UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Gouvernance et supervision décentralisée des chorégraphies inter-organisationnelles

THÈSE

présentée et soutenue publiquement le 27 juin 2013

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Aymen BAOUAB

Composition du jury

Rapporteurs :	Daniela GRIGORI, Professeur à l'Université de Paris Dauphine, LAMSADE, Paris Ladjel BELLATRECHE, Professeur à lSAE-ENSMA, Poitiers
Examinateurs :	Pascal POIZAT, Professeur à l'Université de Paris Ouest Nanterre, LIP6, Paris Christophe CERISARA, Chargé de recherche CNRS (HDR), LORIA, Nancy
Directeurs de thèse :	Claude GODART, Professeur à l'Université de Lorraine, LORIA, Nancy Olivier PERRIN, Professeur à l'Université de Lorraine, LORIA, Nancy

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

Mis en page avec la classe thloria.

Résumé

Durant la dernière décennie, les architectures orientées services (SOA) d'une part et la gestion des processus business (BPM) d'autre part ont beaucoup évolué et semblent maintenant en train de converger vers un but commun qui est de permettre à des organisations complètement hétérogènes de partager de manière flexible leurs ressources dans le but d'atteindre des objectifs communs, et ce, à travers des schémas de collaboration avancée. Ces derniers permettent de spécifier l'interconnexion des processus métier de différentes organisations. La nature dynamique et la complexité de ces processus posent des défis majeurs quant à leur bonne exécution. Certes, les langages de description de chorégraphie aident à réduire cette complexité en fournissant des moyens pour décrire des systèmes complexes à un niveau abstrait. Toutefois, rien ne garantit que des situations erronées ne se produisent pas suite, par exemple, à des interactions "mal" spécifiées ou encore des comportements malhonnêtes d'un des partenaires.

Dans ce manuscrit, nous proposons une approche décentralisée qui permet la supervision de chorégraphies au moment de leur exécution et la détection instantanée de violations de séquences d'interaction. Nous définissons un modèle de propagation hiérarchique pour l'échange de notifications externes entre les partenaires. Notre approche permet une génération optimisée de requêtes de supervision dans un environnement événementiel, et ce, d'une façon automatique et à partir de tout modèle de chorégraphie.

Mots-clés: Chorégraphie de services, processus inter-organisationnel, processus métier, BPM, service web, supervision, décentralisation, CEP.

Abstract

Cross-organizational service-based processes are increasingly adopted by different companies when they can not achieve goals on their own. The dynamic nature of these processes poses various challenges to their successful execution. In order to guarantee that all involved partners are informed about errors that may happen in the collaboration, it is necessary to monitor the execution process by continuously observing and checking message exchanges during runtime. This allows a global process tracking and evaluation of process metrics. Complex event processing can address this concern by analyzing and evaluating message exchange events, to the aim of checking if the actual behavior of the interacting entities effectively adheres to the modeled business constraints.

In this thesis, we present an approach for decentralized monitoring of cross-organizational choreographies. We define a hierarchical propagation model for exchanging external notifications between the collaborating parties. We also propose a runtime event-based approach to deal with the problem of monitoring conformance of interaction sequences. Our approach allows for an automatic and optimized generation of rules. After parsing the choreography graph into a hierarchy of canonical blocks, tagging each event by its block ascendancy, an optimized set of monitoring queries is generated. We evaluate the concepts based on a scenario showing how much the number of queries can be significantly reduced.

Keywords: Choreography, Monitoring, Web service, Business Process Management (BPM), decentralization, Complex Event Processing (CEP).

Remerciements

Je voudrais exprimer mes sentiments les plus sincères envers les personnes qui sans lesquelles ce travail de thèse n'aurait pas pu voir le jour. Leur aide, accompagnement et soutien m'ont été indispensables afin de pouvoir aboutir aux contributions de ma thèse.

Je voudrais tout d'abord exprimer ma reconnaissance envers mes directeurs de thèse : le Professeur Claude Godart et le Professeur Olivier Perrin. Merci de votre aide, de votre disponibilité et de vos encouragements tout au long de cette thèse. Je vous remercie de m'avoir toujours laissé une grande liberté dans mon travail, me permettant ainsi d'acquérir l'autonomie nécessaire à tout travail de recherche.

Je voudrais aussi exprimer ma reconnaissance envers tous les membres du jury pour la grande attention qu'ils ont bien voulu porter à mon travail. Je remercie très sincèrement mes rapporteurs Madame Daniela Grigori et Monsieur Ladjel Bellatreche pour avoir bien accepté d'être mes rapporteurs et pour avoir bien voulu lire et évaluer mon travail de thèse. Je remercie également Monsieur Pascal Poizat et Monsieur Christophe Cerisara pour leur participation au jury de cette thèse et le temps qu'ils ont bien voulu consacrer à l'évaluation de mon travail.

Je remercie très sincèrement Khalid, François, Samir, Gérarld, Claudia et tous les membres de l'équipe SCORE pour leur soutient moral et pour leurs conseils.

Mes plus amicaux remerciements vont aussi à mes collègues du LORIA-INRIA Nancy, notamment Walid, Mehdi, Bilel, Oussema, Ehteshem, Ghazi, Said, Mohammed, Karim, Wahiba, Nabil, Nihel, Najet, Amani, Moutie, Codé, Dorin, Nicole, Khaled et Luca qui ont permis de faire de cette expérience de thèse une expérience riche tant scientifiquement qu'humainement. Merci pour les déjeuners animés, et toutes sortes d'activités. Je tiens à remercier aussi toutes les personnes répondent toujours présentes quand nous avons besoin d'elles.

Je tiens également à adresser mes remerciements les plus sincères à Taoufik, Adnène, Isabelle, Betty, Caroline, Tarek, Jamel, Ines, Reine, Chaza, Ahlem, Maroua, Hatem et Oussema. Merci pour votre soutient en ces moments difficiles.

Un grand merci à toute ma famille qui m'a toujours encouragé. Merci à mes parents et ma femme, qui ont toujours cru en moi, pour leur confiance, leur fierté et leur amour. Merci à ma soeur Ines et à mon frère Ilyes pour leur soutien toujours en douceur.

Je garderai toujours les souvenirs, des premiers jours de ma thèse, où je ne comprenais pas du tout de quoi il pouvait être question mais sentais grandir avec violence l'envie de comprendre.

à ma mère Leila, à mon père Amor,
à ma femme chérie Balkiss,
à mon frère Ilyes et ma sœur Ines.

Table des matières

II Contributions

III Expérimentations, bilan et perspectives

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction

Les nouvelles technologies de l'information et de la communication ont fait l'objet d'investissements considérables ces dernières années. Certaines de ces innovations technologiques ont prouvé leur pertinence en devenant de véritables catalyseurs de la croissance des entreprises. Un des piliers de ces innovations est l'utilisation des processus métiers qui est devenue indispensable pour mieux organiser et gérer les différentes tâches d'une entreprise. La gestion de ces processus métier inclut les concepts, les méthodes et les techniques nécessaires pour la conception, l'administration, la configuration, l'exécution et l'analyse de ces processus [Wes07].

Les besoins des entreprises sont sujets à des changements assez fréquents. L'agilité et la flexibilité deviennent donc des critères essentiels permettant de gagner en compétitivité en s'adaptant le plus rapidement possible aux changements du marché. Un second pilier concerne les architectures orientées service (SOA) «*Service Oriented Architecture*», considérées comme le nouveau paradigme des systèmes d'information. Certaines technologies, plus particulièrement les *services Web*, sont devenues de facto le standard pour permettre l'exécution des applications collaboratives les plus récentes. Outre le découpage des applications en *services* individuels, la standardisation des choix technologiques ou la virtualisation de l'infrastructure, une architecture SOA permet de répondre aisément aux nécessités de changement de l'entreprise afin que celle-ci perçoive les technologies de l'information comme un avantage compétitif sur lequel s'appuyer pour s'adapter au marché [Dav08]. La technologie des services Web est une solution intéressante et élaborée pour l'implantation d'une architecture SOA. Un service Web, étant un composant logiciel auto-descriptif et ouvert, est conçu pour supporter les interactions entre différentes applications, distribuées sur différentes plateformes. Des fournisseurs de services, c'est-à-dire des organisations qui procurent l'implémentation des services, fournissent leurs descriptions et le support technique et commercial relatif.

Alors que l'architecture SOA résout les problèmes de réutilisabilité et d'élimination des données dupliquées dans les infrastructures du système informatique, la gestion des processus métiers BPM «*Business Process Management*» permet aux entreprises de mieux appréhender leurs processus métiers. L'association de ces deux concept améliore significativement le système d'information et la gestion des processus métiers, au service de leur croissance. Les systèmes de gestion de *workflow* actuels sont utilisés pour automatiser la coordination entre tous ces éléments et pour améliorer l'efficacité des collaborations et la gestion des activités [MAM+95]. Le BPM en mode SOA permet d'étendre la gestion informatique de l'entreprise vers les systèmes d'information de ses parties prenantes (e.g. fournisseurs, clients, partenaires techniques ou financiers) et d'enchaî-

ner automatiquement des étapes effectuées par un ou plusieurs fournisseurs de services [Sys08].

Un des concepts intéressants qu'offre le BPM en mode SOA, et qui suscite beaucoup d'intérêt, est la possibilité de créer un nouveau service à valeur ajoutée par composition de services existants, éventuellement offerts par plusieurs entreprises [BSD03]. De nombreux langages et standards existent pour décrire et spécifier de tels services et processus, et ce à plusieurs niveaux : par exemple, l'orchestration décrivant le fonctionnement interne d'un processus, l'interface comportementale d'un processus, et la chorégraphie régissant la coopération entre divers processus et services. L'association entre les standards des services Web et les processus métiers reflète l'intérêt de l'industrie et son orientation vers des applications distribuées qui communiquent avec des services fournis par des fournisseurs ou partenaires.

Aujourd'hui, l'informatique collaborative prend de l'importance et les processus interagissent les uns avec les autres formant des chaînes complexes (e.g. chaîne de sous-traitance, chaîne d'approvisionnement, partenariat) en vue d'accroître leur efficacité. En effet, les processus métier des organisations s'étendent à travers Internet et doivent faire face à des opérations complexes qui peuvent prendre des jours, voire des semaines, au delà des frontières de l'entreprise, à travers plusieurs fuseaux horaires, et sur un périmètre géographique beaucoup plus vaste.

Afin de collaborer efficacement, les organisations doivent coordonner leurs activités [SS09]. Les processus inter-organisationnels requièrent une considération particulière de plusieurs aspects. En effet, la nature dynamique et la complexité des processus métier inter-organisationnels posent des défis majeurs quant à leur bonne exécution (e.g. le passage à l'échelle, l'hétérogénéité, la disponibilité et la sécurité). Certes, les langages de description de chorégraphie (CDL) aident à réduire cette complexité en fournissant des moyens pour décrire des systèmes complexes à un niveau supérieur. Toutefois, rien ne garantit que des situations erronées ne se produisent pas suite, par exemple, à des interactions "mal" spécifiées ou encore des comportements malhonnêtes d'un des partenaires. Par conséquent, une surveillance minutieuse de l'exécution des chorégraphies inter-organisationnelles s'avère être nécessaire.

Durant la dernière décennie, la supervision des compositions de services a fait l'objet de plusieurs travaux de recherche [AFG+08, FGP10, MRD11a, CCMN04, HV09]. Néanmoins, les approches proposées sont limitées aux compositions regroupant des services d'une même zone de confiance et ne peuvent pas être adaptées aux chaînes de production et d'approvisionnement qui s'étendent à travers de nombreuses organisations administrativement indépendantes. Ainsi, il est nécessaire de trouver un moyen efficace pour permettre une propagation instantanée des exceptions internes au delà des frontières organisationnelles.

Le principal objectif de cette thèse est de permettre le suivi de processus collaboratifs et de garantir que tous les partenaires concernés s'informent des erreurs qui se produisent, et ce, dès qu'elles se sont produites. Dans ce sens, nous proposons une approche de supervision dans laquelle les échanges inter-organisationels de messages sont perçus comme des évènements. Ces évènements sont traités par des unités de supervision implantées sous forme de services déployés le long des frontières de chaque organisation et invoqués à chaque détection d'un nouveau message

échangé avec une organisation partenaire. Après la vérification de chaque message par rapport au schéma de la chorégraphie, une notification est automatiquement générée, envoyée et propagée à un sous ensemble pré-calculé de partenaires.

Notre première contribution est de définir un modèle architectural [BPG11, BPBG09] qui va permettre le déploiement de différents composants au niveau de chaque organisation. Ces composants ont la tâche d'intercepter et d'analyser tous les messages de chorégraphies échangés avec le monde extérieur, et ce, d'une manière non intrusive (i.e. sans rien modifier dans les processus et les messages échangés). La deuxième contribution consiste à proposer un mécanisme d'échange de notifications entre les différents composants [BFPG12]. La troisième consiste à trouver comment générer, d'une façon automatique à partir d'un modèle de chorégraphie, un ensemble optimal de requêtes de supervision directement exécutables dans un environnement CEP «*Complex Event Processing*» [BPG12]. Enfin, nous généralisons notre approche de supervision décentralisée afin de traiter un plus large spectre de modèles de chorégraphies [BPG13].

1 Organisation du mémoire

Le présent mémoire est divisé en trois grandes parties. La première partie définit la problématique de notre travail et décrit brièvement nos contributions. Elle propose aussi, un survol sur les notions de base nécessaires à la compréhension de ce mémoire ainsi qu'une étude bibliographique des travaux traitant cette problématique. Elle souligne aussi les limites de ces travaux et présente les apports de notre travail par rapport à ces derniers. La deuxième partie constitue le coeur de notre travail et concerne les solutions que nous avons proposées pour résoudre la problématique posée. La troisième partie est dédiée aux expérimentations, aux bilan et aux perspectives. L'organisation détaillée du manuscrit suit le schéma suivant.

