouve.
  * on ajoute tous les elements dans la liste */
 for(i=0 ; i<l.size() ; i++) {
 if ((l.get(i)).getName().equals("or")) {
 ajouteListe(l.get(i),liste) ;
 }
 else {
 j = i ;
 }
 }
 /* puis on continue le parcours. */
 liste.addAll(listPred(l.get((j==-1?i-1:j)),elem)) ;
 }
 else {
 /* 'elem' a ete trouve.
      *  on ajoute seulement les elements de la branche specifiee */
 liste.addAll(listPred(l.get((j==-1?i-1:j)),elem)) ;
 }
 return liste ;
 }
 }
 }
 }
 catch (Exception e) {
 System.out.println("Erreur dans listPred(Element,String) : " + e.getMessage() ) ;
        e.printStackTrace() ;
 }
 return new LinkedList() ;
}// fin de listPred(Element,String)


/*
 * etablit la liste des predecesseurs de 'elem' en partant de 'racine'
 */
public LinkedList listSucc(Element racine , String elem){
 try {
 if (racine.getTextTrim().equals(elem)){
 /* si la racine est l'element recherche */
 /* on renvoie tous les successeurs */
 LinkedList liste = new LinkedList() ;
 ajouteListe(racine,liste) ;
 liste.remove(racine) ;
 return liste ;
 }
 else {
 if (racine.getChildren().size()!=0) {
 if (!racine.getTextTrim().equals("")) {
 return listSucc(racine.getChild("flow"),elem) ;
 }
 else {
 List<Element> l = racine.getChildren() ;
 LinkedList<Element> liste = new LinkedList<Element>() ;
 /* Parcours de chaque 'or' a fond jusqu'a trouver 'elem' */
 boolean trouve = false ;
```

```
 int i ;
 int j = -1 ;
 for(i=0 ; i<l.size() && !trouve ; i++) {
 if ((l.get(i)).getName().equals("or")) {
 trouve = trouve || contient(l.get(i),elem) ;
 }
 else {
 j = i ;
 }
 }
 if (!trouve){
 /* 'elem' n'a pas ete trouve.
  * on continue sur le 'flow' */
 liste = listSucc(l.get((j==-1?i-1:j)),elem) ;
 }
 else {
 /* 'elem' a ete trouve.
     *  on ajoute les successeurs de 'elem' */
     liste = listSucc(l.get((j==-1?i-1:j)),elem) ;
     /* on ajoute les successeurs du 'flow' */
     if ((l.get(l.size()-1)).getName().equals("flow"))
     ajouteListe(l.get(l.size()-1),liste) ;
 }
 return liste ;
 }
 }
 }
 }
 catch (Exception e) {
 System.out.println("Erreur dans listSucc(Element,String) : " + e.getMessage() ) ;
        e.printStackTrace() ;
 }
 return new LinkedList() ;
}// fin de listPred(Element,String)

/*
 * toutes les activites entre 2 précises
 */
public LinkedList actEntre2(Element racine,String e1,String e2) {
 LinkedList<Element> l1 = listSucc(racine,e1);
 LinkedList<Element> l2 = listPred(racine,e2);
 LinkedList<Element> res = new LinkedList<Element> ();
 Element tmp1;
 Element tmp2;
 boolean trouve;

 for (int i=0 ; i<l1.size() ; i++) {
 tmp1 = l1.get(i);
 trouve = false;
 for (int j=0 ; j<l2.size() & !trouve ; j++) {
 tmp2 = l2.get(j);
 if(tmp1.equals(tmp2)) {
 trouve = true;
 res.add(tmp1);
 }
 }
 }

 return res;
}//fin actEntre2


/*
 * Predecesseur d'un element a un autre
 */
public LinkedList predEntre2(Element racine,String e1,String e2) {
 return actEntre2(racine,e2,e1);
}//fin predEntre2
```

```
/*
 * successeur d'un element a un autre
 */
public LinkedList succEntre2(Element racine,String e1,String e2) {
 return actEntre2(racine,e1,e2);
}//fin succEntre2

/*
 * successeur jusqu'au point de conjonction
 */
public LinkedList succAPointConj(Element racine, String elem) {
 String conj;
 Element elt = rechercheElement(racine,elem);
 Element parent = elt.getParentElement();
 List<Element> l = parent.getChildren();
 Element tmp = null;
 boolean trouve = false;
 while(!trouve) {
 for (int i=0 ; i<l.size() & !trouve ; i++) {
 tmp = l.get(i);
 if(((tmp.getName()).equals("flow")) && (!((tmp.getTextTrim()).equals(elem))) && (!((tmp.getTextTrim()).equals("")
 trouve = true;
 }
 }
 parent = parent.getParentElement();
 l = parent.getChildren();
 }
 conj = tmp.getTextTrim();
 return succEntre2(racine, elem, conj);
}//fin succAPointConj

/*
 * predecesseur jusqu'au point de disjonction
 */
public LinkedList predAPointDisj(Element racine, String elem) {
 String disj;
 Element elt = rechercheElement(racine,elem);
 Element parent = elt.getParentElement();
 boolean trouve = false;
 while(!trouve) {
 if( ((parent.getName()).equals("flow")) && ((parent.getTextTrim()).equals("")) ) {
 trouve = true;
 }
 parent = parent.getParentElement();
 }
 disj = parent.getTextTrim();
 return predEntre2(racine, elem, disj);
}//fin predAPointDisj


/*
 * Etablit la liste des elements qui ne sont pas sur le meme chemin que elem
 */
public LinkedList pasMemeChemin(Element racine , String elem) {
 // liste de tous les elements du graphe
 LinkedList<Element> tousElem = listSucc(racine , racine.getTextTrim()) ;
 tousElem.addFirst(racine) ;
 // liste des predecesseurs de elem
 LinkedList<Element> chemin = listPred(racine , elem) ;
 // liste des successeurs de elem
 LinkedList<Element> succ = listSucc(racine , elem) ;
 // concatenation des predecesseurs de elem avec les successeurs de elem
 chemin.add(rechercheElement(racine,elem)) ;
 chemin.addAll(succ) ;
 // recherche des elements qui ne sont pas sur le meme chemin que elem
 LinkedList<Element> pasChemin = new LinkedList<Element>() ;
 boolean trouve ;
```

```
      for (int i=0 ; i<tousElem.size() ; i++) {
      trouve = false ;
      for (int j=0 ; j<chemin.size() && !trouve ; j++){
      trouve = tousElem.get(i).equals(chemin.get(j)) ;
      }
      if (!trouve) pasChemin.add(tousElem.get(i)) ;
      }
      return pasChemin ;
     }

