UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ADAPTATION DES PROCESSUS COLLABORATIFS PAR COORDINATION DES CHANGEMENTS ET MIGRATION DES INSTANCES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

ENRICO LÉVESQUE

AOÛT 2011

REMERCIEMENTS

Je remercie d'abord mes directeurs de recherche, monsieur Guy Tremblay, directeur, professeur à l'Université du Québec à Montréal (UQÀM), et monsieur Ismaïl Khriss, codirecteur, professeur à l'Université du Québec à Rimouski (UQAR), pour leur soutien financier, les opportunités qu'ils m'ont offertes d'apprendre, leurs conseils et corrections judicieuses, ainsi que leurs encouragements.

J'exprime ma reconnaissance à Ismaïl pour m'avoir soutenu moralement depuis Rimouski, malgré les embûches. Et je dis merci à Guy pour son écoute, sa compréhension et son efficacité. Je considère que vous avez été tous les deux très généreux à mon égard.

Je tiens à remercier l'UQÀM et son administration, plus particulièrement le département d'informatique, ainsi que les professeurs qui m'ont enseigné au cours de la maîtrise : Étienne Gagnon, Guy Tremblay, Jean Privat, Louis Martin, Normand Séguin et Ivan Mafezzini. Merci également à Hafedh Mili pour l'organisation des séminaires du Latece. Merci aux membres du personnel administratif et de soutien, tous, vous contribuez à rendre les lieux et les études agréables. Merci aussi à ceux qui ont accepté de corriger et d'évaluer ce mémoire.

Je remercie mes collègues étudiants du Latece pour leur compagnie, leurs encouragements et les conversations intéressantes.

Je remercie mes parents et ma famille pour leurs encouragements, ainsi que la famille de mon épouse.

Et pour finir, je souhaite exprimer ma profonde gratitude à mon épouse, Annie Lalonde, pour sa compréhension, son sens de l'humour et son soutien. Elle partage vraiment cette réussite avec moi.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES LISTAGES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

BPEL	*Web Services Business Process Execution Language*
BPMN	*Business Process Modeling Notation*
BPMS	*Business Process Management System*
CPC	*Change Protocol for Collaboration*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
MDE	*Model Driven Engineering*
OASIS	*Organization for the Advancement of Structured Information Standards*
OMG	*Object Management Group*
UML	*Unified Modeling Language*
UQÀM	Université du Québec à Montréal
UQAR	Université du Québec à Rimouski
URI	*Uniform Resource Identifier*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
WS-Addressing	*Web Service Addressing*
WSDL	*Web Service Definition Language*
WS-CDL	*Web Services Choreography Description Language*
XML	*Extensible Markup Language*
XPath	*XML Path Language*
XSD	*XML Schema Definition*

RÉSUMÉ

Le changement en entreprise a toujours été une réalité. De ce fait, les processus d'affaires subissent des altérations régulièrement. La flexibilité des systèmes de gestion des processus est donc cruciale. Puisque les processus d'affaires s'exécutent habituellement sur une longue période, la capacité des systèmes d'effectuer ces changements aux processus lorsqu'ils sont en cours d'exécution est quasi indispensable. Grâce aux architectures fondées sur les services, plusieurs organisations indépendantes peuvent contribuer à la réalisation d'une chorégraphie de processus. La complexité de gestion du changement augmente alors en fonction du nombre de participants. Des changements dans de telles chorégraphies doivent être coordonnés au niveau des contrats, des schémas des processus et des instances actives en collaboration. Notre approche de gestion du changement tient compte de tous ces aspects. Nous proposons un protocole pour coordonner le changement dans des chorégraphies de processus, le *Change Protocol for Collaboration* (CPC). Il assure que les changements soient traités par tous les participants. Nous proposons en plus une méthode d'adaptation dynamique des processus collaboratifs par migration des instances, et ce, de manière semi-automatisée. Aucune solution d'adaptation dynamique n'avait encore été proposée tenant compte de la collaboration d'instances actives.

Mots clés : architecture fondée sur les services, processus d'affaires, processus collaboratifs, chorégraphie, évolution des processus, coordination du changement, changement dynamique, migration des instances.

INTRODUCTION

Problématique

L'intérêt global pour la gestion des processus a débuté suite à la publication d'un fameux article (Hammer, 1990), dans lequel l'auteur démontrait que la réingénierie des processus d'affaires est un moyen efficace pour les entreprises d'augmenter leur productivité et leur rentabilité. Kettinger et Grover (1995), faisant une revue de littérature sur la question, écrivaient plus tard :

> *« Global competition, economic downturn, and the potential offered by emerging technologies are pushing firms to fundamentally rethink their business processes. Many firms have reached the conclusion that effecting business process change is the only way to leverage their core competencies and achieve competitive advantage. This belief has led to a near "reengineering frenzy." Thus, we define business process change management as a strategy-driven organizational initiative to improve and (re)design business processes to achieve competitive advantage in performance... through changes in the relationships between management, information, technology, organizational structure, and people. »*

Cet engouement pour la gestion des processus est toujours d'actualité. Des systèmes de gestion des processus sont aujourd'hui offerts pour faciliter la gestion et automatiser l'exécution des processus, mais encore peu des technologies disponibles tiennent compte de la réalité du changement. Pourtant, le besoin de flexibilité est toujours aussi criant. Quelques fournisseurs en industrie et des chercheurs en milieu académique ont conçu des systèmes plus adaptables, mais il y a encore place pour amélioration — notamment, dans

la coordination et la gestion automatisée du changement, et l'adaptation dynamique des processus.

Puisque les processus d'entreprise s'exécutent habituellement sur une longue période, la capacité des systèmes d'effectuer des changements aux processus lorsqu'ils sont en cours d'exécution est quasi indispensable. De plus, grâce à des architectures distribuées, comme les architectures fondée sur les services, plusieurs organisations indépendantes peuvent contribuer à la réalisation d'une chorégraphie de processus. La complexité de gestion du changement augmente alors en fonction du nombre de participants. Des changements dans de telles chorégraphies doivent être coordonnés au niveau des contrats, des schémas des processus et des instances actives en collaboration. Notre approche de gestion du changement tient compte de tous ces aspects.

Contributions

Nous proposons un protocole pour coordonner le changement dans des chorégraphies de processus, le *Change Protocol for Collaboration*. Il assure que les changements soient traités par tous les participants.

Nous proposons en plus une méthode d'adaptation dynamique des processus collaboratifs par migration des instances, et ce, de manière semi-automatisée. Aucune solution d'adaptation dynamique n'avait encore été proposée tenant compte de la collaboration d'instances actives.

Une preuve de concept a été réalisée pour valider l'approche proposée. Nous décrivons dans ce mémoire le système implémentant le CPC et la migration, ainsi que nos méthodes de tests.

Organisation du mémoire

Le premier chapitre traite principalement du cycle de vie des processus d'affaires. Nous définissons les termes utilisés dans le mémoire et nous introduisons les technologies et concepts associés à nos travaux.

Le deuxième chapitre présente une classification du changement dans des processus d'affaires automatisés, avec quelques exemples pour illustrer les différents types de changements.

Le troisième chapitre aborde la question de l'adaptation dynamique des processus. Nous présentons une revue de littérature en considérant trois différentes approches. Nous introduisons ensuite notre approche de migration des instances avec ses avantages et les considérations qui l'ont motivée.

Le quatrième chapitre se consacre à notre approche pour l'adaptation dynamique des processus collaboratifs. Il décrit en détails le *Change Protocol for Collaboration* (CPC). Ce protocole assure la coordination du changement lorsque plusieurs organisations participent à la réalisation d'une chorégraphie de processus. Nous présentons aussi notre algorithme de migration des instances tenant compte des collaborations en cours.

Le dernier chapitre présente la mise en oeuvre du CPC et de la migration. Nous présentons l'architecture globale du système CPC et l'architecture de tests. Nous expliquons la démarche adoptée pour valider notre approche. Nous terminons en discutant des résultats des tests, des limitations de notre preuve de concept et des leçons apprises sur la compensation.

Nous avons inclus trois appendices. L'appendice A fait un survol rapide de la notation BPMN utilisée pour modéliser nos processus. L'appendice B présente le langage BPEL utilisé pour les exécuter. L'appendice C montre le code source des processus implémentant le CPC.

Chapitre I

ÉVOLUTION DES PROCESSUS EN ENTREPRISE

Dans ce chapitre, nous offrons d'abord quelques définitions reliées au domaine des processus d'affaires. Ensuite, nous considérons le cycle de vie des processus d'affaires de façon générale. Nous utilisons ce cadre pour présenter les langages, technologies et concepts utilisés pour nos recherches, spécifiquement pour les phases de conception, d'implémentation et d'exécution des processus. Puis, nous expliquons la distinction entre orchestration et chorégraphie des processus.

1.1 Définitions

Étant donné la diversité des termes reliés aux processus d'affaires, considérons quelques définitions. Toutes ces définitions, sauf la dernière, ont été tirées et traduites du livre de Weske (2007).

Définition 1.1.1 *Un* processus d'affaires (business process) *est composé d'un ensemble d'activités qui sont exécutées en coordination dans un environnement organisationnel et technique. Ces activités réalisent conjointement un objectif d'affaires. Chaque processus d'affaires est exploité par une seule organisation, mais il peut interagir avec les processus d'affaires d'autres organisations.*

Définition 1.1.2 *La* gestion des processus d'affaires (business process management) *comprend les concepts, les méthodes, et les techniques pour appuyer la conception, l'administration, la configuration, l'analyse et l'exploitation des processus d'affaires.*

Définition 1.1.3 *Un* système de gestion des processus d'affaires (business process management system) *est un système logiciel générique comprenant les outils pour permettre les différentes opérations de gestion des processus.*

Définition 1.1.4 *Un* modèle de processus d'affaires (business process model) *comprend un ensemble de modèles d'activités et les contraintes d'exécution entre elles. Une* instance de processus d'affaires *représente un cas concret dans les opérations d'affaires d'une compagnie, comprenant des instances d'activités. Chaque modèle de processus d'affaires agit comme un* schéma *(ou un plan) pour un ensemble d'instances de processus d'affaires, et chaque modèle d'activité agit comme un schéma pour un ensemble d'instances d'activité.*

Définition 1.1.5 *Un* processus automatisé (workflow) *est l'automatisation d'un processus d'affaires, en tout ou en partie, durant lequel des documents, de l'information ou des tâches sont passés d'un participant à l'autre pour prendre action, selon un ensemble de règles procédurales.*

Définition 1.1.6 *Un* partenaire (partner) *est une ressource (humaine, matérielle ou logicielle) qui participe à la réalisation d'une ou plusieurs activités d'un processus. Au moment opportun, le système de gestion des processus d'affaires fait appel aux partenaires pour exécuter les activités du processus.*

Le terme *processus* est employé seul à plusieurs reprises dans ce mémoire lorsque le contexte permet de distinguer entre processus d'affaires, processus automatisé et instance de processus, ou lorsque la distinction n'est pas absolument nécessaire pour la compréhension du texte. Le terme *schéma* est préféré dans ce mémoire pour référer au *modèle* d'un processus.

1.2 Cycle de vie des processus d'affaires

Le cycle de vie des processus d'affaires peut être divisé en quatre phases importantes (voir figure 1.1) :

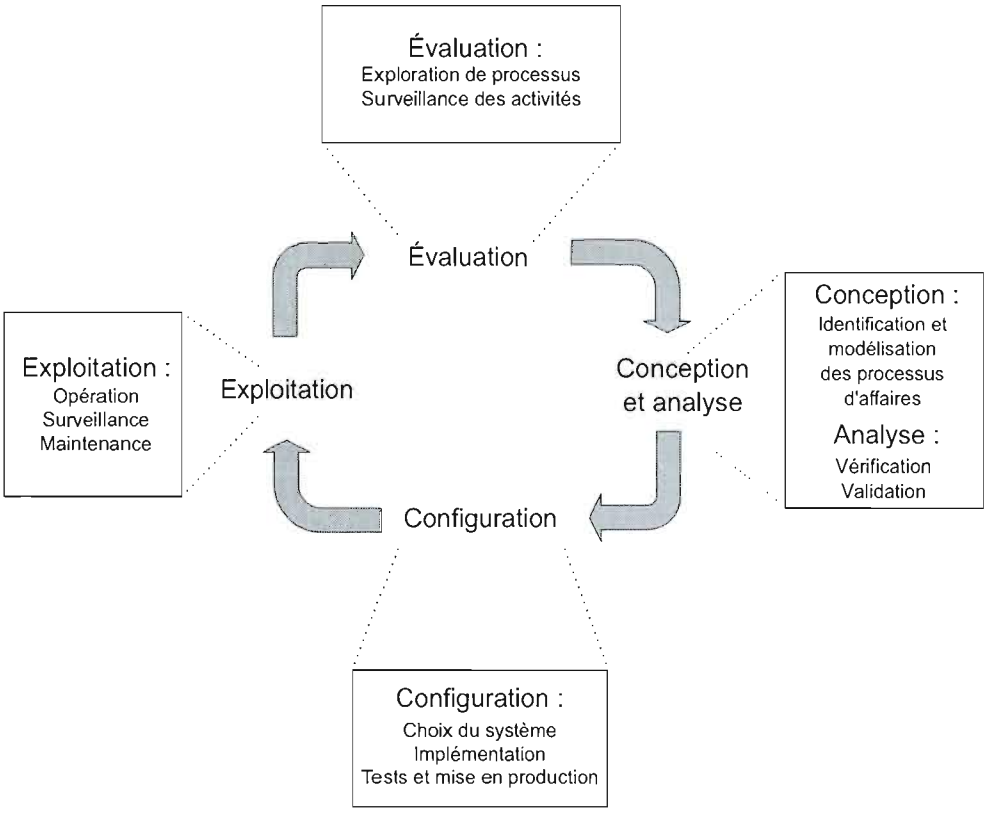

Figure 1.1 Cycle de vie des processus d'affaires. Figure adaptée (Weske, 2007).

i) La phase de conception et d'analyse englobe les activités d'exploration et de modélisation des processus d'affaires, ainsi que les activités de vérification et de validation.

ii) La phase de configuration concerne les activités reliées à l'implémentation, aux tests et au déploiement.

iii) La phase d'exploitation fait référence à l'exécution des instances de processus, ainsi qu'à leur surveillance et leur maintenance.

iv) La phase d'évaluation se base sur les différentes mesures prises au cours de l'exploitation et vise l'amélioration des modèles des processus.

Ce cycle de vie des processus d'affaires n'est bien entendu qu'une représentation simplifiée. En réalité, les activités attachées aux différentes phases peuvent chevaucher plus

d'une phase. Ainsi, il peut arriver pendant la phase d'exploitation que des activités de conception soient effectuées, amenant les processus exploités à subir des modifications. Le changement dans les processus en phase d'exécution est le sujet abordé dans les prochains chapitres de ce mémoire.

Les sous-sections suivantes présentent les notations, langages et progiciels utilisés pour nos expérimentations.

1.2.1 Notation de modélisation

Pour modéliser les processus, nous avons utilisé la notation BPMN *(Business Process Model and Notation)* version 1.2 (OMG, 2010a) développée par l'OMG *(Object Management Group)*. Le premier objectif de cette notation est de proposer une notation graphique normalisée pour la modélisation des processus d'affaires. Cette notation est en train de devenir la norme acceptée en industrie, ainsi qu'en milieu académique.

Nous avons utilisé cette notation parce qu'elle permet de modéliser tant les processus d'affaires que leurs collaborations ; mais aussi (comme mentionné ci-dessous), dans une optique d'ingénierie dirigée par les modèles *(Model Driven Engineering)*, des outils de développement permettent de générer du code exécutable directement à partir de diagrammes BPMN.

Pour une description sommaire de BPMN, veuillez consulter l'appendice A.

1.2.2 Implémentation

Les systèmes de gestion de processus d'affaires (*Business Process Management System*, BPMS) offrent des environnements de développement (éditeurs, simulateurs, vérificateurs, générateurs de code, etc.) pour supporter les spécialistes dans leurs tâches. Typiquement, les modèles de processus d'affaires sont définis graphiquement avec une notation comme BPMN pour ensuite être implémentés dans un langage plus approprié pour l'exécution automatisée des processus, comme BPEL *(Web Services Business Process Execution Language)* (Alves et al., 2007).

Des générateurs de code peuvent être intégrés au BPMS pour générer du code BPEL à partir de diagrammes BPMN. Bien que BPMN soit une notation graphique, son formalisme, inspiré du π-calcul (Milner, Parrow et Walker, 1992), est assez rigoureux pour permettre de générer du code exécutable, en ce qui concerne le flot des activités. Des services Web sont associés aux activités supportées par les partenaires, leurs définitions permettent de générer le code pour effectuer le couplage entre un processus et ses partenaires. Des outils intégrés de *data mapping* permettent de décrire les manipulations des données. Nous avons utilisé dans le cadre de nos recherches *Intalio Designer* (Intalio, 2010b) pour modéliser nos processus avec BPMN et générer du code BPEL.

Les processus BPEL contiennent toutes les informations nécessaires pour l'exécution des processus automatisés faisant partie d'une architecture fondée sur les services (*Service Oriented Architecture*, SOA). Les données sont décrites en XML *(Extensible Markup Language)* (Bray et al., 2008) et spécifiées dans des schémas XSD (*XML Schema Definition*) (Fallside et Walmsley, 2004). Les communications des processus entre eux et avec les ressources applicatives se font grâce à des interfaces de services Web définis avec WSDL *(Web Service Definition Language)* (Christensen et al., 2001).

Il faut tenir compte de plusieurs aspects (ou perspectives) pour implanter des processus automatisés en entreprise. Une description des processus automatisés avec BPEL organisée selon les perspectives des processus, des services et des données est disponible en appendice B.

1.2.3 Exécution

L'implémentation des processus terminée, les modèles (ou schémas) des processus, accompagnés des artefacts additionnels nécessaires à leur exécution, sont déployés dans le ou les BPMS. Les schémas des processus sont utilisés par une composante des BPMS, appelée le moteur des processus, pour exécuter les processus automatisés. Le moteur de processus génère une instance pour chaque exécution d'un processus automatisé.

Le BPMS permet aussi à l'opérateur de consulter ces instances afin de connaître à tout moment les états des processus ou d'exercer un contrôle sur ceux-ci. Habituellement,

il permet également d'effectuer différentes analyses pour l'amélioration continue des processus.

Nous avons utilisé le produit *BPMS Intalio, Community Edition* (Intalio, 2010a) et modifié le moteur de processus *Apache ODE* (The Apache Software Foundation, 2010) pour réaliser notre approche.

1.3 Orchestration vs chorégraphie

Un processus d'affaires, selon la définition 1.1, est géré par une seule entreprise. Le BPMS agit donc comme un agent qui centralise le contrôle des activités du processus, tout comme un chef d'orchestre dans la vie réelle dirige la performance de plusieurs musiciens. On parle alors d'*orchestration* des processus, ou orchestration des services Web lorsque les activités sont accédées via des services Web. BPEL est principalement un langage d'orchestration de services Web.

Les processus d'affaires d'une organisation interagissent fréquemment avec des processus d'autres organisations. Ces interactions peuvent être exprimées dans une *chorégraphie* de processus, ou chorégraphie de services Web lorsque les échanges se font via des services Web. Une chorégraphie implique l'absence d'un agent centralisateur qui contrôle les échanges, comme les danseurs d'une chorégraphie dans la vie réelle. Elle permet d'exprimer une vue globale des échanges de messages entre processus collaboratifs, ainsi que les conditions de synchronisation et contraintes de réalisation.

La figure 1.2 montre le lien entre orchestration et chorégraphie. La chorégraphie s'intéresse uniquement aux échanges externes entre les orchestrations de processus.

Typiquement, plusieurs processus peuvent participer à la réalisation d'un certain nombre d'objectifs d'affaires. Cette collaboration doit être spécifiée à l'intérieur d'une chorégraphie qui sert de contrat entre les parties, sans toutefois spécifier les activités internes à chaque participant. Des langages et notations comme BPMN, les diagrammes de communication UML *(Unified Modeling Language)* (OMG, 2010b), BPEL4Chor *(BPEL For*

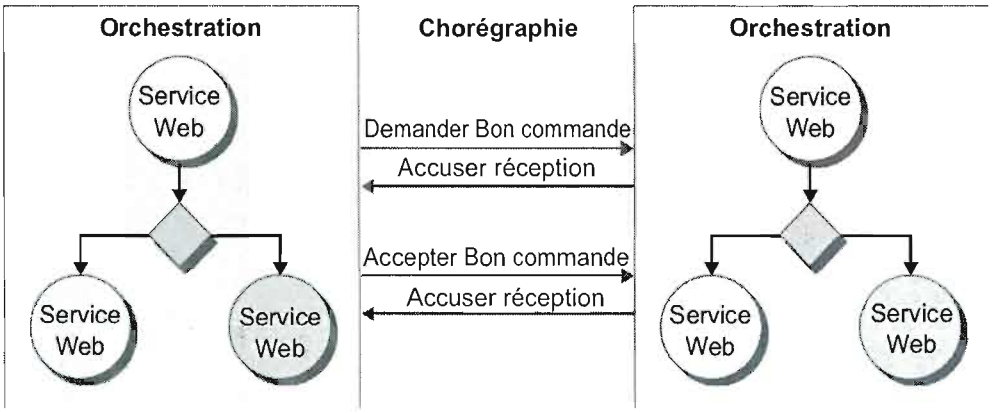

Figure 1.2 Orchestration vs chorégraphie. Figure adaptée (Peltz, 2003).

Choreography) (Decker et al., 2007), WS-CDL *(Web Services Choreography Description Language)* (Kavantzas et al., 2004) ou *Let's Dance* (Zaha et al., 2006a) peuvent servir à exprimer de tels contrats.

Après avoir défini les termes utilisés dans la rédaction de ce mémoire, nous avons présenté le cycle de vie des processus d'affaires, ainsi qu'introduit les notations, langages et progiciels utilisés pour nos expérimentations. Nous avons également établi la distinction entre l'orchestration et la chorégraphie des processus d'affaires. Le prochain chapitre présente les différents types de changements possibles dans des processus automatisés.

Chapitre II

TYPES DE CHANGEMENTS

Dans ce chapitre, nous présentons une classification des changements effectués sur des processus automatisés, avec des exemples à différents niveaux. Le scénario d'un centre de distribution y est introduit, et des modifications sont apportées à ce scénario afin de démontrer les types de changements.

2.1 Classification du changement

Les changements dans les processus automatisés peuvent être classifiés selon trois dimensions : le temps, la portée et la perspective (Khriss et al., 2008).

Temps

On distingue deux temps différents pour appliquer les changements : le temps de conception *(design time)* et le temps d'exécution *(run time)*. Le temps de conception inclut les phases de conception et d'analyse et la phase de configuration. Le temps d'exécution est associé bien sûr à la phase d'exploitation.

Pendant le temps de conception, les changements s'effectuent sur les schémas des processus et les artefacts complémentaires aux processus — il s'agit de *changements statiques*. Pendant le temps d'exécution, les changements sont appliqués à même les instances des processus en cours d'exécution — il s'agit alors de *changements dynamiques*.

Portée

Les changements peuvent affecter toutes les instances des processus ou seulement une partie. Par exemple, un changement peut ne s'appliquer qu'aux instances d'un processus traitant un certain type de clients ou faisant affaire avec une organisation particulière. D'autres changements peuvent affecter toutes les instances d'un processus.

Perspective

Cette dimension comprend la *perspective des processus*, la *perspective des services* et la *perspective des données*. En pratique, il est rare qu'un changement n'affecte qu'une seule perspective. Les sections 2.3, 2.4 et 2.5 présentent des exemples concrets de changements dans chacune de ces perspectives.

2.2 Scénario d'un centre de distribution

Pour illustrer les types de changements, nous allons introduire le scénario simple d'un centre de distribution de produits manufacturés. Le processus démarre lorsqu'un acheteur passe une commande de produits au département des ventes du centre de distribution. Le processus demande alors au crédit l'autorisation de porter le montant de la commande au crédit du client. Sur réponse affirmative, la commande est préparée en entrepôt, puis livrée par un transporteur indépendant. Un accusé de réception est transmis à l'acheteur à la fin du processus. Advenant un refus de crédit, la commande est annulée aussitôt et l'acheteur en est avisé.[1]

Le département de crédit peut être un service interne au centre de distribution, mais il peut s'agir aussi d'une institution financière offrant du crédit aux entreprises. L'entrepôt pourrait bien être une compagnie indépendante offrant les services d'entreposage et d'emballage. De même, le transporteur pourrait être une organisation comme *Federal Express* ou *Postes Canada*.

---

[1] Ce scénario est une variante du *Bookshop* (Coleman, 2005).

La figure 2.1 présente le diagramme BPMN du processus du centre de distribution. Seul le processus du département des ventes est exprimé par le diagramme. Les détails internes des autres processus sont laissés aux soins des organisations qui en sont responsables. Ce diagramme représente à la fois le processus des ventes et le contrat de collaboration entre les différents intervenants : *Acheteur, Ventes, Crédit, Entrepôt* et *Transporteur*. Les flèches pointillées représentent les messages échangés entre partenaires.

Un cercle simple représente un événement qui démarre le processus, par exemple « Recevoir une commande » est un événement avec message qui démarre le processus des ventes. Un cercle double est un événement intermédiaire qui survient au cours du flot des activités. Dans le diagramme, les deux événements intermédiaires attendent l'arrivée d'un message, chacun leur message respectif. Un cercle gras représente la fin du processus. Deux événements peuvent terminer le processus des ventes : « Accuser réception » ou « Annuler la commande ».

Le losange placé à la suite d'« Effectuer une demande de crédit » signifie qu'il y a une décision à prendre, selon la réponse du crédit.

Ce scénario sera utilisé tout au long du chapitre pour illustrer les différents types de changements. Pour plus de détails concernant la notation BPMN, veuillez consulter l'appendice A.

2.3 Changements au niveau des processus

Modifions maintenant le processus des ventes du centre de distribution pour effectuer les trois activités suivantes en parallèle : « Effectuer une demande de crédit », « Emballer le produit » et « Arranger la livraison ». Pour gagner du temps, l'emballage et la livraison débutent avant même d'avoir obtenu la réponse du crédit. Des activités de compensation sont ajoutées pour annuler l'emballage et la livraison au cas où la demande de crédit serait refusée ; la commande est alors annulée. Si le crédit est autorisé, le processus se termine normalement. La figure 2.2 donne le nouveau diagramme BPMN de ce scénario modifié.