- Dans le **chapitre 1**, nous présentons la problématique de notre travail : la supervision centralisée dans le contexte des processus métier inter-organisationnels, est-elle toujours capable de gérer l'évolution continue des entreprises, la complexité de leurs processus, l'externalisation des opérations et les échanges de messages avec les partenaires ? est-elle optimale en termes de coût de communication et qualité de service ? est-elle bien adaptée aux propriétés liés à l'intimité (*privacy*) et la séparation des devoirs ? Nous illustrons notre problématique via un exemple et nous synthétisons les limites des technologies actuelles pour répondre à ce problème. Enfin, nous présentons les principes directeurs de notre travail et les contributions de notre thèse.

- Dans le **chapitre 2**, nous survolons les notions et les définitions des éléments de base nécessaires à la compréhension de ce mémoire. Des concepts comme *SOA, processus métier, BPM* et *EDA* [1] représentent les éléments clés des nouvelles architectures d'entreprises, et sont en relation directe avec notre problématique. Nous présentons également un état de l'art sur les différents standards pour la composition des services web. Dans ce contexte, nous montrons comment une plate-forme logicielle BPM permet d'offrir aux entreprises la

1. Event Driven Architecture

possibilité de modéliser, gérer et optimiser des processus métier pour un gain significatif. Enfin, nous survolons les principaux concepts liés aux architectures orientées événements (EDA) et nous montrons comment elles peuvent compléter SOA et BPM en offrant un couplage très lâche entre les services. Nous montrons aussi l'importance des processeurs CEP et comment ils peuvent détecter des patrons/motifs d'événements complexes diffusés massivement dans un système afin d'évaluer avec précision son état.

– Dans le **chapitre 3**, nous présentons un état de l'art sur les principaux travaux qui portent sur la modélisation, l'analyse, la sécurité et le suivi des compositions des services web. Nous nous intéressons plus particulièrement aux chorégraphies inter-organisationnelles et leur modélisation en BPMN 2.0. Nous discutons des différentes approches qui traitent le sujet de la supervision des processus métiers inter-organisationnels et la vérification de la conformité des chorégraphies durant la phase d'exécution. Nous identifions les problèmes posés par la mise en œuvre de ces approches et l'apport de notre méthodologie par rapport à ces travaux.

– Dans le **chapitre 4**, nous explorons les chorégraphies inter-organisationnelles d'une manière indépendante des langages de spécification et nous offrons un nouveau modèle conceptuel en proposant une architecture générale pour la supervision des collaborations. Nous introduisons les notions de «vue locale» et «vue globale», notamment en étudiant un exemple de modèle de chorégraphie. Ensuite, nous proposons un modèle formel général, simple et compréhensible qui permet la mise en œuvre de notre approche à partir de tout type de langage de spécification. Enfin, nous décrivons l'architecture générale de notre contribution.

– Dans le **chapitre 5**, nous proposons une approche de suivi décentralisé de l'exécution d'une chorégraphie dans laquelle les échanges inter-organisationels de messages sont perçus comme des évènements. Nous montrons l'intérêt et l'applicabilité de notre approche, notamment en étudiant le cas des chorégraphies de chaînes d'approvisionnement inter-entreprises. Nous présentons l'approche de classification hiérarchique des participants et nous définissons les notions de «*super/sous partenaires*» ainsi que de vue de supervision «*EFM-view*». Ensuite, nous décrivons les algorithmes de configuration et d'exécution. Finalement, nous généralisons notre approche à tout type de chorégraphie inter-organisationnelle.

– Le **chapitre 6** est dédié à l'analyse et l'évaluation des événements générés par les échanges de messages entre les partenaires. Le but est de vérifier si le comportement réel des entités en interaction adhère efficacement aux contraintes métier imposées. Pour y parvenir, nous proposons une approche évènementielle permettant la génération automatique d'un ensemble optimisé de requêtes à partir d'un schéma d'interaction défini. Cette approche permet de vérifier ensuite la conformité des séquences d'interaction par rapport au modèle de la chorégraphie, et ce, à chaque échange de message avec ses partenaires. Nous commençons par donner un aperçu sur l'organisation des composants et les différentes étapes de l'approche. Ensuite, nous présentons le mécanisme de fragmentation structurelle ainsi que le mécanisme de génération de règles. Une évaluation de notre approche en terme de

nombre de règles générées est également présentée dans ce chapitre. Enfin, nous montrons comment détecter les violations avec les anti-patrons, et nous proposons un mécanisme d'agrégation de violations.

– Le **chapitre 7** présente les aspects et les choix technologiques liés à l'implémentation de notre approche de supervision. Le but derrière cette implémentation est de tester l'efficacité de notre approche et d'évaluer la surcharge qu'elle génère afin d'étudier, ensuite, son passage à l'échelle. Nous nous basons sur une simulation de génération massive et aléatoire de séquences d'échanges de messages et nous observons et analysons les résultats par la suite. Nous donnons une évaluation globale de l'approche et nous exposons le gain apporté par notre méthode de génération de requêtes.

– Le **chapitre 8** résume nos contributions et dégage les perspectives directes de nos travaux de recherche. Des annexes et une bibliographie guident le lecteur tout au long de sa lecture.

Première partie

Problématique et état de l'art

1 Problématique et contributions

«Etre conscient de son ignorance, c'est tendre vers la connaissance.»

Benjamin Disraeli

Sommaire

Dans ce chapitre, nous exposons la problématique que nous allons traiter dans cette thèse, en mettant l'accent sur l'apport des processus inter-organisationnels décentralisés et les inconvénients de la supervision centralisée. Nous illustrons notre problématique par un exemple qui met en évidence les aspects à traiter lors de la supervision et le suivi au cours d'exécution des processus métiers inter-organisationnels. Enfin, nous présentons brièvement nos contributions concernant la génération automatique des requêtes de supervision et l'échange de notifications entre les participants d'une chorégraphie de services. Dans ce manuscrit, nous utilisons les termes processus métier et composition de services pour dire la même chose, mais nous expliquons les différences dans le chapitre 2.

1.1 Problématique

La capacité d'interconnecter les processus métier inter-organisationnels reçoit une attention accrue dans une économie de plus en plus connectée [Gre06]. Dans le cadre des collaborations B2B [2], l'utilisation de l'échange de données informatisées (EDI [3]) pour les relations commerciales entre les entreprises ne répond plus aux exigences. En effet, l'EDI permet principalement d'établir des relations bilatérales à travers la mise en œuvre de liens spécifiques, et ce, sans supporter les

2. Business to business
3. Electronic Data Interchange

collaborations dynamiques et les interactions entre plusieurs processus. Cependant, de nombreux projets inter-entreprises ont besoin d'un support pour ce type collaboration.

Par opposition aux processus centralisés intra-organisationnels (i.e. au sein d'une même organisation), la configuration décentralisée de processus qui sont déployés à travers les frontières organisationnelles pose de nouvelles exigences en matière de contrôle. Sans coordinateur central, qui risque d'anéantir les performances dans certains scénarios (i.e. goulot d'étranglement), les organisations collaboratrices doivent être en mesure de mettre en place un processus métier inter-organisationnel, et ce, en divulguant les unes aux autres uniquement le strict nécessaire. La vérification de l'exactitude de tels processus ne doit pas forcer chaque partenaire à révéler sa logique métier interne. En effet, la capacité de maintenir secrète cette logique demeure très importante pour maintenir un avantage concurrentiel.

Construire des processus distribués complexes, sans introduire des conséquences inattendues, représente un véritable défi. Les langages de description de chorégraphie (CDL) aident à réduire la complexité de cette tâche en fournissant des moyens pour décrire des processus complexes à un niveau d'abstraction supérieur. La naissance de la notion de chorégraphie de service est souvent déterminée en mettant sur pied les normes externes, les règlements, les politiques, les pratiques et les objectifs métier de chaque organisation partenaire. Toutes ces exigences ont pour effet de restreindre les interactions permises entre les participants à une chorégraphie. Toutefois, cela ne garantit pas nécessairement que les situations erronées ne peuvent pas se produire suite, par exemple, à des interactions "mal" spécifiées ou encore des comportements malhonnêtes d'un des partenaires. En effet, la supervision de chaque exécution doit être prise en considération, et ce, dans le but de vérifier si le comportement réel (en phase d'exécution) des entités en interaction adhère efficacement aux contraintes métier modélisées (en phase conception).

1.1.1 Les processus métier inter-organisationnels : sous-traitance et externalisation

Les processus métier inter-organisationnels impliquent souvent un grand nombre de ressources et d'outils distribués géographiquement. Avec la mondialisation de l'économie, la sous-traitance devient internationale. En effet, la croissance continuelle des entreprises incite de plus en plus à l'externalisation de certaines activités. Par exemple, un constructeur automobile remet à son fournisseur de peintures un service pour le calcul des approvisionnements qui tient compte en temps réel des plannings de production de véhicules. Ce service peut être intégré au sein du SI du fournisseur sous la forme d'un service Web. Il agit sur la production de peintures et la fréquence de livraison. Le calcul du réapprovisionnement devient ainsi une fonction support sous-traitée chez le fournisseur.

> *"La sous-traitance est un contrat par lequel une entreprise, le « donneur d'ordre », demande à une autre entreprise, le « sous-traitant », de réaliser une partie de sa production ou des composants nécessaires à sa production. Les entreprises sous-traitantes sont des entreprises auxquelles sont agréées certaines parties de travail."* [Kno10]

En général, l'externalisation diffère de la simple prestation extérieure de services, et de la simple sous-traitance, dans la mesure où il y a pilotage étroit par l'entreprise donneuse d'ordre et engagement du prestataire externe [Phi07].

"L'externalisation désigne le transfert de tout ou partie d'une fonction d'une organisation (entreprise ou administration) vers un partenaire externe. Elle consiste très souvent en la sous-traitance des activités jugées non-essentielles et non stratégiques." [Mei06]

Pour rationaliser leurs activités, les entreprises abandonnent la gestion interne des fonctions supports pour ne se concentrer que sur leur cœur de métier. Elles doivent donc sous-traiter et obtenir de leurs fournisseurs des contrats de services qui garantissent la meilleure fiabilité d'exécution [Bon05]. L'automatisation des échanges avec le fournisseur est un facteur clef de cette fiabilité. Les risques d'erreur diminuent en fonction de la progression du niveau d'intégration entre les participants. Certes, les compositions de services inter-entreprises semble être la solution pour cette intégration. Cependant, cela ne garantit pas l'absence des exceptions au niveau des applications ou même de l'infrastructure de l'un des partenaires.

D'un autre coté, parmi les inconvénients majeurs de la sous-traitance, nous pouvons citer la perte de maîtrise, le manque de flexibilité de la part du prestataire, le risque de délais trop longs et le manque d'information et de transparence. En plus, le coût peut être parfois plus élevé que la production interne. Afin de pallier ces problèmes, les entreprises doivent adopter une approche de supervision inter-organisationnelle qui va permettre le suivi de l'exécution des fragments de processus externalisés, et ce, de façon abstraite sans pour autant que chaque entreprise dévoile sa logique métier à ses partenaires. En d'autres termes, il y a besoin de propager des données de supervision entre partenaires pour permettre une gestion décentralisée des informations recueillies, et ce, dans le but d'offrir plus de maîtrise au niveau de chaque organisation, et de réduire les délais et les coûts en cas d'occurrence d'exceptions.

1.1.2 La supervision des processus métier

La supervision temps réel de composition de services est le centre d'intérêt de plusieurs travaux de recherche [AFG+07, FGP10, MRD11b, BFPG12, BPG12, CCMN04, HV09]. Les processus métier des entreprises d'aujourd'hui utilisent principalement le traitement des événements complexes (CEP) pour suivre l'exécution d'un processus et signaler tout état incohérent de ses instances individuelles. Au cours des dix dernières années, le domaine CEP a été activement appliqué dans le milieu professionnel. Le BAM «*Business Activity Monitoring*», un concept qui s'intéresse à la supervision des processus et des activités métiers, a été l'une des applications les plus prospères où le CEP a été utilisé. Ce concept propose une exploitation rationnelle des instruments et équipements de l'informatique décisionnelle BI «*Business Intelligence*» afin d'assurer le pilotage des activités et des processus métier les plus critiques. Le BAM cherche ainsi à proposer la meilleure perception de la performance des activités et processus. Du point de vue de l'administrateur, responsable d'activités ou de processus, la solution BAM se concrétise en tableaux de bord de management composés d'indicateurs de performance KPI «*Key Performance*

Indicators», mis à jour de façon instantanée.

S'appuyant sur les KPI, la technologie BAM permet la supervision des processus métier en continu en se basant sur la génération et le traitement des événements de contrôle. La plupart des produits logiciels commerciaux de BPM (e.g. Oracle BAM, Nimbus, Tibco, IBM Tivoli, etc.) comprennent des installations de tableau de bord BAM permettant le suivi, les rapports sur les violations des accords de niveau de service (SLA [4]), et l'affichage des résultats et des métriques sous forme de graphiques. Cependant, vu que les vendeurs de ces applications commerciales sont des concurrents, ces produits ne permettent de surveiller qu'une fraction prédéfinie des événements et des indicateurs de performance, généralement au sein d'une suite d'applications d'un même fournisseur. En outre, ils sont généralement limités à des processus internes qui sont sous contrôle d'une même entité administrative, c'est-à-dire, ne supportent pas des environnements inter-organisationnels distribués et hétérogènes.

Fournir un mécanisme simple permettant la supervision en temps réel des processus inter-organisationnels, là où chaque étape est exécutée par une entreprise différente dans un réseau de collaboration, représente une tâche compliquée. Cela est dû au fait que l'outil de supervision doit faire face à d'énormes volumes de données non structurées provenant de sources différentes. De plus, des erreurs et/ou des échecs d'exécutions peuvent se propager en cascade à travers tous les partenaires. En gérant les agrégations de plusieurs alertes de violations, la technologie CEP peut donner aux administrateurs d'entreprises une meilleure visibilité, et leur fournir des informations plus précises sur l'état actuel des processus contrôlés. Ceci pourra les aider à répondre automatiquement et beaucoup plus rapidement en cas de problème.