     /*
      * Etablit la liste des elements
      */
      LinkedList passN(Element racine , String[] points) {
       // Recherche des predecesseurs et successeurs passant par le premier point
       LinkedList<Element> res = listPred(racine,points[0]) ;
res.add(rechercheElement(racine,points[0])) ;
res.addAll(listSucc(racine,points[0])) ;
// Creation d'un clone pour les comparaisons
LinkedList<Element> comp = new LinkedList(res) ;
// On retire maintenant les elements qui ne sont pas dans la liste des
// predecesseurs/successeurs des points suivants
for (int i=1 ; i<points.length ; i++) {
LinkedList<Element> tmp = listPred(racine,points[i]) ;
tmp.add(rechercheElement(racine,points[i])) ;
tmp.addAll(listSucc(racine,points[i])) ;
for (int j=0 ; j<comp.size() ; j++) {
boolean trouve = false ;
for (int k=0 ; k<tmp.size() && !trouve ; k++) {
trouve = comp.get(j).equals(tmp.get(k)) ;
}
if (!trouve) res.remove(comp.get(j)) ;
}
}
return res ;
     }


     /*
      * test si l'element 'elem' et un successeur de 'elt'
      */
     public boolean contient(Element elt , String elem) {
      boolean test = elt.getTextTrim().equals(elem) ;
      List<Element> l = elt.getChildren() ;
      Iterator<Element> iter = l.iterator() ;
      while(iter.hasNext()){
      Element e = iter.next() ;
      test = test || contient(e,elem) ;
      }
      return test ;
     }// fin de contient(Element,String)


     /*
      * ajoute a la liste 'liste' tous les successeurs de 'elt'
      */
     public void ajouteListe(Element elt , LinkedList liste) {
      if (!elt.getTextTrim().equals("")) liste.add(elt) ;
      List<Element> l = elt.getChildren() ;
      Iterator<Element> iter = l.iterator() ;
      while(iter.hasNext()){
      Element e = iter.next() ;
      ajouteListe(e,liste) ;
      }
     }// fin ajouteListe(Element,LinkedList)


}// fin class ParcoursXPATH
```

Annexe B

La recherche de chemin par le traitement des politiques du flux d'information

La recherche de chemin en vue d'identifier les chemins fiables qui remplacent d'interactions

Dans cette section, nous donnons les détailles de l'algorithme que nous avons implémenté pour trouver les chemin fiables qui remplacent les interactions qui ne sont pas en adéquation avec les politiques du flux d'information. Ce mécanisme est expliqué dans le chapitre 5. Ci-dessous nous présentons les classes principales que nous avons implémentées.

```
Classe Graphe
package tm;


public class Graph {

public static final string can\_route();
public static final string can\_route\_to();
public static final string can\_route\_from();
public static final string can\_route\_to();
int vv=0;
int v=8;// nombre des noeuds
int e=10;//  nombre des arcs.
Noeud[] V=new Noeud[v];
Arret[] E=new Arret[e];
Noeud[] AA=new Noeud[v];


public void saisir(){
Clavier c=new Clavier();
System.out.print("Les politiques: "+"\n");
for(int i=0;i<this.v;i++){
String s=c.lireString();
Noeud no=new Noeud();
no.n=s;
V[i]=no;
}

for(int i=0;i<this.e;i++){
String s;
 Arret ar=new Arret();
```

```
s=c.lireString();

 ar.depar.n=s;
//E[i]=ar;

 s=c.lireString();

 ar.arrive.n=s;
//E[i]=ar1;

 s=c.lireString();
 String montexte = new String(s);
 Integer monnombre=new Integer(montexte);
 int p = monnombre.intValue();

ar.w=p;
E[i]=ar;
}

}
public  int calcul_adequation(Noeud nn){
//Noeud nn=new Noeud();
int id=0;
int k=0;
vv=0;
while(k<this.e){
//if( E[k].depar.n==nn.n || E[k].arrive.n==nn.n){
if(nn.n.equals(E[k].depar.n)){
nn.WW[vv]=E[k].w;
nn.Vois1[vv]=E[k].arrive;vv++;}
if(nn.n.equals(E[k].arrive.n)){
nn.WW[vv]=E[k].w;
nn.Vois1[vv]=E[k].depar;vv++; }


k++;}
for(int u=0;u<this.v;u++){
if(nn.n.equals(this.V[u])) id=u;
}
this.V[id].Vois1=nn.Vois1;
//}
nn.v=vv;
return vv;
}
Noeud calcule_min_d(){
int m=AA[0].d;
Noeud n=new Noeud();
for(int i=0;i<this.v;i++){
if(AA[i].d<=m && AA[i].n.equals("-1")==false){
m=AA[i].d;
n=AA[i];}
}
return n;

}
public void traiter(){
Noeud nu=new Noeud();
V[0].d=0;V[0].pi=V[0];
for(int i=1;i<this.v;i++){
V[i].d=1000;

V[i].pi=null;
V[i].pi.n="null";

}

AA=V;
```

```
int u=0;
int dd=0;

while(u<this.v){

for(int i=u;i<this.v;i++)
for(int j=i;j<this.v;j++){
if(AA[i].d>AA[j].d ){
Noeud t=AA[i];
AA[i]=AA[j];
AA[j]=t;
}
}
nu=AA[u];
//nu=calcule_min_d(u);


calcul_voisin( nu);
for(int i=0;i<nu.v;i++){
if(nu.Vois1[i].d>nu.d+nu.WW[i]){

nu.Vois1[i].d=nu.d+nu.WW[i];
dd=dd+nu.Vois1[i].d;

nu.Vois1[i].pi=nu;
//nu.Vois1[i].pi.n=nu.n;
//System.out.print(nu.n+"----->"+nu.Vois1[i].n+"    "+nu.Vois1[i].d+"\n");
//System.out.print(AA[u].n+"\n");
for(int k=0;k<v;k++){if(nu.Vois1[i].n.equals(AA[k].n)) AA[k]=nu.Vois1[i]; }
}

}
for(int i=0;i<this.v;i++){
System.out.print("("+AA[i].d+","+AA[i].pi.n+")"+"\n");
}
System.out.print("-----------"+"\n");
//System.out.print(nu.n+"---");
//AA[u].n="-1";
u++;
}

System.out.print(Le chemin est: AA"\n");

}

public Graph(int v,int e){

this.vv=0;
this.v=v;
this.e=e;
Noeud[] V1=new Noeud[v];
Arret[] E1=new Arret[e];
Noeud[] AA1=new Noeud[v];
this.AA=AA1;
this.V=V1;
this.E=E1;


}
}

public class Arret {
Noeud depar;
Noeud arrive;
int w;
public Arret(){
Noeud depar=new Noeud();
```