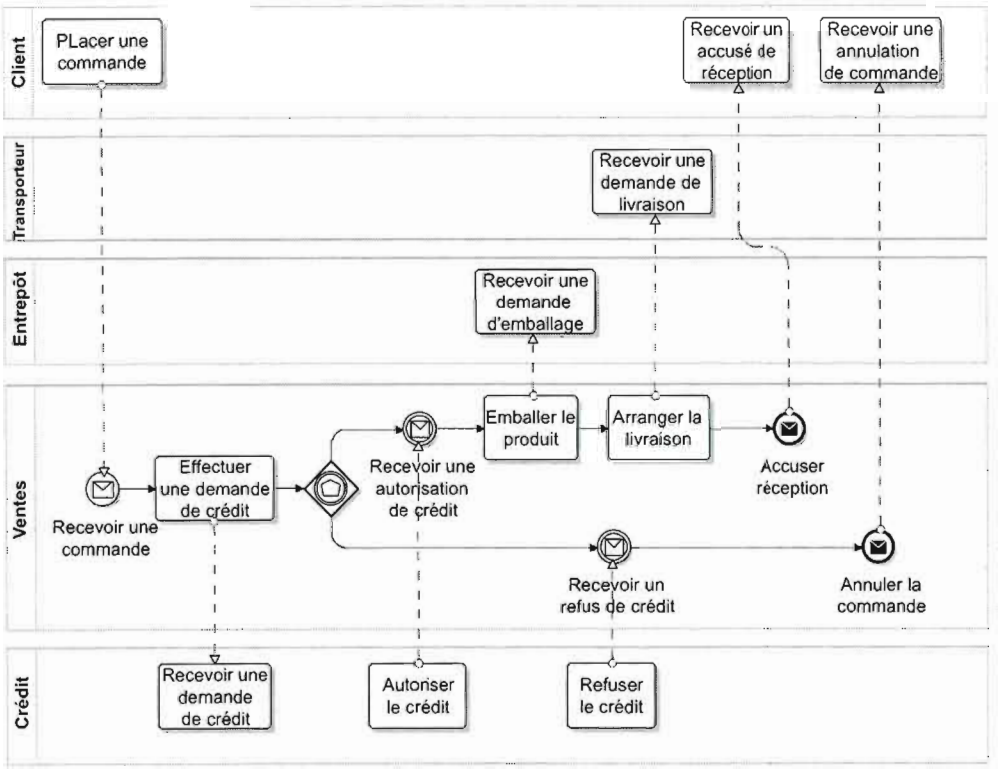

Figure 2.1 Scénario du centre de distribution.

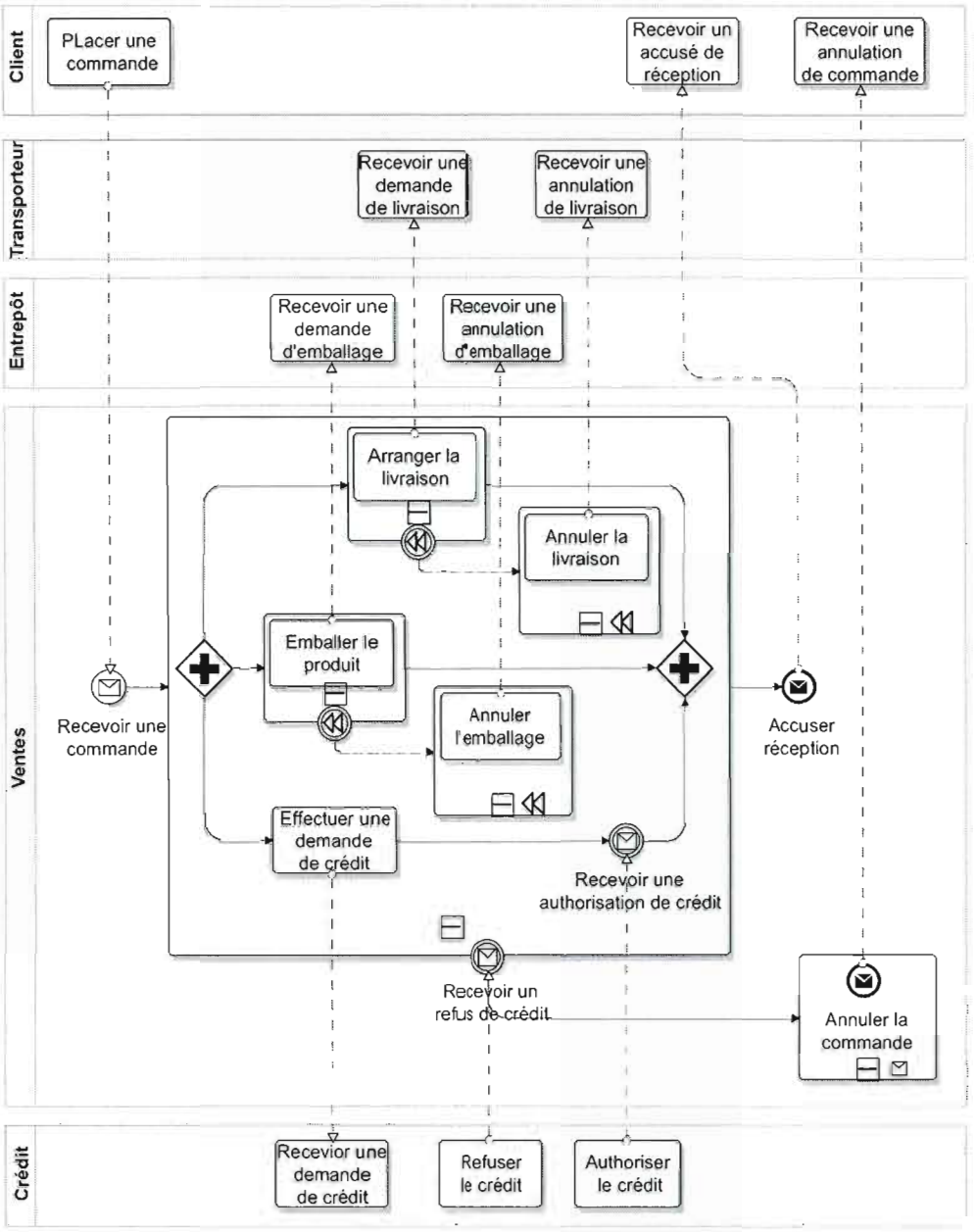

Figure 2.2 Scénario modifié du centre de distribution.

En observant les activités ajoutées par ce nouveau schéma (en gris sur le diagramme), nous pouvons constater les effets d'un tel changement sur la chorégraphie globale. Ces changements amènent les fournisseurs *Entrepôt* et *Transporteur* à modifier leurs processus respectifs afin de supporter de nouvelles opérations d'annulation (perspective des processus). Leurs interfaces de services Web devront être modifiées pour supporter les nouvelles activités en question (perspective des services). De nouveaux types de messages doivent aussi être ajoutés à la configuration des processus (perspective des données).

Supposons que le processus des ventes tel que décrit dans le scénario initial du centre de distribution (figure 2.1) s'exécute déjà depuis plusieurs mois. Imaginons que plusieurs instances de ce processus sont en cours d'exécution au moment où il est modifié (figure 2.2). Disons que des milliers de commandes sont en cours de traitement. Introduire dynamiquement des changements lorsque plusieurs instances de processus collaborent peut entraîner des problèmes difficiles à surmonter, en particulier lorsque les contrats ont été modifiés. La complexité des problèmes augmente selon le nombre de processus impliqués dans la chorégraphie. Il faut s'assurer que les impacts des modifications sur une instance soient répercutés sur les autres instances collaboratrices.

L'approche que nous proposons dans ce mémoire permet d'appliquer des changements à des processus collaboratifs, en tenant compte de tous les participants à un contrat et de toutes les instances en cours d'exécution, et ce, même sur des systèmes répartis dans plusieurs organisations.

2.4 Changements au niveau des services

Les services sont à la base de toute l'infrastructure des processus BPEL. Pour modéliser la couche service, la spécification BPEL utilise la notion de partenaire (*partner*). Un partenaire est un fournisseur ou un consommateur de service. Le processus lui-même est aussi un partenaire.

Une relation (*partner link*) est la description d'une interaction entre deux partenaires, où chaque partenaire est assigné à un rôle particulier. Les rôles sont associés aux types

de ports définis dans les définitions WSDL des services Web. C'est de cette manière que se fait le lien entre WSDL et BPEL.

Pour plus de détails au sujet de la perspective des services du langage BPEL et de sa relation avec WSDL, veuillez consulter la section B.3 en appendice. Pour des exemples de définitions d'interfaces de services Web avec WSDL, veuillez consulter la section C.3 en appendice également.

2.4.1 Exemple de changement de point de service

Une référence de point de service est une adresse physique à laquelle on peut rejoindre un service particulier. Une des fonctionnalités qui permet d'exploiter des processus dynamiques est celle qui consiste à modifier pendant l'exécution les références de point de service. Cette particularité est appelée la liaison dynamique *(dynamic binding)*. Ce type de changement est supporté par les BPMS modernes et fait même partie de la spécification BPEL.

Il est possible d'assigner une nouvelle référence de point de service à une relation BPEL explicitement dans le code. Le code du listage 2.1 est extrait du manuel de l'utilisateur du moteur de processus *Apache ODE* (The Apache Software Foundation, 2010). Deux exemples de la construction BPEL `<copy>` sont présentés dans ce code. Dans le premier exemple, l'adresse du point de service est passée dans un `<service-ref>`. Dans le deuxième, l'adresse est placée dans un `EndpointReference` — construction importée de la spécification *WS-Addressing* (Box et al., 2004).

Ce code en réalité ne sert pas à grand-chose, puisque l'adresse du point de service est inscrite en clair dans le code. Il est donné en guise d'exemple seulement. Le vrai dynamisme devrait nous permettre de modifier les références de point de service pendant l'exécution sans avoir à modifier le code. Voici comment s'y prendre.

Il faut d'abord, quelque part dans un document WSDL, définir un message de type `EndpointReference` (listage 2.2).

Listage 2.1 Exemples d'attributions de références de point de service à une relation BPEL

```
<assign>
  ...
  <copy>
    <from>
      <literal>
        <service-ref>
          <soap:address
              location="http://latece.uqam.ca:8080/ode/dynresponder"/>
        </service-ref>
      </literal>
    </from>
    <to partnerLink="responderPartnerLink"/>
  </copy>
  ...
  <copy>
    <from>
      <literal>
        <service-ref>
          <wsa:EndpointReference
              xmlns:wsa="http://www.w3.org/2005/08/addressing">
            <wsa:To>http://latece.uqam.ca:8080/ode/dynresponder</wsa:To>
          </wsa:EndpointReference>
        <service-ref>
      </literal>
    </from>
    <to partnerLink="responderPartnerLink"/>
  </copy>
  ...
</assing>
```

Listage 2.2 Message WSDL de type *EndpointReference*

```
<wsdl:message name="EndpointMsg">
  <wsdl:part name="payload" element="wsa:EndpointReference"/>
</wsdl:message>
```

Pour utiliser ce type de message, une ligne comme celle ci-dessous doit être ajoutée aux déclarations de variables dans le code BPEL :

```
<bpel:variable name="endpointMsg" type="tns:EndpointMsg"/>
```

Supposons qu'un serveur de règles d'affaires (voir section 3.3) a retourné la valeur de l'adresse à utiliser pour le point de service du transporteur. Cette adresse de type `EndpointReference` a été stockée dans la variable `endpointMsg` qui vient d'être déclarée. Cette valeur (la référence du point de service) peut maintenant être affectée de façon dynamique au rôle du transporteur, en l'affectant directement à la relation nommée `ventesTransporteurPlkVar` (listage 2.3). Le système sait implicitement qu'il s'agit de l'adresse du partenaire externe de la relation.

Listage 2.3 Attribution dynamique d'une référence de point de service

```
<assign>
  <copy>
    <from variable="endpointMsg" part="payload"/>
    <to partnerLink="ventesTransporteurPlkVar"/>
  </copy>
</assign>

<bpel:invoke partnerLink="ventesTransporteurPlkVar"
    portType="Transporteur:PourVentes" operation="Arranger_Livraison"
    inputVariable="arranger_LivraisonMsg"
    name="Transmettre_Bon_Livraison"/>
```

Voilà du moins comment on peut s'y prendre pour le moteur de processus BPEL *Apache ODE*. Chaque moteur a ses propres particularités. *Apache ODE* permet entre autres d'assigner directement une valeur de type `string` à un `partnerLink` afin de simplifier les manipulations.

Pour des exemples de code exécutable BPEL utilisant la liaison dynamique, vous pouvez consulter les listages des processus *Master* (section C.1) et *Slave* (section C.2) placés en appendice.

2.4.2 Exemple de changement de type de service

Chaque participant dans une chorégraphie doit implémenter sa part du contrat. Pour ce faire, des interfaces sont fournies définissant les différentes opérations accessibles chez un fournisseur. La définition d'une interface de service est le type de ce service. Typiquement, avec des processus BPEL, les types de services sont définis dans des documents WSDL.

Par exemple, dans le scénario initial du centre de distribution (figure 2.1), le service du transporteur fournit l'opération `Arranger_Livraison` et le service de l'entrepôt fournit l'opération `Emballer_Commande`, afin de supporter les activités du processus des ventes. Le service du processus des ventes fournit à son tour l'opération `Placer_Commande` pour le client, et les opérations `Autoriser_Credit` et `Refuser_Credit` pour le crédit.

Pour que le transporteur et l'entrepôt puissent à l'avenir supporter les cas d'annulation d'une livraison (figure 2.2), de nouvelles définitions des services Web sont nécessaires, afin d'ajouter les nouvelles opérations `Annuler_Livraison` et `Annuler_Commande`. Il faut ensuite utiliser les bonnes versions des documents WSDL et s'assurer qu'il n'y ait pas de problèmes d'intégration.

En phase de conception, les problèmes d'intégration en grande partie sont résolus par des outils de validation statique, inclus dans les environnement de développement. Cependant, les difficultés surviennent quand il est nécessaire d'appliquer des changements de types de services aux instances de processus en cours d'exécution ; car cela implique qu'il faille modifier les paramètres de couplage entre services et processus, mais certains BPMS permettent quand même d'effectuer de tels changements dynamiquement.

2.5 Changements au niveau des données

Des schémas XML sont utilisés pour définir les types de données utilisés dans les processus BPEL et les services Web. Ils peuvent être partagés et importés au besoin. Un exemple de schéma XML peut être consulté en appendice (section C.4).

Supposons que le centre de distribution adopte un nouveau schéma XML pour le bon de commande. Ce schéma pourrait, par exemple, avoir été suggéré par une firme de consultation qui a convaincu les cadres de l'entreprise d'utiliser les normes EDI *(Electronic Data Interchange)* pour les échanges numériques. Le nouveau schéma du bon de commande aurait été inspiré de la norme *EDI 850 Purchase Order* (BISAC SCEDI committee, 2005) et remplacerait à l'avenir le bon de commande utilisé jusqu'à présent.

Le code du processus pour accéder aux éléments du bon de commande doit être modifié en conséquence. Par exemple, le code pour copier les valeurs du bon de commande vers la demande de crédit doit subir des transformations afin de traduire les *nouveaux* types de valeurs du bon de commande vers les *anciens* types attendus dans la demande de crédit.

Puisque les données sont utilisées pour représenter les états des processus et qu'elles font partie des définitions des interfaces de services, il n'arrive pas souvent que des changements au niveau des données ne concernent qu'une seule perspective. Après chaque modification, il faut s'assurer qu'il n'y a pas de problèmes d'intégration.

Les éditeurs inclus dans les BPMS permettent habituellement de valider les différents schémas et définitions de façon statique, un peu comme dans les environnements de développement intégrés des langages modernes. Ces outils de validation statique permettent aux développeurs de gagner du temps à l'intégration en détectant les erreurs en phase de conception. Certains BPMS permettent même de simuler l'exécution des processus avant de les déployer, ce qui permet de détecter des fautes qui n'étaient pas détectées statiquement, comme certains types de blocage ou de vivacité.

L'intégration des types de données est un problème courant lorsqu'on introduit des changements au niveau des données qui peut être résolu statiquement par des outils efficaces en phase de conception. Le vrai problème cependant survient, comme pour les types de services, lorsqu'on a besoin d'appliquer ces modifications aux instances de processus en cours d'exécution.

Nous avons présenté dans ce chapitre une classification du changement, avec des exemples de changement dans toutes les perspectives identifiées, sans toutefois en faire une liste exhaustive. L'objectif était de donner une idée au lecteur des implications à multiples niveaux que peuvent avoir les changements dans des processus automatisés. Le prochain chapitre aborde la question de l'adaptation dynamique des processus.

Chapitre III

ADAPTATION DYNAMIQUE DES PROCESSUS

Dans ce chapitre, nous discutons de l'adaptation dynamique des processus. Les diverses solutions amenées pour résoudre le problème peuvent se regrouper sous trois catégories : l'évolution des schémas, le changement spécifique d'instance et la flexibilité des schémas. Nous présentons des approches dans ces trois catégories. Nous identifions ensuite le problème de l'exactitude du changement lorsque des processus collaboratifs sont en jeu. Nous terminons en présentant les avantages et les contributions de notre approche collaborative.

La classification déjà amenée à la section 2.1 fait une distinction sur les types de changements par rapport au temps, à la portée et à la perspective. Des changements statiques peuvent être faits en conception sur les schémas des processus. Les changements dynamiques sont appliqués sur les instances des processus en cours d'exécution. Les changements peuvent affecter toutes les instances d'un processus ou seulement quelques instances spécifiques. Ces mêmes changements peuvent toucher à une ou à plusieurs des perspectives des processus, des services ou des données.

Cette classification est nécessaire pour déterminer quels types de fonctionnalités un BPMS devra supporter pour offrir la flexibilité requise pour un changement. Elle s'inspire des différentes classifications déjà proposées (van der Aalst et al., 1999; Casati et al., 2000; Kammer et al., 2000; Karastoyanova et Buchmann, 2004; Weber, Reichert et Rinderle-Ma, 2008). Nous avons gardé cette classification à l'esprit pour organiser et rédiger cette revue de littérature.

3.1 Évolution des schémas

Un schéma de processus, typiquement, est découpé en plusieurs activités, composites ou atomiques. Les activités composites sont aussi appelées sous-processus. Un processus peut être représenté (mais pas nécessairement) par un arbre, les activités atomiques étant ses feuilles. Certaines approches utilisent différentes variantes de graphes et de réseaux pour représenter les processus.

Peu importe la méthode utilisée pour représenter un processus, l'objectif principal dans un contexte d'évolution des schémas est de permettre des transformations sur les schémas, tout en assurant l'exactitude des schémas après chaque modification.

Un bon nombre d'approches pour l'adaptation dynamique des processus se concentrent sur l'évolution des schémas pour traiter les besoins de changement. Certains auteurs considèrent la migration des instances comme une évolution dynamique des schémas des processus, car il s'agit d'une répercussion dynamique sur les instances actives de modifications d'abord apportées statiquement sur les schémas.

La migration des instances consiste à faire passer les instances d'une version d'un schéma à l'autre, tout en assurant l'exactitude et la continuité des processus. Nous avons privilégié cette approche pour notre propre solution.

Dans l'approche de migration proposée par Casati et al. (1998), des primitives de changements sur les données et sur le flot d'exécution restreignent les modifications applicables sur les schémas. Ces primitives sont appliquées en respectant certaines règles qui permettent aux nouvelles instances de demeurer compatibles après la migration. S'il s'avérait impossible de migrer une instance vers un nouveau schéma, à cause des activités déjà exécutées, l'approche suggère un schéma temporaire. Lorsque la situation ne permet pas de passer par un schéma temporaire, des compensations peuvent s'effectuer sur les anciennes activités pour ramener l'instance à un état compatible au nouveau schéma.

Liu, Orlowska et Li (1998) introduisent un langage simple pour définir les politiques à suivre pour la migration des instances. Pour chaque activité de l'instance en exécution,

une action est donnée pour la rendre compatible avec le nouveau schéma. Il peut s'agir d'une compensation, d'un remplacement d'activité ou simplement d'exécuter l'ancienne activité comme si de rien n'était. L'administrateur du processus utilise ce langage pour définir les cas de migration et le système effectue par la suite la migration telle que définie. Un graphe de passage est généré à partir de ces politiques pour chaque processus afin de pouvoir déterminer les activités de chaque instance à exécuter lors de la migration. Ce graphe de passage est analysé pour vérifier l'exactitude des changements.

Kradolfer et Geppert (1999) basent aussi leur approche sur un ensemble d'opérations effectuées sur les schémas, mais les opérations spécifient plutôt comment passer d'une version d'un composant à une autre. Les versions d'un schéma de processus est constitué d'un arbre dont les composants sont des sous-processus et des activités. À chaque composant est associé un numéro de version, en commençant par [0] pour la première version, puis subséquemment 1, 2, 3... La particularité de cet arbre de versions est qu'il indique quelles opérations doivent être appliquées sur les composants pour passer d'une version à l'autre. La migration d'une version vers l'autre d'un processus se fait en effectuant ces opérations sur les composants qui le composent. Des règles de transformation sont imposées lors de la migration pour assurer la validité de chaque opération. Cette approche permet de passer d'une version à n'importe laquelle de ses descendants par dérivations subséquentes, ou *vice versa* en effectuant les opérations inverses.

ML-DEWS (Ellis et Keddara, 2000) est présenté comme un langage de modélisation du processus de migration. Le langage sert à décrire le contexte de migration, les règles d'application, les opérations à effectuer sur les activités des instances, etc. L'objectif des auteurs est de proposer un langage et un métamodèle servant à définir toute migration.

Contrairement aux approches présentées jusqu'ici, celle de Sadiq (2000) fournit, elle, un algorithme générique de migration. Il n'est donc pas nécessaire de spécifier les différentes opérations à effectuer sur les instances pour réaliser la migration, le schéma modifié suffit. Un graphe de compatibilité *(compliance graph)* est généré pour chaque instance à partir de sa trace d'exécution. Pour chaque activité modifiée, une activité de compensation

ou de remplacement est insérée dans le graphe. Le graphe de compatibilité est ensuite exécuté à la place de l'instance. Les arcs d'un noeud dans le graphe du processus déterminent si une activité a été modifiée, permettant ainsi de considérer les changements dans la structure du processus. L'avantage d'une telle méthode est bien sûr son algorithme générique de migration applicable à toutes les instances. L'intervention humaine est tout de même nécessaire pour associer des activités de compensation et de remplacement avec chacun des noeuds de l'ancien schéma. Par contre, l'approche se concentre sur la perspective des processus seulement, sans considérer les autres perspectives.

Qiu et Wong (2007) proposent eux aussi une gestion automatique de la migration. Les instances sont réexécutées suivant les nouveaux schémas, mais les activités déjà complétées ne sont pas réexécutées. Trois conditions servent à identifier les activités *(nodes)* pouvant être ignorées lors de la réexécution des instances : (1) elles n'ont pas changées, (2) elles ont été complétées par l'ancienne instance et (3) elles n'ont pas besoin d'être exécutées de nouveau. Pour vérifier la dernière condition, toutes les activités faisant partie du chemin pour s'y rendre doivent avoir été ignorées également. L'approche prend en considération toutes les perspectives, incluant la perspective des données. Aucune intervention humaine n'est nécessaire pour diriger les opérations lors de la migration. Par contre, la compensation n'est pas utilisée pour amener les instances à un état compatible, advenant que des activités déjà exécutées ne puissent être ignorées.

L'approche de migration que nous proposons s'apparente aux deux dernières par son algorithme générique, la réexécution des instances, les activités ignorées et la compensation. Nous reviendrons plus tard sur les avantages de notre approche (section 3.5).

3.2 Changement spécifique d'instance

D'autres approches appliquent les changements directement sur les instances, sans modifier les schémas de façon générale. Elles permettent d'adapter des instances spécifiques de processus pendant qu'elles sont en cours d'exécution. Des restrictions sont imposées sur les opérations acceptables afin de garder l'exactitude des processus après chaque modifi-

cation. Aux fins de vérification lors des changements, les processus sont habituellement basés sur des modèles formels.

$ADEPT_{flex}$ (Reichert et Dadam, 1998) est un bon exemple de ce type d'approche. Les auteurs mentionnent deux préoccupations principales lors des changements : la perspective des processus *(flow of control)* et la perspective des données *(flow of data)*. Pour chaque perspective, un ensemble de règles et de primitives de changements basées sur les schémas sont fournies assurant l'exactitude des changements.

Zhao et Liu (2007) proposent une approche basée sur les différentes versions de chaque activité d'un schéma. L'approche définit un certain nombre d'opérations acceptables (ajout, suppression, remplacement) modifiant les noeuds et les arcs d'un graphe de transformation. Ce graphe évolue après chaque modification, en gardant la trace de toutes les versions des activités du schéma. Il est entretenu pour chaque schéma et le système l'utilise pour exécuter les activités appropriées des instances.

Le système *Bonita* (Charoy, Guabtni et Faura, 2006) offre une approche particulière. Le schéma et l'instance sont un seul et même objet dans le système. Les processus sont définis à la volée pour chaque instance. Il est possible de partir d'un autre processus pour démarrer ou modifier une instance — très utile pour explorer des processus coopératifs non spécifiés d'avance. L'exactitude des processus n'est pas une inquiétude puisque l'on suppose que les problèmes peuvent être réglés à l'exécution ; tous les types de changements sont donc permis et tous les participants peuvent modifier le processus. Pour ce faire, le moniteur de processus est également un éditeur. Un système de contrôle de versions s'occupe de l'intégrité des données et des scripts peuvent être associés à chaque activité pour déclencher des opérations automatisées.

3.3 Flexibilité des schémas

Les approches présentées jusqu'ici présupposent qu'on ne connaisse pas les besoins de changement *a priori*. Une autre approche d'adaptation dynamique consiste à prévoir les changements dès la conception des processus, en construisant des schémas plus flexibles.

La liaison dynamique (déjà mentionnée à la section 2.4) est considérée comme une forme de flexibilité des schémas, ainsi que le changement de type de service (mentionnée également dans la même section), puisque ces modifications s'effectuent sur chaque instance à l'exécution. Quelques auteurs traitent de cette forme de dynamisme associée à la perspective des services (Karastoyanova et Buchmann, 2004; Casati et al., 2000).

Une autre forme de flexibilité consiste à extraire les règles d'affaires des processus et à les stocker sur un serveur (Goh, Koh et Domazet, 2001). En modifiant les règles d'affaires sur ce serveur, il est possible de modifier, aux points identifiés comme *décisions d'affaires*, le comportement d'un processus pendant son exécution. Ces décisions d'affaires sont des activités du processus qui interrogent le serveur de règles d'affaires. Chaque règle est composée d'un événement déclencheur, d'une condition et d'une action, pouvant inclure le retour d'un résultat (décision), le déclenchement d'autres événements ou l'appel à des applications. Les règles d'affaires peuvent être juxtaposées pour former une chaîne de décisions, formant une espèce de sous-processus en dehors de la logique de l'instance.

Weber, Reichert et Rinderle-Ma (2008) identifient quatre patrons de changements pouvant être classifiés dans cette catégorie d'approche par flexibilité des schémas : *Late Selection of Process Fragments, Late Modeling of Process Fragments, Late Composition of Process Fragments* et *Multi-instance Activity*.