Dans ce manuscrit, nous considérons que les organisations qui collaborent ne révèlent qu'une partie de leur comportement et de leurs processus. En effet, la logique métier dite «interne» (i.e. les activités et les processus locaux au sein d'une organisation) reste invisible aux partenaires. Ainsi, le processus global peut être considéré comme une chorégraphie, qui régit l'ordre et la structure des messages échangés afin d'aboutir à un comportement collaboratif coordonné entre deux ou plusieurs participants en interaction [KBR+05].

Une chorégraphie peut parfois échouer en raison d'une circonstance exceptionnelle ou d'une erreur qui s'est produite au sein d'un des services qui la composent. Le nombre d'exceptions peut augmenter si la coordination n'est pas bien encadrée. Pour s'assurer de l'intégrité du processus d'exécution, les participants se mettent généralement d'accord sur un coordinateur centralisé appartenant à une même zone de confiance et permettant de gérer l'exécution des primitives non fonctionnelles telles que le routage des messages et les fonctions de sécurité. Cette solution est généralement non applicable, particulièrement lorsqu'il est impossible de trouver une unité de confiance commune à tous les participants.

En absence de tout coordinateur central, il faut trouver comment décentraliser ces primitives sur l'ensemble des participants. Ainsi, des vérifications supplémentaires (e.g. vérification de la structure des messages échangés et de la conformité de la séquence des interactions) par rapport au schéma de processus prédéfini doivent être ajoutées au sein de chaque organisation

4. Service Level Agreement

participante.

1.1.3 Exemple de motivation

Pour illustrer notre problématique, nous avons choisi un simple exemple d'achat et livraison de produits qui met bien en évidence tous les problèmes. L'exemple est présenté dans le schéma de la *figure* 1.1. La figure montre une chorégraphie entre quatre participants : un revendeur, un fournisseur et deux expéditeurs (les autres participants impliqués dans ce scénario tels que les banques ont été omis pour des raisons de simplicité). D'abord, le revendeur envoie une demande de bon de commande avec des détails sur les produits requis et les montants nécessaires au fournisseur (interaction 1). Ensuite, le fournisseur vérifie la demande et avise le revendeur dans le cas où il n'y a pas de produits disponibles (interaction 2c). Dans le cas contraire, le fournisseur traite la commande et sélectionne un des deux expéditeurs pour livrer les produits sélectionnés (interaction 2a ou 2b). Enfin, l'expéditeur choisi termine le processus en envoyant les produits au revendeur (interactions 3a ou 3b).

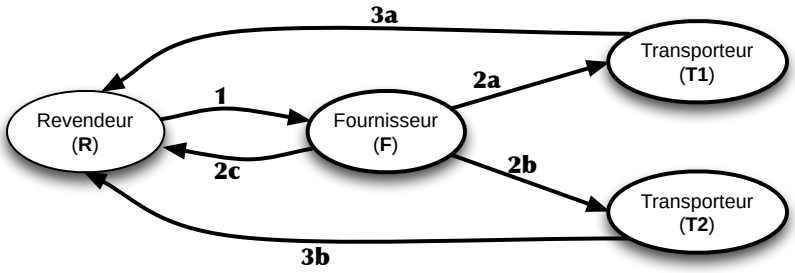


FIGURE 1.1 – Scénario d'échange de messages

Sans avoir à se référer à un coordinateur central de confiance, chaque participant doit être en mesure de vérifier, à tout moment de l'exécution de la chorégraphie, que tous les messages entrants et sortants sont conformes à ceux attendus selon le schéma d'échange prédéfini. Par exemple, le fournisseur a le choix entre trois appels de service (2a , 2b ou 2c) en fonction de sa logique interne qui n'est pas visible pour les autres participants. En contre partie, le revendeur recevra à la fin un et un seul des messages (2c , 3a ou 3b). Si ce n'est pas le cas, le revendeur doit détecter un comportement non conforme à ce qui a été prédéfini et éventuellement informer les autres participants concernés. En effet, dans une telle situation, le revendeur peut être le seul participant qui est au courant de ce qui s'est passé et donc le seul capable de notifier les autres, ce qui est de premier intérêt en l'absence d'un coordinateur central.

Au cours de la phase d'exécution, de nombreuses chorégraphies partagées entre différents participants peuvent être instanciées. De plus, une même chorégraphie peut être instanciée plusieurs fois. Ainsi, chaque organisation peut avoir plusieurs interactions externes associées aux différentes instances de chorégraphies. De toute évidence, il est nécessaire de vérifier la cohérence de tous les messages entrants et sortants par rapport à l'étape actuelle dans chacune des instances de chorégraphies auxquelles l'organisation est en train de participer. En effet, l'accès aux ressources doit

être limité suivant le principe du moindre privilège (*principle of least privilege* [SS75]). Lorsque les données sont transmises entre les participants, il doit y avoir des restrictions d'accès sur le flux entrant et sortant en fonction de l'étape en cours. Ainsi, l'ensemble des règles d'accès doit être dynamique et minimal, et ce, à n'importe quel moment durant l'exécution. Ceci dit, un message qui ne peut pas être associé à une des interactions attendues de l'étape actuelle de l'une des chorégraphies instanciées doit être refusé et/ou signalé aux applications de surveillance comme une violation potentielle. Le contrôle des invocations entrantes au niveau de chaque organisation participante, dans un stade précoce, vise à réduire les attaques courantes (e.g. des attaques type DoS [5]). En outre, un tel contrôle agit comme une défense contre les opérations malveillantes qui visent à exploiter les autorisations d'accès initialement allouées aux partenaires à des fins collaboratives. En d'autres termes, il représente une exigence essentielle pour faire échouer toute tentative de forger une instance par un participant malveillant dans le but d'accéder illégalement à une ressource.

D'un autre coté, le contrôle du flux sortant au niveau de chaque organisation participante permet d'empêcher la divulgation des informations à des partenaires inappropriés. Ainsi, chaque message sortant doit être vérifié par rapport au schéma de collaboration prédéfini avant d'être envoyé, et ce, de façon dynamique et en fonction de l'étape courante.

1.2 Contributions de la thèse

Pour répondre à cette problématique, nous proposons une approche qui permet le suivi temps réel de l'évolution de l'exécution des processus métiers inter-organisationnels, dans laquelle les échanges inter-organisationels sont perçus comme des événements. En effet, notre approche repose uniquement sur les changements d'état d'une chorégraphie, c'est-à-dire quand un message de chorégraphie est envoyé ou reçu. Elle permet le suivi de la conformité des interactions entre les participants d'une manière discrète (i.e. non intrusive). En d'autres termes, les messages échangés ne sont pas modifiés et les pairs ne sont pas conscients des moniteurs (i.e. écoute passive). Elle permet de vérifier le comportement de chaque participant en analysant les évènements générés pour chaque échange de message détecté.

L'objectif principal de cette thèse est de fournir un mécanisme de supervision à la fois décentralisé (i.e. s'exécutant au niveau de chaque participant et échangeant des données de supervision entre partenaires) et efficace (i.e. pas trop de surcharge en terme de notifications échangés et de règles générées). Dans ce contexte, nous avons proposé une approche événementielle permettant :

1. d'automatiser la génération de requêtes de supervision (i.e. directement à partir du modèle de la chorégraphie),
2. de minimiser la surcharge en terme d'échange de données de supervision : l'envoi des notifications se fait de façon ciblée (uniquement vers un ensemble pré-calculé de participants),
3. et d'optimiser le nombre de règles de vérification de la conformité des séquences d'interaction afin de réduire le temps de calcul pour la détection instantanée des violations.

5. Denial of Service (déni de service)

Ainsi, le travail présenté dans cette thèse porte sur l'analyse et l'évaluation des événements générés par les échanges de messages entre les partenaires. Le but est de vérifier si le comportement réel des entités en interaction adhère précisément aux contraintes métier imposées. Les événements sont traités par des composants spéciaux. Ces composants sont implantés sous forme de services déployés le long des frontières de chaque organisation et invoqués à chaque détection de nouveau message inter-organisationnel. L'interception des messages (entrants et sortants) et l'invocation de ces services secondaires se fait en utilisant un mini-orchestrateur comme celui décrit dans notre modèle SMSOA [BPBG09]. Après la vérification du message par rapport à une politique générée automatiquement à partir du schéma de la chorégraphie, une notification est automatiquement générée en cas d'une violation détectée. Une telle approche permet de bien séparer les aspects fonctionnels et non fonctionnels et de respecter ainsi le principe de "séparation des préoccupations". Elle offre aussi une traçabilité de l'exécution des processus en échangeant les notifications entre les participants. La *figure* 1.2 donne un aperçu sur nos contributions.

La première contribution de cette thèse [BPG11] consiste à explorer les chorégraphies inter-organisationnelles d'une manière indépendante des langages de spécification et offrir un nouveau modèle conceptuel en proposant une architecture générale pour la supervision des collaborations. Dans le *chapitre* 4, nous proposons un modèle formel général, simple et compréhensible qui va permettre la mise en œuvre de notre approche à partir de tout type de langage de spécification. Nous introduisons la notion de vérification événementielle dans le cadre des chorégraphies de services et nous décrivons l'architecture générale de notre contribution. L'architecture que nous présentons (*cf. zone A de la figure* 1.2) consiste à fournir trois composants qui seront intégrés et déployés au sein de chaque organisation qui participe à une chorégraphie :

1. le contrôleur de flux externe (EFC) «*External Flow Controller*» pour assurer l'interception des messages échangés et la vérification de la structure de chaque message,
2. le superviseur de flux externe (EFM) «*External Flow Monitor*» pour la vérification du séquencement des messages et l'échange de notifications entre partenaires (i.e. avec les EFMs des autres participants),
3. et l'entrepôt de politique de flux externe (EFP) «*External Flow Policy*» pour le stockage et la gestion de règles de supervision.

La deuxième contribution [BFPG12, BPG13] consiste à proposer un mécanisme automatisé et décentralisé pour l'échange de données de supervision entre les participants d'une chorégraphie (*cf. chapitre* 5). Nous définissons de nouveaux canaux pour l'échange de notifications entre les différents EFMs. Notre approche n'est pas intrusive, c'est-à-dire que ces canaux nouvellement définis sont indépendants des canaux dans lesquels les messages de la chorégraphie sont échangés. L'EFM peut voir « passivement » ce qui est échangé sans rien modifier dans le modèle de chorégraphie. Afin de ne pas induire une surcharge du réseau, nous proposons une approche d'envoi ciblé au lieu de « broadcaster » toutes les notifications à tous les participants. En effet, l'envoi des notifications se fait uniquement vers un ensemble minimal de participants (que nous jugeons intéressés) selon le type de l'interaction dans le modèle prédéfini de la chorégraphie. Pour ce faire, nous présentons une nouvelle approche de classification hiérarchique des participants (i.e. arbre CPT) et nous définissons les notions de *super/sous partenaires* ainsi que la vue de

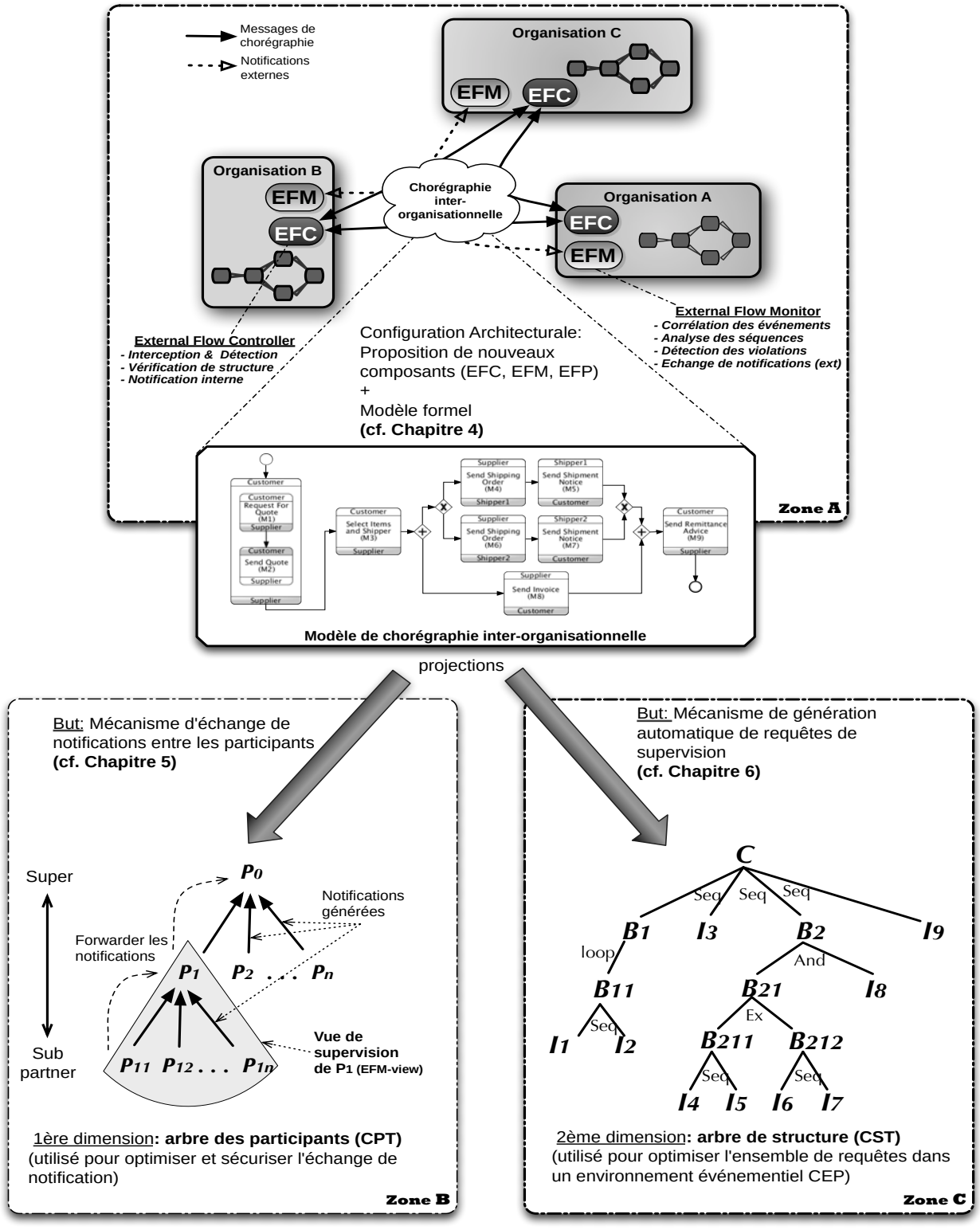


FIGURE 1.2 – Aperçu sur les contributions de la thèse

supervision *EFM-view* (*cf. zone B de la figure* 1.2). Nous généralisons notre approche d'échange de notifications afin de traiter un plus large spectre de modèles de chorégraphies [BPG13].