```
this.depar=depar;
Noeud arrive=new Noeud();
this.arrive=arrive;
}
}

Classe Nud
package tm;

public class Noeud {

String n;
int v;
int d;
Noeud pi;
Noeud[] Vois1=new Noeud[71];
int[] WW=new int[71];

public Noeud(){
this.d=1000;
Noeud[] Vois1=new Noeud[71];
int[] WW=new int[71];
this.WW=WW;
this.Vois1=Vois1;
}
}
```

Annexe C

La preuve du théorème de l'exactitude

Dans cette annexe, nous détaillons la preuve du théorème que nous avons utilisé dans le chapitre 5. Le but de ce théorème est de prouver la relation d'équivalence entre la spécification centralisée d'un procédé de composition et ses procédés coopérants dérivés. Il est à noter que ce théorème valide les algorithmes de transformation au lieux de procéder une vérification sur les procédés coopérants dérivés. Pour ce faire, nous utilisons des définitions qui complément celles qui sont décrites dans les chapitres 4 et 5.

Définition 1 *Une transition est un tuple* $(A_s, A_c, lbl) \in (2^{\mathcal{A}}, \mathcal{A}, \mathcal{C}onds)$ *où* A_s *est un ensemble d'activités sources,* A_c *est un ensemble d'activités cibles et lbl est les états de conditions de transitions.*

Nous pouvons définir la sémantique du procédé comme une automate à états finis en identifiant (a) un ensemble d'états, (b) un ensemble de relations de transition, (c) un ensemble d'états initiaux, celui-ci est un singleton, (d) un ensemble d'états finaux, celui-ci est un singleton. Par rapport au théorème que nous essayons de prouver, l'ensemble (a) est le contexte que nous allons définir dans la définition 2 ci-dessous. L'ensemble (b) est la fonction *step* que nous allons définir dans la définition ci-dessus. Les ensembles (c) et (d) sont définis avec ces définitions.

Définition 2 *Soit P une spécification de procédé centralisée, un contexte particulier* cp_i *de procédé,* $i > 1$*, une transition* $t = (source, cible, E[C]A)$*,* $t \in \mathcal{T}$ *est l'ensemble de toutes les transitions de procédé, activée à l'instant i si et seulement si les conditions suivantes sont vraies :*

- *si* $cp_i \in \mathcal{CP} \wedge eval(E, cp_i, cp_{i+1}) =$ *générée,*
- *si* $source = \emptyset$ *et* $\exists t' \in \mathcal{T}$ *au même étape de transition qui précède un point OR-split,*
- *si* $source = \emptyset$ *et* $\exists t' \in \mathcal{T}$ *au même étape de transition qui précède un point AND-split.*

Une transition peut être évaluée si plusieurs critères sont vraies en même temps. Les deux critères principaux qui nous intéressent est l'activité source de la dépendance de contrôle est dans un état qui permet l'exécution de l'activité suivante par rapport au flux de contrôle et les données sur lesquelles les conditions de transitions sont exprimées sont initialisées. Après l'évaluation d'une transition, une nouvelle étape de l'exécution nommée cp_{i+1} est valable. Un contexte particulier initial est défini comme cp_0 pour les contextes particuliers non-initialisés. Un contexte particulier peut changer si au moins une transition devient vraie. La fonction $eval()$ est

utilisée pour évaluer les conditions exprimées des transitions. Plus formellement, on peut noter une transition comme suit. A partir d'un contexte particulier cp_{i-1}, et un ensemble de transition $\mathcal{T}_i$, $i > 0$, le contexte cp_i est défini par l'induction suivante :

- si $s \in cp_{i-1} \wedge \nexists_{t \in \mathcal{T}_{i-1}} \in source(t)$ alors s est dans cp_i ;
- si $\exists_{t \in \mathcal{T}_{i-1}} s \in target(t)$ alors $s \in cp_i$

où $source(t)$ et $target(t)$ sont les ensembles des états quittés et entrées, respectivement, des transitions qui sont évaluées. Intuitivement, un état est un élément d'un nouveau contexte. Quand une transition est évaluée, le composant action d'ECA est exécuté ce qui fait que les autres éléments du contexte tels que les événements, conditions et les variables sont changés. Plus formellement, on peut définir le changement du contexte peut être défini comme dans la définition 3.

Définition 3 *Soit un ensemble de conditions, $\mathcal{C}$, d'événements E et de données $\mathcal{D}$, un contexte à l'instant i est défini comme suit. $cp_i : \mathcal{C} \cup E \cup \mathcal{D} \rightarrow \{vrai, faux\} \cup \emptyset$, cp_i est défini :*

- *$eval(x)$, si $eval(x) \neq \emptyset$,*
- *$\emptyset$, $eval(x) = \emptyset$,*
- *$cp_{i-1}(x)$, pour les autres cas.*

Le contexte peut changer s'il y a des évaluations explicites à cause des *external stimuli*.

Homomorphisme des spécifications Dans cette section, nous présentons les éléments principaux qui nous ont permis de vérifier la préservation de la sémantique centralisée des procédés décentralisés. L'équivalence de la sémantique de deux systèmes est décrit par le même comportement externe qu'on peut observer sous le même external stimuli. Cela veut dire que les procédés décentralisés exécutent les activités respectives dans le même ordre que la spécification centralisée. Comme nous l'avons mentionné déjà, en considérant le procédé comme une structure algébrique avec son contexte pour son ensemble, et les opérations d'évaluation comme les lois de composition algébriques, on peut assurer des transitions équivalentes, sous le même external stimuli, entre les deux spécifications par la vérification du homomorphisme.

Définition 4 *Opération de transformation Soit un procédé centralisé P avec un ensemble de contextes $\mathcal{CP}$ et un ensemble de transitions $\mathcal{T}$. Soit P' l'union des procédés coopérants correspondants à P avec l'ensemble de contextes $\mathcal{CP}'$ et de transitions $\mathcal{T}'$. La fonction qui relie un contexte $cp \in \mathcal{CP}$ à un contexte $cp' \in \mathcal{CP}'$ est définie comme ci-dessous :*

- *$d_{cp}(cp) = cp'$ avec $cp' = \{a_{start}, a_1, a_2, ..., a_n\} \cup \{\forall\ a_i \in P | in_1, in_2, in_m\} \cup \{\forall\ a_i \in P | out_1, out_2, ..., out_n\}$*

La définition ci-dessus exprime les relations entre les éléments de deux algèbres. Par exemple deux activités interconnectées de la spécification centralisée a_1 et a_2 sont trouvées comme étant suivies des activités de connexions venant après a_1 et avant a_2. La dépendance de contrôle qui les lit est trouvée comme deux dépendances de contrôle dans les procédés coopérants produits. Nous cherchons toujours à prouver que $d_{cp}(step(cp)) = step'(d_{cp}(cp))$. Pour ce faire, nous définissons les lemmes suivantes qui vont nous aider à prouver le théorème.

Lemme 1 *Soit P un procédé. Soit P' un autre procédé qui diffère de P de telle sorte que seulement le composant conditionnel des transitions C est restreint comme suit. t = (source, cible, $E[C]A$) est changé en t' = (source, target, $E[C \wedge in_{source}A$). Soit $t \in \mathcal{T}_i$, et, $t' \in \mathcal{T}'_i$ les transitions correspondantes à l'instant i des procédés P et P', l'assertion est vraie pour $\mathcal{T}_i$ et $\mathcal{T}'_i$: $t \in \mathcal{T}_i \Leftrightarrow t' \in \mathcal{T}'_i$, $\forall i > 0$.*

Cette lemme assure que les transitions sont évaluées en même temps si les restrictions exprimées sur les données de conditions. Cela veut dire que quand les transitions sont évaluées, les transitions définies avec les mêmes conditions sont également évaluées par les procédés coopérants. Mais dans ce cas, la question essentielle est est-ce que une transition, même si une transition initiale, sera les mêmes effets sur l'exécution du procédé.