L'activité multi-instance est considérée comme un patron de changements à cause des possibilités de dynamisme qu'elle offre. Par exemple, nous avons utilisé l'activité multi-instance dans nos processus pour effectuer des liaisons dynamiques avec plusieurs fournisseurs d'un même service (voir figure 5.2).

Sadiq, Sadiq et Orlowska (2001) proposent un modèle de schéma défini partiellement, avec des *pockets of flexibility* à concrétiser lors de l'exécution. Une activité spéciale nommée *build activity* fournit les règles à suivre pour générer à l'exécution la partie concrète du schéma, à partir d'un ensemble de canevas prédéterminés. Une approche semblable est utilisée dans une autre solution (Adams et al., 2006; Adams et al., 2007), mais en passant par un service externe pour les fragments des processus (comme

pour le serveur de règles d'affaires). Orrïens, Yang et Papazoglou (2003) proposent une composition dynamique de services Web (processus BPEL) à partir de règles d'affaires fournies par l'opérateur en cours d'exécution.

Le serveur de règles d'affaires, la liaison dynamique et, bien sûr, l'activité multi-instance sont devenus des fonctionnalités courantes de plusieurs BPMS commerciaux, tentant d'offrir plus de flexibilité à leurs clients. Malheureusement, la plupart de ces BPMS commerciaux ne se limitent qu'à ces trois fonctionnalités (Intalio, 2011; Microsoft, 2011).

3.4 Exactitude du changement

Presque tous les travaux présentés aux sections précédentes traitent de l'exactitude du changement. Il s'agit d'une préoccupation importante lors de changements dans des processus en exécution. Une analyse de différentes méthodes utilisées pour assurer l'exactitude du changement est présentée par Rinderle, Reichert et Dadam (2004).

L'exactitude des processus collaboratifs est aussi un problème important abordé dans la littérature. Wodtke et Weikum (1997) présentent une théorie appliquant une technique nommée *Orthogonalization of State Charts* pour distribuer un processus global (défini par un diagramme d'états) sur un ensemble de composants orthogonaux. L'équivalence entre la forme globale et celle distribuée est prouvée formellement. Le diagramme global peut alors servir à la vérification avant la distribution vers ses composants.

Le langage de chorégraphie WS-CDL (Kavantzas et al., 2004) a aussi été proposé dans cette optique (Carbone, Honda et Yoshida, 2007). Les membres du groupe *π4 Technologies Foundation* (2011) ont développé le projet *Pi4SOA* basé sur ce langage. L'outil permet de modéliser une chorégraphie, la vérifier, la simuler et l'implémenter avec plusieurs processus distribués.

Pour autant que nous sachions, aucune étude n'a encore abordé le problème d'exactitude du changement lorsqu'on effectue des changements dans une chorégraphie en cours d'exécution.

3.5 Proposition

La coordination du changement lorsqu'il s'agit de processus collaboratifs est un autre défi de recherche identifié dans un numéro spécial du journal *Information Systems Frontiers* (Liu, Li et Zhao, 2009). Notre *Change Protocol for Collaboration* apporte d'abord une solution statique au problème. L'algorithme de migration que nous proposons y apporte une solution dynamique.

Les changements effectués dans une chorégraphie *en cours d'exécution* doivent être répercutés dynamiquement sur toutes les instances des processus la réalisant. Il s'agit d'un problème difficile à résoudre.

Pour y remédier, nous proposons une migration des instances qui tient compte des collaborations actives : nous effectuons la migration considérant un ensemble d'instances interreliées, pouvant être distribuées sur plusieurs serveurs distincts. La synchronisation entre les instances en migration est réalisée de façon dynamique. Notre algorithme générique effectue une réexécution des instances en ignorant les activités déjà complétées et non modifiées. Il utilise la compensation pour annuler le travail qui n'est plus applicable. La migration s'effectue sans intervention humaine, pourvu que des activités de compensation aient été spécifiées au préalable.

Nous avons introduit dans ce chapitre les diverses approches d'adaptation dynamique des processus, en insistant sur la migration des instances. Les autres approches ont été présentées pour donner un aperçu de la diversité des solutions proposées et de la portée du problème. Nous avons identifié le problème du changement dans des processus collaboratifs et introduit notre solution pour résoudre ce problème. Le prochain chapitre fait la description de cette solution.

Chapitre IV

ADAPTATION DYNAMIQUE DES PROCESSUS COLLABORATIFS

Dans ce chapitre, nous présentons d'abord le *Change Protocol for Collaboration* (CPC) et ses caractéristiques. Nous faisons ensuite la description de notre algorithme de migration des instances. Les descriptions du CPC et de l'algorithme de migration présentées dans ce chapitre sont adaptées d'un premier article paru précédemment (Khriss et al., 2008).

Comme démontré au chapitre 2, les changements au niveau des différentes perspectives peuvent avoir des répercussions sur les chorégraphies. Les changements introduits dans le scénario modifié du centre de distribution (figure 2.2) affectent le contrat entre *Ventes* et *Entrepôt*, et entre *Ventes* et *Transporteur*. Les processus exploités du côté de chaque fournisseur doivent être modifiés afin de supporter les nouvelles fonctionnalités.

Il y a principalement deux problèmes à considérer lors de changements dans une chorégraphie de processus :

1. la synchronisation du changement entre les partenaires ;
2. la migration des instances *en cours d'exécution* vers les nouveaux schémas.

Nous proposons une nouvelle approche pour régler ces deux problèmes. Elle consiste en :

1. un protocole de changement pour les chorégraphies de processus ;
2. un algorithme de migration des instances pour les collaborations actives.

4.1 Aperçu du protocole

CPC est un protocole de validation en deux phases. Il consiste en cinq messages échangés entre un maître (initiateur du changement) et ses esclaves (les autres partenaires devant réagir au changement). Ces messages sont les suivants : *Notify*, *Accept*, *Deny*, *Proceed* et *Cancel*.

Un ou l'autre des scénarios suivants a lieu, si le changement est accepté ou refusé (figures 4.1 et 4.2). Lorsqu'un des partenaires (maître m) désire effectuer un changement qui affecte le contrat, il notifie les autres (esclaves $s1$, $s2$ et $s3$). En recevant le message *Notify* ($1a$, $1b$ et $1c$), les esclaves entrent dans l'état *Notifié* ; les esclaves sont initialement à l'état *Latent*. Un esclave peut accepter d'adapter ses processus par le renvoi du message *Accept* ($2a$, $2b$ et $2c$) ou refuser par le message *Deny* ($2b'$). Lorsqu'un esclave accepte le changement, il entre dans l'état *Accepté*, sinon il prend l'état *Refusé*.

Le maître a donc deux cas à considérer. S'il reçoit le message *Accept* de la part de tous les esclaves, il leur indique qu'ils peuvent procéder au changement ($3a$, $3b$ et $3c$). À la réception du message *Proceed*, les esclaves entrent dans l'état *Procédé*. Le deuxième cas survient lorsque le maître reçoit le message *Deny* de la part d'un des esclaves. Dans ce cas, le maître informe tous les esclaves que le changement est annulé par le message *Cancel* ($3a'$, $3b'$ et $3c'$). Les esclaves entrent donc dans l'état *Annulé*[1].

Précisons que le fait d'annuler un changement ne donne pas nécessairement un droit de *veto* aux esclaves ; mais plutôt, si un esclave refuse un changement, il ne pourra pas continuer sa participation au contrat. Le maître n'aura qu'à soumettre le changement de nouveau dès qu'il aura remplacé le partenaire sortant.

La figure 4.3 montre un diagramme d'activités UML décrivant le comportement du maître, et la figure 4.4 montre le diagramme d'états UML pour l'esclave. Les actions

[1] S'il advenait que le message *Proceed* ou *Cancel* ne venait pas de la part du maître, les esclaves resteraient à l'un ou l'autre des états *Accepté* ou *Refusé*. Après un certain temps, des avertissements pourraient être transmis par les systèmes aux administrateurs.

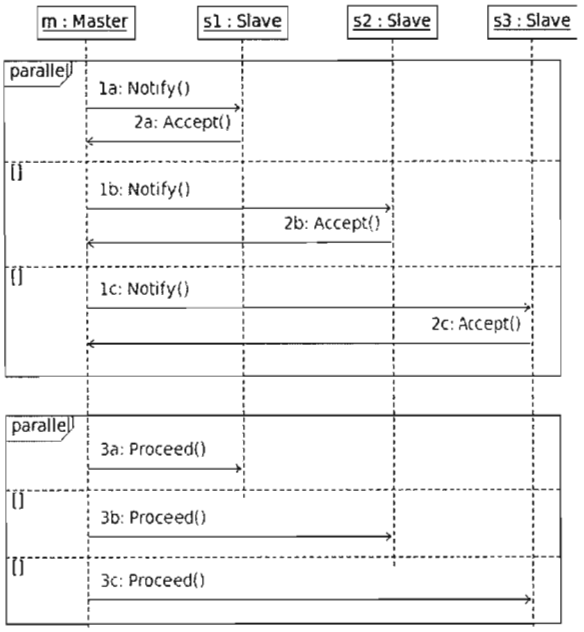

Figure 4.1 Échanges entre maître et esclaves lorsque le changement est accepté par tous les esclaves.

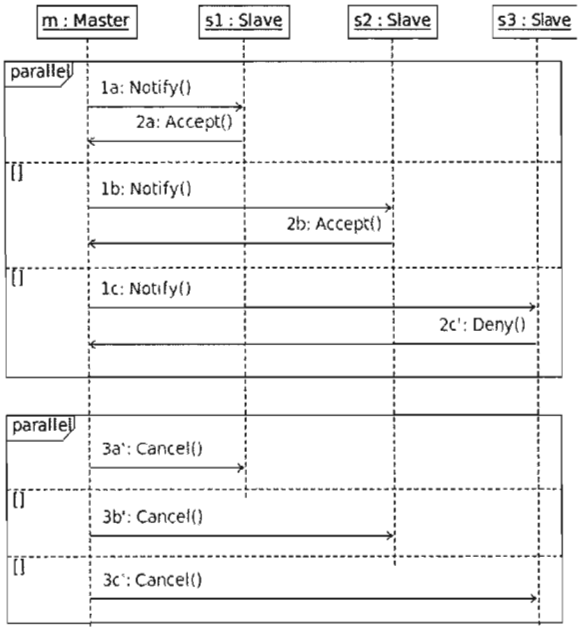

Figure 4.2 Échanges entre maître et esclaves lorsque le changement est refusé par un (ou plusieurs) des esclaves.

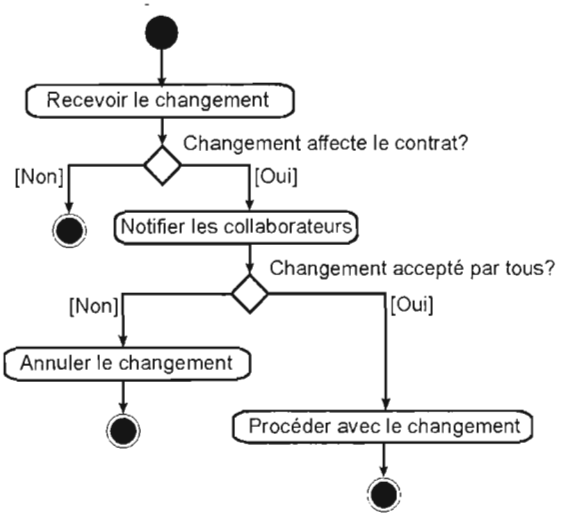

Figure 4.3 Comportement du maître. Figure adaptée (Khriss et al., 2008).

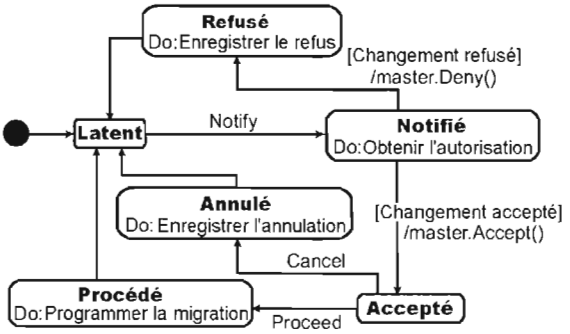

Figure 4.4 États des esclaves. Figure adaptée (Khriss et al., 2008).

effectuées par les esclaves *(Obtenir l'autorisation, Enregistrer le refus, Enregistrer l'annulation* et *Programmer la migration)* sont expliquées à la sous-section 4.3. À la fin de l'une ou l'autre des trois dernières actions, l'esclave retourne à l'état *Latent.*

4.2 Description des messages

Le message *Notify* contient les champs suivants :

- *messageID* : identifiant du message ;
- *effectiveDate* : date à laquelle le changement sera en vigueur ;
- *expirationDate* : date à laquelle ce processus sera expiré ;
- *scope* : étendue du changement, *schema* ou *instance* ;
- *runningInstances* : vrai ou faux, indique si le changement doit être appliqué aux instances déjà en cours d'exécution ;
- *instanceReferences* : obligatoire lorsque le changement ne concerne qu'un sous-groupe des instances d'un processus (*scope* = *instance*), le cas échéant, ce champ permet d'identifier les instances à modifier ;
- *contractRefID* : identifiant du contrat modifié ;
- *newContractSchema* : schéma du nouveau contrat ;
- *newContractRefId* : identifiant du nouveau contrat ;
- *masterAddress* : adresse du point de service du maître ;
- *slaveAddresses* : adresses des points de services des esclaves.

Les messages *Accept*, *Cancel* et *Proceed* contiennent les champs suivants :

- *messageID* : identifiant du message ;
- *notifyMessageRefID* : correspond à l'identifiant du message *Notify*.

Le message *Deny* contient les champs suivants :

- *messageID* : identifiant du message ;
- *notifyMessageRefID* : correspond à l'identifiant du message *Notify* ;
- *reasons* : texte donnant les raisons du refus.

Le schéma XML définissant les messages du CPC peut être consulté en appendice (section C.4).

4.3 Actions des partenaires esclaves

Quatre actions (voir figure 4.4) doivent être supportées par les partenaires esclaves.

L'action *Obtenir l'autorisation* demande une autorisation couvrant deux aspects. Tout d'abord, le changement doit être autorisé selon les politiques de l'organisation : il s'agit purement d'une décision d'affaires. Par la suite, une autorisation technique doit être obtenue. Cette dernière est obtenue après la vérification de l'exactitude du nouveau contrat. Si la vérification est satisfaisante, le système génère un ensemble de propositions pour corriger le processus local afin de satisfaire le nouveau contrat. Ce processus local doit ensuite être validé et complété manuellement par l'analyste. S'il advenait un problème incontournable lors de ces étapes empêchant l'intégration du changement, un message *Deny* est généré.

L'action *Programmer la migration* effectue la migration des instances actives de l'ancien vers le nouveau schéma du processus, mais uniquement lorsque l'étendue du changement concerne les instances (soit $scope = instance$, ou $scope = schema$ et $runningInstances = true$). La migration est décrite en détail à la section 4.4. Lorsque l'étendue du changement ne concerne que le schéma ($scope = schema$ et $runningInstances = false$), il

n'y a pas de migration à effectuer ; le nouveau schéma est programmé pour entrer en vigueur pour toutes les nouvelles instances du processus, selon les dates spécifiées dans les champs *effectiveDate* et *expirationDate* du message *Notify*.

Les actions *Enregistrer le refus* et *Enregistrer l'annulation* enregistrent ces événements dans un but administratif.

4.4 Migration des instances

La migration d'une instance $I1$ du schéma $S1$ vers le schéma $S2$ s'effectue en quatre étapes.

1. Créer une nouvelle instance $I2$ à partir du nouveau schéma $S2$.
2. Obtenir la *trace des échanges* d'$I1$, nommée T.
3. Compenser les activités d'échange complétées d'$I1$, de types *invoke*, ne faisant plus partie de $S2$ et réalisées par des partenaires qui ne supportent pas la migration[2] (appelés ci-après les *échanges non supportés*).
4. Exécuter conditionnellement $I2$ à partir de $S2$ et T.

La première étape est évidente. La seconde consiste à obtenir la trace des échanges de $I1$, de l'activité d'échange de démarrage jusqu'à la dernière activité d'échange exécutée au moment de sa suspension. Par exemple, pour une instance du processus des ventes du centre de distribution (figure 2.1), si la dernière activité d'échange exécutée est « Effectuer une demande de crédit », la trace contiendra les deux échanges suivants :

1. L'échange initial de l'Acheteur contenant le « Bon de commande » reçu par l'activité « Recevoir une commande ».

[2]Un partenaire ne pouvant pas supporter la migration survient lorsque ses activités ne s'exécutent pas à travers un BPMS. Par exemple, l'activité « Déterminer un itinéraire de voyage » pourrait être confiée au BPMS d'un partenaire, le BPMS exécuterait alors un processus pour déterminer l'itinéraire de voyage ; alors que l'activité « Effectuer une réservation de billets d'avion » pourrait bien s'exécuter chez un partenaire sans l'utilisation d'un BPMS.

2. L'échange vers le Crédit contenant la « Demande de crédit » envoyé par l'activité « Effectuer une demande de crédit ».

La troisième étape consiste à compenser, en ordre inverse de leur exécution, les activités des *échanges non supportés* de type *invoke* qui ne figurent pas dans le nouveau schéma $S2$.[3] Pour le scénario modifié du centre de distribution (figure 2.2), il n'y aurait rien à compenser puisque toutes les activités *invoke* se retrouvent dans le nouveau schéma.

La quatrième étape consiste à effectuer une exécution conditionnelle de la nouvelle instance $I2$ selon le schéma $S2$ et la trace T. L'exécution conditionnelle s'effectue comme d'habitude, sauf pour les cas d'exceptions indiqués par les règles suivantes[4].

Règle R1 Pour chaque *échange non supporté* de type *receive*, l'échange correspondant contenu dans la trace d'$I1$ est injecté dans $I2$.

Règle R2 Pour chaque activité *invoke* ou *reply* déjà complétée par $I1$ (appartenant donc à T), lorsqu'il s'agit d'un échange supporté par la migration, peu importe s'il y a modification des paramètres de l'échange, un message de synchronisation est envoyé au partenaire. Ce message informe le partenaire du progrès de la migration. Il permet ainsi de synchroniser la migration entre les différentes instances. Le message de synchronisation contient le message à envoyer au partenaire.

Règle R3 Pour une activité *invoke* déjà complétée par $I1$, lorsqu'il s'agit plutôt d'un *échange non supporté*, si le type de service et l'adresse du fournisseur n'ont pas changé et les paramètres en entrée sont toujours les mêmes, l'échange est ignoré. La valeur de retour contenue dans la trace T est utilisée pour la suite du processus. Si, par contre, le service ou le message a changé, l'ancienne activité *invoke* est compensée et la nouvelle est exécutée à sa place.

---

[3] BPEL permet de spécifier à même le schéma des activités de compensation pour chaque *invoke*.

[4] Comme le CPC, les règles de migration sont une idée originale d'Ismail Khriss *et al.* (2008). Une première version de ces règles a d'abord été présentée dans l'article précité, la version présente fait suite aux observations que nous avons faites lors de nos expérimentations

Règle R4 Pour l'activité *reply* déjà complétée par $I1$, lorsqu'il s'agit d'un *échange non supporté*, dans le cas où il n'y a pas de changement au service ou au message, l'échange est ignoré. Par contre, s'il y a changement, l'activité *reply* ne peut être compensée. Le système génère alors une faute. L'impact d'un tel changement doit être pris en considération chez le partenaire avant de reprendre la migration.

Règle R5 Recevoir un message de synchronisation a le même effet que recevoir le message qu'il contient. Le processus continue l'exécution s'il est en train d'attendre ce message ou le BPMS démarre une nouvelle instance du processus lorsqu'il s'agit d'une activité de démarrage d'instance (`createInstance=yes`).

Règle R6 L'activité *wait* qui attend pour une période de temps $P1$ est exécutée seulement pour une période égale à $P1 - P2$. $P2$ est égal au moment prévu pour l'exécution de la prochaine activité B duquel on soustrait le moment de l'exécution de la dernière activité A ($P2 = B - A$). $P2$ correspond donc au temps qui s'est écoulé entre les exécutions d'$I1$ et $I2$. L'activité *wait* est exécutée avant l'activité B.

Règle R7 Toutes les activités *invoke* concernant des *échanges non supportés*, qui ne sont pas déjà compensées, qui ont été complétées par $I1$ et qui ne peuvent être atteintes par $I2$, sont compensées dans l'ordre inverse de leur exécution.

Règle R8 Toutes les activités *reply*, qui ont été complétées par $I1$ et qui ne peuvent être atteintes par $I2$, sont considérées comme des cas possibles d'erreurs, mais seulement lorsqu'il s'agit d'*échanges non supportés*.

Règle R9 À la fin de la migration, les partenaires exploitant des processus ne faisant plus partie de la collaboration doivent en être avisés. Il peut s'agir de processus retirés du nouveau contrat ou ignorés lors de seconde exécution. L'instance $I1$ de chacun des processus retirés ou ignorés est alors compensée en entier, chez le partenaire, selon $R7$ et $R8$.

Règle R10 S'il advenait qu'une compensation critique ne puisse s'effectuer, la migration devrait être annulée.

Revenons maintenant à notre exemple du centre de distribution. Lors de la migration, le processus des ventes est donc réexécuté à partir du nouveau schéma. La nouvelle instance ($I2$) commence à être exécutée à partir du début. $I2$ attend le message de synchronisation venant de l'Acheteur ($R5$). À l'arrivée du message, la « Demande de crédit », l'« Avis d'emballage » et le « Bon de livraison » sont préparés. Puisque l'activité « Effectuer une demande de crédit » est présente dans la trace, un message de synchronisation est envoyé au crédit ($R2$). Le processus continue son cours normalement : les activités « Emballer le produit » et « Arranger la livraison » sont exécutées sans condition, et ainsi de suite jusqu'à la fin du processus. Il n'y a pas de compensation à effectuer (selon $R7$ ou $R8$) puisque toutes les activités exécutées d'$I1$ ont été atteintes par $I2$.

Nous avons fourni dans ce chapitre la description détaillée du CPC, avec les comportements du maître et des esclaves. Nous avons également expliqué notre approche de migration des instances. Le prochain chapitre se consacre à l'implémentation de notre preuve de concept.

Chapitre V

PREUVE DE CONCEPT

Ce chapitre traite de la mise en oeuvre du CPC et de l'algorithme de migration. Nous décrivons l'architecture du système CPC, ainsi que l'architecture du système de tests réalisé pour notre preuve de concept. Ensuite nous décrivons les cas de tests et la validation effectuée. Nous terminons par des discussions sur les résultats des tests, les limitations du système de tests et le problème de compensation dans un contexte de collaboration.

5.1 Architecture du système CPC

Le système CPC est constitué de plusieurs composants : les outils d'administration, le serveur de processus, les processus *Master* et *Slave*, les processus modifiés, le module de migration et une base de données. La figure 5.1 présente l'architecture du système CPC. Un système CPC doit exister pour chaque participant de la chorégraphie dont les activités sont exécutées à travers un BPMS. Les systèmes communiquent à travers des services Web. Le système CPC peut jouer à la fois les rôles de maître et d'esclave.

Le serveur de processus déploie et exécute les processus automatisés. Les processus *Master* et *Slave* réalisant le *Change Protocol for Collaboration* (CPC) y sont déployés. Ils constituent une partie importante du système puisqu'ils concrétisent les comportements du maître et de l'esclave, tout dépendant du rôle que le système est appelé à jouer. Les processus concernés par les changements y sont également déployés.

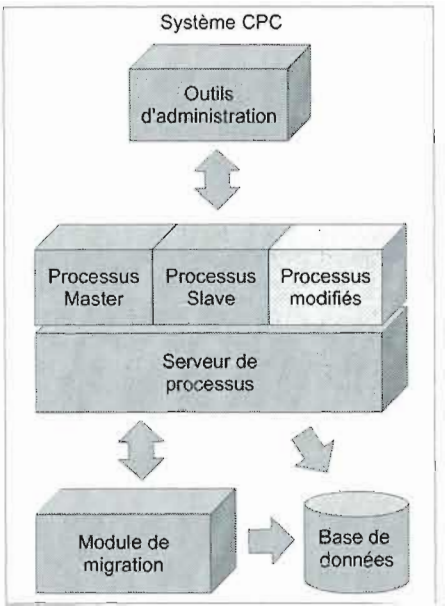

Figure 5.1 Architecture du système CPC.

Une base de données permet d'assurer la persistance des instances des processus et aussi de stocker les données d'administration du système.

Les processus *Master* et *Slave*, le module de migration et les outils d'administration sont expliqués ci-dessous plus en détail.

5.1.1 Processus *Master* et *Slave*

Le diagramme BPMN de la figure 5.2 montre les processus *Master* et *Slave* tels qu'ils ont été conçus pour le système. Ils sont décrits par la suite en pseudo-code (listages 5.1 et 5.2). Deux types d'analystes participent au scénario d'une demande de changement, l'analyste maître, qui initie la demande de changement, et les analystes esclaves, qui répondent à la demande.