La troisième contribution [BPG12] est dédiée à l'analyse et l'évaluation des événements générés par les échanges de messages entre les partenaires (*cf. chapitre* 6). Le but est de vérifier si le comportement réel des entités en interaction adhère aux contraintes métier imposées. Pour y parvenir, nous proposons une approche évènementielle permettant la génération automatique d'un ensemble optimisé de requêtes à partir d'un schéma d'interaction défini. Cette approche permet de vérifier la conformité des séquences d'interaction par rapport au modèle de la chorégraphie, et ce, à chaque échange de message avec ses partenaires. La vérification se fait au niveau de chaque participant par rapport à sa vue de supervision *EFM-view*. Pendant la phase de configuration (qui se déroule une fois par modèle de chorégraphie), nous procédons à la fragmentation du modèle de chorégraphie en blocs SESE (single entry / single exit). Cette fragmentation nous permet de construire ce que nous appelons un arbre de structure de chorégraphie CST «*Choreography Structure Tree*» (*cf. zone C de la figure* 1.2). Ce dernier, qui est unique pour chaque modèle, représente l'arborescence des blocs. A partir de l'arbre CST généré et en fonction des règles de génération par patron, un ensemble optimal de requêtes CEP spécifiques au modèle en question est automatiquement généré. Nous distinguons deux types de requêtes générées : requêtes pour la vérification du séquencement des événements et requêtes de génération des évènements de haut niveau (i.e. les évènements de fin d'exécution d'un bloc *END-events*).

Afin de tester l'efficacité de notre approche et d'évaluer la surcharge qu'elle génère, nous nous basons sur une simulation de génération de séquences d'échanges de messages et nous observons et analysons les résultats obtenus (*cf. chapitre* 7). Nous commençons par un test d'exécution de nos algorithmes. Nous exposons par la suite les différentes expériences que nous avons effectuées grâce à un mécanisme de génération massive et aléatoire d'événements d'échange de messages. Ensuite, nous introduisons les résultats obtenus et donnons une évaluation globale de l'approche. Nous expliquons ainsi les résultats de la simulation et le gain apporté par notre méthode de génération de requêtes. Nous soulignons aussi l'aspect passage à l'échelle qui représente un de nos objectifs majeurs.

1.3 Synthèse

Dans ce chapitre, nous avons exposé la problématique de notre travail. Plus particulièrement, nous avons montré comment l'évolution des architectures IT des entreprises, la sous-traitance et l'externalisation des activités et des processus, ont poussé les entreprises à passer d'une architecture traditionnelle basée sur un orchestrateur centralisé, à une architecture moderne basée sur la collaboration et la coopération. Nous avons présenté le besoin qu'éprouve chaque organisation de superviser ses processus externalisés et d'échanger des données de supervision avec ses partenaires, et illustré la problématique par un exemple concret. Nous avons aussi, souligné l'intérêt d'un mécanisme de supervision instantanée et décentralisé. Enfin, nous avons exposé les différentes contributions développées tout au long de cette thèse pour répondre aux problèmes posés.

Dans le chapitre suivant, nous allons survoler les notions et les définitions des éléments de base nécessaires à la compréhension de ce mémoire. Nous introduisons les technologies SOA et BPM et leur rôle pour améliorer la compétitivité et l'agilité de l'entreprise. Ensuite, nous survolons les principaux concepts liés aux architectures orientées événements (EDA) et nous montrons comment elles peuvent compléter les architectures SOA et BPM en offrant un couplage très lâche entre les services. Enfin, nous soulignons l'importance des processeurs CEP qui permettent de détecter des patrons/motifs d'événements complexes diffusés massivement dans un système afin d'évaluer avec précision son état, et nous montrons comment ils peuvent être utilisés dans la supervision des processus métier.

2 Concepts et État de l'art

«I have deep faith that the principle of the universe will be beautiful and simple.»

Albert Einstein

Sommaire

Introduction

Dans ce chapitre, nous allons présenter les notions et les concepts de base que nous jugeons nécessaires à la compréhension de ce mémoire. Des concepts comme *SOA, BPM, EDA* et *CEP* représentent aujourd'hui les éléments clés des nouvelles architectures des systèmes d'informations des entreprises, et sont en relation directe avec la problématique de notre thèse. Nous commençons d'abord par expliquer les apports des architectures orientées services (SOA) dans une organisation par rapport aux anciennes architectures (section 2.1). Nous poursuivons par une présentation des différents concepts liés aux processus métiers (section 2.2). Nous expliquons ainsi comment la technologie workflow a pu fournir un excellent support pour la modélisation, l'analyse et l'automatisation des processus métiers (section 2.2.2). Ensuite, nous présentons les modèles de composition de service (section 2.3) en mettant l'accent sur la différence entre orchestration et chorégraphie de services. Nous introduisons la technologie CEP et la génération et le traitement des événements complexes dans un contexte SOA (section 2.4). Nous concluons ce chapitre par une description du BPM événementiel (*EdBPM*) en montrant comment ce concept permet de combiner à la fois les avantages de SOA, BPM et CEP (section 2.5).

2.1 Les architectures orientées services (SOA)

Les premières architectures [DA07] à base de services sont DCOM «*Distributed Component Object Model*» et ORB «*Object Request Brokers*» qui est basée sur la spécification de CORBA [omg92]. DCOM3 est la technologie proposée par Microsoft pour réaliser des appels distants à des objets COM «*Component Object Microsoft*», dont une version améliorée est COM+ [Pla99]. CORBA est un modèle d'architecture dont le but était de faciliter l'interaction des composants, indépendamment de leur plateforme d'origine et du langage dans lequel ils étaient développés. A cette fin, CORBA utilisait un modèle d'interface appelé IDL «*Interface Definition Language*» qui permet à un composant d'exposer aux autres les fonctionnalités qu'il offre. CORBA a été défini pour fonctionner dans un environnement hétérogène, mais il s'est avéré très complexe à mettre en oeuvre. L'OMG «*Object Management Group*» a défini le protocole IIOP «*Internet Inter ORB Protocol*» pour communiquer avec des objets CORBA à travers le réseau.

Les architectures orientées services (SOA) ont émergé à la suite de CORBA comme des architectures préférées pour la conception, le développement et l'intégration des systèmes d'informations (SI) de nouvelle génération [Dav09]. Ces architectures reposent sur la réorganisation des applications en ensembles fonctionnels appelés services et l'exposition des informations nécessaires sur ces services pour qu'ils soient facilement utilisés par les clients. L'objectif est donc de décomposer les différentes fonctionnalités d'un SI en un ensemble de fonctions basiques offertes par des services et de décrire finement un schéma d'interaction entre ces services. Ces applications-services peuvent être exécutés sur des plateformes hétérogènes dans un environnement distribué, et fournissent des fonctionnalités à d'autres entités (clients) [CCMN04].

"Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations." [BH06]

Entre autres, ces architectures représentent une nouvelle façon d'intégrer et de manipuler les différentes briques et composants applicatifs d'un système d'information et de gérer les liens qu'ils entretiennent.

"Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs." [BEA05]

En utilisant une analogie simple, SOA transforme les applications de l'entreprise en pièces de Lego, capables de s'emboiter dans toutes sortes de configurations et réutilisables pour différentes constructions ce qui augmente considérablement la flexibilité du système d'information. En outre, la deuxième valeur ajoutée d'une architecture SOA pour l'entreprise est son agilité, i.e., sa capacité à réduire sensiblement les coûts de maintenance et d'adaptation au changement.

2.1.1 Les services

Le service est le composant clef d'une architecture SOA. C'est une entité de traitement qui représente une fonction ou fonctionnalité bien définie qui est divisée en opérations. Un service peut implémenter plusieurs interfaces, et aussi plusieurs services peuvent implémenter une même interface. Les services interagissent et communiquent entre eux.

Les principales caractéristiques d'un service sont les suivantes :

Large Granularité « coarse-grained » : Les opérations d'un service peuvent encapsuler plusieurs fonctions et donc opérer sur un périmètre de données large au contraire de la notion de composant technique. En effet, il est possible de composer des services et de combiner ainsi les différentes fonctionnalités offertes par les fournisseurs de services ce qui permet d'offrir des services à forte valeur ajoutée aux clients.

Couplage faible « loosely-coupled » : Les services sont connectés aux clients et autres services via des standards (e.g. des documents XML [6]) [BPSM+08] qui assurent le découplage, i.e., la réduction des dépendances. En d'autres termes, les échanges entre les services ne dépendent pas de l'implémentation sur laquelle les applications offertes par ces derniers reposent.

6. Extensible Markup Language

2.1.2 Les services Web : une instance de SOA

Les service Web constituent une technologie à part entière qui ouvre de réelles opportunités au sein des systèmes d'informations des entreprises. Cette technologie représente indéniablement un nouvel atout pour l'entreprise étendue. En effet, un service web est un composant logiciel accessible à travers des intranets, des extranets et l'Internet moyennant l'utilisation des technologies du web (HTTP, URI, XML,...) [ACKM03]. Une architecture basée sur les services Web permet aux applications de communiquer facilement quelque soit la plate-forme sous-jacente.

"Un service Web est une application ou un composant logiciel (i) identifié par un URI, (ii) dont ses interfaces et ses liens (binding) peuvent être décrits en XML, (iii) sa définition peut être découverte par d'autres services Web et (iv) qui peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet." (Le consortium W3C [W3C92])

Techniquement parlant, les services Web sont des composants logiciels qui encapsulent des fonctionnalités métiers de l'entreprise et accessibles via des protocoles standards du Web. Un service Web est décrit dans un document WSDL [7] [W3C03c], précisant les méthodes pouvant être invoquées, leur signature, et les points d'accès du service (URL, port, etc.). Ces méthodes sont accessibles via des protocoles comme SOAP [8] [W3C03a] ou REST [9] [RR07] : la requête et la réponse sont des messages XML, transportés par HTTP [FFBL+96], SMTP [YM08] ou encore FTP [PCR85].

Plus d'interopérabilité et ouverture des SI sur le monde extérieur : Basés sur des protocoles de transport standards d'Internet comme le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux pare-feu sans nécessiter des changements sur les règles de filtrage. L'utilisation de standards et protocoles ouverts avec des formats de données compréhensibles facilite l'échange entre divers logiciels fonctionnant sur diverses plates-formes. La technologie service web permet donc de répondre à une ancienne problématique d'invocation de composants applicatifs dans une architecture distribuée.

2.1.3 La description des services avec WSDL

Officiellement recommandé par le W3C, le langage WSDL (Web Service Description Language) [W3C03c] permet de décrire l'interface d'un service web de façon universelle à l'aide d'une grammaire XML. Il définit les paramètres nécessaires pour invoquer un service, le format et le type des données retournées. Concrètement, un fichier WSDL est un document XML qui utilise XML Schema [FW04] dans lequel on trouve le nom du service, son emplacement et

7. Web Service Description Language
8. Simple Object Access Protocol
9. REpresentational State Transfer

comment y accéder (i.e. l'adresse du service Web sous la forme d'une URI [10] et les protocoles permettant son invocation) ainsi que la liste de ses opérations avec tous les paramètres d'entrée et de réponse. Le WSDL contient aussi des données détaillées sur des informations sur l'encodage technique des données. Les documents WSDL peuvent être référencés dans des annuaires permettant la recherche et la découverte des services disponibles (e.g. UDDI [11] [W3C03b]).

2.1.4 Les styles d'architecture : SOAP et REST

Un style d'architecture n'est pas un standard. En revanche, il peut s'appuyer sur des standards (e.g. URI pour le nommage des ressources, HTTP comme protocole de communication, XML pour les documents et les données, etc.). Dans le cadre des services Web, SOAP et REST sont les deux styles dominants.

SOAP SOAP [W3C03a] est, avant tout, un protocole de communication en charge de l'acheminement d'un message structuré en XML lors de l'invocation d'un service, et ce, en se basant sur des protocoles de transport réseau TCP/IP [Ste93] (e.g. HTTP, SMTP [YM08] ou encore FTP [PCR85]). Le choix du protocole est dirigé par les contraintes techniques du système ou encore le mode de communication désiré (synchrone ou asynchrone). Des messages de type document (e.g. ebXML [JJDM06]) peuvent ainsi être transmis de façon unidirectionnelle ou avec des dialogues requête-réponse supportant RPC [12]. Le protocole SOAP définit un ensemble de règles pour définir le contenu (i.e. les données véhiculées) et la structure du message (i.e. l'enveloppe). L'enveloppe qui est l'élément racine du document XML à transmettre est composée de deux parties : une en-tête «*SOAP header*» et un corps «*SOAP body*» (*cf. Figure* 2.1). L'en-tête SOAP est optionnelle et est typiquement utilisée pour transmettre des données d'authentification ou de gestion de session. Le corps SOAP fournit un mécanisme de transmission d'information.