Lemme 2 *Soit P un procédé et P' un procédé limité par la contrainte définie dans la lemme précédente. Soit $\mathcal{A}(t)$, $\mathcal{C}ond(t)$, $\mathcal{D}(t)$ les activités, les conditions et les données associées à une transition t du procédé P et respectivement $\mathcal{A}(t')$, $\mathcal{C}ond(t')$, $\mathcal{D}'(t)$ correspond à ceux du P'. Les assertions suivantes sont vraies : $\mathcal{A}(t) = \mathcal{A}(t')$, $\mathcal{C}ond(t) = \mathcal{C}ond(t')$ et $\mathcal{D}(t) = \mathcal{D}'(t)$ pour $\forall t \in \mathcal{T}$ et $\forall t' \in \mathcal{T}'$.*

Par rapport à cette dernière lemme, les transitions restreintes par les états des transitions antérieures ont les mêmes effets dans les deux procédés.

Théorème 1 *La transformation d_{cp} permettant de relier le contexte d'un procédé au contexte des procédés coopérants correspondants constitue un homomorphisme. Cela s'exprime au travers du fait que pour toute étape $i >= 0$ et pour tout contexte de système cs_i du diagramme d'état originel, l'équation $d_{cp}(step(cp_i)) = step^{'}(d_{cp}(cp_i))$ doit être satisfaite, oú step est la fonction pour passer d'un contexte à un autre et $step^{'}$ est celle des procédés coopérants (tous les deux définis au travers des sémantiques opérationnelles développées les définitions initiales).*

Preuve du théorème 1

A partir du fait que la fonction *step* est en réalité une fonction composite, la dérivation formelle de l'homomorphisme doit se référer au contexte *cp*. Il est alors nécessaire de montrer que, pour toutes configurations de contexte cp_i, et $step(cp_i)$ aboutit à la même configuration que celle de $d_{cp}^{-1}(step^{'}(d_{cp}(cp_i))$.

Considérons un procédé p=($\mathcal{CP}, \mathcal{T}$), son ensemble d'activité $A = \{a_1, ..., a_n\}$ et son ensemble de transitions $\mathcal{T} = \{t_1, ..., t_m\}$. Soit $cp_i = \{e_1, ..., e_p\}$ le contexte actuel. D'abord nous allons transformer le procédé $p^{'} := d_{cp}(p) = (\mathcal{CP}^{'}, \mathcal{T}^{'})$ oú $p^{'} := d_{cp}(p)$, $T^{'} := d_{cp}(\mathcal{T}) \cup \mathcal{T}^{'}_{default}$ et l'ensemble des activités transformé $A^{'} := \{a_{start}, a_1,...,a_p,in\_a_1,...,in\_a_n,$
$out\_a_1,...,out\_a_n\}$. La preuve est organisée en quatre étapes :

- ***Etape 1 :*** D'abord nous la fonction *step* à cp_i, c'est à dire exécuter $step(cp_i)$:
  Etant donné cp_i, l'ensemble des transitions d'activitées τ_i peut être calculé. Supposons que $\tau_i = \{t_k, ...t_1\}$. L'activation de telles transitions permet de quitter certains états : $\bigcup_{t\in\tau_i}\{source^*(t)\}$ et d'en rejoindre d'autres : $\bigcup_{t\in\tau_i}\{target^*(t)\}$. Sans pour autant perdre la généralité, nous assumons que l'ensemble des activités quittés est $\{a_1, ..., a_{j-1}\}$ et l'ensemble des activités rejointes est $\{a_{p+1}, ..., a_q\}$. D'autre part, $cp_{i+1} = \{e_j, ..., e_p, e_{p+1}, ..., e_q\}$. Pour le calcul du contexte cp_{i+1}, nous assumons que $\mathcal{C}onds^+(t_i)$, $\mathcal{C}onds^-(t_i)$, $\mathcal{D}(t_i)$, et $A(t_i)$, que sont respectivement l'ensemble des conditions vraies, l'ensemble des conditions fausses, l'ensemble des données générés, et l'ensemble des activités exécutées, ainsi que l'ensemble de stimuli externes $\{es_{i+1}(x)\}$ sont connus. Conformément aux lemmes 1 et 2, le nouveau contexte cp_{i+1} est parfaitement défini. Ainsi, $step(cp_i) = (cp_{i+1})$.
- ***Etape 2 :*** Maintenant, nous allons appliquer d_{cp} à $\mathcal{A}$ et d_{cp} à $\mathcal{T}$:
  $cp_i^{'} = d_{cp}(cp_i)$=$\{a_1, ..., a_n, a_0, in\_a_n, out\_a_1, ..., out\_a_n\}$.
  $cp_i^{'} = d_{cp}(cp_i) = cp_i$.

- ***Etape 3 :*** Maintenant, nous allons appliquer la fonction *step* à $cp_i^{'}$:
  D'abord, nous calculons l'ensemble des transitions d'activation τ_i^i des procédés coopérants. Etant donné la lemme 2, $\tau_i^i = d_{cp}(\tau_i)$. En prenant en compte le contexte, nous savons, d'après les définition 2 et 3, que chaque transition qui est un élément de n'importe quel $d_{cp}(t)$, $t \in \{t_k, ..., t_1\}$ ait soit une activité source qui est un élément de $\{in_s | s \in source^*\}(t)\}$, soit une activité cible qui est un élément de $\{in_s | s \in source^*\}(t)\}$ Par rapport au contexte défini, les assertions suivantes sont vraies : les changements du contexte peuvent avoir leur raisons dans l'exécution des activités précédentes ou bien par un external stimuli. En conséquence, la fonction de transformation implémentée par des algorithmes dans les chapitres précédents, une transition t_i est associée à une autre transition t_i' avec les mêmes conditions *lbl*. De ce résultat, on peut passer à l'étape suivante.
- ***Etape 4 :*** Le dernier point porte sur l'injectivité de l'opération de d_{cp} qui est mentionnée dans les lemmes 1 et 2 pour montrer l'association de cp'_{i+1} à cp_{i+1}. $cp_{i+1} = d_{cp}^{-1}(cp'_{i+1})$, $d_{cp}^{-1}(cp'_{i+1}) = d_{cp}^{-1}(\{a_1, a_2, a_n, a_0, out\_a_1, out\_a_1, .., out\_a_1, ... in\_a_1, out\_a_2, out\_a_n\}) \Rightarrow \{a_1, a_n, ..., out\_a\_n\}$.

De ces opérations, les étapes 4 et 1 présentent des résultats équivalents ce qui fait que d_{cp} est un homomorphisme $\forall i > 0$.

Bibliographie

[Adi and Etzion2004] Asaf Adi and Opher Etzion. Amit - the situation manager. *VLDB J.*, 13(2) :177–203, 2004.

[Alonso *et al.*1996] Gustavo Alonso, Roger Günthör, Mohan Kamath, Divyakant Agrawal, Amr El Abbadi, and C. Mohan. Exotica/fmdc : A workflow management system for mobile and disconnected clients. *Distributed and Parallel Databases*, 4(3) :229–247, 1996.

[Alonso *et al.*1997] Gustavo Alonso, Claus Hagen, Hans-Jörg Schek, and Markus Tresch. Distributed processing over stand-alone systems and applications. In *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB*, pages 575–579, 1997.

[Alonso *et al.*2004] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services : Concepts, Architectures and Applications.* Springer-Verlag Berlin heidelberg New York, 2004.

[Andonoff *et al.*2005] Eric Andonoff, Lotfi Bouzguenda, and Chihab Hanachi. Specifying web workflow services for finding partners in the context of loose inter-organizational workflow. In *Business Process Management*, pages 120–136, 2005.

[Androutsellis-Theotokis and Spinellis2004] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4) :335–371, 2004.

[Aoyama *et al.*2002] Mikio Aoyama, Sanjiva Weerawarana, Hiroshi Maruyama, Clemens A. Szyperski, Kevin J. Sullivan, and Doug Lea. Web services engineering : promises and challenges. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE*, pages 647–648, 2002.

[Atluri and Warner2005] Vijayalakshmi Atluri and Janice Warner. Supporting conditional delegation in secure workflow management systems. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, SACMAT*, pages 49–58, 2005.

[Atluri *et al.*2001] Vijayalakshmi Atluri, Soon Ae Chun, and Pietro Mazzoleni. A chinese wall security model for decentralized workflow systems. In *ACM Conference on Computer and Communications Security*, pages 48–57, 2001.