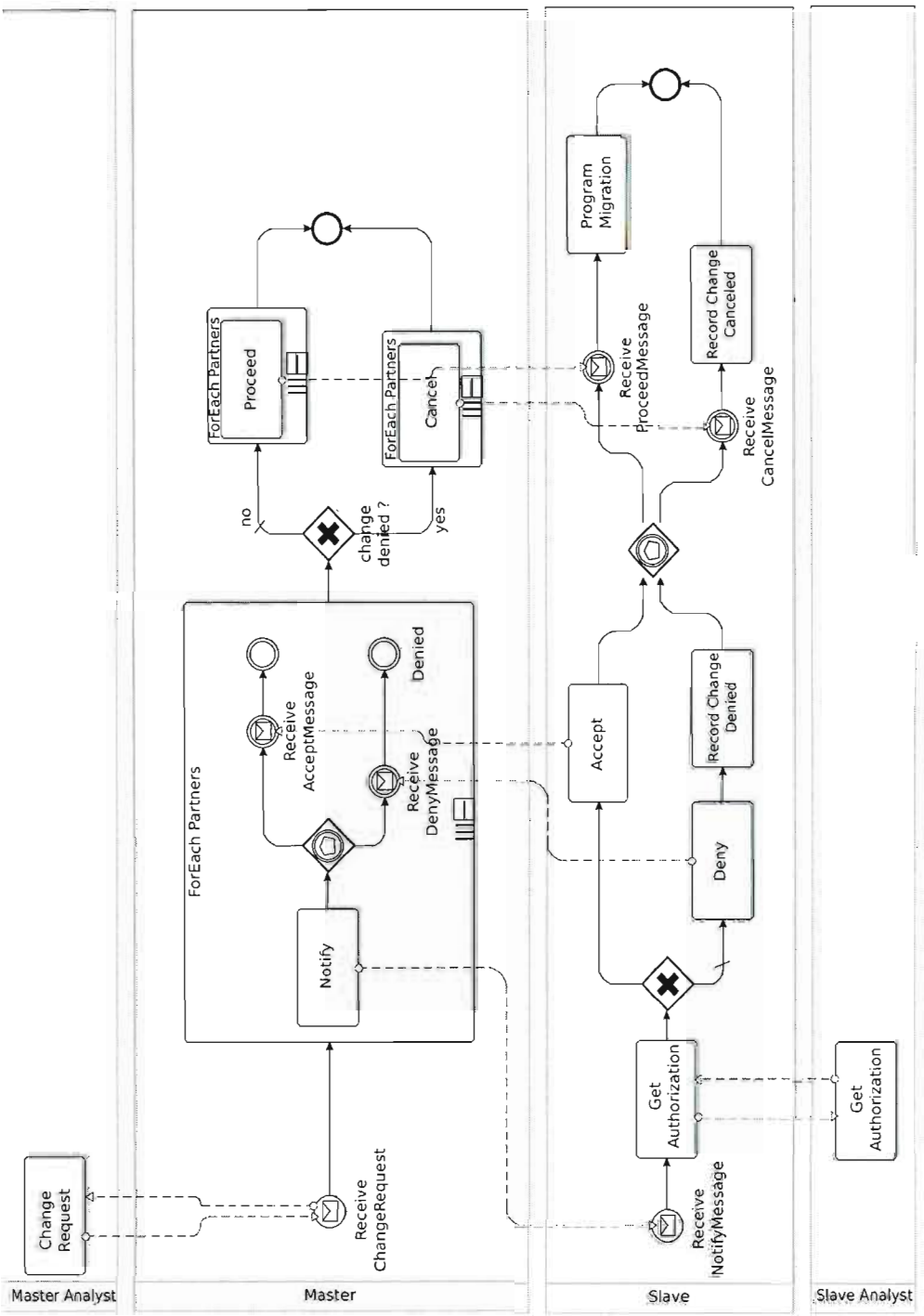

Figure 5.2 Diagramme BPMN des processus *Master* et *Slave* du système CPC.

Listage 5.1 Processus *Master*

```
process MASTER
    Receive ChangeRequest                          ▷ demande de l'analyste maître
    denied ← FALSE
    for each address in ChangeRequest.slaveAddresses do      ▷ multi-instances
        SLAVE ← address                                      ▷ liaison dynamique
        SLAVE.Notify                  ▷ message Notify envoyé à chaque esclave
        Initier la corrélation pour cette itération
        Attendre la réponse de l'esclave
        if Receive AcceptMessage then
            Ne rien faire
        else if Receive DenyMessage then
            denied ← TRUE
            Ne plus attendre les autres réponses non reçues
        end if
    end for
    if not denied then
        for each adress in ChangeRequest.slaveAddresses do   ▷ multi-instances
            SLAVE ← address                                  ▷ liaison dynamique
            SLAVE.Proceed             ▷ message Proceed envoyé à chaque esclave
        end for
    else
        for each adress in ChangeRequest.slaveAddresses do   ▷ multi-instances
            SLAVE ← address                                  ▷ liaison dynamique
            SLAVE.Cancel              ▷ message Cancel envoyé à chaque esclave
        end for
    end if
end process
```

Listage 5.2 Processus *Slave*

```
process SLAVE
    Receive NotifyMessage
    Initier la corrélation pour cette instance
    ADMINISTRATION.GetAuthorization          ▷ intervention de l'analyste esclave
    MASTER ← NotifyMessage.masterAddress     ▷ liaison dynamique
    if changement autorisé then
        MASTER.Accept
    else
        MASTER.Deny
        DATABASE.RecordChangeDenied
    end if
    Attendre la réponse du maître
    if Receive ProceedMessage then
        DATABASE.ProgramMigration
    else if Receive CancelMessage then       ▷ peut survenir à tout moment
        DATABASE.RecordChangeCanceled
    end if
end process
```

5.1.2 Module de migration

Suite à une demande de changement approuvée, lorsque la migration des instances est requise, le système CPC déclenche la migration au moment indiqué, et ce pour tous les partenaires participant au contrat. Le module de migration interagit avec le serveur de processus pour réaliser notre algorithme de migration (voir section 4.4). Il permet de synchroniser l'exécution conditionnelle des activités réalisées par des partenaires supportant la migration et de simuler les *échanges non supportés*.

Les *échanges non supportés* de type *receive* doivent être injectés pendant l'exécution conditionnelle à partir de la trace des échanges.

S'il advenait, suite à la migration, qu'un *échange non supporté* de type *invoke* subisse une transformation (il n'est plus identique à celui contenu dans la trace des échanges[1]), le module de migration doit alors en déclencher la compensation.

[1]Selon l'étape 2 de l'algorithme de migration (section 4.4), la trace des échanges est obtenue avant de débuter la migration. Pour chaque échange déjà survenu entre le processus et ses partenaires, une trace des entrées/sorties est gardée pour consultation lors de la migration. Ce qui permet de déterminer si une activité/échange déjà exécutée a subi des modifications lors de la migration.

Par contre, les échanges supportés peuvent être modifiés sans égard à leurs anciennes valeurs, puisque la réexécution pendant la migration de toutes les instances de la chorégraphie tient compte des transformations que ces échanges peuvent causer.

Puisqu'il est impossible de compenser une activité *reply*, lorsque ce type d'*échange non supporté* a été modifié, le module de migration génère une faute. La migration entre dans l'état *Interrompue*. Si la migration doit être annulée par la suite, le module doit faire un *roll back* de la migration et avertir les autres partenaires CPC qu'ils doivent également annuler la migration engagée, avant de placer la migration dans l'état *Annulée*.

Pour éviter d'attendre plus longtemps que nécessaire lors de la réexécution des instances, le module de migration modifie le temps d'attente de l'activité *wait* au besoin (Règle R6, section 4.4), considérant le temps déjà écoulé lors de l'exécution des premières instances.

5.1.3 Outils d'administration

Les outils d'administration permettent aux analystes d'interagir avec le système. La figure 5.3 montre le diagramme de cas d'utilisation des outils d'administration. Les cas d'utilisation sont décrits brièvement ci-dessous.

Trois acteurs sont identifiés dans le diagramme : l'opérateur, l'analyste maître et l'analyste esclave. L'analyste maître est celui qui initie la demande de changement et l'analyste esclave celui qui accepte ou refuse le changement. Il s'agit de deux types différents d'opérateurs, selon le rôle qu'ils sont appelés à jouer dans le CPC.

Effectuer une demande de changement : L'analyste maître déclenche à travers le module d'administration le processus de négociation, tel que réalisé par les processus *Master* et *Slave*.

Connaître l'état des demandes de changements : L'opérateur interroge les outils d'administration pour connaître l'état des demandes de changements (voir figure 4.4). Le système retourne la liste des demandes de changements avec leurs numéros d'identification et leurs états respectifs. L'opérateur peut alors consulter le contenu du message *Notify* associé à une demande.

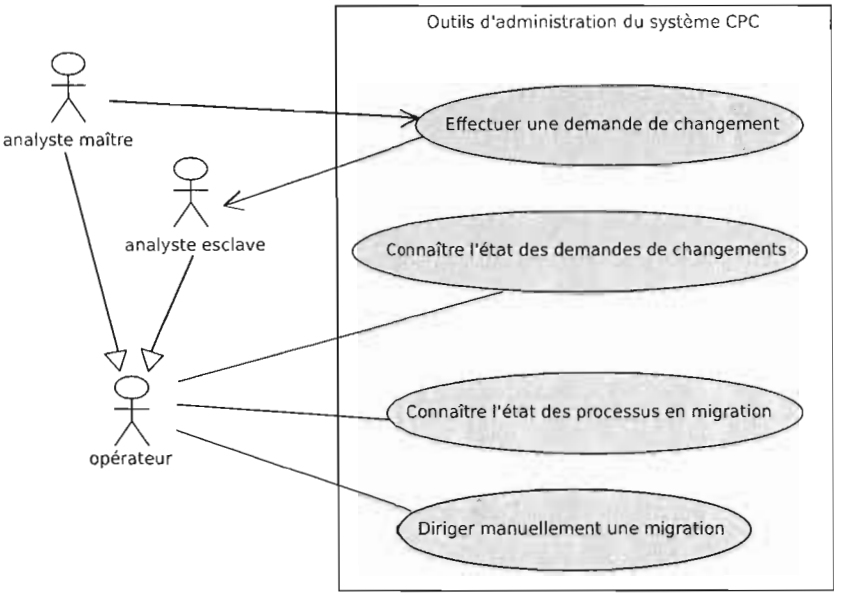

Figure 5.3 Diagramme de cas d'utilisation du module d'administration.

Connaître l'état des processus en migration : L'opérateur interroge le module d'administration pour connaître l'état des processus en migration (voir tableau 5.1). Le système retourne la liste des processus en migration, leurs identifiants, leurs états respectifs, ainsi que les numéros des instances concernées par la migration. L'opérateur peut alors consulter les détails des processus et les états des instances. Il peut également extraire la trace des échanges d'une instance.

Diriger manuellement une migration : Aux fins de tests, en cas d'erreur ou pour toute autre raison, l'opérateur peut diriger lui-même une migration à travers le module d'administration. Il peut déclencher une migration, la suspendre, la reprendre ou l'annuler. Le système répond en conséquence, bien sûr en assurant la synchronisation avec les autres partenaires CPC.

Tableau 5.1 États de la migration

État	Étape	Description
Programmée		La migration a été programmée et le système attend le moment indiqué pour déclencher la migration.
En cours*	Initialisation	Comprend la création des instances, l'obtention de la trace des échanges et la compensation des échanges exclus du nouveau schéma.
	Exécution	Exécution conditionnelle des instances suivant les règles de migration.
	Finalisation	Comprend la compensation des échanges ignorés et la notification des processus retirés et ignorés.
Terminée		La migration s'est terminée avec succès.
Interrompue†		La migration a été interrompue.
Annulée†		La migration a été annulée.

* Voir section 4.4 pour plus de détails sur les étapes de la migration.

† Voir sous-section 5.1.2 pour des explications sur ces états de la migration.

5.2 Validation du système CPC

Cette section décrit le système de tests développé pour valider notre approche. Nous fournissons ensuite les détails sur les cas de tests effectués et les résultats obtenus.

5.2.1 Architecture du système de tests

La figure 5.4 montre les principales composantes du système de tests. Nous avons déployé trois systèmes identiques sur trois machines différentes pour effectuer nos tests : un maître et deux esclaves.

Chaque système est composé du serveur de processus *Intalio Server Community Edition* (Intalio, 2010a) sur lequel sont déployés les deux processus *Master* et *Slave*.

Le diagramme BPMN des processus *Master* et *Slave* (figure 5.2) a été réalisé grâce à l'outil *Intalio Designer* (Intalio, 2010b). Le code BPEL des processus a été généré directement à partir du diagramme BPMN. Des modifications manuelles ont toutefois été nécessaires dans les expressions XPATH du code pour effectuer le couplage dynamique entre maître et esclaves. Le code BPEL peut être consulté en appendice C.

Le module de migration qui a été développé/modifié pour réaliser notre approche fait partie du moteur de processus BPEL *Apache ODE* (The Apache Software Foundation, 2010). *Apache ODE* est une application codée en *Java* (Oracle, 2011a). Elle a été testée et déployée sur le serveur Web *Jetty* (Eclipse, 2011).

Les processus modifiés sont déployés sur notre implémentation d'*Apache ODE*.

Nous avons utilisé *Intalio Server* pour les processus du CPC à cause d'un *bug* dans la dernière version « stable » d'*Apache ODE* concernant les corrélations d'activités multi-instances. *Intalio Server* utilise une version différente d'*Apache ODE* sur laquelle le *bug* en question n'est pas présent.

Le serveur de base de données *MySQL Community Server* (Oracle, 2011b) supporte chaque système.

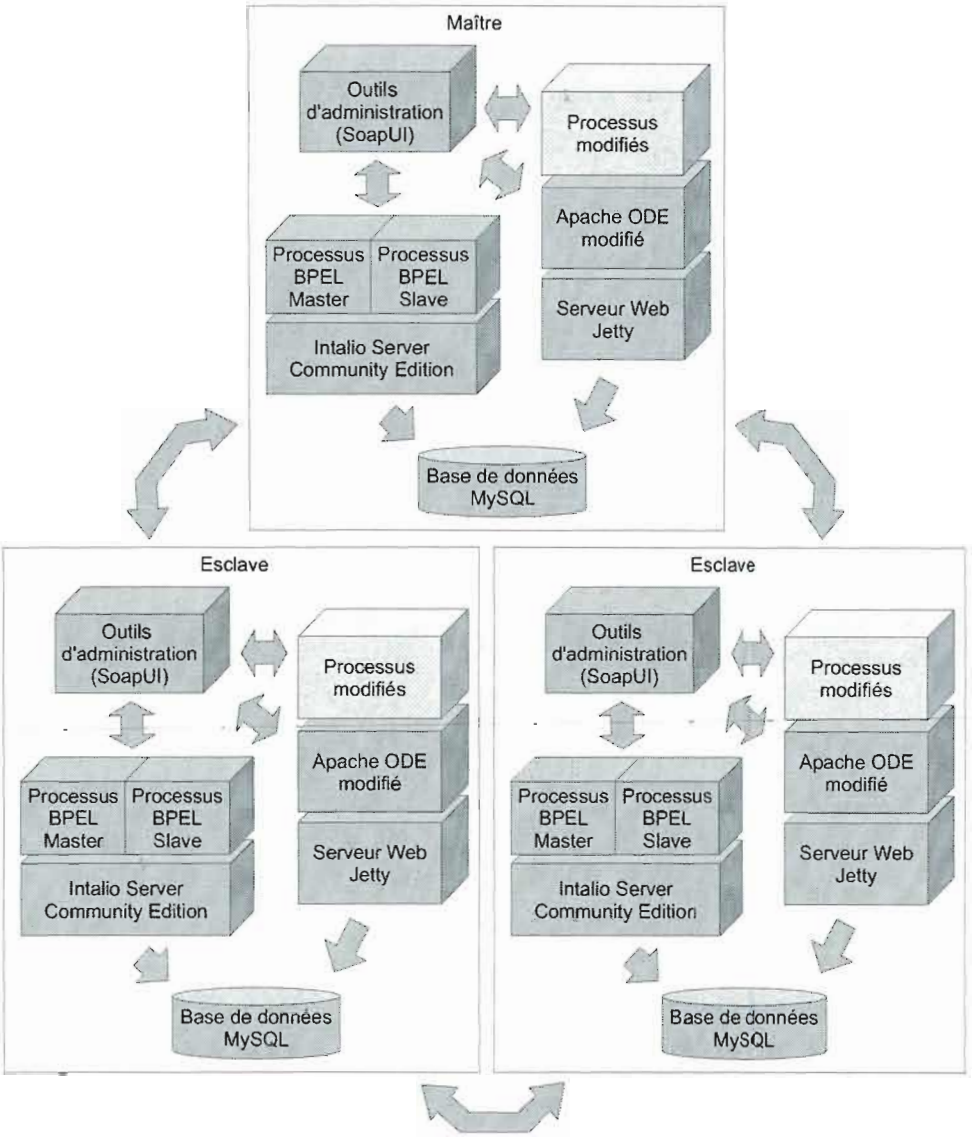

Figure 5.4 Architecture du système de tests.

Actuellement, pour les outils d'administration, nous utilisons le logiciel *SoapUI* (eviware, 2010) qui permet de simuler et d'effectuer des tests sur des services Web. Grâce à ce logiciel, l'opérateur interroge le moteur *Apache ODE* via son service Web et examine le contenu des messages SOAP/XML retournés. L'interface d'administration d'*Apache ODE* a été modifiée pour répondre à nos besoins. *SoapUI* permet également aux analystes d'interagir avec les services Web des processus du CPC et ceux des processus en migration.

5.2.2 Participation au projet *Open Source Apache ODE*

Le moteur de processus BPEL *Apache ODE* effectue la migration des instances en injectant les échanges d'une ancienne instance d'un processus dans une nouvelle exécution. C'est en partie ce dont nous avions besoin pour réaliser notre approche. Ce type de migration par injection des échanges a été implémenté tout d'abord dans un sous-projet d'*Apache ODE* nommé *Replayer*, auquel nous avons participé. Le *Replayer* a été introduit dans la version stable du moteur de processus *Apache ODE* (Version 1.3.4) (voir `http://ode.apache.org/instance-replayer.html`). Notre preuve de concept a été développée à partir de cette version.

La migration effectuée par *Apache ODE* ne supporte pas les changements au niveau des services et des données : toutes les données injectées et les services invoqués doivent demeurer identiques. Les modifications aux processus sont prises en compte lors de la réexécution, mais elles ne peuvent causer de changements sur les collaborations. Nous avons donc travaillé avec les développeurs du projet *Replayer* pour réaliser la synchronisation des instances pendant la migration.

Le code *Open Source* du moteur *Apache ODE* développé pour notre solution (incluant le *Replayer* modifié) peut être consulté et téléchargé à partir du lien suivant :

Replayer for collaborative processes
`http://github.com/enricoJL/ode`

5.2.3 Implémentation du module de migration

Le *Replayer* d'*Apache ODE* a été modifié pour réaliser les étapes de migration selon les règles décrites à la section 4.4. Nous faisons ci-dessous une brève description de ce qui a été implémenté.

Le module de migration (*Replayer* modifié) injecte aux instances en migration les *échanges entrants non supportés* ($R1$).

Le module vérifie chaque *invoke* à partir de la trace avant de le traiter. S'il s'agit d'un échange supporté, le module envoie le message directement, comme si c'était un message de synchronisation ($R2$). La synchronisation se fait par l'envoi du message directement au partenaire.

Lorsqu'il s'agit d'un *échange non supporté* de type *invoke* ($R3$), le module vérifie si l'échange est demeuré inchangé lors de la migration. Si tel est le cas, le module passe à l'instance la valeur du message de retour contenu dans la trace, sans que le véritable échange n'ait lieu. Si, par contre, un changement de valeur est détecté dans le message, le module identifie l'ancien échange pour compensation, et il effectue un nouvel *invoke* avec les nouvelles valeurs.

Le module effectue les mêmes vérifications qu'aux deux paragraphes précédents pour le *reply* ($R2$ et $R4$).

Le module intercepte tous les messages entrants lors de la migration. Si un message concerne un des processus identifiés pour migration, il effectue le traitement approprié. Le module crée l'instance en migration au moment de la réception du premier message ; s'il s'agit d'un message subséquent, le module retrouve l'instance et lui passe le message ($R5$). L'exécution continue selon les règles de la migration.

Apache ODE associe un *timestamp* à chaque échange dans la trace. Le module de migration utilise ce *timestamp* pour gérer la différence de temps lors de l'activité *wait* pour ne pas attendre plus longtemps que nécessaire au moment de l'exécution conditionnelle ($R6$).

Le module identifie les *échanges sortants non supportés* (*invoke* et *reply*) qui sont ignorés ou modifiés pendant la migration ($R7$ et $R8$). Il produit un rapport à la fin de la migration listant les activités à compenser. Le module n'effectue pas de compensation. Nous avons mis de côté la compensation lors de cette première étape de développement.

Le code pour réaliser notre prototype a été inséré à même les classes du *Replayer*. La figure 5.5 montre un extrait du diagramme de classes du module de migration. Ce sont principalement les classes `Replayer`, `ReplayerScheduler`, `ReplayerContext` et `ReplayerBpelRuntimeContextImpl` ont été modifiées. D'autres classes et schémas XML qui ne figurent pas dans ce diagramme ont aussi subi des modifications.

Le module de migration est réalisé par les classes mentionnées ci-dessus. Le moteur de processus est réalisé en partie par la classe `BpelEngineImpl`. La classe `ProcessInstanceDAO` correspond à l'état en cours d'une instance du processus, elle est utilisée pour enregistrer les données de l'instance. La classe `BpelProcess` réfère au schéma du processus.

La logique d'exécution d'une instance est contenue dans la classe `BpelRuntimeContextImpl`, elle représente une instance en cours d'exécution. Par polymorphisme, en faisant hériter la classe `ReplayerBpelRuntimeContextImpl` de cette classe, le comportement habituel lors de l'exécution d'une instance peut être remplacé par celui de la migration. Une instance de la classe `ReplayerBpelRuntimeContextImpl` est associée à l'instance du processus à sa création par le *Replayer*. Les comportements des activités *receive*, *invoke*, *reply* et *wait* sont ainsi modifiés de façon non intrusive dans le moteur de processus. La classe `ReplayerScheduler` est utilisée pour enregistrer et gérer les événements pendant la migration.

5.2.4 Cas de tests et résultats

Nous avons utilisé deux versions d'une chorégraphie à trois processus pour valider la migration : *Proc1*, *Proc2* et *Proc3*. Dans un premier temps, nous avons déployé les trois processus et effectué la migration sur un seul système de tests (figure 5.6). Puis nous avons tenté la migration sur trois systèmes distincts installés sur trois machines diffé-

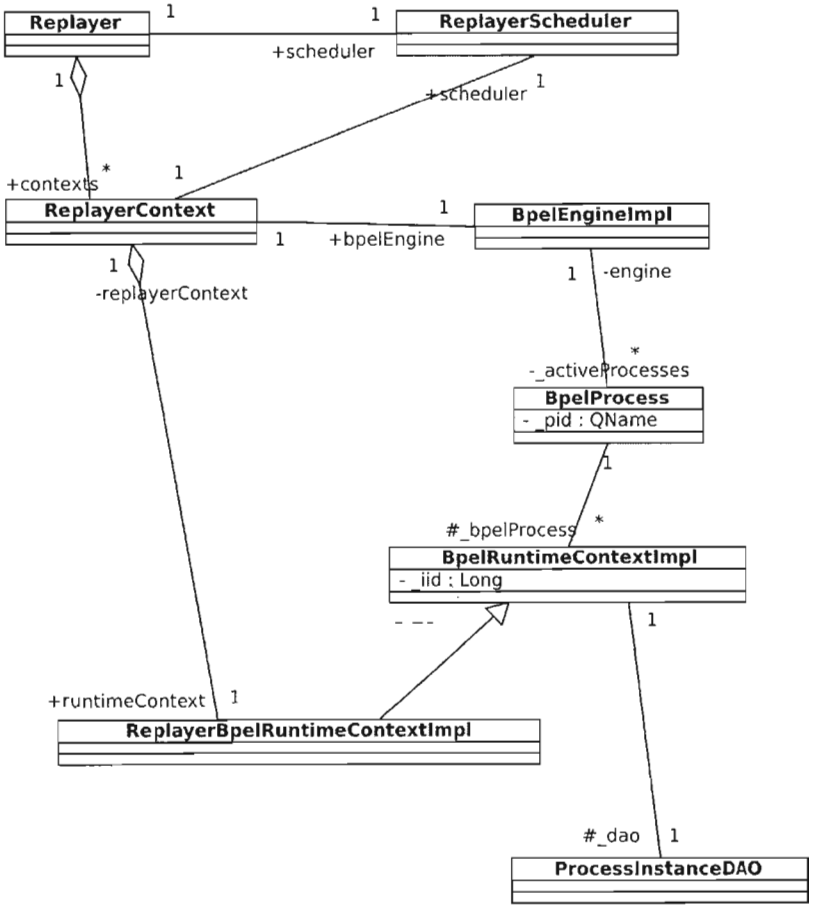

Figure 5.5 Classes implémentant le Replayer d'Apache ODE.

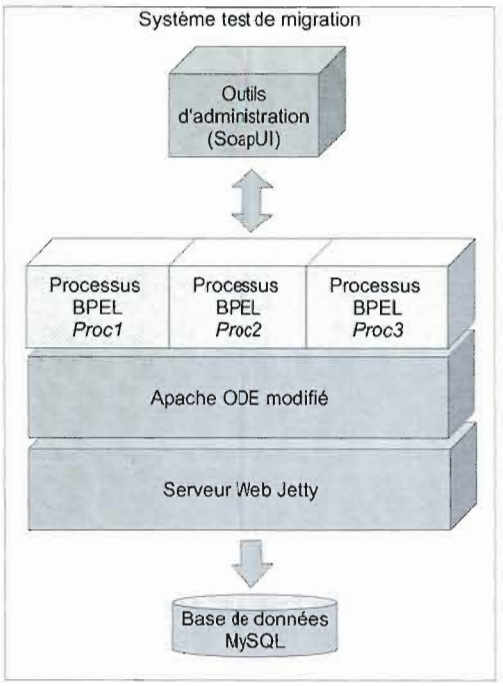

Figure 5.6 Système de tests pour la migration en local.

rentes, un système pour chaque processus (figure 5.7) — ceci correspondait davantage au problème particulier qui nous intéressait.

Les figures 5.8 et 5.9 montrent les deux versions des processus de tests. Les échanges identifiés *e2, e3, e5, e6* et *e7* sont des échanges supportés par la migration, puisqu'ils sont effectués entre les processus de la chorégraphie ; par contre, les échanges *e1, e4, ext1, ext2* et *ext3* ne sont pas supportés : ils retiennent donc notre attention pour déterminer les cas de compensations.

Lorsque les instances des processus *Proc2* (version 1) et *Proc3* (version 1) sont suspendues aux activités *rec-e5* et *rec-e4* respectivement, et que l'instance de *Proc1* (version 1) est suspendue à l'activité *rec-e6*, les activités *rec-e1, rec-e2, rec-e3, inv-e2, inv-e3, inv-ext1* et *inv-ext2* ont été exécutées. Lors de la migration, les instances sont réexécutées à

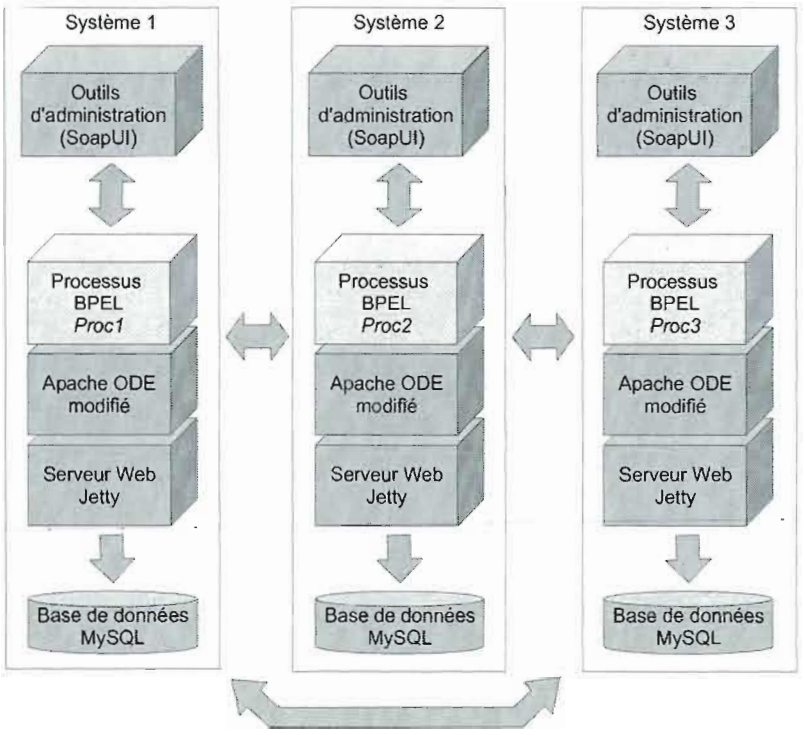

Figure 5.7 Systèmes de tests pour la migration distribuée.