FIGURE 2.1 – Structure d'un message SOAP

REST (REpresentational State Transfer) Le style architectural REST [RR07] estime qu'il n'est, dans bien des cas, pas nécessaire de faire appel aux couches d'abstraction proposées par SOAP et XML-RPC, et que les méthodes de HTTP, combinées avec de bonne URIs, suffisent

10. Uniform Resource Identifier
11. Universal Description, Discovery and Integration
12. Remote Procedural Call

amplement dans la majorité des cas. Cette philosophie, contrairement à celle de SOAP, se base sur le concept de ressource plutôt que de service. Les services dits RESTful ont permis d'apporter de réels gains en termes d'usage et de performance. En effet, l'absence de contrat WSDL spécifique pour chaque service a permis de réduire le couplage entre le client et le fournisseur de service et minimiser les différences d'interface et de sémantique au sein de ressources hétérogènes. Contrairement à SOAP qui induit un typage fort au travers du WSDL, une ressource REST n'est pas typée, et peut être représentée via tout type de format d'échange de données même avec du simple texte. Face à ces avantages, plusieurs entreprises n'ont pas hésité à abandonner l'architecture SOAP au profit d'une architecture REST. Certaines entreprises combinent les deux styles architecturaux : REST pour plus de simplicité, et SOAP pour le support de W3C et des grandes compagnies (e.g. Microsoft, IBM, Oracle, etc.).

2.1.5 SCA : Service Component Architecture

Les spécifications SCA visent à simplifier la construction d'architectures SOA en adressant plus particulièrement l'aspect composition (e.g. comment packager un composant logiciel afin que d'autres applications puissent l'utiliser) et l'aspect assemblage (e.g. comment les composants peuvent fonctionner ensemble). De plus, elles adressent les politiques d'accès aux composants (e.g. restriction d'accès, authentification , etc.). En effet, l'assemblage SCA permet de proposer un découpage en fonctionnalités alignées sur l'organisation de l'entreprise tout en définissant comment un nouveau composant (ou une application existante) peut exposer des services métiers, et comment ces services sont assemblés en «composites». En d'autres termes, les applications se présentent comme un assemblage de composants où chaque composant implante une partie de la logique métier et peut dépendre de services et/ou exposer, à son tour, tout ou une partie de son comportement comme un service. Ceci facilite l'intégration de nouveaux composants et rend la réutilisation de briques existantes plus simple.

Cependant, toutes les communications entre les composants et les composites sont définies au sein d'un même domaine et se font d'une manière spécifique au fournisseur. En effet, une application SCA qui désire communiquer avec une autre application SCA dans un domaine différent ne peut pas voir cette dernière comme une application SCA ; car l'utilisation de SCA n'est pas visible en dehors de son domaine. Pour communiquer entre domaines (i.e. communication inter-organisationnelle) ou avec des applications non-SCA, un composant doit généralement permettre l'accès via des services Web ou d'autres mécanismes interopérables.

2.1.6 ESB : Enterprise Service Bus

Les ESB sont les héritiers directs des EAI (Enterprise Application Integration) [13] et représentent aujourd'hui la technologie d'intégration et de médiation inter-applicative pour la mise en oeuvre d'une architecture orientée service généralement basée sur des services Web SOAP (*cf. Figure* 2.2). Reprenant les caractéristiques architecturales des solutions d'EAI, les ESB s'appuient

13. Une solution middleware dans laquelle un composant central assure la médiation physique entre le client et sa cible

sur un ensemble de standards parmi lesquels :

- Les Services Web pour gérer les communications synchrones.
- XML pour définir les formats des messages.
- JMS (Java Message Service) pour la communication asynchrone.
- JCA (Java Connector Architecture) pour la connexion aux progiciels et systèmes d'information existants appelés legacy (ERP [14], CRM [15], etc.).

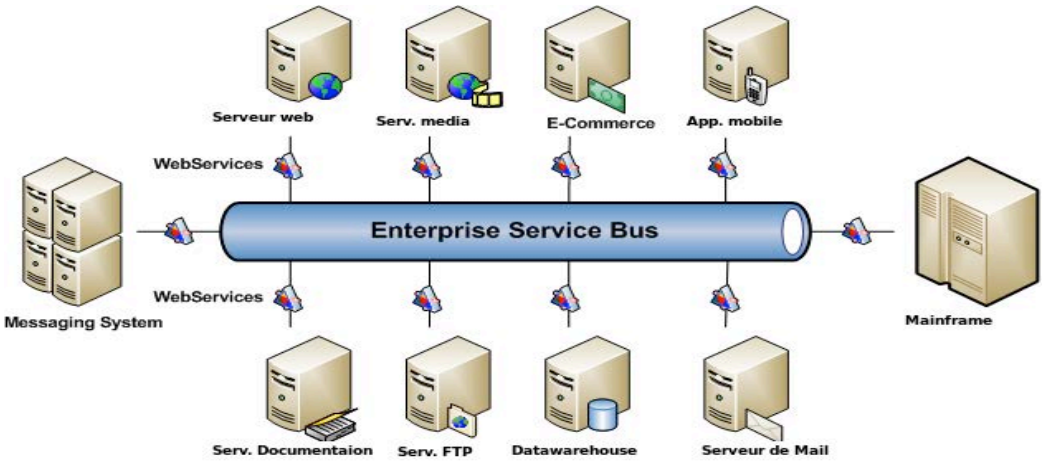


FIGURE 2.2 – Enterprise Service Bus (ESB)

Synthétiquement, le rôle de l'ESB est d'assurer la connexion et la médiation entre les services. Il permet, de plus, le suivi et la fiabilisation de l'activité des services. Il représente ainsi l'infrastructure qui optimise les échanges entre consommateurs et fournisseurs de services. Il peut donc prendre en charge le routage, la transformation des données, les transactions, la sécurité et la qualité de service.

2.1.7 Les avantages de la SOA

Le paradigme d'Architecture Orientée Services, plus particulièrement lorsqu'elle est basée sur la technologie des services web, offre un ensemble de principes d'exécution qui sont en adéquation avec les pré-requis fonctionnels des plate-formes des entreprises [Ram06, Yil08] :

- **Couplage faible,** les échanges entre collaborateurs ne dépendent pas de l'implémentation sur laquelle les applications offertes par ces derniers reposent. En effet, la possibilité d'établir des communications avec les services, qui sont découverts dynamiquement, exige un couplage faible entre le client et le fournisseur du service à invoquer. La notion de couplage faible est une des caractéristiques essentielles des technologies des services Web. Cette notion exclut toutes les hypothèses sur les plate-formes spécifiques que le client ou le fournisseur impose. Ceci est valable pour le format et le protocole qui caractérisent l'interaction de ces derniers [Fdh11].

14. Enterprise Resource Planning
15. Customer Relationship Management

- **Interopérabilité,** d'après [AF01, CNW01], les architectures orientées services ont pour objectif de permettre à des applications hétérogènes s'exécutant au sein de différentes entreprises de pouvoir inter-opérer à travers des services offerts par les applications. L'hétérogénéité des applications n'est pas seulement considérée au niveau des langages d'implémentation des applications, mais aussi au niveau des modèles d'interaction, des protocoles de communication et de la qualité des services. En effet, l'interopérabilité est présente à plusieurs niveaux dans une architecture SOA. Tout d'abord, entre les services eux-mêmes : en effet, rien n'oblige à utiliser la même plate-forme entre les différents services formant une application à part entière. Ensuite, au niveau des clients et des services : les architectures logicielle et matérielle peuvent être totalement différentes, l'essentiel est la mise à disposition des protocoles des services web sur ces architectures, indépendamment du langage et des logiciels utilisés.
- **Dynamicité,** liée au fait que les services web reposent sur des communications entre un client et un service. Si la localisation d'un service est souvent stable (pour permettre de le localiser sur le long terme), il n'en est pas de même pour le client, car celui-ci est souvent connecté à travers Internet par un fournisseur d'accès ne lui attribuant pas une adresse IP fixe par exemple [Fdh11].
- **Composition de services web,** les fonctionnalités offertes par les fournisseurs de services peuvent être combinées pour offrir des services à forte valeur ajoutée aux clients. Ceci est réalisé via des langages spécifiques de descriptions comportementales qui permettent la description du comportement du service composé, en indiquant comment les communications vont se réaliser entre les différents services [Fdh11].

2.1.8 Modèle générique d'architecture

L'architecture SOA peut être décrite de la façon suivante :

– Le système d'information de l'entreprise est découpé en services. Le niveau de granularité devient le service et non plus l'application [Cru03].
– Les services délèguent les traitements aux applications tout en fournissant une interface indépendante de ces applications.
– Les services transverses utilisés par les applications sont mutualisés (e.g. la notification, la sécurité ou le *logging* sont accessibles sous forme de services). Il n'est plus nécessaire, par exemple, de re-développer la gestion de sécurité et/ou du *logging* pour chaque application.

Les échanges entre les services sont gérés en utilisant des fonctionnalités de *Service Management*. Il devient alors possible de diagnostiquer et de prévenir par exemple les problèmes d'indisponibilité ou de montée en charge d'une application. Le *Service Management* permet aussi d'avoir une vue générale sur l'ensemble des composants en remontant des informations basées sur des indicateurs métiers à destination des décideurs de l'entreprise.

Nous présentons la vue d'ensemble d'un modèle générique d'architecture orientée services. La *figure* 4.5 permet de décrire un cadre global pour les fonctions importantes à mettre en place au sein d'une organisation ainsi que les différents modes d'utilisation des services Web [Bon05]. Sur cette figure, nous discernons trois modes d'accès au services : interne au SI (e.g. WS 1,

WS 2, WS 3), externe depuis des organisations clientes (e.g. WS 4, WS 5, WS 6) ou vers des fournisseurs externes (e.g. WS 7). Les services Web WS 1 et WS 4 invoquent d'autres services Web internes (SI production-distribution) lorsqu'ils s'exécutent pour constituer des orchestrations de services. Les services WS 3 et WS 6 déclenchent des traitements par l'intermédiaire de connecteurs propriétaires.

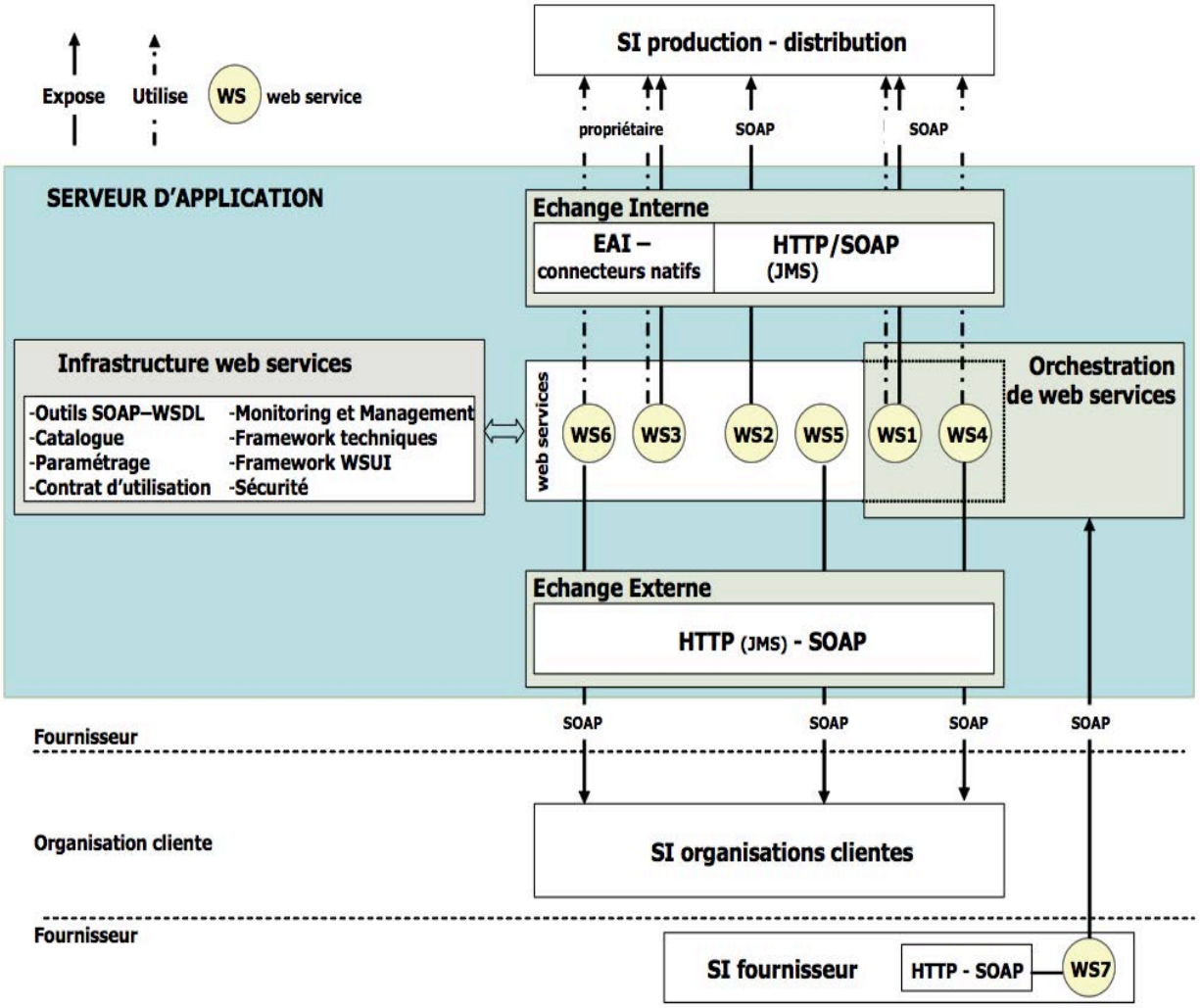


FIGURE 2.3 – Modèle d'architecture général autour des services Web [Bon05].