[Baresi *et al.*2007] Luciano Baresi, Andrea Maurino, and Stefano Modafferi. Distributed bpel processes. In *Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering, SEKE*, pages 692–697, 2007.

[Barros *et al.*2005] Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Service interaction patterns. In Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Proceedings of 3rd International Conference, of Business Process Management, BPM*, volume 3649 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2005.

[Benatallah *et al.*2002] Boualem Benatallah, Quan Z. Sheng, Anne H. H. Ngu, and Marlon Dumas. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the 18th International Conference on Data Engineering, ICDE*, pages 297–308, 2002.

[Benatallah *et al.*2006] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing web service protocols. *Data Knowl. Eng.*, 58(3) :327–357, 2006.

[Berardi *et al.*2005] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *Proceedings of the 31st International Conference on Very Large Data Bases, VLDB*, pages 613–624. ACM, 2005.

[Bergholtz *et al.*2004] Maria Bergholtz, Prasad Jayaweera, Paul Johannesson, and Petia Wohed. A pattern and dependency based approach to the design of process models. In Paolo Atzeni, Wesley W. Chu, Hongjun Lu, Shuigeng Zhou, and Tok Wang Ling, editors, *Proceedings of the 23nd International Conference on Conceptual Modeling, ER*, volume 3288 of *Lecture Notes in Computer Science*, pages 724–739. Springer, 2004.

[Bertino *et al.*1999] Elisa Bertino, Elena Ferrari, and Vijayalakshmi Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1) :65–104, 1999.

[Bertino *et al.*2004] Elisa Bertino, Elena Ferrari, and Anna Cinzia Squicciarini. Trust-x : A peer-to-peer framework for trust establishment. *IEEE Trans. Knowl. Data Eng.*, 16(7) :827–842, 2004.

[Bertino *et al.*2005] Elisa Bertino, Beng Chin Ooi, Yanjiang Yang, and Robert H. Deng. Privacy and ownership preserving of outsourced medical data. In *Proceedings of the 21st International Conference on Data Engineering, ICDE*, pages 521–532. IEEE Computer Society, 2005.

[Bertino2004] Elisa Bertino. Purpose based access control for privacy protection. In *20èmes Journées Bases de Données Avancées, BDA*, page 243, 2004.

[Betin-Can *et al.*2005] Aysu Betin-Can, Tevfik Bultan, and Xiang Fu. Design for verification for asynchronously communicating web services. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW*, pages 750–759, 2005.

[Bondy and Murty1976] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. The Macmillan Press Ltd., 1976.

[Box *et al.*2000] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn., Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1. W3c note, W3C, 2000.

[BPMI.org2001] BPMI.org. Business process modeling language (bpml). Working draft 0.4, Business Process Management Initiative, 2001.

[Bultan *et al.*2006] Tevfik Bultan, Jianwen Su, and Xiang Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1) :18–25, 2006.

[Cai *et al.*1996] Ting Cai, Peter A. Gloor, and Saurab Nog. Dartflow : A workflow management system on the web using transportable agents. Technical Report TR96-283, 1996.

[Carlsson and Gustavsson2001] Bengt Carlsson and Rune Gustavsson. The rise and fall of napster - an evolutionary approach. In *Active Media Technology*, pages 347–354, 2001.

[Chafle *et al.*2004] Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. In *WWW (Alternate Track Papers & Posters)*, pages 134–143, 2004.

[Chafle *et al.*2005] Girish Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Orchestrating composite web services under data flow constraints. In *IEEE International Conference on Web Services, ICWS*, pages 211–218. IEEE Computer Society, 2005.

[Chen and Hsu2001] Qiming Chen and Meichun Hsu. Inter-enterprise collaborative business process management. In *Proceedings of the 17th International Conference on Data Engineering, ICDE*, pages 253–260, 2001.

[Chinnici *et al.*2002] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana. Web services description language (wsdl) version 1.2. Technical report, W3C, 2002.

[com1993] Com : Component object model technologies. Technical report, Microsoft, 1993.

[Coon2002] M. D. Coon. Peer-to-peer workflow management white paper. Technical report, 2002. http ://www.proteustechnologies. com/cmm/docs/P2P_Workflow_Whitepaper.do c.

[cor] *Common Object Request Broker Architecture*. http ://www.omg.org.

[Curbera *et al.*2002] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Ieee internet computing : Spotlight - unraveling the web services web : An introduction to soap, wsdl, and uddi. *IEEE Distributed Systems Online*, 3(4), 2002.

[Curbera *et al.*2003] Francisco Curbera, Rania Khalaf, Frank Leymann, and Sanjiva Weerawarana. Exception handling in the bpel4ws language. In *Business Process Management*, pages 276–290, 2003.

[Dahl *et al.*] O.-J. Dahl, Edsger W. Dijkstra, and Charles Antony Richard Hoare. *Structured Programming*. Academic Press.

[Damiani *et al.*2007] Maria Luisa Damiani, Elisa Bertino, Barbara Catania, and Paolo Perlasca. Geo-rbac : A spatially aware rbac. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007.

[Deng *et al.*2006] ShuiGuang Deng, Ying Li, Haijiang Xia, Jian Wu, and Zhaohui Wu. Exploring the flexible workflow technology to automate service composition. In *First Asian Semantic Web Conference, ASWC*, volume 4185 of *Lecture Notes in Computer Science*, pages 444–458. Springer, 2006.

[Dijkstra1959] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[Doyen2005] Guillaume Doyen. *Supervision des réseaux et services pair à pair*. PhD thesis, Université Henri Poincaré - Nancy I, 12 2005.

[Draft] Uddi Open Draft. Uddi version 2.0 api specification.

[Dumas and Fauvet2006] Marlon Dumas and Marie-Christine Fauvet. Les services web. *Intergiciel et Constructions d'Applications Réparties*, (chapter 4) :77–102, 2006.

[Emerson1990] E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B : Formal Models and Sematics (B)*, pages 995–1072. 1990.

[emu2002] Emule project, 2002. http ://www.emule-project.net/.

[Fakas and Karakostas2004] Georgios John Fakas and Bill Karakostas. A peer to peer (p2p) architecture for dynamic workflow management. *Information & Software Technology*, 46(6) :423–431, 2004.

[Fakas *et al.*2003] Georgios John Fakas, Bill Karakostas, and Stelios Christophi. A peer to peer (p2p) dynamic workflow management system based on web services. In *Proceedings of the International Conference on Web Services, ICWS*, pages 428–431, 2003.

[Fischer2007] Layna Fischer. *2007 BPM & Workflow Handbook Methods, Concepts, Case Studies and Standards in Business Process Management and Workflow*. Future Strategies Inc., 2007.

[Fu *et al.*2005] Xiang Fu, Tevfik Bultan, and Jianwen Su. Synchronizability of conversations among web services. *IEEE Trans. Software Eng.*, 31(12) :1042–1055, 2005.

[Gauron2006] Philippe Gauron. *Interconnexion et routage efficaces pour des procédures de recherche décentralisées dans les systèmes P2P*. PhD thesis, l'Université Paris-Sud 11, 2006.

[Godart1993] Claude Godart. Coo : A transaction model to support cooperating software developers coordination. In Ian Sommerville and Manfred Paul, editors, *Proceedings of the 4th European Software Engineering Conference, ESEC*, volume 717 of *Lecture Notes in Computer Science*, pages 361–379. Springer, 1993.