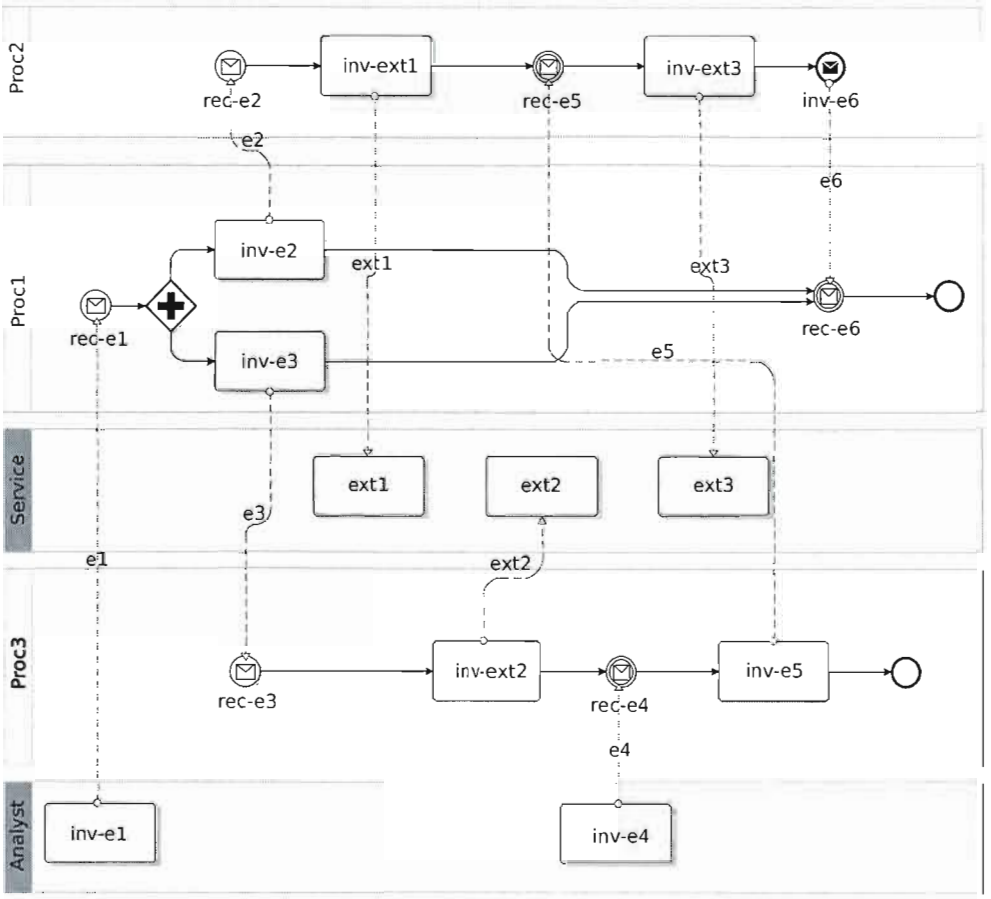

Figure 5.8 Chorégraphie de trois processus asynchrones — version 1.

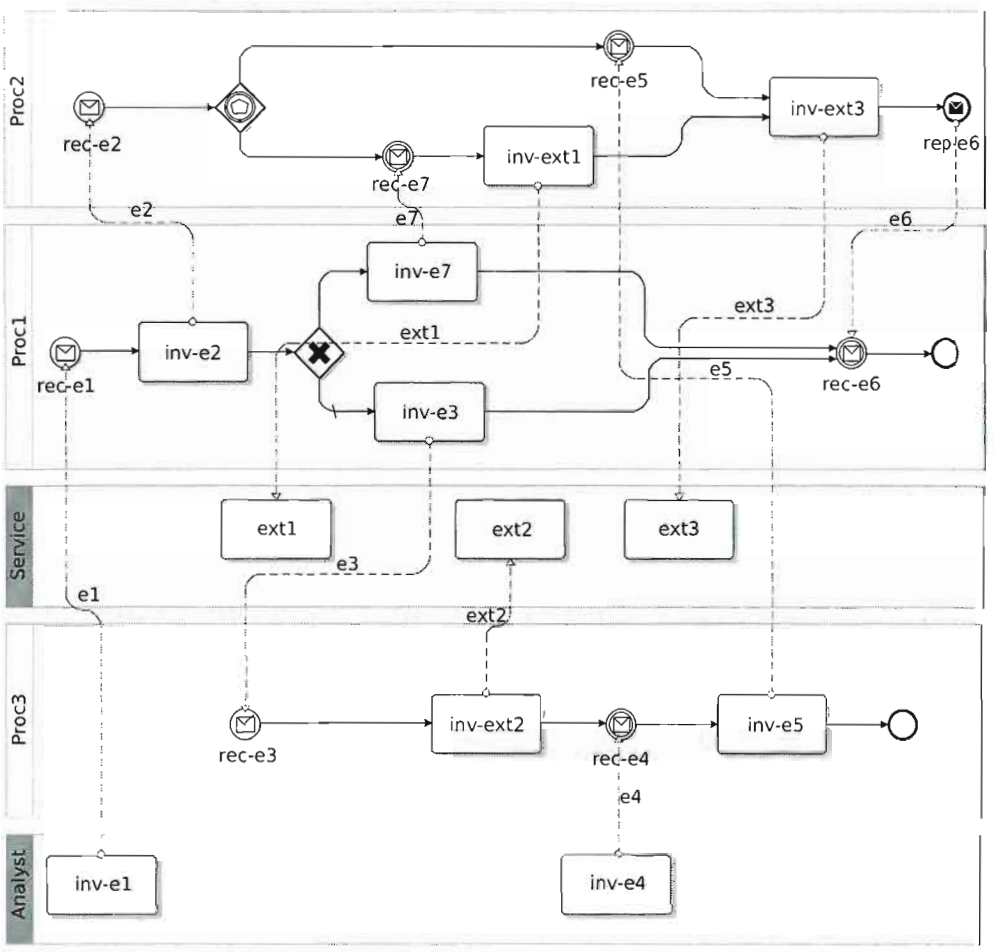

Figure 5.9 Chorégraphie de trois processus asynchrones — version 2.

partir de la deuxième version des processus. À cause du choix exclusif inséré dans *Proc1* (version 2), un seul des échanges *e3* ou *e7* peut avoir lieu. Deux scénarios différents d'exécution peuvent alors se produire. Les deux scénarios de tests ci-dessous (*Test 1* et *Test 2*) ont été utilisés pour valider les règles de migration.

Test 1 : *e7* ignoré. Si l'activité *inv-e7* est ignorée pendant la migration, puisque l'exécution de *inv-ext1* (*Proc2*, version 2) dépend de la réception de *e7*, *inv-ext1* doit être compensée, car elle a déjà été exécutée par l'ancienne instance.

Test 2 : *e3* ignoré. Si l'activité *inv-e3* est ignorée pendant la migration, l'échange *e3* n'a pas lieu, ce qui empêche la nouvelle instance de *Proc3* (version 2) de démarrer. Par conséquent, les activités *inv-e3* et *inv-ext2* doivent être compensées.

Dans un premier temps, lorsque les trois processus ont été déployés à l'intérieur d'un même serveur (figure 5.6), la migration s'est effectuée sans problème et les compensations ont été identifiées correctement.

Dans un deuxième temps, lorsque nous avons effectué une migration distribuée sur plusieurs serveurs (figure 5.7), la migration a échoué. Dans l'architecture distribuée, lorsqu'une instance en migration demeure en attente d'un message de synchronisation qui ne vient pas de l'interne, le *Replayer* suppose que la migration est terminée. Il s'agit d'une limitation de l'implémentation actuelle.

5.3 Discussions

5.3.1 Appréciation des résultats des tests

Les résultats des tests nous permettent de valider l'algorithme de migration proposé par notre approche. Nous avons démontré que la trace des échanges est suffisante pour effectuer la migration des instances de processus collaboratifs. Nous avons également démontré que notre algorithme générique avec compensation s'avère efficace pour résoudre les problèmes de migration dans des chorégraphies complexes.

Les tests que nous avons effectués nous permettent de valider le traitement des *échanges non supportés* par la migration et l'identification des compensations les concernant. Ils nous permettent de valider le traitement des échanges supportés et d'observer le comportement des collaborations pendant la migration. Les tests nous ont confirmé que nous n'avons pas besoin de compenser les échanges supportés lorsque nous effectuons une migration par réexécution de toutes les instances d'une collaboration.

Pour régler le cas d'échec rencontré lors du test de la migration distribuée, il faut tout d'abord forcer les systèmes à rester en mode migration jusqu'à ce que les instances soient terminées. Cependant, il faut tenir compte du cas d'un premier message de synchronisation qui ne viendrait jamais — ce qui implique que l'instance ne serait pas créée, donc que le module de migration attendrait indéfiniment ce message. Dans ce cas, le système devra déterminer si des processus ont été ignorés lors de l'exécution conditionnelle et notifier les partenaires, afin qu'ils puissent terminer la migration et effectuer les compensations requises. Cette solution n'a pas encore été implémentée. La règle $R9$ (section 4.4) a été ajoutée aux règles de migration suite aux tests effectués. Cette fonctionnalité sera donc implémentée dans une phase subséquente de développement.

Malheureusement, à cause des limitations du module de migration implémenté, nous n'avons pas encore pu réaliser concrètement la migration dans un environnement distribué ; mais nous avons montré que l'algorithme fonctionne dans le cas de base. Nous avons bon espoir d'en réaliser la preuve concrète dans une architecture distribuée sous peu. D'autres cas de tests devront alors être exécutés pour vérifier l'algorithme dans diverses autres situations.

5.3.2 Limitations du système de tests

Les outils d'administration tels que présentés dans l'architecture du système CPC restent encore à implémenter. L'opérateur consulte la trace d'exécution des processus du CPC via *SoapUI* pour arriver à connaître l'état des demandes de changements. De même, il doit consulter les traces des échanges pour connaître l'état des processus en migration.

De plus, il est impossible pour l'instant de faire un suivi adéquat des migrations en cours, car le système de tests ne fait pas le suivi des états de la migration.

La migration doit être lancée manuellement par l'opérateur. L'interface d'administration d'*Apache ODE* permet de lancer la migration. L'opérateur doit fournir la trace de l'instance à migrer. Il peut la demander au système qui lui fournira une première trace, mais il doit la modifier manuellement afin d'y ajouter certaines directives.

Cette première étape d'initialisation manuelle est nécessaire pour tous les processus de la chorégraphie. L'analyste déclenche lui-même la migration parce que la trace des échanges utilisée pour la migration nécessite des manipulations et des détections qui n'ont pas encore été automatisées. Plus précisément, il doit identifier les types d'échanges (échange supporté ou non) et s'il s'agit d'une instance qui démarre la chorégraphie ou qui doit attendre un premier message de synchronisation pour démarrer.

Comme il a été déjà mentionné, la compensation n'a pas été automatisée. Le système de tests développé pour l'instant ne fait qu'identifier les activités à compenser. Les notifications des partenaires ayant des processus retirés ou ignorés lors de la migration ne sont d'ailleurs pas encore implémentées. Rappelons que notre objectif était tout d'abord de réaliser une preuve de concept.

5.3.3 Compensation

Bien que nous n'ayons pas implémenté la compensation dans notre preuve de concept, nous avons tout de même pu en faire quelques observations.

Les processus en collaboration peuvent être considérés comme un seul mégaprocessus. Un processus peut être exécuté autant de fois qu'on le désire si aucun impact ne se fait sentir sur les activités concrètes de celui-ci. C'est ce qui se passe lorsqu'on simule les *échanges non supportés* pendant la migration. Puisque toutes les instances des processus dans une chorégraphie sont réexécutées lors de la migration, les échanges supportés peuvent modifier à leur guise les instances en migration. Cependant, puisque les *échanges*

non supportés par la migration sont simulés lors de l'exécution conditionnelle, si des changements les concernent, les activités correspondantes doivent être compensées et réexécutées de nouveau pour pouvoir tenir compte des modifications.

Pour revenir à l'exemple déjà mentionné, l'activité « Déterminer un itinéraire de voyage » d'un processus en migration pourrait être confiée à un partenaire qui exécute alors un processus pour la réaliser. Si l'instance du processus qui construit l'itinéraire de voyage subit des changements lors de la migration, ils seront pris en compte chez le partenaire lors de son exécution conditionnelle grâce à la synchronisation des instances. L'activité « Effectuer une réservation de billets d'avion » par contre ne peut être considérée de la même manière, puisque nous avons supposé que le partenaire responsable de cette activité ne supporte pas la migration. L'échange est donc simulé lors de l'exécution conditionnelle si ses paramètres n'ont subi aucun changement, autrement l'activité est compensée et réexécutée avec les nouveaux paramètres.

Donc, lorsqu'on effectue l'exécution conditionnelle de plusieurs processus collaboratifs, seules les activités qui concernent des *échanges non supportés* par la migration ont besoin d'être compensées. Notre algorithme de migration a été ajusté suite à ces observations.

Nous avons présenté dans ce chapitre l'architecture du système CPC et l'architecture du système de tests réalisé pour notre preuve de concept. Les processus *Master* et *Slave* ont été présentés, ainsi que le module de migration et les outils d'administration. Puis nous avons expliqué notre démarche de validation et présenté les résultats des tests effectués. Pour terminer, nous avons discuté des résultats des tests, des limitations du système de tests et du problème de la compensation dans un contexte collaboratif.

CONCLUSION

Après avoir introduit le sujet de ce mémoire, c'est-à-dire, l'adaptation dynamique des processus collaboratifs, nous avons présenté le cycle de vie des processus d'affaires et les technologies et concepts associés. Nous avons fourni une classification et des exemples de changements pour illustrer les différentes perspectives qui peuvent être touchées lors de changements dans des processus automatisés. Nous avons présenté différentes approches d'adaptation dynamique des processus, en mettant l'accent sur la migration des instances dans un contexte de collaboration.

Nous avons ensuite présenté notre solution et fait la description du *Change Protocol for Collaboration* (CPC). Nous avons expliqué notre approche de migration des instances de processus collaboratifs. L'architecture du système CPC a ensuite été présentée, ainsi que l'architecture du système de tests implémenté pour la validation. Les cas de tests et les résultats ont été expliqués. Bien qu'il reste encore des fonctionnalités à implémenter, notre preuve de concept est suffisamment complète pour nous permettre de valider l'approche proposée.

Nous avons appris certaines leçons lors de nos expérimentations. Entre autres, lorsqu'on effectue une réexécution de plusieurs processus collaboratifs, seules les activités qui concernent des échanges non supportés par la migration ont réellement besoin d'être compensées.

L'algorithme de migration, tel que nous le proposons, permet d'identifier quels sont les échanges non supportés par la migration qui subissent un changement lors d'une modification dans une chorégraphie complexe, et ce, de façon locale à chaque processus, simplifiant considérablement le problème de compensation.

Nous apportons donc une solution au problème de la coordination du changement dans des chorégraphies complexes en cours d'exécution. Nous proposons un algorithme générique de migration des instances avec compensation applicable à tous les types de changements et valide pour toutes les instances actives. Rappelons qu'aucune solution d'adaptation dynamique n'avait encore été proposée dans la littérature tenant compte de la collaboration.

La prochaine étape de nos travaux consiste à réaliser la compensation et les autres fonctionnalités manquantes, après quoi, nous pourrons tester la migration dans diverses situations plus complexes. Nous avons également identifié le problème d'exactitude du changement dans un contexte dynamique et collaboratif — nos travaux futurs pourront explorer cette nouvelle avenue de recherche.

APPENDICE A

DESCRIPTION DE LA NOTATION BPMN

BPMN *(Business Process Model and Notation)* (OMG, 2010a) est une notation pour définir les processus d'affaires d'une organisation. Nous présentons dans cet appendice les principaux éléments de cette notation[1].

Les éléments de la notation se regroupent en quatre catégories : les objets de flot, les connecteurs, les artefacts et les couloirs d'activités (figure A.1). Les objets de flot regroupent les événements, les activités et les passerelles. Le flot de séquence, le flot de messages et l'association sont les trois types de connecteurs qui relient les objets entre eux. Les artefacts sont des objets qui ajoutent de l'information contextuelle au diagramme. Le couloir et ses segments forment la dernière catégorie. Un couloir encadre le processus d'un participant dans une chorégraphie. Les segments d'un couloir permettent de séparer les activités d'un participant par rôles dans une organisation ou par département.

Il existe deux types d'activités : les tâches (activités atomiques) et les sous-processus (compositions d'activités). Les activités sont représentées par un rectangle à coins arrondis (figure A.2). Elles peuvent être exécutées une seule fois ou à plusieurs reprises dans une boucle séquentielle. La même activité peut aussi être exécutée en plusieurs instances. Les sous-processus peuvent être montrés sans leurs activités (boîte fermée) ou avec leurs activités (boîte ouverte).

---

[1]Les figures présentées dans cet appendice sont adaptées de (White, 2006).

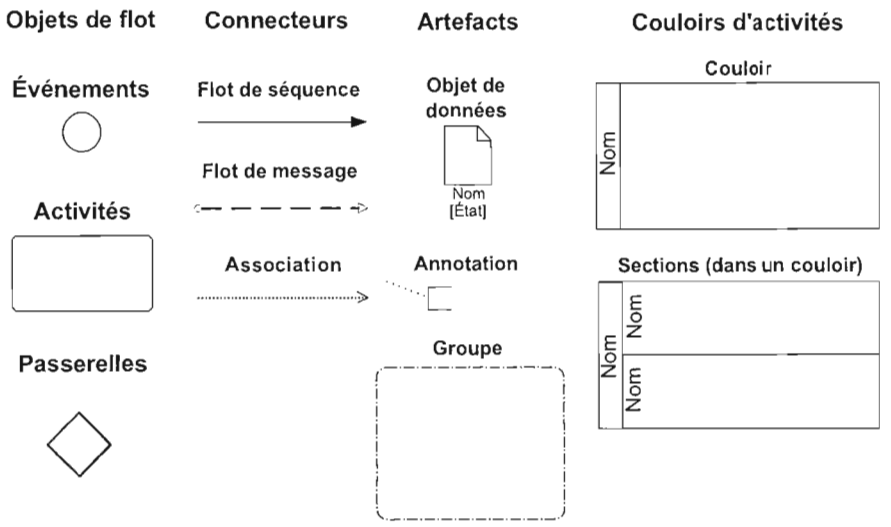

Figure A.1 Les éléments de la notation BPMN se regroupent en quatre catégories.

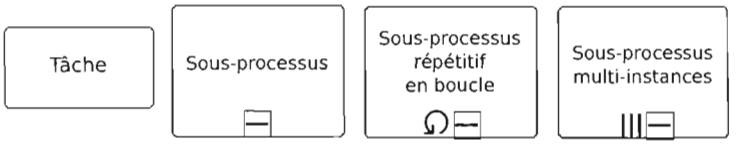

Figure A.2 Les activités sont constituées de tâches et de sous-processus.

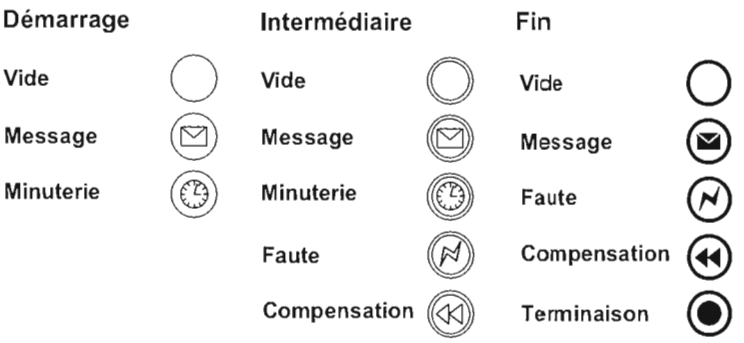

Figure A.3 Principaux types d'événements classifiés selon leurs catégories.

Un événement correspond à l'occurrence d'une situation donnée dans le cours du processus. Les événements peuvent démarrer, interrompre ou terminer le flot. Ils se subdivisent donc en trois catégories : événements de démarrage, intermédiaires ou de fin. Ils sont représentés respectivement par un cercle simple, un double cercle et un cercle gras (figure A.3).

Les événements de démarrage attendent un événement déclencheur (un message, un signal ou une minuterie), alors que les événements de fin génèrent un événement (un message ou un signal). Les événements intermédiaires peuvent attendre ou générer un événement. Le type de l'événement est indiqué à l'intérieur du cercle. Un événement vide n'attend rien ou ne fait rien.

Lorsqu'un événement intermédiaire est placé entre deux activités, il indique que quelque chose se produit pendant l'exécution du processus (figure A.4). Il peut par exemple représenter la réception d'un message ou l'envoi d'un message. S'il est placé sur la frontière inférieure d'une activité, il indique que l'activité doit s'interrompre à l'arrivée de l'événement. Ce dernier type est utilisé pour gérer les fautes et les exceptions, ainsi que les compensations.

Les passerelles permettent de contrôler le flot du processus lorsqu'il existe plusieurs possibilités d'exécution. Elles sont placées aux intersections des différentes séquences de

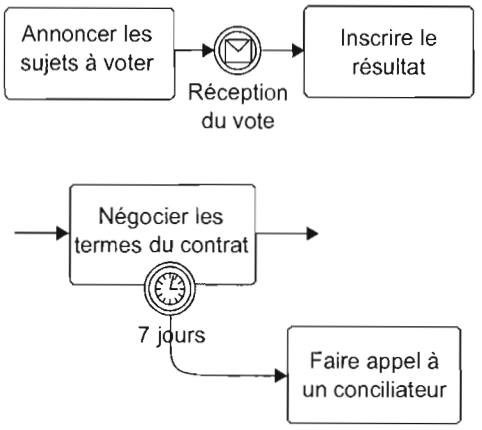

Figure A.4 Les événements intermédiaires peuvent être placés entre des activités (événement) ou sur la frontière d'une activité (interruption).

flot. Tous les types de passerelles sont représentés par un losange. Différents symboles sont placés à l'intérieur du losange pour indiquer le type. Seulement trois types de passerelles sont utilisés dans les diagrammes de ce mémoire : le *Choix exclusif basé sur les données*, le *Choix exclusif basé sur les événements* et le *Parallèle*.

Le choix exclusif est utilisé à un point d'intersection lorsque plusieurs alternatives sont possibles, mais qu'une seule doit être exécutée. Le *Choix exclusif basé sur les données* du processus prend la décision par rapport à une condition sur la valeur d'une variable (figure A.5). Le *Choix exclusif basé sur les événements* associe chaque branchement à un événement particulier, il prend le branchement du premier événement survenu (figure A.6). Le *Parallèle* permet d'indiquer que toutes les branches s'exécutent en parallèle (figure A.7).

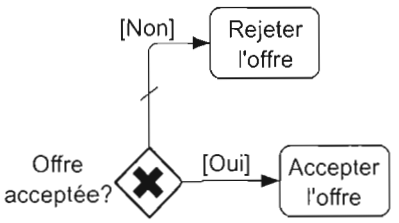

Figure A.5 Passerelle — Choix exclusif basé sur les données.

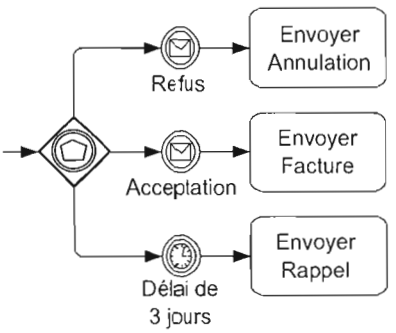

Figure A.6 Passerelle — Choix exclusif basé sur les événements.

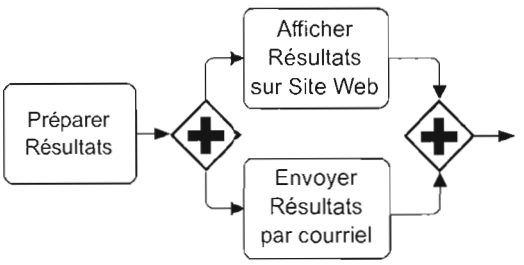

Figure A.7 Passerelle — Parallèle.

APPENDICE B

DÉFINITION DES PROCESSUS AVEC BPEL

Nous présentons dans cet appendice le langage BPEL plus en détail. Nous démontrons comment les trois perspectives des processus d'affaires présentées au chapitre 2 sont spécifiées grâce à BPEL, WSDL et XML.

B.1 Présentation de BPEL

BPEL *(Web Services Business Process Execution Language)* (Alves et al., 2007) est un langage de composition de services Web proposé par le groupe OASIS *(Organization for the Advancement of Structured Information Standards)*. Il sert à définir et orchestrer des interactions entre plusieurs partenaires. Il permet de modéliser les processus automatisés de l'entreprise. Il demeure avant tout un langage d'orchestration (Decker, 2009), c'est-à-dire qu'il est destiné à être interprété et exécuté sur un seul tiers par une machine virtuelle, appelé le moteur de processus *(process engine)*.

La définition du schéma d'un processus avec BPEL (figure B.1) contient la structure du flot des activités du processus, les services Web impliqués, la description des variables et des messages utilisés, ainsi que des mécanismes pour gérer les exceptions et les transactions. Des notions de rôles et de partenaires permettent de faire le lien entre le processus et les services. Ces différents concepts sont décrits en détail dans les sections suivantes.[1]

[1]Une description intéressante du langage a été faite par Orriëns, Yang et Papazoglou (2003) en associant les différents aspects d'un processus d'affaires (comment, pourquoi, quand, qui, quoi) aux éléments de BPEL.