Les différentes modules de ce modèle d'architecture (e.g. infrastructure services Web, connecteurs pour l'échange interne et externe, orchestration) sont contenues dans un serveur d'application (e.g. Glassfish, Tomcat, JBoss, IBM Websphere). Parmi les technologies de serveurs d'application nous pouvons citer *JEE*[16] du monde java et *.Net* de Microsoft. Dans le cas de JEE par exemple, les services peuvent être formés par des classes java. La logique fonctionnelle peut être accédée via une interface JMS[17].

16. Java Enterprise Edition, ou Java EE (anciennement J2EE)
17. Java Message Service

Le module d'échange interne assure la communication entre les applicatifs internes d'une organisation. Le transport peut être propriétaire ou standardisé en services Web. Ce module permet l'unification de la communication entre les applicatifs au travers de connecteurs, la transformation des données échangées « *data mapping* » et la gestion des transactions distribuées.

Le module d'échange externe utilise généralement les standards service Web pour la communication. Il permet donc de transporter les messages SOAP par l'intermédiaire de HTTP. Un transport par JMS peut aussi être utilisé dans le cas où l'organisation distante utilise également JMS.

2.2 La gestion des processus métiers (BPM)

Dans cette section, nous présentons les différents concepts de bases liés aux processus métiers ainsi qu'à leur gestion.

2.2.1 Processus métiers

Un processus métier consiste en un ensemble d'activités exécutées et coordonnées dans le cadre d'un environnement organisationnel et technique, pour réaliser un objectif prédéterminé. Un processus métier est généralement affecté à une seule organisation, mais peut interagir avec d'autres processus métiers appartenant à d'autres organisations dites partenaires. La gestion de ces processus inclut les concepts, les méthodes et les techniques nécessaires pour la conception, l'administration, la configuration, l'exécution et l'analyse des processus métiers [Wes07].

"A business process is a structured, measured set of activities designed to produce a specific output for a particular customer or market... A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs : a structure for action... Processes are the structure by which an organization does what is necessary to produce value for its customers." [Dav93]

Processus intra/inter-organisationnel Quand un processus métier est exécuté par une seule organisation, c'est-à-dire, sans aucune interaction avec d'autres processus métiers exécutés par d'autres organisations, ce processus est dit *intra-organisationnel*. Toutefois, un grand nombre de processus métiers interagissent avec d'autres processus appartenant à d'autres organisations formant ainsi des processus *inter-organisationnels*. De telles interactions représentent des collaborations entre les entreprises en question, dans le cadre par exemple d'un scénario B2B « *Business-to-Business* ».

2.2.2 Gestion des processus métiers

La gestion des processus métiers BPM « *Business Process Managenement* » considère que chaque produit fourni par une entreprise est le résultat d'exécution d'un ensemble d'activités. En effet, les processus métiers sont essentiels pour comprendre le fonctionnement des entreprises et se présentent aujourd'hui comme un outil clé pour l'organisation des différentes activités et l'amélioration des interactions entre elles. Les systèmes d'information jouent un rôle important dans la gestion des processus métiers, vu que la plupart des entreprises de nos jours ont tendance à automatiser leurs processus et que leurs activités sont de plus en plus supportées par les systèmes d'information. En effet, les activités d'un processus métier peuvent être exécutées manuellement, par des employés ou à l'aide du système d'information [Wes07]. Certaines activités sont totalement automatisées sans aucune intervention humaine. Sur le plan organisationnel, les processus métiers jouent aussi un rôle important au niveau de la conception et la réalisation de systèmes d'information flexibles. Ces systèmes d'information fournissent une base technique pour la création rapide de nouvelles fonctionnalités qui participent à leur tour à la création de nouveaux produits et pour l'adaptation des fonctionnalités qui existent aux nouveaux besoins du marché.

Niveaux d'abstraction de BPM

Le BPM représente la première initiative visant à offrir un niveau d'abstraction par rapport aux technologies sous-jacentes et à capitaliser sur l'existant [Cru03]. L'objectif principal était de définir des processus répondant aux besoins métiers et fonctionnels tout en se basant sur les services offerts par le SI. Il s'agit de la réunification des différentes visions métiers, et techniques. Un processus métier est généralement modélisé en trois niveaux (cf. *figure* 2.4) :

- **Le niveau métier :** vue de haut niveau qui permet de définir les principales étapes et l'impact sur l'organisation de l'entreprise. Ce niveau est généralement défini par les décideurs, et les responsables méthodes de l'entreprise.
- **Le niveau fonctionnel :** formalisation des interactions entre les différents participants fonctionnels du processus. Ce niveau est généralement modélisé par les équipes fonctionnelles.
- **Le niveau technique :** vue de bas niveau qui permet de définir le lien entre les activités/participants modélisés dans le niveau fonctionnel, et les applications/services du SI. Ce niveau est généralement réalisé par les architectes et les équipes techniques de l'entreprise.

Standardisation de BPM

La standardisation du BPM se fait à différents niveaux [Cru03] :

- **Modélisation du processus :** la notation graphique doit permettre la modélisation métier, et le renseignement des informations techniques pour rendre les processus exécutables. Afin que les outils de modélisation et d'implémentation puissent communiquer, il est aussi nécessaire d'avoir un format d'import/export et/ou une notation graphique commune à

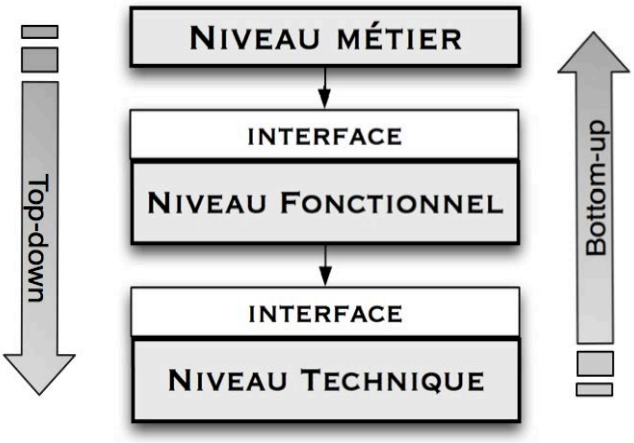


FIGURE 2.4 – Niveaux d'abstraction de BPM [Fdh11].

tous les outils de modélisation.

- **Exécution du processus :** les éléments déployés et exécutés sur les serveurs BPMS «systèmes de gestion des processus métiers» doivent être standardisés afin de garantir la portabilité des processus réalisés sur différentes plate-formes.
- **Connectivité :** les applications invoquées par les processus métier doivent fournir des connecteurs standards et indépendants de toute architecture, i.e., systèmes d'exploitation, bases de données, plateforme (e.g. Java, .Net), etc.

Workflow

Les processus métiers ont permis aux entreprises de réduire considérablement les coûts des transactions (le flux de travail est gouverné par des processus) [BV05]. Toutefois, cette tâche ne s'avère pas facile surtout dans les organisations qui gèrent un grand nombre d'informations non nécessairement structurées. Vu les problèmes de coordination, il y a besoin de plus de flexibilité et d'une adaptation rapide au changements du marché [BAC+07]. Dans ce sens, la technologie workflow apparait comme une solution pour répondre à la plupart de ces problèmes. En effet, l'approche *workflow* est une technologie clé pour l'automatisation des processus métiers. Cette technologie supporte les processus métiers qui intègrent des applications aussi bien que ceux qui impliquent des tâches humaines [Wes07].

L'effort majeur pour standardiser le workflow a été fait par la WfMC (Workflow Management Coalition) [wfm98]. La WfMC est un consortium international d'éditeurs de workflow, d'utilisateurs, d'analystes et de groupes de recherches, dont les objectifs sont la promotion des technologies de workflow et la définition de standards. Parmi ces standards, nous pouvons citer des langages de définition de workflow et API [18] pour accéder au WfMS (Système de Gestion de Workflow) [LOP05].

La WfMC définit un workflow comme étant la vision automatisée de la totalité ou d'une partie

18. Application Programming Interface

d'un processus durant laquelle des documents, des informations, ou des tâches sont transmises d'un participant à un autre en suivant des règles procédurales. De façon plus pratique, le workflow décrit le circuit de validation, les tâches à accomplir entre les différents acteurs d'un processus, les délais, les modes de validation, et fournit à chacun des acteurs les informations nécessaires pour la réalisation de sa tâche [wfm98].

"The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [wfm98]

Système de gestion de workflow

Un système de gestion de workflow est un système qui définit, crée et gère l'exécution de workflows grâce à l'utilisation d'un logiciel capable d'interpréter les définitions de processus, d'interagir avec les participants et, lorsque cela est requis, d'invoquer les outils et les applications [Fdh11].

"Les systèmes de gestion de workflows sont des collecticiels qui offrent des mécanismes de modélisation et d'automatisation de processus métiers. Ils sont parmi les systèmes les plus élaborés pour définir et exécuter des processus. Ils permettent, en particulier, de décrire explicitement les méthodes de travail réalisant un processus, de les expérimenter et de mesurer leurs qualités." [Gaa06]

Processus vs Workflow

Si un processus permet de décrire d'une manière informelle les méthodes de travail d'un groupe de personnes et les règles qui les régissent, un workflow permet de formaliser, de structurer, d'automatiser (dans la mesure du possible) et d'exécuter ces méthodes de travail. La *figure* 2.5 définit la relation entre un processus et un workflow, et décrit les relations entre les différents autres éléments terminologiques que nous allons décortiquer dans la section suivante.

2.2.3 Terminologie et concepts de base

Modèle de processus un modèle de processus est la représentation d'un processus dans une forme qui permet sa manipulation automatique (modélisation, exécution) par un système de gestion de workflow [wfm98]. Un modèle de processus est constitué d'un réseau d'activités et des dépendances entre elles, des critères pour spécifier le démarrage et la terminaison d'un processus et des informations sur les activités individuelles (participants, applications, données informatiques).

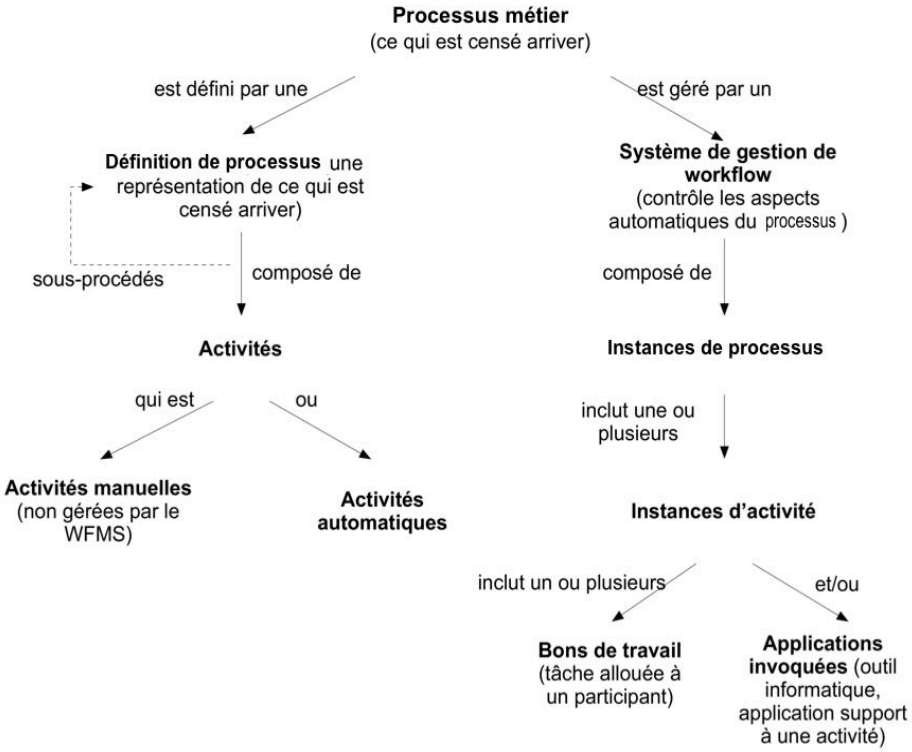


FIGURE 2.5 – Relations entre les concepts de base [Fdh11].

"Un modèle de processus est un graphe acyclique dans lequel les noeuds représentent les étapes d'exécution et les arcs représentent le flot de contrôle et le flot de données entre ces étapes." [wfm98]

Instance de processus une instance de processus est un cas d'une exécution d'un modèle de processus, incluant des instances de données associées. Chaque instance représente une exécution du processus qui est contrôlée séparément, a son propre état interne et sa propre identité externe. Cette exécution peut inclure le stockage et la synthèse de données des traces d'exécution (log) [wfm98].

Modèle vs instance de processus tandis qu'un modèle de processus métier représente un ensemble de modèles d'activités ainsi que les contraintes d'exécution entre elles, une instance de processus représente un cas concret d'exécution des activités qui le composent.

Activité une activité est une description d'un fragment de travail qui constitue une étape logique à l'intérieur d'un processus [wfm98]. Elle peut être manuelle ou automatique. Pour s'exécuter, une activité utilise des ressources humaines et/ou machines. Quand une ressource humaine est requise, la réalisation de l'activité est attribuée à un participant. Chaque activité a un nom, un type, des pré/post conditions et des contraintes d'ordonnancement [Gaa06].

Instance d'activité une instance d'activité représente une activité au sein d'une instance de processus. Chaque instance d'activité représente une invocation simple de l'activité au sein d'exactement une instance de processus et utilise les données associées à cette instance. Plusieurs instances d'activité peuvent être associées au même moment à une instance de processus (e.g. cas d'activités parallèles ou en boucle) mais une instance d'activité ne peut pas être associée à plus d'une instance de processus [wfm98].