[Gokkoca *et al.*1997] Esin Gokkoca, Mehmet Altinel, Ibrahim Cingil, Nesime Tatbul, Pinar Koksal, and Asuman Dogac. Design and implementation of a distributed workflow enactment service. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems, CoopIS*, pages 89–98, 1997.

[Goldfard1990] C.-F. Goldfard. *The SGML Handbook*. Oxford Clarendon Press, 1990.

[Gray and Reuter1993] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Technologies*. Morgan Kaufmann Publishers, 1993.

[Grinter *et al.*1999] Rebecca E. Grinter, James D. Herbsleb, and Dewayne E. Perry. The geography of coordination : dealing with distance in r&d work. In *Proceedings of the International Conference on Supporting Group Work, GROUP*, pages 306–315, 1999.

[Grundy and Hosking1998] John C. Grundy and John G. Hosking. Serendipity : Integrated environment support for process modelling, enactment and work coordination. *Autom. Softw. Eng.*, 5(1) :27–60, 1998.

[Guabtni and Charoy2004] Adnene Guabtni and François Charoy. Multiple instantiation in a dynamic workflow environment. In Anne Persson and Janis Stirna, editors, *Proceedings of the 16th International Conference on Advanced Information Systems Engineering, CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2004.

[Guabtni *et al.*2006] Adnene Guabtni, François Charoy, and Claude Godart. Using isolation spheres for cooperative processes correctness. In *Proceedings of the 10th International Conference on CSCW in Design, CSCWD*, pages 21–26. IEEE Computer Society, 2006.

[Hagen and Alonso2000] Claus Hagen and Gustavo Alonso. Exception handling in workflow management systems. *IEEE Trans. Software Eng.*, 26(10) :943–958, 2000.

[Halevy *et al.*2006] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Ten-year best paper award talk session, data integration : The teenage years. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB*, pages 9–16. ACM Press, 2006.

[Harel and Naamad1996] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Trans. Softw. Eng. Methodol.*, 5(4) :293–333, 1996.

[Haugland *et al.*] Solveig Haugland, Mark Cade, and Anthony Orapallo. *J2EE 1.4 : The Big Picture*. Prentice Hall.

[jav] *Java Distributed Systems Home Page.* http ://www.sun.com/rmi ?

[Jøsang *et al.*2006] Audun Jøsang, Elizabeth Gray, and Michael Kinateder. Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems*, 4(2) :139–161, 2006.

[Joshi *et al.*2005] James Joshi, Elisa Bertino, and Arif Ghafoor. An analysis of expressiveness and design issues for the generalized temporal role-based access control model. *IEEE Trans. Dependable Sec. Comput.*, 2(2) :157–175, 2005.

[Kahn1974] Gilles Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[Kazhamiakin and Pistore2006] Raman Kazhamiakin and Marco Pistore. Choreography conformance analysis : Asynchronous communications and information alignment. In *Proceedings of Third International Workshop on Web Services and Formal Methods, WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2006.

[Kazhamiakin *et al.*2006] Raman Kazhamiakin, Marco Pistore, and Luca Santuari. Analysis of communication models in web service compositions. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web, WWW*, pages 267–276. ACM, 2006.

[Khalaf and Leymann2006] Rania Khalaf and Frank Leymann. Role-based decomposition of business processes using bpel. In *ICWS*, pages 770–780, 2006.

[Kiepuszewski *et al.*2000] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *Advanced Information Systems Engineering, 12th International Conference, CAiSE*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2000.

[Kindler *et al.*2000] Ekkart Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of workflow applications : Local criteria for global soundness. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer, 2000.

[King1983] John Leslie King. Centralized versus decentralized computing : organizational considerations and management options. *ACM Comput. Surv.*, 15(4) :319–349, 1983.

[Koutsonikola and Vakali2004] Vassiliki A. Koutsonikola and Athena Vakali. Ldap : Framework, practices, and trends. *IEEE Internet Computing*, 8(5) :66–72, 2004.

[Krzysztof Apt1997] Ernst-Rüdiger Olderog Krzysztof Apt. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, 1997.

[Leung and Reghbati1987] Hareton K. N. Leung and Hassan K. Reghbati. Comments on program slicing. *IEEE Trans. Software Eng.*, 13(12) :1370–1371, 1987.

[Leymann and Roller2000] Frank Leymann and Dieter Roller. *Production Workflow - Concepts and Techniques.* PTR Prentice Hall, 2000.

[Liu and Kumar2005] Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. In Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Proceedings of 3rd International Conference, of Business Process Management, BPM*, volume 3649 of *Lecture Notes in Computer Science*, pages 268–284. Springer, 2005.

[Ly *et al.*2008] Linh Thao Ly, Stefanie Rinderle, and Peter Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1) :3–23, 2008.

[Malik and Bouguettaya2007] Zaki Malik and Athman Bouguettaya. Evaluating rater credibility for reputation assessment of web services. In *Proceedings of 8th International Conference on Web Information Systems Engineering, WISE*, volume 4831 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2007.

[Marconi *et al.*2006] Annapaola Marconi, Marco Pistore, and Paolo Traverso. Implicit vs. explicit data-flow requirements in web service composition goals. In *Service-Oriented Computing - ICSOC 2006, 4th International Conference, ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 459–464. Springer, 2006.

[Marjanovic2000] Olivera Marjanovic. Dynamic verification of temporal constraints in production workflows. In *Australasian Database Conference*, pages 74–81, 2000.

[Mecella *et al.*2006] Massimo Mecella, Mourad Ouzzani, Federica Paci, and Elisa Bertino. Access control enforcement for conversation-based web services. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web, WWW*, pages 257–266. ACM, 2006.

[Milner1999] Robin Milner. *Communicating and Mobile Systems : the π-Calculus.* Cambridge Univ. Press, 1999.

[Muth *et al.*1998] Peter Muth, Dirk Wodtke, Jeanine Weißenfels, Angelika Kotz Dittrich, and Gerhard Weikum. From centralized workflow specification to distributed workflow execution. *J. Intell. Inf. Syst.*, 10(2) :159–184, 1998.

[Myers and Liskov1997] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *SOSP '97 : Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 129–142, New York, NY, USA, 1997. ACM Press.

[Nanda and Karnik2004] Mangala Gowri Nanda and Neeran M. Karnik. Synchronization analysis for decentralizing composite web services. *Int. J. Cooperative Inf. Syst.*, 13(1) :91–119, 2004.

[Nanda *et al.*2004] Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA*, pages 170–187, 2004.

[Nezhad *et al.*2006] Hamid R. Motahari Nezhad, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Web services interoperability specifications. *IEEE Computer*, 39(5) :24–32, 2006.

[OASIS2005] OASIS. Web Services Business Process Execution Language, version 2.0 (retrieved october 18, 2006), 2005. http ://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html.