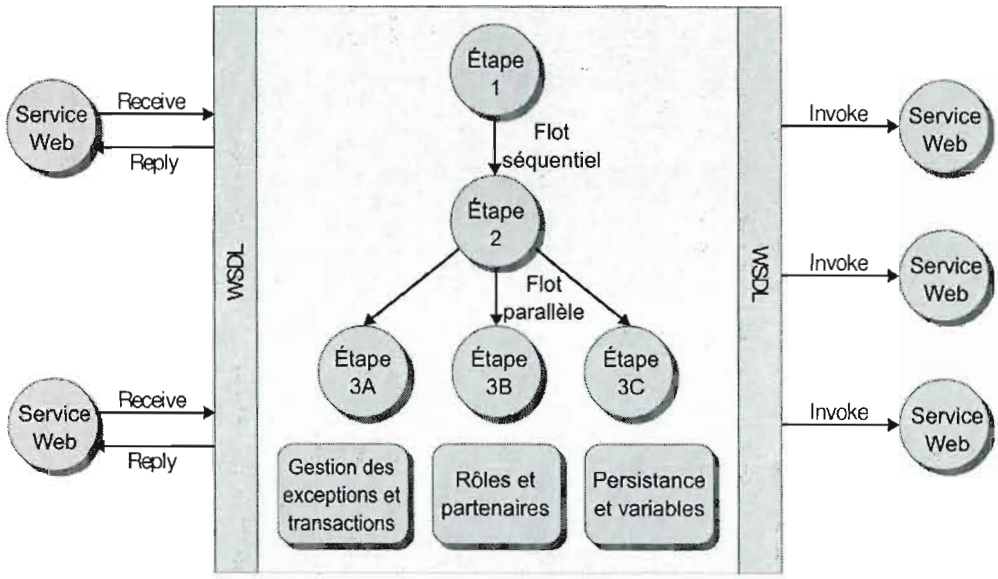

Figure B.1 Composition de services Web avec BPEL. Figure adaptée (Peltz, 2003).

B.2 Perspective des processus

Le schéma d'un processus BPEL commence par la balise `<process>` contenant un certain nombre d'attributs : les *namespaces* utilisés, le nom du processus et l'URI *(Uniform Resource Identifier)* pointant vers le schéma du langage à utiliser[2].

Les déclarations globales présentées au tableau B.1 sont placées au début du schéma. Les activités présentées au tableau B.2 sont insérées ensuite pour décrire le comportement du processus. Les *activités de base* sont des activités dites atomiques, et les *activités structurées* sont des conteneurs d'activités qui permettent de contrôler et de structurer l'exécution. Ces dernières activités peuvent bien sûr contenir d'autres activités structurées.

La construction `<scope>` est un contexte local dans lequel on peut déclarer des variables (voir tableau B.1, Déclarations locales) et d'autres mécanismes d'exécution (voir ta-

[2]BPEL fournit deux schémas différents selon le type du processus, un schéma pour les processus exécutables et un schéma pour les processus abstraits.

Tableau B.1 Déclarations BPEL

Déclarations globales	
`<process>`	Racine du processus et contexte global
`<extension>`	Ajout de nouveaux éléments au langage
`<import>`	Importation de documents (WSDL, XSD ou autres)
`<partnerLinkType>`	Déclaration de types d'interactions entre partenaires
Déclarations globales ou locales	
`<partnerLink>`	Lien concret avec un partenaire
`<messageExchange>`	Échange avec un partenaire
`<variable>`	Message ou variable interne
`<correlationSet>`	Ensemble de propriétés de corrélation

bleau B.2, Contextes). La construction `<process>` (la racine du processus) est un contexte global, les mécanismes associés au contexte peuvent aussi y être insérés.

Un processus abstrait n'est pas exécutable. Il contient des activités partiellement définies ou omises qui doivent être implémentées dans un processus exécutable. Les activités abstraites sont identifiées par la balise `<opaqueActivity>`.

Les activités principales des processus BPEL sont les activités d'entrée/sortie, sans elles, le processus n'a aucune raison d'être. Elles sont décrites ci-dessous.

L'activité `<invoke>` permet d'appeler les opérations des fournisseurs de services Web (listage B.1). Un `partnerLink` y est spécifié comme attribut pour indiquer de quel partenaire il s'agit. L'attribut `operation` précise l'opération visée du service invoqué. L'attribut `inputVariable` indique la variable message à transmettre en paramètre. L'attribut `outputVariable` indique la variable message où placer le résultat de la réponse à la requête. D'autres indications peuvent être ajoutées si nécessaire pour la corrélation, la gestion des fautes et la compensation.

Tableau B.2 Activités BPEL

Activités de base	
`<invoke>`	Utiliser des opérations via des services Web
`<receive>`, `<reply>`	Fournir des opérations via un service Web
`<assign>`	Attribuer des valeurs aux variables et aux relations
`<throw>`	Signaler une faute interne
`<wait>`	Suspendre l'exécution
`<empty>`	Ne rien faire
`<extensionActivity>`	Ajouter une nouvelle activité au langage
`<exit>`	Terminer immédiatement l'exécution du processus
`<rethrow>`	Propager les fautes
Activités structurées	
`<sequence>`	Exécution séquentielle
`<if>`	Exécution conditionnelle
`<while>`	Exécution répétitive
`<repeatUntil>`	Exécution répétitive
`<pick>`	Sélection par événement
`<flow>`	Exécution parallèle à dépendances conditionnelles
`<forEach>`	Exécution parallèle de plusieurs branches
Contextes	
`<scope>`	Contexte local
`<compensationHandler>`	Exécution compensatoire
`<faultHandlers>`	Exécutions alternatives en cas de fautes
`<eventHandlers>`	Réactions à des événements
Abstractions	
`<opaqueActivity>`	Activité abstraite

Listage B.1 Invocation d'un service Web avec BPEL

```
<bpel:invoke partnerLink="ventesCreditPlkVar"
    operation="Verifier_Credit" inputVariable="verifier_CreditMsg"
    name="Effectuer_Demande_Credit"/>
```

L'activité <receive> reçoit les messages transmis au processus (listage B.2). Elle est associée à un partnerLink définissant le rôle du processus. On y spécifie l'opération à fournir et la variable où placer le message reçu. L'indication createInstance="yes" permet de dire au système que cette activité crée une nouvelle instance du processus. La construction <correlation set... initiate="yes"/> permet d'indiquer au système que les valeurs de corrélation sont initiées à partir du message reçu.

Listage B.2 Recevoir un message avec BPEL

```
<bpel:receive partnerLink="ventesAcheteurPlkVar"
    operation="Placer_Commande" variable="placer_CommandeMsg"
    createInstance="yes" name="Recevoir_Bon_Commande">
  <bpel:correlations>
    <bpel:correlation set="noCommande-corr" initiate="yes"/>
  </bpel:correlations>
</bpel:receive>
```

La déclaration <correlationSet> permet de déclarer les propriétés des messages qui identifient uniquement une instance du processus, comme un numéro de commande. Le code du listage B.3 déclare un correlationSet nommé noCommande-corr qui spécifie un ensemble de propriétés à utiliser pour la corrélation : l'ensemble ne contient qu'une seule propriété, noCommande. Cette propriété est définie avec ses propertyAlias dans le document WSDL du service Web du processus. Les propertyAlias indiquent les parties des messages du service Web qui correspondent à une propriété. Le code du listage B.4 définit la propriété noCommande de type unsignedLong. Une propertyAlias y est définie qui pointe vers la partie du message Placer_CommandeRequete qui correspond au numéro de commande.

D'autres constructions servent à spécifier la réception de messages et fonctionnent de la même manière que <receive>. Il s'agit de l'élément <onMessage> de l'activité <pick> et de l'élément <onEvent> de la construction <eventHandlers>.

Listage B.3 Déclaration d'une corrélation BPEL

```
<bpel:correlationSets>
  <bpel:correlationSet name="noCommande-corr"
     properties="this:noCommande"/>
</bpel:correlationSets>
```

Listage B.4 Définitions d'une propriété de corrélation

```
<vprop:property name="noCommande" type="xs:unsignedLong"/>
<vprop:propertyAlias propertyName="this:noCommande"
    messageType="this:Placer_CommandeRequete" part="body">
  <vprop:query>noCommande/text()</vprop:query>
</vprop:propertyAlias>
```

L'activité `<reply>` sert à répondre à un partenaire lors d'un échange de type requête-réponse. La syntaxe est presque la même que `<receive>`, seul l'attribut `createInstance` ne s'applique pas (listage B.5).

Listage B.5 Répondre à une requête avec BPEL

```
<bpel:reply partnerLink="ventesAcheteurPlkVar"
    operation="Placer_Commande" variable="accuser_ReceptionMsg"
    name="Accuser_Reception"/>
```

Parmi les activités structurées, la construction `<flow>` est particulièrement intéressante. Elle encadre l'exécution parallèle d'activités, tout en permettant de définir des branchements conditionnels (*links*) entre des activités non structurées. Des conditions peuvent être ajoutées aux branchements. Ces mêmes conditions sont utilisées pour spécifier des règles de distribution et de fusion lorsqu'une activité devient la source ou la destination de plusieurs branchements.

La construction `<forEach>` permet de traiter plusieurs éléments d'une collection. Il est utilisé entre autres pour lancer des appels à plusieurs fournisseurs d'un même type de service par liaison dynamique *(dynamique binding)* (Barreto et al., 2007).

La construction `<faultHandlers>` contient les différents `<catch>` et `<catchAll>` pour attraper les événements qui provoquent des cas d'exceptions.

Tableau B.3 Éléments d'une définition de service avec WSDL

Élément	Défini par
Type	Schéma XML
Message	Parties *(parts)*
	Types
Type de ports *(portType)*	Opérations
	Messages
Liaison *(binding)*	Type de ports
	Protocole
	Opérations
	Formats des données des messages
Service	Ports
	Liaisons
	Adresses de points de service *(address endpoint)*

La construction `<compensationHandler>` permet de spécifier des activités compensatoires à exécuter pour annuler des activités déjà exécutées.

Le lecteur est invité à consulter la norme OASIS (Alves et al., 2007) et le manuel l'accompagnant (Barreto et al., 2007) pour plus d'explications.

B.3 Perspective des services

B.3.1 Définitions des services Web avec WSDL

La spécification WSDL (Christensen et al., 2001) utilise plusieurs concepts pour définir l'interface d'un service Web. Les différents éléments d'une définition de service Web avec WSDL sont présentés au tableau B.3.

- Les *types* sont définis par des schémas XML.
- Le *message* permet de faire abstraction des types XML et peut contenir plusieurs parties. Chaque *partie* est d'un type particulier.
- Le *type de ports* est un ensemble d'opérations abstraites. La signature abstraite d'une opération est définie par son nom et par les messages requis en entrée, en sortie ou en cas d'erreur.
- La *liaison* est la définition concrète des opérations et des messages du service. Une liaison doit être définie pour chaque type de ports. Un *protocole* de communication est spécifié, SOAP *(Simple Object Access Protocol)* ou HTTP *(HyperText Transfer Protocol)*. Des informations concrètes nécessaires pour chaque *opération* sont indiquées. Des *formats de données* sont établis pour tous les messages associés aux opérations.
- Le *service* regroupe un certain nombre de *ports*. Chaque port est associé à une liaison. Le port fournit l'*adresse du point de service* pour accéder aux opérations de la liaison.

Pour plus de détails concernant le langage WSDL, veuillez consulter la spécification (Christensen et al., 2001).

B.3.2 Éléments BPEL reliés à la perspective des services

BPEL s'appuie sur WSDL pour les définitions des services, les éléments WSDL font donc aussi partie de ses éléments (voir tableau B.4).

Un *partenaire* est une entité qui fournit des services. Il peut s'agir d'un autre processus BPEL ou d'une application quelconque qui offre un service Web accessible à travers le réseau.

Une *référence de point de service* est en fait une *adresse réseau* qui pointe vers le fournisseur réel du point de service. Cet élément provient de la spécification WS-Addressing *(Web Service Addressing)* (Box et al., 2004).

Un *type de relations* définit un échange qui existe entre le partenaire et le processus lui-même. Des *rôles* sont définis et des *types de ports* leurs sont associés correspondant aux types définis dans les documents WSDL.

Tableau B.4 Éléments BPEL reliés à la perspective des services

Élément	Défini par
Type de relations *(partnerLinkType)*	Collaborateurs *(partners)*
	Rôles
	Types de ports *(portTypes)*
Relation *(partnerLink)*	Type de relations
	Rôle du partenaire
	Rôle du processus
	Référence de point de service
Référence de point de service *(endpointReference)*	Adresse réseau

Une *relation* concrétise un type de relation. Les rôles du partenaire et du processus sont identifiés. Il est possible d'attribuer dynamiquement une référence de point de service à une relation pendant l'exécution du processus. La relation contient donc l'adresse concrète du point de service à utiliser pour un service.

Le listage B.6 montre le code WSDL dans lequel deux types de relations (`partnerLinkType`) sont définis : le type pour définir l'interaction entre Ventes et Crédit (`VentesCreditPlk`) et le type pour Ventes et Entrepôt (`VentesEntrepotPlk`). Les types de ports correspondent à ceux définis dans les interfaces WSDL.

Listage B.6 Définitions de types de relations

```
<pnlk:partnerLinkType name="VentesCreditPlk">
  <pnlk:role name="Credit_pour_Ventes" portType="Credit:PourVentes"/>
  <pnlk:role name="Ventes_pour_Credit" portType="Ventes:PourCredit"/>
</pnlk:partnerLinkType>
<pnlk:partnerLinkType name="VentesEntrepotPlk">
  <pnlk:role name="Entrepot_pour_Ventes" portType="Entrepot:PourVentes"/>
</pnlk:partnerLinkType>
```

Ce code contenu dans un document WSDL est importé dans le schéma BPEL du processus avec les autres définitions de services Web.

Les relations (`partnerLink`) sont ensuite déclarées dans le schéma BPEL comme il est indiqué au listage B.7. Le rôle du processus (`myRole`) et celui du partenaire (`partnerRole`) sont identifiés pour chaque relation. Les noms des rôles correspondent à ceux définis dans les types de relations.

Listage B.7 Déclaration des relations BPEL

```
<bpel:partnerLinks>
  <bpel:partnerLink name="ventesCreditPlkVar"
      partnerLinkType="dist:VentesCreditPlk" initializePartnerRole="yes"
      myRole="Ventes_pour_Credit" partnerRole="Credit_pour_Ventes"/>
  <bpel:partnerLink name="ventesEntrepotPlkVar"
      partnerLinkType="dist:VentesEntrepotPlk"
      initializePartnerRole="yes" partnerRole="Entrepot_pour_Ventes"/>
  ...
</bpel:partnerLinks>
```

Les relations sont utilisées par la suite dans les activités d'entrée/sortie pour interagir avec un partenaire (voir listages B.1 et B.2). L'adresse réseau d'un service Web peut être spécifiée statiquement dans le document WSDL, mais elle peut aussi être contenue dans une référence de point de service et attribuée dynamiquement à une relation. Ainsi, plusieurs relations d'un même type peuvent être gardées en mémoire pendant l'exécution d'un processus contenant des adresses différentes et utilisées pour un même service selon le contexte.

B.4 Perspective des données

On distingue deux sortes de variables dans un schéma BPEL, les variables de messages et les variables internes. Les types de messages sont importés directement des définitions WSDL. Les types de variables internes peuvent être importés à partir d'autres documents XSD ou spécifiés à l'intérieur même du schéma du processus.

Les variables sont déclarées à l'intérieur de la construction `<variables>` (listage B.8).

Listage B.8 Déclaration de variables BPEL

```
<bpel:variables>
  <bpel:variable name="placer_CommandeMsg"
      messageType="this:Placer_CommandeRequete"/>
  <bpel:variable name="autoriser_CreditMsg"
      messageType="this:Autoriser_CreditRequete"/>
  <bpel:variable name="acheteur" type="tns:AcheteurType"/>
  <bpel:variable name="temp" type="xs:string"/>
  ...
</bpel:variables>
```

Les messages (`messageType`) sont utilisés pour communiquer avec les services Web. Leurs types proviennent des fichiers WSDL importés. Les autres types de variables proviennent de schémas XML et elles sont utilisées pour des manipulations internes au processus.

Pour accéder aux éléments d'une variable, le langage XPath *(XML Path Language)* (Clark et DeRose, 1999) est utilisé. Les valeurs sont copiées à partir d'une autre variable ou spécifiées directement dans le code.

APPENDICE C

CODE SOURCE DES PROCESSUS DU CPC

C.1 Schéma BPEL du processus *Master*

cpc-Master.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
    executable" xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/
    varprop" xmlns:pnlk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3
    .org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:ode="http://www.apache.org/ode/type/extension"
    xmlns:Slave="http://latece.uqam.ca/cpc/Slave" xmlns:this="http://
    latece.uqam.ca/cpc/Master" xmlns:tns="http://latece.uqam.ca/cpc/xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:CPC_Interface="http://latece.uqam.ca/cpc/CPC_Interface"
    xmlns:diag="http://latece.uqam.ca/cpc" xmlns:xml="http://www.w3.org/
    XML/1998/namespace" xmlns:bpmn="http://www.intalio.com/bpms"
    xmlns:atomic="http://ode.apache.org/atomicScope" queryLanguage="
    urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0" expressionLanguage="
    urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"  name="Master"
    targetNamespace="http://latece.uqam.ca/cpc/Master">
  <bpel:import namespace="http://latece.uqam.ca/cpc" location="cpc.wsdl"
      importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc/Master" location="cpc
      -Master.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc/Slave" location="cpc-
      Slave.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:partnerLinks>
    <bpel:partnerLink name="slaveAndMasterPlkVar" partnerLinkType="
        diag:SlaveAndMaster" initializePartnerRole="yes" myRole="
        Master_for_Slave" partnerRole="Slave_for_Master"/>
    <bpel:partnerLink name="cPC_InterfaceAndMasterPlkVar" partnerLinkType
        ="diag:CPC_InterfaceAndMaster" myRole="Master_for_CPC_Interface"/
        >
  </bpel:partnerLinks>
  <bpel:variables>
```

```
    <bpel:variable name="thisRequestChangeRequestMsg" messageType="
        this:RequestChangeRequest"/>
    <bpel:variable name="denied" type="xs:string"/>
    <bpel:variable name="cmpt" type="xs:int"/>
    <bpel:variable name="thisAcceptRequestMsg" messageType="
        this:AcceptRequest"/>
    <bpel:variable name="thisRequestChangeResponseMsg" messageType="
        this:RequestChangeResponse"/>
    <bpel:variable name="thisDenyRequestMsg" messageType="
        this:DenyRequest"/>
  </bpel:variables>
  <bpel:sequence>
    <bpel:receive partnerLink="cPC_InterfaceAndMasterPlkVar" portType="
        this:ForCPC_Interface" operation="RequestChange" variable="
        thisRequestChangeRequestMsg" createInstance="yes"  name="
        RequestChange" ></bpel:receive>
    <bpel:assign name="init-variables-Master" >
      <bpel:copy >
        <bpel:from>
          <bpel:literal></bpel:literal>
        </bpel:from>
        <bpel:to>$denied</bpel:to>
      </bpel:copy>
      <bpel:copy >
        <bpel:from>
          <bpel:literal></bpel:literal>
        </bpel:from>
        <bpel:to>$cmpt</bpel:to>
      </bpel:copy>
      <bpel:copy >
        <bpel:from>
          <bpel:literal>
            <this:RequestChangeResponse>
              <tns:changeRequestRefID></tns:changeRequestRefID>
            </this:RequestChangeResponse>
          </bpel:literal>
        </bpel:from>
        <bpel:to>$thisRequestChangeResponseMsg.body</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:assign  name="RequestChange-1" >
      <bpel:copy>
        <bpel:from>$ode:pid</bpel:from>
        <bpel:to>$thisRequestChangeResponseMsg.body/
           tns:changeRequestRefID</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:reply partnerLink="cPC_InterfaceAndMasterPlkVar" portType="
        this:ForCPC_Interface" operation="RequestChange" variable="
        thisRequestChangeResponseMsg"  name="RequestChange-2" ></
        bpel:reply>
    <bpel:scope  name="outer-ForEach_Partners" >
      <bpel:forEach  name="ForEach_Partners"  parallel="yes" counterName=
          "ForEach_PartnersCounter">
```

```
<bpel:startCounterValue>1</bpel:startCounterValue>
<bpel:finalCounterValue>count($thisRequestChangeRequestMsg.body/
    tns:slaveAddresses)</bpel:finalCounterValue>
<bpel:scope  name="ForEach_Partners-1" >
  <bpel:correlationSets>
    <bpel:correlationSet name="corr" properties="this:refID"/>
  </bpel:correlationSets>
  <bpel:variables>
    <bpel:variable name="slaveNotifyRequestMsg" messageType="
        Slave:NotifyRequest"/>
    <bpel:variable name="thisDenyRequestMsg" messageType="
        this:DenyRequest"/>
    <bpel:variable name="thisAcceptRequestMsg" messageType="
        this:AcceptRequest"/>
  </bpel:variables>
  <bpel:sequence>
    <bpel:assign name="init-variables-ForEach_Partners" >
      <bpel:copy >
        <bpel:from>
          <bpel:literal>
            <Slave:NotifyRequest>
              <tns:messageID></tns:messageID>
              <tns:effectiveDate></tns:effectiveDate>
              <tns:expirationDate></tns:expirationDate>
              <tns:scope></tns:scope>
              <tns:runningInstances></tns:runningInstances>
              <tns:instanceReferences></tns:instanceReferences>
              <tns:contractRefID></tns:contractRefID>
              <tns:newContractSchema></tns:newContractSchema>
              <tns:newContractRefID></tns:newContractRefID>
              <tns:masterAddress></tns:masterAddress>
              <tns:slaveAddresses></tns:slaveAddresses>
            </Slave:NotifyRequest>
          </bpel:literal>
        </bpel:from>
        <bpel:to>$slaveNotifyRequestMsg.body</bpel:to>
      </bpel:copy>
    </bpel:assign>
    <bpel:assign  name="Notify" >
      <bpel:copy>
        <bpel:from>$thisRequestChangeRequestMsg.body</bpel:from>
        <bpel:to>$slaveNotifyRequestMsg.body</bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>$thisRequestChangeRequestMsg.body/
            tns:slaveAddresses[$ForEach_PartnersCounter]/text()</
            bpel:from>
        <bpel:to partnerLink="slaveAndMasterPlkVar"
            endpointReference="partnerRole"/>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>concat($ode:pid, "-", $ForEach_PartnersCounter
            )</bpel:from>
```

```
    <bpel:to>$slaveNotifyRequestMsg.body/tns:messageID</
        bpel:to>
  </bpel:copy>
</bpel:assign>
<bpel:invoke partnerLink="slaveAndMasterPlkVar" portType="
    Slave:ForMaster" operation="Notify" inputVariable="
    slaveNotifyRequestMsg"  name="Notify-1" >
  <bpel:correlations>
    <bpel:correlation set="corr" initiate="yes"/>
  </bpel:correlations>
</bpel:invoke>
<bpel:pick  name="Exclusive_Event-Based_Gateway" >
  <bpel:onMessage partnerLink="slaveAndMasterPlkVar" portType
      ="this:ForSlave" operation="Accept" variable="
      thisAcceptRequestMsg"  name="Accept" >
    <bpel:correlations>
      <bpel:correlation set="corr" initiate="no"/>
    </bpel:correlations>
    <bpel:sequence>
      <bpel:assign name="init-variables-ForEach_Partners" >
        <bpel:copy >
          <bpel:from>
            <bpel:literal>
              <Slave:NotifyRequest>
                <tns:messageID></tns:messageID>
                <tns:effectiveDate></tns:effectiveDate>
                <tns:expirationDate></tns:expirationDate>
                <tns:scope></tns:scope>
                <tns:runningInstances></tns:runningInstances>
                <tns:instanceReferences></
                    tns:instanceReferences>
                <tns:contractRefID></tns:contractRefID>
                <tns:newContractSchema></
                    tns:newContractSchema>
                <tns:newContractRefID></tns:newContractRefID>
                <tns:masterAddress></tns:masterAddress>
                <tns:slaveAddresses></tns:slaveAddresses>
              </Slave:NotifyRequest>
            </bpel:literal>
          </bpel:from>
          <bpel:to>$slaveNotifyRequestMsg.body</bpel:to>
        </bpel:copy>
      </bpel:assign>
      <bpel:empty  name="Empty_Intermediate_Event" />
    </bpel:sequence>
  </bpel:onMessage>
  <bpel:onMessage partnerLink="slaveAndMasterPlkVar" portType
      ="this:ForSlave" operation="Deny" variable="
      thisDenyRequestMsg"  name="Deny" >
    <bpel:correlations>
      <bpel:correlation set="corr" initiate="no"/>
    </bpel:correlations>
    <bpel:sequence>
      <bpel:assign name="init-variables-ForEach_Partners" >
```

```
                <bpel:copy >
                  <bpel:from>
                    <bpel:literal>
                      <Slave:NotifyRequest>
                        <tns:messageID></tns:messageID>
                        <tns:effectiveDate></tns:effectiveDate>
                        <tns:expirationDate></tns:expirationDate>
                        <tns:scope></tns:scope>
                        <tns:runningInstances></tns:runningInstances>
                        <tns:instanceReferences></
                            tns:instanceReferences>
                        <tns:contractRefID></tns:contractRefID>
                        <tns:newContractSchema></
                            tns.newContractSchema>
                        <tns:newContractRefID></tns:newContractRefID>
                        <tns:masterAddress></tns:masterAddress>
                        <tns:slaveAddresses></tns:slaveAddresses>
                      </Slave:NotifyRequest>
                    </bpel:literal>
                  </bpel:from>
                  <bpel:to>$slaveNotifyRequestMsg.body</bpel:to>
                </bpel:copy>
              </bpel:assign>
              <bpel:assign  name="denied" >
                <bpel:copy>
                  <bpel:from>"denied"</bpel:from>
                  <bpel:to>$denied</bpel:to>
                </bpel:copy>
              </bpel:assign>
            </bpel:sequence>
          </bpel:onMessage>
        </bpel:pick>
      </bpel:sequence>
    </bpel:scope>
  </bpel:forEach>
</bpel:scope>
<bpel:if>
  <bpel:condition>$denied = "denied"</bpel:condition>
  <bpel:scope  name="outer-ForEach_Partners-1" >
    <bpel:forEach  name="ForEach_Partners-2"  parallel="yes"
        counterName="ForEach_PartnersCounter">
      <bpel:startCounterValue>1</bpel:startCounterValue>
      <bpel:finalCounterValue>count($thisRequestChangeRequestMsg.body
          /tns:slaveAddresses)</bpel:finalCounterValue>
      <bpel:scope  name="ForEach_Partners-3" >
        <bpel:variables>
          <bpel:variable name="slaveCancelRequestMsg" messageType="
              Slave:CancelRequest"/>
        </bpel:variables>
        <bpel:sequence>
          <bpel:assign name="init-variables-ForEach_Partners" >
            <bpel:copy >
              <bpel:from>
                <bpel:literal>
```