"Activities instances represent the actual work conducted during business processes, the actual unit of work." [Wes07]

Transition une transition est un point dans l'exécution d'une instance de processus où une activité se termine et une autre démarre. Une transition peut être inconditionnelle (i.e. la terminaison de l'activité précédente déclenche le démarrage de l'activité suivante) ou conditionnelle (i.e. le déclenchement est gardé par une condition logique) [wfm98]. Une pré-condition (resp. post-condition) est une condition logique portant sur les données relevantes qui est évaluée au moment de l'exécution pour décider si une instance d'activité peut démarrer (resp. terminer).

Données de processus à chaque modèle correspond un ensemble de données qui décrivent toutes les informations nécessaires pour son exécution. Ces données incluent (i) des informations requises en entrée des activités, (ii) des informations requises pour l'évaluation des conditions et (iii) des données qui doivent être échangées entre les activités [Bhi05].

Flot de Contrôle séparer la logique d'exécution des détails techniques d'implémentation permet d'analyser le modèle de processus et de raisonner dessus [Bhi05]. Plusieurs langages ont été proposés pour la spécification des modèles de workfow. Certains langages se basent sur les techniques de modélisations existantes comme les réseaux de Petri [vdAvH04], les diagrammes d'activités [Gro99, CSE+00], l'algèbre des processus [MPW92], etc. D'autres langages sont spécifiques. Cette variété de langages et de concepts pénalise l'interopérabilité entre les systèmes de workflows et rend difficile une compréhension commune des systèmes existants et leur comparaison. Afin de remédier à ces problèmes, Van der Aalst *et al.* [ABHK00] ont défini les patrons de workflows dans lesquels ils décrivent d'une façon abstraite les différentes formes d'interactions recensées dans les systèmes de workflow, et ce, dans l'objectif de donner une vue commune et assez complète sur l'existant.

"Dans les langages de modélisation de processus, le flot de contrôle fait référence à l'ordre dans lequel les activités sont exécutées et c'est l'évaluation des règles de transition entre les activités, des pré-conditions et des post-conditions des activités qui décident de la navigation dans le modèle de processus et du flot de contrôle." [GPB+09]

Patrons de contrôle afin de simplifier la modélisation des processus, des composants génériques réutilisables ont été identifiés. On les appelle des patrons de processus [GPB+09]. Ces derniers varient de constructeurs de flot de contrôle simples comme le routage séquentiel à des patrons complexes nécessitant des mécanismes de routage plus avancés comme le patron *discriminateur* ou *instances-multiples*. Un ensemble de patrons de processus a été identifié initialement par la WFMC puis plus largement par van der Aalst et al. [VDATHKB03]. Nous résumons l'ensemble des patrons les plus utilisés dans la *table* 2.1.

2.2.4 Classification des processus métier

Nous pouvons classifier les processus métiers en deux types : les processus abstraits et les processus exécutables. D'un autre point de vue, il est possible aussi de distinguer les processus selon leur structuration : processus structurés et processus non structurés.

Processus abstraits ils définissent une représentation haut niveau des processus métiers [Cru03] en spécifiant les échanges de messages publics entre les différentes parties sans, toutefois, spécifier le comportement interne de chacun d'eux. Le processus ainsi décrit n'est pas exécutable. En d'autres termes, la définition des processus abstraits est indépendante des aspects techniques : on ne précise pas quelles sont les actions effectivement réalisées par les activités (e.g. comment une demande de devis est reçue, quelle application est utilisée pour vérifier la disponibilité des produits, etc.).

Processus exécutables à l'opposé des processus abstraits, les processus exécutables décrivent le fonctionnement interne du processus. Un processus exécutable représente la nature et l'ordre des différents échanges entre les différentes parties. Il définit le processus métier lui-même et est directement exécutable par un moteur d'orchestration. Le cheminement interne et les conditions sont donc visibles dans sa description.

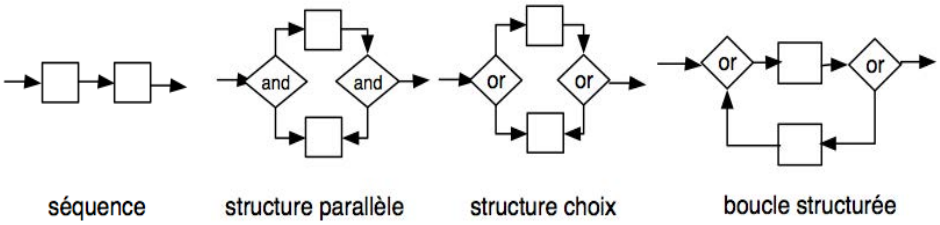
Processus structurés un processus structuré est un processus restreint par un ensemble de règles de construction liées au flux de contrôle [KtHB00, vdABCC05]. Dans un processus structuré, à chaque patron *split* (e.g. OR-split, AND-split) doit correspondre un élément *join* de même type (e.g. OR-join, AND-join). De plus, les paires *split-join* sont proprement imbriqués et les blocs se situant entre ces paires n'ont qu'un seul point d'entrée et un seul point de sortie. En effet, il existe quatre types de base de processus structurés : séquence, choix, parallèle et boucle (cf. *figure* 2.6.a).

Processus non structurés processus ne vérifiant pas les contraintes de structuration (i.e. contraintes sur le flux de contrôle). Le schéma de la *figure* 2.6.b montre un exemple de modèle de processus non structuré. Malgré que l'analyse de ces processus reste complexe, ces derniers permettent une meilleure expressivité. Dans ce sens, plusieurs travaux récents ont été proposés

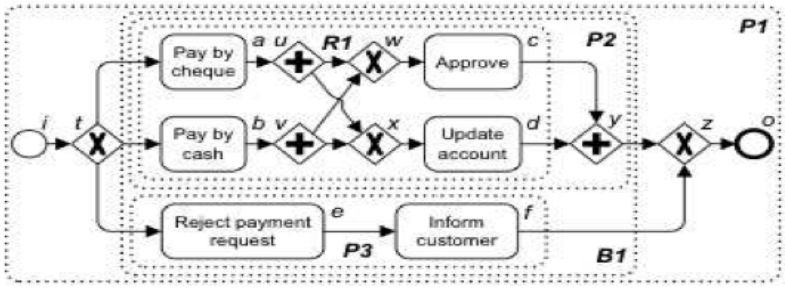
Catégorie	Patron	Description
Flot de contrôle de base	**Séquence**	Une activité d'un processus est activée juste après la terminaison d'une autre.
	Branchemepnt Multiple	AND-split est un point dans le workflow, où un itinéraire unique se sépare en deux ou plusieurs itinéraires différents dans le but de réaliser deux ou plusieurs activités en parallèle.
	Synchronisation	AND-join est un point dans le workflow, où deux ou plusieurs itinéraires parallèles convergent vers un itinéraire unique, avec synchronisation.
	Choix exclusif	XOR-split : un itinéraire s'ouvre sur plusieurs itinéraires possibles et que le cas d'exécution suit un seul de ces itinéraires.
	Jonction simple	XOR-join : deux ou plusieurs itinéraires alternatives convergent vers une même activité sans synchronisation.
Branchement et synchronisation avancés	**Choix Multiple**	OR-split : un itinéraire s'ouvre sur plusieurs itinéraires possibles et que le cas d'exécution suit un ou plusieurs de ces itinéraires, selon les conditions de transition.
	Jonction Multiple	Deux ou plusieurs itinéraires convergent vers un itinéraire unique et que l'on assure l'activation de ce dernier autant de fois qu'il y a d'itinéraires actifs.
	Jonction Synchronisée	Deux ou plusieurs itinéraires convergent vers un itinéraire unique et que l'on assure la synchronisation des itinéraires actifs.
	Discriminateur	Deux ou plusieurs itinéraires convergent vers un itinéraire unique dont on assure l'activation une seule fois.
Patron Instances Multiples (IM)	**IM fixées lors de la conception**	Plusieurs instances, d'une activité ou d'un bloc d'activités sont créés. Ces instances doivent synchroniser avant de passer à la suite du processus. Le nombre d'instances est connu au moment de la conception.
	IM fixées au cours de l'exécution	Plusieurs instances, d'une activité ou d'un bloc d'activités sont créés. Ces instances doivent synchroniser avant de passer à la suite du processus. Le nombre d'instances n'est connu qu'au cours de l'exécution.
Boucles	**Répéter**	Répéter une activité ou un bloc d'activités jusqu'à ce que la condition de sortie soit évaluée à faux. La condition de répétition est évaluée à la fin de la boucle.
	Tant que	Répéter une activité ou un bloc d'activités tant que la condition d'entrée est évalué à vrai. La condition de répétition est évaluée à l'entrée de la boucle.

Table 2.1 – Patrons de base dans les processus métier

pour transformer des processus non structurés en des modèles structurés équivalents [vdABCC05,



a) Patrons de base des processus structurés



b) Exemple de processus non structuré

FIGURE 2.6 – Processus structurés et processus non structurés.

SO00b, SO00a, PGBD10, DGBD09].

2.3 BPM en mode SOA : Composition de services

Pour faire face aux changements stratégiques, les systèmes d'information des entreprises doivent supporter des évolutions rapides et simples, ce que l'on nomme l'agilité. Devenir agile c'est aussi prendre en considération les processus et leur organisation, optimiser les ressources, rationaliser le travail et vouloir s'améliorer de façon continue tout en évitant d'incessants nouveaux développements informatiques et minimisant les transformations et compilations de programmes qui risquent d'alourdir sans cohérence le système d'information. Par ailleurs, la croissance de la taille de l'organisation risque également d'entraîner des changements importants dans l'organisation de l'activité. Ces changements sont souvent à la source de dysfonctionnements.

En combinant BPM avec SOA, l'organisation et les processus de travail peuvent changer en ne modifiant que l'orchestration et non pas les activités composantes. Comme toute gestion de processus, le BPM en mode SOA permet d'automatiser le fonctionnement selon des critères bien définis par les collaborateurs, réduit le temps de réalisation et minimise les risques d'erreur, tout en restant flexible et agile.

L'objectif de la composition de services est de créer de nouvelles fonctionnalités en combinant des fonctionnalités offertes par d'autres services déjà existants, en vue d'apporter une nouvelle valeur ajoutée [vdADtH03, Bhi05]. La composition de service, étant donnée une spécification de haut niveau, implique la capacité de sélectionner, de composer et de faire inter-opérer des services

existants. Contrairement aux *processus métiers* traditionnels qui sont exécutés de manière prévisible et répétitive dans un environnement statique, les processus orientés services s'exécutent dans un environnement versatile dans lequel le nombre de services disponibles peut évoluer rapidement. De plus, la forte compétition engendrée par la multitude de fournisseurs de services oblige les entreprises à adapter leurs services pour mieux répondre aux besoins des clients et ce à moindre coût [Fdh11]. L'utilisation d'un protocole de coordination permet de gérer de façon appropriée l'ordre et le type des messages échangés.

La composition de services web peut être étudiée selon deux points de vue complémentaires [DA07] : (i) un point de vue global dans lequel l'ensemble des partenaires participant à la composition est considéré ; ce modèle est appelé «chorégraphie», (ii) un point de vue local ou privé dans lequel seul le processus interne d'un participant est modélisé ; le modèle qui en découle est appelé «orchestration».

Nous rappelons qu'une composition de services peut aussi être décrite en termes d'un processus abstrait ou exécutable.

2.3.1 Orchestration vs Chorégraphie, centralisé / décentralisé

Orchestration de services Une orchestration définit l'enchaînement des services (i.e. la logique et les séquences d'invocation) selon un schéma prédéfini, et permet de les exécuter à travers des processus métier ou des workflows inter/intra-entreprises. En d'autres termes, l'orchestration de services permet la description d'un ensemble d'actions internes (dans lesquelles un service donné peut ou doit s'engager afin de remplir ses fonctions) et d'actions communicatives (i.e. d'envoi et de réception de messages) ainsi que les dépendances entre ces actions.

Chorégraphie de services Une chorégraphie est typiquement associée à l'échange de messages publics entre les services, plutôt qu'à un processus métier spécifique exécuté par un seul participant [Pel03]. Elle décrit, d'une part, un ensemble d'interactions qui peuvent ou doivent avoir lieu entre un ensemble de services/participants qui sont généralement représentés de façon abstraite par des rôles, et d'autre part, les dépendances entre ces interactions. La chorégraphie définit la séquence de messages pouvant impliquer plusieurs parties et plusieurs sources, incluant les clients, les fournisseurs, et les partenaires.

Orchestration vs Chorégraphie L'orchestration offre une vision centralisée, i.e., le processus est toujours contrôlé du point de vue d'un des partenaires métier. La chorégraphie est de nature plus collaborative, i.e., chaque participant impliqué dans le processus décrit le rôle qu'il joue dans l'interaction. Dans une chorégraphie, il n'y a pas de coordinateur central. La logique qui contrôle les interactions entre les services composants est distribuée entre les participants. Chaque service impliqué dans la chorégraphie connaît exactement avec qui interagir et quand exécuter les opérations dont il est responsable [DA07]. Les opérations internes des participants ne sont pas considérées.

2.3.2 Modélisation des processus métier

Comme nous l'avons déjà mentionné, la modélisation des processus métiers d'une entreprise consiste à représenter sa structure et son fonctionnement selon un certain point de vue. Dans ce sens, plusieurs langages de modélisation ont été proposés dans la littérature. Globalement, si nous regardons leurs syntaxes et leurs sémantiques, tous ces langages sont sensiblement similaires [KBP00]. D'un point de vue un peu réducteur, nous parvenons à retrouver des activités exécutées par des acteurs et reliées entre elles par des flux orientés par des points de décisions. Les différences se situent principalement au niveau de la représentation graphique et l'ensemble de patrons supportés. Nous citons les deux langages de modélisation les plus connus à savoir BPMN et UML. Ces derniers sont dotés d'une notation graphique permettant de faciliter la tâche de conception des processus et d'améliorer sa lisibilité. Cependant, ces langages ne sont pas directement exécutables. Une étude comparative des deux standards a été élaborée dans [DtH01, KLL09].