[Omicini and Ossowski2004] Andrea Omicini and Sascha Ossowski. Coordination and collaboration activities in cooperative information systems. *Int. J. Cooperative Inf. Syst.*, 13(1) :1–7, 2004.

[Osterweil1987] Leon J. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering, ICSE*, pages 2–13. ACM Press, 1987.

[Osterweil1997] Leon J. Osterweil. Software processes are software too, revisited : An invited talk on the most influential paper of icse 9. In *19th International Conference on Software Engineering, ICSE*, pages 540–548. ACM Press, 1997.

[Ouyang *et al.*2006] Chun Ouyang, Marlon Dumas, Stephan Breutel, and Arthur H. M. ter Hofstede. Translating standard process models to bpel. In *Advanced Information Systems Engineering, 18th International Conference, CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2006.

[Park and Hwang2003] Joon S. Park and Junseok Hwang. Role-based access control for collaborative enterprise in peer-to-peer computing environments. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 93–99, New York, NY, USA, 2003. ACM Press.

[Paul *et al.*1997] Santanu Paul, Edwin Park, and Jarir K. Chaar. Rainman : A workflow system for the internet. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[Pautasso and Alonso2005] Cesare Pautasso and Gustavo Alonso. Flexible binding for reusable composition of web services. In *Software Composition*, volume 3628 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2005.

[Pistore *et al.*2005] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. Automated composition of web services by planning at the knowledge level. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1252–1259. Professional Book Center, 2005.

[Port and Kaiser1998] Daniel Port and Gail E. Kaiser. Collaborative work : Collaborative technologies for evolving software systems. *IEEE Internet Computing*, 2(6) :79–83, 1998.

[Rao *et al.*2006] Jinghai Rao, Peep Küngas, and Mihhail Matskin. Composition of semantic web services using linear logic theorem proving. *Inf. Syst.*, 31(4-5) :340–360, 2006.

[Reichert and Dadam1998] Manfred Reichert and Peter Dadam. $\text{Adept}_{\text{flex}}$-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.*, 10(2) :93–129, 1998.

[Reichert *et al.*2004] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. On the modeling of correct service flows with bpel4ws. In *Informationssysteme im E-Business und E-Government, Beiträge des Workshops der GI-Fachgruppe, EMISA*, volume 56 of *LNI*, pages 117–128. GI, 2004.

[Rinderle *et al.*2005] Stefanie Rinderle, Barbara Weber, Manfred Reichert, and Werner Wild. Integrating process learning and process evolution - a semantics based approach. In *Business Process Management*, pages 252–267, 2005.

[Rusinkiewicz and Georgakopoulos1994] Marek Rusinkiewicz and Dimitrios Georgakopoulos. From transactions to transactional workflows : Technology and applications. In *Proceedings of the First International Workshop on Advances in Databases and Information Systems, ADBIS*, pages 118–119, 1994.

[Rusinkiewicz and Sheth1995] Marek Rusinkiewicz and Amit P. Sheth. Specification and execution of transactional workflows. In *Modern Database Systems*, pages 592–620. ACM Press and Addison-Wesley, 1995.

[Sadiq and Orlowska1999] Wasim Sadiq and Maria E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In *Proceedings of the Advanced Information Systems Engineering, 11th InternationalConference, CAiSE*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 1999.

[Sadiq and Orlowska2000a] Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2) :117–134, 2000.

[Sadiq and Orlowska2000b] Wasim Sadiq and Maria E. Orlowska. On business process model transformations. In *ER*, pages 267–280, 2000.

[Sadiq *et al.*2006] Wasim Sadiq, Shazia W. Sadiq, and Karsten Schulz. Model driven distribution of collaborative business processes. In *Proceedings of the IEEE International Conference on Services Computing, SCC*, pages 281–284. IEEE Computer Society, 2006.

[Schollmeier2001] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *1st International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE Computer Society, 2001.

[Schuldt *et al.*1999] Heiko Schuldt, Gustavo Alonso, and Hans-Jörg Schek. Concurrency control and recovery in transactional process management. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS*, pages 316–326. ACM Press, 1999.

[Schuler *et al.*2004] Christoph Schuler, Roger Weber, Heiko Schuldt, and Hans-Jörg Schek. Scalable peer-to-peer process management - the osiris approach. In *Proceedings of the IEEE International Conference on Web Services, ICWS*, pages 26–34, 2004.

[SDSC1999] SDSC. San diego super computing center, 1999. Matrix.

[Sheth and Kochut1997] Amit Sheth and K.J Kochut. Workflow applications to research agenda : Scalable and dynamic work coordination and collaboration systems. In *NATO Advanced Study Institute on Workflow Management Systems and Interoperability, Istanbul, Türkiye,*, 1997.

[Sheth1991] Amit P. Sheth. Federated database systems for managing distributed, heterogeneous, and autonomous databases. In *VLDB*, page 489, 1991.

[Srivastava and Velegrakis2007] Divesh Srivastava and Yannis Velegrakis. Intensional associations between data and metadata. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data„ SIGMOD Conference*, pages 401–412. ACM, 2007.

[Stricker *et al.*1995] Thomas Stricker, James M. Stichnoth, David R. O'Hallaron, Susan Hinrichs, and Thomas R. Gross. Decoupling synchronization and data transfer in message passing systems of parallel computers. In *International Conference on Supercomputing*, pages 1–10, 1995.

[Thain *et al.*2005] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice : the condor experience. *Concurrency - Practice and Experience*, 17(2-4) :323–356, 2005.

[Thatte2001] Satish Thatte. Xlang - web services for business process design. Initial public draft, Microsoft Corporation, 2002-03-22 2001.

[van der Aalst and Berens2001] Wil M. P. van der Aalst and P. J. S. Berens. Beyond workflow management : product-driven case handling. In *Proceedings of ACM International Conference on Supporting Group Work, GROUP*, pages 42–51, 2001.

[van der Aalst and Weske2001] Wil M. P. van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *Proceedings 13th International Conference of Advanced Information Systems Engineering, CAiSE*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2001.

[van der Aalst *et al.*2000] Wil M. P. van der Aalst, Alistair P. Barros, Arthur H. M. ter Hofstede, and Bartek Kiepuszewski. Advanced workflow patterns. In *Proceedings of the* 7^{th} *International Cooperative Information Systems Conference, CoopIS*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2000.

[van der Aalst1998] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1) :21–66, 1998.

[van der Aalst1999] Wil M. P. van der Aalst. Interorganizational workflows : An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 3(34) :335–367, 1999.

[van der Aalst2005] W.M.P. van der Aalst. Process mining in cscw systems. In *Proceedings of the 9th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD*, pages 1–8. Coventry University/IEEE Computer Society Press, 2005.

[W3C] W3C. Extensible markup language (xml).