```
                <Slave:CancelRequest>
                  <tns:messageID></tns:messageID>
                  <tns:notifyMessageRefID></tns:notifyMessageRefID>
                </Slave:CancelRequest>
              </bpel:literal>
            </bpel:from>
            <bpel:to>$slaveCancelRequestMsg.body</bpel:to>
          </bpel:copy>
        </bpel:assign>
        <bpel:assign  name="Cancel" >
          <bpel:copy>
            <bpel:from>$thisRequestChangeRequestMsg.body/
                tns:slaveAddresses[$ForEach_PartnersCounter]/text()
                </bpel:from>
            <bpel:to partnerLink="slaveAndMasterPlkVar"
                endpointReference="partnerRole"/>
          </bpel:copy>
          <bpel:copy>
            <bpel:from>concat($ode:pid, "-", $
                ForEach_PartnersCounter)</bpel:from>
            <bpel:to>$slaveCancelRequestMsg.body/
                tns:notifyMessageRefID</bpel:to>
          </bpel:copy>
        </bpel:assign>
        <bpel:invoke partnerLink="slaveAndMasterPlkVar" portType="
            Slave:ForMaster" operation="Cancel" inputVariable="
            slaveCancelRequestMsg"  name="Cancel-1" ></bpel:invoke>
      </bpel:sequence>
    </bpel:scope>
  </bpel:forEach>
</bpel:scope>
<bpel:else>
  <bpel:scope  name="outer-ForEach_Partners-2" >
    <bpel:forEach  name="ForEach_Partners-4"  parallel="yes"
        counterName="ForEach_PartnersCounter">
      <bpel:startCounterValue>1</bpel:startCounterValue>
      <bpel:finalCounterValue>count($thisRequestChangeRequestMsg.
          body/tns:slaveAddresses)</bpel:finalCounterValue>
      <bpel:scope  name="ForEach_Partners-5" >
        <bpel:variables>
          <bpel:variable name="slaveProceedRequestMsg" messageType=
              "Slave:ProceedRequest"/>
        </bpel:variables>
        <bpel:sequence>
          <bpel:assign name="init-variables-ForEach_Partners" >
            <bpel:copy >
              <bpel:from>
                <bpel:literal>
                  <Slave:ProceedRequest>
                    <tns:messageID></tns:messageID>
                    <tns:notifyMessageRefID></
                        tns:notifyMessageRefID>
                  </Slave:ProceedRequest>
                </bpel:literal>
```

```
                    </bpel:from>
                    <bpel:to>$slaveProceedRequestMsg.body</bpel:to>
                  </bpel:copy>
                </bpel:assign>
                <bpel:assign  name="Proceed" >
                  <bpel:copy>
                <bpel:from>$thisRequestChangeRequestMsg.body/
                    tns:slaveAddresses[$ForEach_PartnersCounter]/text()</
                    bpel:from>
                <bpel:to partnerLink="slaveAndMasterPlkVar"
                    endpointReference="partnerRole"/>
                  </bpel:copy>
                  <bpel:copy>
                    <bpel:from>concat($ode:pid, "-", $
                        ForEach_PartnersCounter)</bpel:from>
                    <bpel:to>$slaveProceedRequestMsg.body/
                        tns:notifyMessageRefID</bpel:to>
                  </bpel:copy>
                </bpel:assign>
                <bpel:invoke partnerLink="slaveAndMasterPlkVar" portType=
                    "Slave:ForMaster" operation="Proceed" inputVariable="
                    slaveProceedRequestMsg"  name="Proceed-1" ></
                    bpel:invoke>
              </bpel:sequence>
            </bpel:scope>
          </bpel:forEach>
        </bpel:scope>
      </bpel:else>
    </bpel:if>
    <bpel:empty  name="Empty_End_Event" />
  </bpel:sequence>
</bpel:process>
```

C.2 Schéma BPEL du processus *Slave*

cpc-Slave.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
    executable" xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/
    varprop" xmlns:pnlk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3
    .org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:ode="http://www.apache.org/ode/type/extension"
    xmlns:DataBase="http://latece.uqam.ca/cpc/DataBase" xmlns:this="http:
    //latece.uqam.ca/cpc/Slave" xmlns:cpc-CPC_Interface="http://latece.
    uqam.ca/cpc/CPC_Interface" xmlns:Master="http://latece.uqam.ca/cpc/
    Master" xmlns:tns="http://latece.uqam.ca/cpc/xsd" xmlns:CPC_Interface
    ="http://latece.uqam.ca/cpc/CPC_Interface" xmlns:diag="http://latece.
    uqam.ca/cpc" xmlns:cpc-DataBase="http://latece.uqam.ca/cpc/DataBase"
    xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns:bpmn="http://
    www.intalio.com/bpms" xmlns:atomic="http://ode.apache.org/atomicScope
    " queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
    expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath2.0"
    name="Slave"  targetNamespace="http://latece.uqam.ca/cpc/Slave">
  <bpel:import namespace="http://latece.uqam.ca/cpc/CPC_Interface"
      location="cpc-Interface.wsdl" importType="http://schemas.xmlsoap.
      org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc/DataBase" location="
      cpc-DataBase.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc" location="cpc.wsdl"
      importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc/Master" location="cpc
      -Master.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:import namespace="http://latece.uqam.ca/cpc/Slave" location="cpc-
      Slave.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
  <bpel:partnerLinks>
    <bpel:partnerLink name="slaveAndMasterPlkVar" partnerLinkType="
        diag:SlaveAndMaster" initializePartnerRole="yes" myRole="
        Slave_for_Master" partnerRole="Master_for_Slave"/>
    <bpel:partnerLink name="slaveAndDataBaseForPortForSlavePlkVar"
        partnerLinkType="diag:SlaveAndDataBaseForPortForSlavePlk"
        initializePartnerRole="yes" partnerRole="DataBase_for_Slave"/>
    <bpel:partnerLink name="cPC_InterfaceAndSlaveForPortForSlavePlkVar"
        partnerLinkType="diag:CPC_InterfaceAndSlaveForPortForSlavePlk"
        initializePartnerRole="yes" partnerRole="CPC_Interface_for_Slave"
        />
  </bpel:partnerLinks>
  <bpel:correlationSets>
    <bpel:correlationSet name="corr" properties="this:refID"/>
  </bpel:correlationSets>
  <bpel:variables>
    <bpel:variable name="thisProceedRequestMsg" messageType="
        this:ProceedRequest"/>
    <bpel:variable name="thisNotifyRequestMsg" messageType="
        this:NotifyRequest"/>
```

```
  <bpel:variable name="cpc-DataBaseProgram_MigrationMsg" messageType="
      cpc-DataBase:Program_MigrationRequest"/>
  <bpel:variable name="cpc-DataBaseProgram_MigrationResponseMsg"
      messageType="cpc-DataBase:Program_MigrationResponse"/>
  <bpel:variable name="cpc-CPC_InterfaceGet_AuthorizationMsg"
      messageType="cpc-CPC_Interface:Get_AuthorizationRequest"/>
  <bpel:variable name="cpc-CPC_InterfaceGet_AuthorizationResponseMsg"
      messageType="cpc-CPC_Interface:Get_AuthorizationResponse"/>
  <bpel:variable name="cpc-DataBaseRecord_Change_CanceledMsg"
      messageType="cpc-DataBase:Record_Change_CanceledRequest"/>
  <bpel:variable name="cpc-DataBaseRecord_Change_CanceledResponseMsg"
      messageType="cpc-DataBase:Record_Change_CanceledResponse"/>
  <bpel:variable name="cpc-DataBaseRecord__Change_DeniedMsg"
      messageType="cpc-DataBase:Record__Change_DeniedRequest"/>
  <bpel:variable name="cpc-DataBaseRecord__Change_DeniedResponseMsg"
      messageType="cpc-DataBase:Record__Change_DeniedResponse"/>
  <bpel:variable name="thisCancelRequestMsg" messageType="
      this:CancelRequest"/>
  <bpel:variable name="masterDenyRequestMsg" messageType="
      Master:DenyRequest"/>
  <bpel:variable name="masterAcceptRequestMsg" messageType="
      Master:AcceptRequest"/>
</bpel:variables>
<bpel:sequence>
  <bpel:receive partnerLink="slaveAndMasterPlkVar" portType="
      this:ForMaster" operation="Notify" variable="thisNotifyRequestMsg
      " createInstance="yes"  name="Notify" >
    <bpel:correlations>
      <bpel:correlation set="corr" initiate="yes"/>
    </bpel:correlations>
  </bpel:receive>
  <bpel:assign name="init-variables-Slave" >
    <bpel:copy >
      <bpel:from>
        <bpel:literal>
          <cpc-DataBase:Program_MigrationRequest>
            <tns:processID></tns:processID>
            <tns:initiatorProcessID></tns:initiatorProcessID>
            <tns:proceedMessageRefID></tns:proceedMessageRefID>
            <tns:notifyMessage>
              <tns:messageID></tns:messageID>
              <tns:effectiveDate></tns:effectiveDate>
              <tns:expirationDate></tns:expirationDate>
              <tns:scope></tns:scope>
              <tns:runningInstances></tns:runningInstances>
              <tns:instanceReferences></tns:instanceReferences>
              <tns:contractRefID></tns:contractRefID>
              <tns:newContractSchema></tns:newContractSchema>
              <tns:newContractRefID></tns:newContractRefID>
              <tns:masterAddress></tns:masterAddress>
              <tns:slaveAddresses></tns:slaveAddresses>
            </tns:notifyMessage>
          </cpc-DataBase:Program_MigrationRequest>
        </bpel:literal>
```

```
  </bpel:from>
  <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-CPC_Interface:Get_AuthorizationRequest>
        <tns:messageID></tns:messageID>
        <tns:effectiveDate></tns:effectiveDate>
        <tns:expirationDate></tns:expirationDate>
        <tns:scope></tns:scope>
        <tns:runningInstances></tns:runningInstances>
        <tns:instanceReferences></tns:instanceReferences>
        <tns:contractRefID></tns:contractRefID>
        <tns:newContractSchema></tns:newContractSchema>
        <tns:newContractRefID></tns:newContractRefID>
        <tns:masterAddress></tns:masterAddress>
        <tns:slaveAddresses></tns:slaveAddresses>
      </cpc-CPC_Interface:Get_AuthorizationRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-CPC_InterfaceGet_AuthorizationMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-DataBase:Record_Change_CanceledRequest></cpc-
          DataBase:Record_Change_CanceledRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-DataBaseRecord_Change_CanceledMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-DataBase:Record__Change_DeniedRequest></cpc-
          DataBase:Record__Change_DeniedRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-DataBaseRecord__Change_DeniedMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <Master:DenyRequest>
        <tns:messageID></tns:messageID>
        <tns:notifyMessageRefID></tns:notifyMessageRefID>
        <tns:reasons></tns:reasons>
      </Master:DenyRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$masterDenyRequestMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
```

```
    <bpel:from>
      <bpel:literal>
        <Master:AcceptRequest>
          <tns:messageID></tns:messageID>
          <tns:notifyMessageRefID></tns:notifyMessageRefID>
        </Master:AcceptRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$masterAcceptRequestMsg.body</bpel:to>
  </bpel:copy>
</bpel:assign>
<bpel:assign  name="Get_Authorization" >
  <bpel:copy>
    <bpel:from>$thisNotifyRequestMsg.body</bpel:from>
    <bpel:to>$cpc-CPC_InterfaceGet_AuthorizationMsg.body</bpel:to>
  </bpel:copy>
</bpel:assign>
<bpel:invoke partnerLink="cPC_InterfaceAndSlaveForPortForSlavePlkVar"
     portType="cpc-CPC_Interface:ForSlave" operation="
    Get_Authorization" inputVariable="cpc-
    CPC_InterfaceGet_AuthorizationMsg" outputVariable="cpc-
    CPC_InterfaceGet_AuthorizationResponseMsg"  name="
    Get_Authorization-1" ></bpel:invoke>
<bpel:if>
  <bpel:condition>"true" = $cpc-
      CPC_InterfaceGet_AuthorizationResponseMsg.body/
      tns:isChangeAccepted</bpel:condition>
  <bpel:sequence>
    <bpel:assign  name="Accept" >
      <bpel:copy>
        <bpel:from>$ode:pid</bpel:from>
        <bpel:to>$masterAcceptRequestMsg.body/tns:messageID</bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>$thisNotifyRequestMsg.body/tns:messageID</
            bpel:from>
        <bpel:to>$masterAcceptRequestMsg.body/tns:notifyMessageRefID<
            /bpel:to>
      </bpel:copy>
      <bpel:copy>
        <bpel:from>$thisNotifyRequestMsg.body/tns:masterAddress/text
            ()</bpel:from>
        <bpel:to partnerLink="slaveAndMasterPlkVar" endpointReference
            ="partnerRole"/>
      </bpel:copy>
    </bpel:assign>
    <bpel:invoke partnerLink="slaveAndMasterPlkVar" portType="
        Master:ForSlave" operation="Accept" inputVariable="
        masterAcceptRequestMsg"  name="Accept-1" ></bpel:invoke>
  </bpel:sequence>
  <bpel:else>
    <bpel:sequence>
      <bpel:assign  name="Deny" >
        <bpel:copy>
```

```
            <bpel:from>$cpc-CPC_InterfaceGet_AuthorizationResponseMsg.
                body/tns:reasons</bpel:from>
            <bpel:to>$masterDenyRequestMsg.body/tns:reasons</bpel:to>
          </bpel:copy>
          <bpel:copy>
            <bpel:from>$thisNotifyRequestMsg.body/tns:messageID</
                bpel:from>
            <bpel:to>$masterDenyRequestMsg.body/tns:notifyMessageRefID<
                /bpel:to>
          </bpel:copy>
          <bpel:copy>
            <bpel:from>$thisNotifyRequestMsg.body/tns:masterAddress/
                text()</bpel:from>
            <bpel:to partnerLink="slaveAndMasterPlkVar"
                endpointReference="partnerRole"/>
          </bpel:copy>
        </bpel:assign>
        <bpel:invoke partnerLink="slaveAndMasterPlkVar" portType="
            Master:ForSlave" operation="Deny" inputVariable="
            masterDenyRequestMsg"  name="Deny-1" ></bpel:invoke>
        <bpel:invoke partnerLink="slaveAndDataBaseForPortForSlavePlkVar
            " portType="cpc-DataBase:ForSlave" operation="
            Record__Change_Denied" inputVariable="cpc-
            DataBaseRecord__Change_DeniedMsg" outputVariable="cpc-
            DataBaseRecord__Change_DeniedResponseMsg"  name="
            Record__Change_Denied" ></bpel:invoke>
      </bpel:sequence>
    </bpel:else>
  </bpel:if>
  <bpel:pick  name="Exclusive_Event-Based_Gateway" >
    <bpel:onMessage partnerLink="slaveAndMasterPlkVar" portType="
        this:ForMaster" operation="Cancel" variable="
        thisCancelRequestMsg"  name="Cancel" >
      <bpel:correlations>
        <bpel:correlation set="corr" initiate="no"/>
      </bpel:correlations>
      <bpel:sequence>
        <bpel:assign name="init-variables-Slave" >
          <bpel:copy >
            <bpel:from>
              <bpel:literal>
                <cpc-DataBase:Program_MigrationRequest>
                  <tns:processID></tns:processID>
                  <tns:initiatorProcessID></tns:initiatorProcessID>
                  <tns:proceedMessageRefID></tns:proceedMessageRefID>
                  <tns:notifyMessage>
                    <tns:messageID></tns:messageID>
                    <tns:effectiveDate></tns:effectiveDate>
                    <tns:expirationDate></tns:expirationDate>
                    <tns:scope></tns:scope>
                    <tns:runningInstances></tns:runningInstances>
                    <tns:instanceReferences></tns:instanceReferences>
                    <tns:contractRefID></tns:contractRefID>
                    <tns:newContractSchema></tns:newContractSchema>
```

```
          <tns:newContractRefID></tns:newContractRefID>
          <tns:masterAddress></tns:masterAddress>
          <tns:slaveAddresses></tns:slaveAddresses>
        </tns:notifyMessage>
      </cpc-DataBase:Program_MigrationRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body</bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-CPC_Interface:Get_AuthorizationRequest>
        <tns:messageID></tns:messageID>
        <tns:effectiveDate></tns:effectiveDate>
        <tns:expirationDate></tns:expirationDate>
        <tns:scope></tns:scope>
        <tns:runningInstances></tns:runningInstances>
        <tns:instanceReferences></tns:instanceReferences>
        <tns:contractRefID></tns:contractRefID>
        <tns:newContractSchema></tns:newContractSchema>
        <tns:newContractRefID></tns:newContractRefID>
        <tns:masterAddress></tns:masterAddress>
        <tns:slaveAddresses></tns:slaveAddresses>
      </cpc-CPC_Interface:Get_AuthorizationRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-CPC_InterfaceGet_AuthorizationMsg.body</
      bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-DataBase:Record_Change_CanceledRequest></cpc-
          DataBase:Record_Change_CanceledRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-DataBaseRecord_Change_CanceledMsg.body</
      bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <cpc-DataBase:Record__Change_DeniedRequest></cpc-
          DataBase:Record__Change_DeniedRequest>
    </bpel:literal>
  </bpel:from>
  <bpel:to>$cpc-DataBaseRecord__Change_DeniedMsg.body</
      bpel:to>
</bpel:copy>
<bpel:copy >
  <bpel:from>
    <bpel:literal>
      <Master:DenyRequest>
```

```
                <tns:messageID></tns:messageID>
                <tns:notifyMessageRefID></tns:notifyMessageRefID>
                <tns:reasons></tns:reasons>
              </Master:DenyRequest>
            </bpel:literal>
          </bpel:from>
          <bpel:to>$masterDenyRequestMsg.body</bpel:to>
        </bpel:copy>
        <bpel:copy >
          <bpel:from>
            <bpel:literal>
              <Master:AcceptRequest>
                <tns:messageID></tns:messageID>
                <tns:notifyMessageRefID></tns:notifyMessageRefID>
              </Master:AcceptRequest>
            </bpel:literal>
          </bpel:from>
          <bpel:to>$masterAcceptRequestMsg.body</bpel:to>
        </bpel:copy>
      </bpel:assign>
      <bpel:invoke partnerLink="slaveAndDataBaseForPortForSlavePlkVar
          " portType="cpc-DataBase:ForSlave" operation="
          Record_Change_Canceled" inputVariable="cpc-
          DataBaseRecord_Change_CanceledMsg" outputVariable="cpc-
          DataBaseRecord_Change_CanceledResponseMsg"  name="
          Record_Change_Canceled" ></bpel:invoke>
    </bpel:sequence>
  </bpel:onMessage>
  <bpel:onMessage partnerLink="slaveAndMasterPlkVar" portType="
      this:ForMaster" operation="Proceed" variable="
      thisProceedRequestMsg"  name="Proceed" >
    <bpel:correlations>
      <bpel:correlation set="corr" initiate="no"/>
    </bpel:correlations>
    <bpel:sequence>
      <bpel:assign name="init-variables-Slave" >
        <bpel:copy >
          <bpel:from>
            <bpel:literal>
              <cpc-DataBase:Program_MigrationRequest>
                <tns:processID></tns:processID>
                <tns:initiatorProcessID></tns:initiatorProcessID>
                <tns:proceedMessageRefID></tns:proceedMessageRefID>
                <tns:notifyMessage>
                  <tns:messageID></tns:messageID>
                  <tns:effectiveDate></tns:effectiveDate>
                  <tns:expirationDate></tns:expirationDate>
                  <tns:scope></tns:scope>
                  <tns:runningInstances></tns:runningInstances>
                  <tns:instanceReferences></tns:instanceReferences>
                  <tns:contractRefID></tns:contractRefID>
                  <tns:newContractSchema></tns:newContractSchema>
                  <tns:newContractRefID></tns:newContractRefID>
                  <tns:masterAddress></tns:masterAddress>
```

```
            <tns:slaveAddresses></tns:slaveAddresses>
          </tns:notifyMessage>
        </cpc-DataBase:Program_MigrationRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body</bpel:to>
  </bpel:copy>
  <bpel:copy >
    <bpel:from>
      <bpel:literal>
        <cpc-CPC_Interface:Get_AuthorizationRequest>
          <tns:messageID></tns:messageID>
          <tns:effectiveDate></tns:effectiveDate>
          <tns:expirationDate></tns:expirationDate>
          <tns:scope></tns:scope>
          <tns:runningInstances></tns:runningInstances>
          <tns:instanceReferences></tns:instanceReferences>
          <tns:contractRefID></tns:contractRefID>
          <tns:newContractSchema></tns:newContractSchema>
          <tns:newContractRefID></tns:newContractRefID>
          <tns:masterAddress></tns:masterAddress>
          <tns:slaveAddresses></tns:slaveAddresses>
        </cpc-CPC_Interface:Get_AuthorizationRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$cpc-CPC_InterfaceGet_AuthorizationMsg.body</
        bpel:to>
  </bpel:copy>
  <bpel:copy >
    <bpel:from>
      <bpel:literal>
        <cpc-DataBase:Record_Change_CanceledRequest></cpc-
            DataBase:Record_Change_CanceledRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$cpc-DataBaseRecord_Change_CanceledMsg.body</
        bpel:to>
  </bpel:copy>
  <bpel:copy >
    <bpel:from>
      <bpel:literal>
        <cpc-DataBase:Record__Change_DeniedRequest></cpc-
            DataBase:Record__Change_DeniedRequest>
      </bpel:literal>
    </bpel:from>
    <bpel:to>$cpc-DataBaseRecord__Change_DeniedMsg.body</
        bpel:to>
  </bpel:copy>
  <bpel:copy >
    <bpel:from>
      <bpel:literal>
        <Master:DenyRequest>
          <tns:messageID></tns:messageID>
          <tns:notifyMessageRefID></tns:notifyMessageRefID>
```

```
                    <tns:reasons></tns:reasons>
                  </Master:DenyRequest>
                </bpel:literal>
              </bpel:from>
              <bpel:to>$masterDenyRequestMsg.body</bpel:to>
            </bpel:copy>
            <bpel:copy >
              <bpel:from>
                <bpel:literal>
                  <Master:AcceptRequest>
                    <tns:messageID></tns:messageID>
                    <tns:notifyMessageRefID></tns:notifyMessageRefID>
                  </Master:AcceptRequest>
                </bpel:literal>
              </bpel:from>
              <bpel:to>$masterAcceptRequestMsg.body</bpel:to>
            </bpel:copy>
          </bpel:assign>
          <bpel:assign  name="Program_Migration" >
            <bpel:copy>
              <bpel:from>$thisNotifyRequestMsg.body</bpel:from>
              <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body/
                  tns:notifyMessage</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>$cpc-CPC_InterfaceGet_AuthorizationResponseMsg.
                  body/tns:initiatorProcessID</bpel:from>
              <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body/
                  tns:initiatorProcessID</bpel:to>
            </bpel:copy>
            <bpel:copy>
              <bpel:from>$cpc-CPC_InterfaceGet_AuthorizationResponseMsg.
                  body/tns:processID</bpel:from>
              <bpel:to>$cpc-DataBaseProgram_MigrationMsg.body/
                  tns:processID</bpel:to>
            </bpel:copy>
          </bpel:assign>
          <bpel:invoke partnerLink="slaveAndDataBaseForPortForSlavePlkVar
              " portType="cpc-DataBase:ForSlave" operation="
              Program_Migration" inputVariable="cpc-
              DataBaseProgram_MigrationMsg" outputVariable="cpc-
              DataBaseProgram_MigrationResponseMsg"  name="
              Program_Migration-1" ></bpel:invoke>
        </bpel:sequence>
      </bpel:onMessage>
    </bpel:pick>
    <bpel:empty  name="Empty_End_Event" />
  </bpel:sequence>
</bpel:process>
```