[W3C2005] W3C. *Web Services Choreography Description Language Version 1.0 (Candidate Recommendation)*, 2005.

[Weerawarana *et al.*2005] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Dan Ferguson. *Web Services Platform Architecture*. Prentice Hall, 2005.

[WfMC1996] WfMC. Workflow standard - interoperability abstract specification version 1.0. Document number wfmc-tc-1012, Workflow Management Coalition, 1996.

[Wodtke and Weikum1997] Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In Foto N. Afrati and Phokion G. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory - ICDT '97*, pages 230–246, Delphi, Greece, 1997.

[Wodtke *et al.*1997] Dirk Wodtke, Jeanine Weißenfels, Gerhard Weikum, Angelika Kotz Dittrich, and Peter Muth. The mentor workbench for enterprise-wide workflow management. In *SIGMOD Conference*, pages 576–579, 1997.

[Wohed *et al.*2003] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. Analysis of web services composition languages : The case of bpel4ws. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *Proceedings of the 22nd International Conference on Conceptual Modeling, ER*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2003.

[Wombacher2006] Andreas Wombacher. Decentralized consistency checking in cross-organizationalworkflows. In *ighth IEEE International Conference on E-Commerce Technology (CEC 2006) / Third IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2006)*, page 7. IEEE Computer Society, 2006.

[wsf] Technical report.

[Yang and Liu2006] Zaihan Yang and Chengfei Liu. Implementing a flexible compensation mechanism for business processes in web service environment. volume 0, pages 753–760, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[Yildiz and Godart2007a] Ustun Yildiz and Claude Godart. Centralized versus decentralized conversation-based orchestrations. In *9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007)*, pages 289–296, 2007.

[Yildiz and Godart2007b] Ustun Yildiz and Claude Godart. Dynamic decentralized service orchestrations. In Bruno Encarnacao Joaquim Filipe, José Cordeiro and Vitor Pedrosa, editors, *Proceedings of the* 3^{th} *International Conference on Web Information Systems and Technologies, WEBIST*, pages 36–45, Spain, 2007. INSTICC Press. **(Best Student Paper Award)**.

[Yildiz and Godart2007c] Ustun Yildiz and Claude Godart. Enhancing secured service interoperability with decentralized orchestration. In *Proceedings of the* 23^{th} *International Conference on Data Engineering Workshops, ICDE Workshops*, pages 725–733. IEEE Computer Society, 2007.

[Yildiz and Godart2007d] Ustun Yildiz and Claude Godart. Information flow control with decentralized service compositions. In *Proceedings of the* 5^{th} *IEEE International Conference on Web Services, ICWS*, pages 7–19. IEEE Computer Society Press, 2007.

[Yildiz and Godart2007e] Ustun Yildiz and Claude Godart. Synchronization solutions for decentralized service orchestrations. In Stefania Galizia, editor, *Proceedings of the Second IEEE International Conference on Internet and Web Applications and Services, ICIW*, pages 39–46, Mauritius, 2007. IEEE Computer Society Press.

[Yildiz and Godart2007f] Ustun Yildiz and Claude Godart. Towards decentralized service orchestrations. In *Proceedings of the* 27^{th} *ACM Symposium on Applied Computing, SAC*, pages 1662–1666. ACM Press, 2007.

[Yildiz2008] Ustun Yildiz. Contrôle du flux d'information dans une composition de services. In *Actes du XXVIème Congrès Informatique des organisations et systèmes d'information et de décision, INFORSID*. Lavoisier - Hermès, 2008.

[Yu *et al.*2004] Weihai Yu, Yan Wang, and Calton Pu. A dynamic two-phase commit protocol for self-adapting services. In *IEEE International Conference on Services Computing, SCC*, pages 7–15, 2004.

[Zhao *et al.*2005] Xiaohui Zhao, Chengfei Liu, and Yun Yang. An organisational perspective on collaborative business processes. In *Proceedings of the 3rd International Conference Business Process Management*, pages 17–31, 2005.

[Özsu and Valduriez1999] Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Inc., 1999.

Résumé

Les travaux de recherche de cette thèse portent sur la modélisation et la gestion des procédés métiers orientés services. Le travail s'intéresse aux procédés d'un point de vue de gestion décentralisée où les services composés peuvent établir des interactions de pairà-pair. Dans un premier temps, nous présentons une méthode qui permet de dériver des procédés coopérants à partir d'une spécification centralisée. Il s'agit des algorithmes qui analysent un procédé centralisé pour le traduire en procédés coopérants, en transformant le flux de contrôle et le flux de données du procédé d'origine en interactions équivalentes de type pair-à-pair. Un des apports de la décentralisation, qui répond à une nouvelle exigence des procédés orientés vers les services, est l'établissement des interactions de pair-à-pair qui respectent le flux d'information des services composés. La deuxième partie du travail est la proposition d'un langage permettant d'exprimer des politiques de flux d'information. Par la suite, nous étudions l'intégration des politiques du flux d'information dans les procédés coopérants. Le choix dun service entrant dans une composition peut être effectué dynamiquement, au moment de l'exécution du procédé, de sorte que l'ensemble des services composés n'est pas connu à priori. Une compréhension de la stratégie de choix dynamique des services et leur intégration dans le cadre des contributions proposées dans son ensemble est pour cela une étape centrale. Pour ce faire, une méthodologie qui automatise le processus de déploiement dynamique des procédés coopérants est proposée. Le travail présente une architecture logicielle qui valide les concepts proposés.

Abstract

This thesis studies decentralized management of business processes in the context of service oriented architectures. We present a generic approach that enables decentralized executions with cooperating processes by implemeting the centralized semantic with P2P interactions. Precisely, we present our method that derives the latter. We focus on the sophisticated control/data flow and conversational aspects that run counter to naive intuition, most of which, we explain using deeper analysis of the algorithms and data structures that we employed. In the second part of the manuscrit, we focus on information flow control that consists of planning the interactions of composed services in order to satisfy different security restrictions concerning the propagation of information in a composition. We examine the questions of what the information flow is and how it can be modeled and computed in service-oriented business processes. We present the design and implementation of a decentralized workflow management solution for the control of information flow. The derived process fragments are deployed on composed services and they enable them to establish P2P interactions with each other with respect to information flow policies. The last contribution takes on the challenge of implementing dynamic and decentralized process deployment following the same approach. Our conceptual contributions are implemented with state-of-art technologies.