C.3 Définitions WSDL des interfaces du CPC

cpc-Master.wsdl

```
<?xml version='1.0' encoding='utf-8'?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:CPC_Interface="http://latece.uqam.ca/cpc/CPC_Interface"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xml="http://www.w3.
    org/XML/1998/namespace" xmlns:bpel="http://docs.oasis-open.org/wsbpel
    /2.0/process/executable" xmlns:diag="http://latece.uqam.ca/cpc"
    xmlns:tns="http://latece.uqam.ca/cpc/xsd" xmlns:Slave="http://latece.
    uqam.ca/cpc/Slave" xmlns:vprop="http://docs.oasis-open.org/wsbpel
    /2.0/varprop" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:pnlk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:this="http://latece.uqam.ca/cpc/Master" xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance" targetNamespace="http://latece.uqam.
    ca/cpc/Master">
    <wsdl:types>
        <xs:schema elementFormDefault="qualified" targetNamespace="http:
            //latece.uqam.ca/cpc/Master">
            <xs:import namespace="http://latece.uqam.ca/cpc/xsd"
                schemaLocation="cpc.xsd"/>
            <xs:element name="RequestChangeRequest" type="tns:NotifyType"
                />
            <xs:element name="RequestChangeResponse" type="
                tns:ChangeResponseType"/>
            <xs:element name="DenyRequest" type="tns:DenyType"/>
            <xs:element name="AcceptRequest" type="tns:CancelType"/>
        </xs:schema>
    </wsdl:types>
    <wsdl:import namespace="http://latece.uqam.ca/cpc/Slave" location="
        cpc-Slave.wsdl"/>
    <wsdl:message name="RequestChangeRequest">
        <wsdl:part name="body" element="this:RequestChangeRequest"/>
    </wsdl:message>
    <wsdl:message name="RequestChangeResponse">
        <wsdl:part name="body" element="this:RequestChangeResponse"/>
    </wsdl:message>
    <wsdl:message name="DenyRequest">
        <wsdl:part name="body" element="this:DenyRequest"/>
    </wsdl:message>
    <wsdl:message name="AcceptRequest">
        <wsdl:part name="body" element="this:AcceptRequest"/>
    </wsdl:message>
    <wsdl:portType name="ForCPC_Interface">
        <wsdl:operation name="RequestChange">
            <wsdl:input message="this:RequestChangeRequest" name="
                RequestChange"/>
            <wsdl:output message="this:RequestChangeResponse" name="
                RequestChangeResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:portType name="ForSlave">
        <wsdl:operation name="Accept">
```

```
        <wsdl:input message="this:AcceptRequest" name="Accept"/>
    </wsdl:operation>
    <wsdl:operation name="Deny">
        <wsdl:input message="this:DenyRequest" name="Deny"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CanonicBindingForCPC_Interface" type="
    this:ForCPC_Interface">
    <soap:binding style="document" transport="http://schemas.xmlsoap.
        org/soap/http"/>
    <wsdl:operation name="RequestChange">
        <soap:operation style="document" soapAction="http://latece.
            uqam.ca/cpc/Master/ForCPC_Interface/RequestChange"/>
        <wsdl:input name="RequestChange">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="RequestChangeResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="CanonicBindingForSlave" type="this:ForSlave">
    <soap:binding style="document" transport="http://schemas.xmlsoap.
        org/soap/http"/>
    <wsdl:operation name="Deny">
        <soap:operation style="document" soapAction="http://latece.
            uqam.ca/cpc/Master/ForSlave/Deny"/>
        <wsdl:input name="Deny">
            <soap:body use="literal"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="Accept">
        <soap:operation style="document" soapAction="http://latece.
            uqam.ca/cpc/Master/ForSlave/Accept"/>
        <wsdl:input name="Accept">
            <soap:body use="literal"/>
        </wsdl:input>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CanonicServiceForCPC_Interface">
    <wsdl:port name="canonicPort" binding="
        this:CanonicBindingForCPC_Interface">
        <soap:address location="http://latece.no-ip.org:8080/ode/
            processes/cpc/cpc/Master/CPC_Interface"/>
    </wsdl:port>
</wsdl:service>
<wsdl:service name="CanonicServiceForSlave">
    <wsdl:port name="canonicPort" binding="
        this:CanonicBindingForSlave">
        <soap:address location="http://latece.no-ip.org:8080/ode/
            processes/cpc/cpc/Master/Slave"/>
    </wsdl:port>
</wsdl:service>
<vprop:property name="refID" type="xs:string"/>
```

```
    <vprop:propertyAlias propertyName="this:refID" messageType="
        this:DenyRequest" part="body">
        <vprop:query>tns:notifyMessageRefID/text()</vprop:query>
    </vprop:propertyAlias>
    <vprop:propertyAlias propertyName="this:refID" messageType="
        this:AcceptRequest" part="body">
        <vprop:query>tns:notifyMessageRefID/text()</vprop:query>
    </vprop:propertyAlias>
    <vprop:propertyAlias propertyName="this:refID" messageType="
        Slave:NotifyRequest" part="body">
        <vprop:query>tns:messageID/text()</vprop:query>
    </vprop:propertyAlias>
</wsdl:definitions>
```

cpc-Slave.wsdl

```
<?xml version='1.0' encoding='utf-8'?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:pnlk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:bpel="http:
    //docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:vprop="
    http://docs.oasis-open.org/wsbpel/2.0/varprop" xmlns:this="http://
    latece.uqam.ca/cpc/Slave" xmlns:DataBase="http://latece.uqam.ca/cpc/
    DataBase" xmlns:cpc-CPC_Interface="http://latece.uqam.ca/cpc/
    CPC_Interface" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance" xmlns:Master="http://
    latece.uqam.ca/cpc/Master" xmlns:tns="http://latece.uqam.ca/cpc/xsd"
    xmlns:CPC_Interface="http://latece.uqam.ca/cpc/CPC_Interface"
    xmlns:diag="http://latece.uqam.ca/cpc" xmlns:cpc-DataBase="http://
    latece.uqam.ca/cpc/DataBase" xmlns:xml="http://www.w3.org/XML/1998/
    namespace" targetNamespace="http://latece.uqam.ca/cpc/Slave">
    <wsdl:types>
        <xs:schema elementFormDefault="qualified" targetNamespace="http:
            //latece.uqam.ca/cpc/Slave">
            <xs:import namespace="http://latece.uqam.ca/cpc/xsd"
                schemaLocation="cpc.xsd"/>
            <xs:element name="NotifyRequest" type="tns:NotifyType"/>
            <xs:element name="ProceedRequest" type="tns:ProceedType"/>
            <xs:element name="CancelRequest" type="tns:CancelType"/>
        </xs:schema>
    </wsdl:types>
    <wsdl:message name="NotifyRequest">
        <wsdl:part name="body" element="this:NotifyRequest"/>
    </wsdl:message>
    <wsdl:message name="ProceedRequest">
        <wsdl:part name="body" element="this:ProceedRequest"/>
    </wsdl:message>
    <wsdl:message name="CancelRequest">
        <wsdl:part name="body" element="this:CancelRequest"/>
    </wsdl:message>
    <wsdl:portType name="ForMaster">
        <wsdl:operation name="Cancel">
            <wsdl:input message="this:CancelRequest" name="Cancel"/>
        </wsdl:operation>
        <wsdl:operation name="Notify">
            <wsdl:input message="this:NotifyRequest" name="Notify"/>
        </wsdl:operation>
        <wsdl:operation name="Proceed">
            <wsdl:input message="this:ProceedRequest" name="Proceed"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="CanonicBindingForMaster" type="this:ForMaster">
        <soap:binding style="document" transport="http://schemas.xmlsoap.
            org/soap/http"/>
        <wsdl:operation name="Notify">
            <soap:operation style="document" soapAction="http://latece.
                uqam.ca/cpc/Slave/ForMaster/Notify"/>
            <wsdl:input name="Notify">
```

```
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
        <wsdl:operation name="Proceed">
            <soap:operation style="document" soapAction="http://latece.
                uqam.ca/cpc/Slave/ForMaster/Proceed"/>
            <wsdl:input name="Proceed">
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
        <wsdl:operation name="Cancel">
            <soap:operation style="document" soapAction="http://latece.
                uqam.ca/cpc/Slave/ForMaster/Cancel"/>
            <wsdl:input name="Cancel">
                <soap:body use="literal"/>
            </wsdl:input>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="CanonicServiceForMaster">
        <wsdl:port name="canonicPort" binding="
            this:CanonicBindingForMaster">
            <soap:address location="http://latece.no-ip.org:8080/ode/
                processes/cpc/cpc/Slave/Master"/>
        </wsdl:port>
    </wsdl:service>
    <vprop:property name="refID" type="xs:string"/>
    <vprop:propertyAlias propertyName="this:refID" messageType="
        this:NotifyRequest" part="body">
        <vprop:query>tns:messageID/text()</vprop:query>
    </vprop:propertyAlias>
    <vprop:propertyAlias propertyName="this:refID" messageType="
        this:ProceedRequest" part="body">
        <vprop:query>tns:notifyMessageRefID/text()</vprop:query>
    </vprop:propertyAlias>
    <vprop:propertyAlias propertyName="this:refID" messageType="
        this:CancelRequest" part="body">
        <vprop:query>tns:notifyMessageRefID/text()</vprop:query>
    </vprop:propertyAlias>
</wsdl:definitions>
```

cpc.wsdl

```
<?xml version='1.0' encoding='utf-8'?>
<wsdl:definitions xmlns:Slave="http://latece.uqam.ca/cpc/Slave" xmlns:xs=
    "http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xmlns:tns="http://latece.uqam.ca/cpc/xsd"
    xmlns:diag="http://latece.uqam.ca/cpc" xmlns:cpc-DataBase="http://
    latece.uqam.ca/cpc/DataBase" xmlns:cpc-CPC_Interface="http://latece.
    uqam.ca/cpc/CPC_Interface" xmlns:Master="http://latece.uqam.ca/cpc/
    Master" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:pnlk="
    http://docs.oasis-open.org/wsbpel/2.0/plnktype" xmlns:soap="http://
    schemas.xmlsoap.org/wsdl/soap/" xmlns:bpel="http://docs.oasis-open.
    org/wsbpel/2.0/process/executable" xmlns:vprop="http://docs.oasis-
    open.org/wsbpel/2.0/varprop" xmlns:bpdm="http://www.intalio/designer/
    business-process-data-modeling" targetNamespace="http://latece.uqam.
    ca/cpc">
    <wsdl:import namespace="http://latece.uqam.ca/cpc/CPC_Interface"
        location="cpc-Interface.wsdl"/>
    <wsdl:import namespace="http://latece.uqam.ca/cpc/DataBase" location=
        "cpc-DataBase2.wsdl"/>
    <wsdl:import namespace="http://latece.uqam.ca/cpc/Master" location="
        cpc-Master.wsdl"/>
    <wsdl:import namespace="http://latece.uqam.ca/cpc/Slave" location="
        cpc-Slave.wsdl"/>
    <pnlk:partnerLinkType name="CPC_InterfaceAndSlaveForPortForSlavePlk">
        <pnlk:role name="CPC_Interface_for_Slave" portType="cpc-
            CPC_Interface:ForSlave"/>
    </pnlk:partnerLinkType>
    <pnlk:partnerLinkType name="SlaveAndDataBaseForPortForSlavePlk">
        <pnlk:role name="DataBase_for_Slave" portType="cpc-
            DataBase:ForSlave"/>
    </pnlk:partnerLinkType>
    <pnlk:partnerLinkType name="CPC_InterfaceAndMaster">
        <pnlk:role name="Master_for_CPC_Interface" portType="
            Master:ForCPC_Interface"/>
    </pnlk:partnerLinkType>
    <pnlk:partnerLinkType name="SlaveAndMaster">
        <pnlk:role name="Slave_for_Master" portType="Slave:ForMaster"/>
        <pnlk:role name="Master_for_Slave" portType="Master:ForSlave"/>
    </pnlk:partnerLinkType>
</wsdl:definitions>
```

C.4 Définition XSD des messages du CPC

cpc.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
    latece.uqam.ca/cpc/xsd" xmlns:tns="http://latece.uqam.ca/cpc/xsd"
    elementFormDefault="qualified">

    <complexType name="NotifyType">
      <sequence>
        <element name="messageID" type="string"/>
        <element name="effectiveDate" type="dateTime"/>
        <element name="expirationDate" type="dateTime"/>
        <element name="scope" type="tns:ScopeType"/>
        <element name="runningInstances" type="boolean"/>
        <element name="instanceReferences" type="string" minOccurs="0"
            maxOccurs="unbounded"/>
        <element name="contractRefID" type="string"/>
        <element name="newContractSchema" type="string"/>
        <element name="newContractRefID" type="string"/>
        <element name="masterAddress" type="string"/>
        <element name="slaveAddresses" type="string" maxOccurs="unbounded
            "/>
      </sequence>
    </complexType>

    <simpleType name="ScopeType">
      <restriction base="string">
        <enumeration value="schema"/>
        <enumeration value="instance"/>
      </restriction>
    </simpleType>

    <complexType name="AcceptType">
      <sequence>
        <element name="messageID" type="string"/>
        <element name="notifyMessageRefID" type="string"/>
      </sequence>
    </complexType>

    <complexType name="ProceedType">
      <sequence>
        <element name="messageID" type="string"/>
        <element name="notifyMessageRefID" type="string"/>
      </sequence>
    </complexType>

    <complexType name="CancelType">
      <sequence>
        <element name="messageID" type="string"/>
        <element name="notifyMessageRefID" type="string"/>
      </sequence>
    </complexType>
```

```
    <complexType name="DenyType">
      <sequence>
        <element name="messageID" type="string"/>
        <element name="notifyMessageRefID" type="string"/>
        <element name="reasons" type="string"/>
      </sequence>
    </complexType>

    <complexType name="MigrationType">
      <sequence>
        <element name="processID" type="string" maxOccurs="unbounded"/>
        <element name="initiatorProcessID" type="string" minOccurs="0"/>
        <element name="proceedMessageRefID" type="string"/>
        <element name="notifyMessage" type="tns:NotifyType"/>
      </sequence>
    </complexType>

    <complexType name="ChangeResponseType">
      <sequence>
        <element name="changeRequestRefID" type="string"/>
      </sequence>
    </complexType>

    <complexType name="NotifyResponseType">
      <sequence>
        <element name="notifyMessageRefID" type="string"/>
      </sequence>
    </complexType>

    <complexType name="AuthorizationType">
      <sequence>
        <element name="notifyMessageRefID" type="string"/>
        <element name="isChangeAccepted" type="boolean"/>
        <element name="processID" type="string" maxOccurs="unbounded"/>
        <element name="initiatorProcessID" type="string"/>
        <element name="reasons" type="string"/>
      </sequence>
    </complexType>


    <element name="notify" type="tns:NotifyType"/>
    <element name="accept" type="tns:AcceptType"/>
    <element name="deny" type="tns:DenyType"/>
    <element name="proceed" type="tns:ProceedType"/>
    <element name="cancel" type="tns:CancelType"/>
    <element name="migration" type="tns:MigrationType"/>
    <element name="changeResponse" type="tns:ChangeResponseType"/>
    <element name="notifyResponse" type="tns:NotifyResponseType"/>
    <element name="authorization" type="tns:AuthorizationType"/>
    <element name="listIds" type="string"/>

</schema>
```

BIBLIOGRAPHIE

Adams, M., A. ter Hofstede, D. Edmond, et W. van der Aalst. 2006. *Worklets : A Service-Oriented Implementation of Dynamic Flexibility in Workflows*. Coll. Meersman, R. et Z. Tari, éditeurs, Coll. « *On the Move to Meaningful Internet Systems 2006 : CoopIS, DOA, GADA, and ODBASE* ». T. 4275, série *Lecture Notes in Computer Science*, p. 291–308. Springer Berlin / Heidelberg.

Adams, M., A. H. M. Ter Hofstede, W. M. P. Van Der Aalst, et D. Edmond. 2007. « Dynamic, extensible and context-aware exception handling for workflows ». In *OTM'07 : Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems*, p. 95–112, Berlin, Heidelberg. Springer-Verlag.

Alves, A., A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. v. d. Rijn, P. Yendluri, et A. Yiu. 2007. WS-BPEL : Web Service Business Process Executional Language v2.0. Organization for the Advancement of Structured Information Standards (OASIS). `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` [En ligne]. Consulté le 23 avril 2011.

Barreto, C., V. Bullard, T. Erl, J. Evdemon, D. Jordan, K. Kand, D. König, S. Moser, R. Stout, R. Ten-Hove, I. Trickovic, D. van der Rijn, et A. Yiu. 2007. Web Service Business Process Executional Language Version 2.0, Primer. Organization for the Advancement of Structured Information Standards (OASIS). `http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm` [En ligne]. Consulté le 23 avril 2011.

Bauer, T. et P. Dadam. 1997. « A distributed execution environment for large-scale workflow management systems with subnets and server migration ». In *Cooperative Information Systems, 1997. COOPIS '97., Proceedings of the Second IFCIS International Conference on*, p. 99–108.

BISAC SCEDI committee. 2005. 850 Purchase Order, Version : X12-4010. Federal Information Processing Standards Publications (FIPS PUBS).

Box, D., E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana, et S. Winkler. 2004. Web Services Addressing (WS-Addressing). World Wide Web Consor-

tium (W3C). http://www.w3.org/Submission/ws-addressing [En ligne]. Consulté le 30 novembre 2010.

Bray, T., J. Paoli, C. Sperberg-McQueen, E. Maler, et F. Yergeau. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium (W3C). http://www.w3.org/TR/REC-xml/ [En ligne]. Consulté le 30 novembre 2010.

Carbone, M., K. Honda, et N. Yoshida. 2007. « A Calculus of Global Interaction based on Session Types », *Electronic Notes in Theoretical Computer Science*, vol. 171, no. 3, p. 127–151. Proceedings of the Second International Workshop on Developments in Computational Models (DCM 2006).

Casati, F., S. Ceri, B. Pernici, et G. Pozzi. 1998. « Workflow Evolution », *Data & Knowledge Engineering*, vol. 24, no. 3, p. 211–238.

Casati, F., S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, et M.-C. Shan. 2000. « Adaptive and Dynamic Service Composition in eFlow ». In *Advanced Information Systems Engineering : 12th International Conference, CAiSE 2000, Stockholm. Proceedings.* T. 1789/2000, p. 13.

Charoy, F., A. Guabtni, et M. V. Faura. 2006. « A dynamic workflow management system for coordination of cooperative activities ». In Heidelberg, S. B. ., éditeur, *Business Process Management Workshops.* T. Volume 4103/2006, p. 205–216. Springer Berlin / Heidelberg.

Chen, Q. et M. Hsu. 2001. « Inter-enterprise collaborative business process management », *Data Engineering, International Conference on*, vol. 0, p. 0253.

Christensen, E., F. Curbera, G. Meredith, et S. Weerawarana. 2001. Web Services Description Language (WSDL) 1.1. World Wide Web Consortium (W3C). http://www.w3.org/TR/wsdl [En ligne]. Consulté le 30 novembre 2010.

Clark, J. et S. DeRose. 1999. XML Path Language (XPath) Version 1.0. World Wide Web Consortium (W3C). http://www.w3.org/TR/xpath/ [En ligne]. Consulté le 10 décembre 2010.

Coleman, J. 2005. « Examining BPEL's Compensation Construct », *Rigorous Engineering of Fault-Tolerant Systems (REFT 2005)*, p. 122.

Decker, G. 2009. « Design and analysis of process choreographies ». Thèse de Doctorat, Potsdam, Germany, Hasso Plattner Institute, University of Postsdam.

Decker, G., O. Kopp, F. Leymann, et M. Weske. 2007. « BPEL4Chor : Extending BPEL for modeling choreographies ». In *IEEE International Conference on Web Services. ICWS 2007*, p. 296–303.

Eclipse. 2011. Jetty. http://www.eclipse.org/jetty [En ligne]. Consulté le 24 février 2011.

Ellis, C. et K. Keddara. 2000. « *ML-DEWS* : Modeling language to support dynamic evolution within workflow systems », *Computer Supported Cooperative Work (CSCW)*, vol. 9, p. 293–333.

Ellis, C., K. Keddara, et G. Rozenberg. 1995. « Dynamic change within workflow systems ». In *COCS '95 : Proceedings of conference on Organizational computing systems*, p. 10–21, New York, NY, USA. ACM.

eviware. 2010. eviware, Products, soapUI. `http://www.eviware.com/soapUI/soapui-products-overview.html` [En ligne]. Consulté le 10 novembre 2010.

Fallside, D. C. et P. Walmsley. 2004. XML Schema Part 0 : Primer Second Edition. World Wide Web Consortium (W3C). `http://www.w3.org/TR/xmlschema-0/` [En ligne]. Consulté le 30 novembre 2010.

Goh, A., Y. K. Koh, et D. S. Domazet. 2001. « ECA rule-based support for workflows », *Artificial Intelligence in Engineering*, vol. 15, no. 1, p. 37–46.

Hammer, M. 1990. « Reengineering work : Don't automate, obliterate », *Harvard business review*, vol. 68, no. 4, p. 104–112.

Havey, M. 2006. Modeling Web Services Choreography with New Eclipse Tool : Dancing with BPMN and new Eclipse tool pi4soa. SOA World Magazine, `http://soa.sys-con.com/node/175396`. [En ligne]. Consulté le 8 septembre 2010.

Intalio. 2010a. Intalio | Community Website. `http://community.intalio.com` [En ligne]. Consulté le 30 novembre 2010.

———. 2010b. Intalio, The Private Cloud Company – Designer. `http://www.intalio.com/bpms/designer` [En ligne]. Consulté le 3 septembre 2010.

———. 2011. Intalio | BPMS. `http://www.intalio.com/bpms` [En ligne]. Consulté le 10 août 2011.

Joeris, G. et O. Herzog. 1998. « Managing evolving workflow specifications ». In *Cooperative Information Systems, 1998. Proceedings. 3rd IFCIS International Conference on*, p. 310–319.

Kammer, P. J., G. A. Bolcer, R. N. Taylor, A. S. Hitomi, et M. Bergman. 2000. « Techniques for supporting dynamic and adaptive workflow », *Computer Supported Cooperative Work (CSCW)*, vol. 9, p. 269–292.

Karastoyanova, D. et A. Buchmann. 2004. « Extending web service flow models to provide for adaptability ». In *Proceedings of OOPSLA '04, Workshop on Best Practices and Methodologies in Service-oriented Architectures : Paving the Way to Web-services Success, Vancouver, Canada, 24 October 2004*. ACM.

Kavantzas, N., D. Burdett, G. Ritzinger, T. Fletcher, et Y. Lafon. 2004. Web Services Choreography Description Language Version 1.0. World Wide Web Consor-

tium (W3C). http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/ [En ligne]. Consulté le 30 novembre 2010.

Kettinger, W. J. et V. Grover. 1995. « Special section : Toward a theory of business process change management. », *Journal of Management Information Systems*, vol. 12, no. 1, p. 9–30.

Khriss, I. 2005. « Automating the construction of business processes from UML business scenarios ». In *Proceedings of Montreal Conference on eTechnologies 2005 (MCeTech'05)*.

Khriss, I., E. Lévesque, G. Tremblay, et A. Jacques. 2008. « Towards adaptability support in collaborative business processes ». In *MCETECH'08 : Proceedings of the 2008 International MCETECH Conference on e-Technologies*, p. 34–45, Washington, DC, USA. IEEE Computer Society.

Kradolfer, M. et A. Geppert. 1999. « Dynamic workflow schema evolution based on workflow type versioning and workflow migration ». In *Cooperative Information Systems, 1999. CoopIS'99. Proceedings. 1999 IFCIS International Conference on*, p. 104–114.

Liu, C., Q. Li, et X. Zhao. 2009. « Challenges and opportunities in collaborative business process management : Overview of recent advances and introduction to the special issue », *Information Systems Frontiers*, vol. 11, p. 201–209.

Liu, C., M. E. Orlowska, et H. Li. 1998. *Automating Handover in Dynamic Workflow Environments*. Coll. Pernici, B. et C. Thanos, éditeurs, Coll. « *Advanced Information Systems Engineering* ». T. 1413, série *Lecture Notes in Computer Science*, p. 159–171. Springer Berlin / Heidelberg.

Mendling, J. et M. Hafner. 2005. « From inter-organizational workflows to process execution : Generating BPEL from WS-CDL ». In *On the Move to Meaningful Internet Systems 2005 : OTM Workshops*, p. 506–515. Springer.

Microsoft. 2011. Microsoft Biztalk Server. http://www.microsoft.com/biztalk [En ligne]. Consulté le 10 août 2011.

Milner, R., J. Parrow, et D. Walker. 1992. « A calculus of mobile processes, I », *Information and Computation*, vol. 100, no. 1, p. 1 – 40.

OMG. 2010a. Business process model and notation (BPMN). Object Management Group (OMG). http://bpmn.org [En ligne]. Consulté le 30 novembre 2010.

———. 2010b. OMG unified modeling language^{tm} (OMG UML). Object Management Group (OMG). http://www.omg.org/spec/UML [En ligne]. Consulté le 1er décembre 2010.

Oracle. 2011a. Java. http://www.java.com [En ligne]. Consulté le 24 février 2011.

——. 2011b. MySQL - The world's most popular open source database. http://www.mysql.com [En ligne]. Consulté le 24 février 2011.

Orriëns, B., J. Yang, et M. Papazoglou. 2003. *A Framework for Business Rule Driven Web Service Composition*. Coll. « Conceptual Modeling for Novel Application Domains ». T. 2814, série *Lecture Notes in Computer Science*, p. 52–64. Springer Berlin / Heidelberg.

Ouyang, C., W. M. Van Der Aalst, M. Dumas, et A. H. Ter Hofstede. 2006. « Translating BPMN to BPEL », *BPM Center Report*, vol. BPM-06-02.

Peltz, C. 2003. « Web services orchestration and choreography », *Computer*, vol. 36, p. 46–52.

Pi4 Technologies Foundation. 2011. Welcome to the Pi4 Technologies Foundation. Sourceforge, http://pi4soa.sourceforge.net/. [En ligne]. Consulté le 10 mars 2011.

Qiu, Z. et Y. Wong. 2007. « Dynamic workflow change in PDM systems », *Computers in Industry*, vol. 58, no. 5, p. 453–463.

Reichert, M. et P. Dadam. 1998. « Adeptflex - Supporting Dynamic Changes of Workflows Without Losing Control », *Journal of Intelligent Information Systems*, vol. Volume 10, Number 2 / March, 1998, p. 93–129.

Rinderle, S., M. Reichert, et P. Dadam. 2004. « Correctness criteria for dynamic changes in workflow systems : a survey », *Data Knowl. Eng.*, vol. 50, no. 1, p. 9–34.

Sadiq, S., W. Sadiq, et M. Orlowska. 2001. « Pockets of flexibility in workflow specification ». In Heidelberg, S. B. ., éditeur, *Conceptual Modeling - ER 2001 : 20th International Conference on Conceptual Modeling*. T. 2224/2001. Springer Berlin / Heidelberg.

Sadiq, S. W. 2000. « Handling dynamic schema change in process models ». In *Database Conference, 2000. ADC 2000. Proceedings. 11th Australasian*, p. 120–126.

Sadiq, S. W., M. E. Orlowska, et W. Sadiq. 2005. « Specification and validation of process constraints for flexible workflows », *Information Systems*, vol. 30, no. 5, p. 349–378.

The Apache Software Foundation. 2010. Apache ODE. http://ode.apache.org [En ligne]. Consulté le 25 août 2010.

van der Aalst, W. 2001. « Exterminating the dynamic change bug : A concrete approach to support workflow change », *Information Systems Frontiers*, vol. 3, p. 297–317.

van der Aalst, W. M. P., T. Basten, H. M. W. Verbeek, P. A. C. Verkoulen, et M. Voorhoeve. 1999. « Adaptive workflow : On the interplay between flexibility and support », *Interprise Information Systems*, p. 61–68.

van der Aalst, W. M. P. et S. Jablonski. 2000. « Dealing with workflow change : Identi-

fication of issues an solutions », *Int'l Journal of Computer Systems, Science and Engineering*, vol. 15, no. 5, p. 267–276.

van der Aalst, W. M. P., A. H. M. ter Hofstede, et B. Kiepuszewski. 2003. « Workflow Patterns », *Distributed and Parallel Databases*, vol. 14, no. 1, p. 5–51.

Weber, B., M. Reichert, et S. Rinderle-Ma. 2008. « Change patterns and change support features — Enhancing flexibility in process-aware information systems », *Data Knowledge Engineering*, vol. 66, no. 3, p. 438 – 466.

Weske, M. 2007. *Business Process Management : Concepts, Languages, Architectures.* Springer-Verlag New York Inc.

White, S. A. 2006. Introduction to BPMN (Tutorial). Object Management Group (OMG). `http://www.bpmn.org/Documents/OMG_BPMN_Tutorial.pdf` [En ligne]. Consulté le 6 décembre 2010.

Wodtke, D. et G. Weikum. 1997. *A formal foundation for distributed workflow execution based on state charts.* Coll. Afrati, F. et P. Kolaitis, éditeurs, Coll. « *Database Theory - ICDT'97* ». T. 1186, série *Lecture Notes in Computer Science*, p. 230–246. Springer Berlin / Heidelberg.

Zaha, J., A. Barros, M. Dumas, et A. ter Hofstede. 2006a. *Let's Dance : A Language for Service Behavior Modeling.* Coll. Meersman, R. et Z. Tari, éditeurs, Coll. « *On the Move to Meaningful Internet Systems 2006 : CoopIS, DOA, GADA, and ODBASE* ». T. 4275, série *Lecture Notes in Computer Science*, p. 145–162. Springer Berlin / Heidelberg.

Zaha, J. M., M. Dumas, A. ter Hofstede, A. Barros, et G. Decker. 2006b. « Service Interaction Modeling : Bridging Global and Local Views », *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, p. 45–55.

Zhao, X. et C. Liu. 2006. *Tracking over Collaborative Business Processes.* Coll. Dustdar, S., J. Fiadeiro, et A. Sheth, éditeurs, Coll. « *Business Process Management* ». T. 4102, série *Lecture Notes in Computer Science*, p. 33–48. Springer Berlin / Heidelberg.

———. 2007. « Version management in the business process change context ». In *Proceedings of the 5th international conference on Business process management.* Coll. « BPM'07 », p. 198–213, Berlin, Heidelberg. Springer-Verlag.