BPMN (*Business Process Modeling Notation*) la notation BPMN est une initiative du BPMI «*Business Process Management Initiative*» et est maintenant maintenue par l'OMG «*Object Management Group*». L'objectif était de définir une notation graphique permettant la modélisation des processus métiers qui soit compréhensible par tous les utilisateurs de l'entreprise (i.e. les analystes métier, les développeurs et les utilisateurs qui vont gérer et superviser l'exécution de ces processus) [KLL09]. La notation BPMN décrit statiquement les processus. BPMN a permis d'améliorer les possibilités des notations traditionnelles en gérant les concepts de procédures B2B, comme les processus publics et privés, et les chorégraphies (à partir de sa version 2.0), ainsi que des concepts de modélisation avancée, comme la gestion des exceptions et la compensation des transactions. Toutefois, c'est un langage de conception des processus et ce n'est pas un langage d'exécution.

UML (*Unified Modeling Langage*) a été défini par l'OMG [omg92]. Dans ses versions 1.X, UML proposait un ensemble de diagrammes permettant de faire de la modélisation de processus métiers même si le but premier de ce langage était la conception orientée objet [KLL09]. UML devait néanmoins être complété par des stéréotypes supplémentaires pour que l'ensemble des concepts métiers puissent être exprimés. De plus, certaines erreurs sémantiques ont apparu dans les versions 1.X et ont été corrigées dans la version 2.0. De nouveaux diagrammes ont également été ajoutés dans cette version mais restent peu intuitifs par rapport à BPMN. Cependant, UML dispose de son méta-modèle et d'un format d'échange ce qui rend les diagrammes UML plus facilement exploitables et outillables.

2.3.3 Exécution des processus métier : Le langage BPEL

Plusieurs langages ont été proposés dans la perspective de mise en oeuvre des compositions de services. Nous citons le plus important WS-BPEL [BIM03].

BPEL (*Business Process Execution Language*) initialement connu sous le nom BPEL4WS, renommé par la suite WS-BPEL, cette spécification est plus connue sous le nom BPEL [BIM03]. Le standard BPEL est une spécification proposée conjointement par IBM, Microsoft, et BEA, et représente une convergence des idées initialement proposées par les langages XLANG [Cor02] et WSFL [Ley01]. BPEL est un effort pour standardiser la composition de services web [ACD+03, WvdAP+03]. Il est aussi un langage pour définir et gérer des activités d'un processus métier. Ce langage permet de décrire des protocoles d'interactions et de collaborations entre les services web sur lesquels s'appuie le processus. BPEL utilise le modèle de contrôle classique des flots de tâches (workflows) pour décrire des processus métiers invoquant des services web [DA07]. BPEL supporte les processus exécutables et abstraits, et ce, grâce à son mécanisme générique de spécification de concepts.

Les processus BPEL interagissent en invoquant d'autres services web. Un processus BPEL peut être exposé comme un service (i.e. l'interface d'un processus BPEL est décrite en WSDL) et recevoir donc des invocations d'autres services web [Pap03]. La relation entre le processus et un partenaire est une relation pair-à-pair. Un partenaire peut être le consommateur d'un service que le processus produit, ou le producteur d'un service que le processus consomme. BPEL utilise le concept de *partner links* pour modéliser les liens avec les partenaires (i.e. les collaborations pair-à-pair), et ce, en se basant sur les *portType* des services web. Une instance du processus est créée à la réception de messages ou événements de la part d'un consommateur. BPEL utilise la notion de *endpoint reference* pour sélectionner dynamiquement des services à invoquer pour des tâches spécifiques. Un système de corrélation est employé par BPEL afin de corréler entre plusieurs instances d'un même processus.

Le processus dans BPEL est constitué d'activités et d'un flot de contrôle. Les activités peuvent être primitives ou structurées (i.e. pouvant se décomposer encore en plusieurs sous activités). BPEL a les caractéristiques d'un langage structuré en blocs (caractéristique issue de XLANG), ainsi que celles d'un flot de tâches (caractéristique issue de WSFL). WS-Coordination [NRFJ07] et WS-transaction [WS-08] peuvent être utilisés pour étendre un processus BPEL.

Des règles de correspondance entre BPMN et BPEL existent. Cependant, BPMN n'a pas de format d'échange. On ne peut donc pas exporter les diagrammes BPMN dans un outil utilisant BPEL.

2.4 Le traitement des événements complexes (CEP)

Selon le glossaire [LS08] publié en 2008 par l'EPTS «*Event Process Technical Society*», le traitement des événements complexes (CEP) «*Complex Event Processing*» concerne tout calcul ou traitement exécutant des opérations sur des événements complexes telles que la création, la lecture, la transformation et l'abstraction des données portées par ces événements. En d'autres termes, CEP traite les concepts d'analyse d'événements porteurs d'informations prévenant de diverses sources de données [Luc11].

"Le CEP se focalise, quant à lui, sur l'extraction d'informations plus complexes dans des environnements générant des flux, tels que la détection de patrons d'événements complexes." [Gas08]

Le but de CEP, en tant que concept, est de répondre à tout besoin exigeant en termes de traçabilité, réactivité, disponibilité et prise de décision. La complexité de mise en œuvre réside dans la définition d'algorithmes performants permettant le calcul et la corrélation d'événements ainsi que de processus et règles métiers optimisés [Luc02a].

2.4.1 Architecture orientée événements (EDA)

Le concept d'architecture orientée événements (EDA) «*Event Driven Architecture*» reprend les idées de l'orienté services. Par opposition à l'architecture SOA où un fournisseur rend un service à la demande d'un consommateur, en architecture EDA, un service prévient par émission d'un événement qu'il vient de réaliser une opération donnée. Ensuite, c'est aux clients potentiels (i.e. consommateurs d'événements) de traiter cet événement.

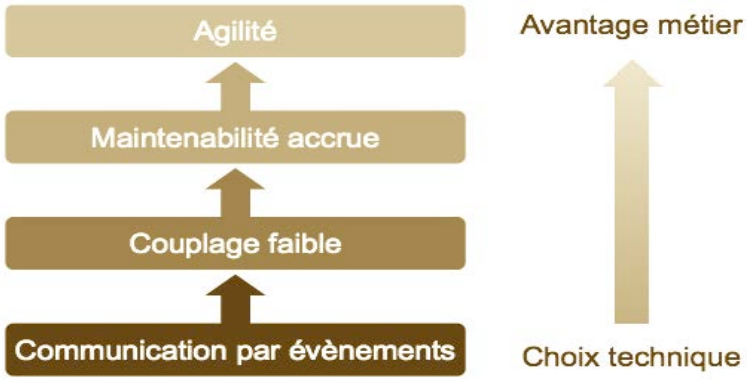


FIGURE 2.7 – Avantages d'une architecture orientée événements (EDA)

EDA est avant tout une forme d'architecture de médiation avec une forte cohérence interne par l'utilisation d'un format d'échange pivot, le plus souvent XML. Cette architecture représente aussi un modèle d'interaction applicative avec des couplages externes lâches par l'utilisation d'événements (i.e. envoi asynchrone à la place des requêtes / réponses). Ces avantages permettent d'aboutir à une maintenabilité accrue et meilleure agilité (cf. *figure* 2.7). En effet, les composants d'un système informatique, bien qu'interagissant, doivent rester isolés le plus possible les uns des autres. L'orienté événements propose de limiter la communication entre composants à la diffusion de messages asynchrones et sans destinataires, appelés événements et représentant un changement d'état du système [Zim10]. Les événements sont ensuite détectés par les autres composants, qui prennent alors des actions en conséquence. La nécessité de devoir traiter un grand nombre d'événements et d'y reconnaître des motifs (i.e. des patrons d'événements) conduit à l'utilisation de composants spécialisés comme les processeurs CEP qui permettent le traitement massif d'événements complexes.

2.4.2 Terminologie et concepts de base

Événement La notion d'événement dans CEP est identique à celle de l'EDA. Un événement est l'enregistrement d'un fait. Sous sa forme la plus simple, il indique le moment et le lieu du fait, accompagné de données connexes [TIB06]. Par exemple, l'enregistrement d'une nouvelle commande d'un client dans un système de ventes peut être considéré comme événement.

Plus formellement, un événement est défini comme un changement d'état d'un système donné (i.e. variation d'un des paramètres définissant l'état d'un système). Il est associé à un *timestamp* qui définit le temps de son occurrence [Luc08].

Événement dérivé, composé ou complexe Un événement complexe est l'abstraction d'un ensemble d'événements dits membres. Par exemple, un événement complexe «*La commande d'un client C a été finie*» peut avoir comme membres «*Début de commande*», «*facture envoyée*», «*paiement accepté*» et «*livraison accomplie*». La relation entre un événement complexe et ses événements membres est appelée *agrégation*. L'occurrence de l'événement complexe se traduit par l'occurrence de tous ses événements membres.

Un événement dérivé représente, par contre, une notion plus technique dans le sens où elle est liée aux objets événements eux-mêmes. Un événement dérivé est généré à l'issue d'un traitement d'un ou plusieurs événements. Il représente tout simplement le résultat de l'application d'une règle de génération. Un événement composé, quant à lui, est une collection d'événements satisfaisant un patron (motif). Il est généralement construit en utilisant des constructeurs particuliers, comme une conjonction, disjonction, etc.

Patrons d'événement Un patron d'événement «*event pattern*» permet la détection de séquences d'événements liés par des relations temporelles, booléennes, de similarité, d'indépendance ou de causalité. Lorsqu'une séquence d'événements correspond à un patron, elle constitue alors une instance de ce patron. Comme exemple de patrons, nous pouvons citer :

- tous les événements A1 ayant l'argument *arg1* supérieur à 300.
- chaque événement A1 suivi d'un événement A2 avec l'argument *arg3* de A1 égale à l'argument *arg2* de A2.
- chaque trois événements A1 espacés de moins de 3 secondes.

Causalité La relation de causalité entre deux événements A et B se détermine soit par inférence, soit par le contexte liant les deux événements. Dans le cas où un événement A a causé un événement B, aucune horloge ne peut attribuer à B un instant d'occurrence plus petit que celui attribué à A.

Traitement des requêtes en continu Un système CEP est capable de gérer un très grand nombre de requêtes extrêmement complexes. Il prend en compte le fait que les données à traiter (i.e. les événements) sont vues comme un flux qui entre et sort en permanence. Ceci ressemble à

l'interrogation des bases de données mais avec une inversion de la logique. En effet, les requêtes changent dans une base de données alors que les données sont stockées de façon permanente et changent rarement. Dans un système CEP, c'est l'inverse : les données changent fréquemment et les requêtes sont généralement statiques et fixées à l'avance (cf. *figure* 2.8).

"There is a paradigm shift between procedural code and databases, when compared to continuous processing. In a traditional database, the data is (relatively) static and the queries are dynamic. An event processing system is the dual of that. In an event processing system, the queries are static and the data is dynamic." [Win10]

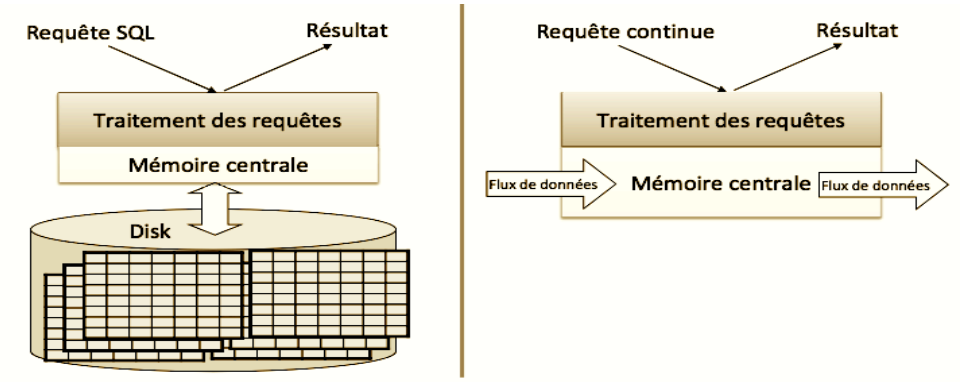


FIGURE 2.8 – Différence entre un système de bases de données (à gauche) et un système CEP (à droite) [Gab11]

Cette similarité est à l'origine du fait que la plupart des langages de traitement d'événement, appelés aussi langages EPL «*Event Processing Languages*», ont une syntaxe ressemblante à celle du langage SQL utilisé dans les bases de données.

Traitement de flux d'événements (ESP) Un flux d'événements «*Event Stream*» est une séquence linéaire d'événements ordonnée par rapport à un axe temps. Cette séquence peut être bornée et contenir des événements de différents types.

Nuage d'événements Un nuage d'événements «*Event Cloud*» est une séquence d'événements partiellement ordonnée dans le temps. La séquence est généralement engendrée par des relations (entre les événements) autres que le temps. Typiquement, un nuage d'événements est créé par des événements produits par une ou plusieurs sources (e.g. systèmes distribués) et peut inclure plusieurs flux d'événements. En d'autres termes, dans un *cloud*, il n'y a pas nécessairement de règles permettant d'ordonner les événements. Un flux d'événements peut être considéré comme un cas très particulier de nuage.

2.4.3 Réseaux d'agents de traitement d'événements (EPN)

Un réseaux d'agents de traitement d'événements (EPN) «*Event Processing Network*» est généralement composé de deux ou plusieurs agents permettant de réaliser des traitements sur des événements provenant d'une ou plusieurs sources et à destination d'un ou plusieurs consommateurs. Le mécanisme «*publish/subscribe*» est généralement utilisé pour permettre à chaque agent générateur (resp. consommateur) d'événements de publier (resp. se souscrire à) un ou plusieurs flux d'événements (cf. *figure* 7.4). Parmi les composants de base d'un tel réseau, nous pouvons citer :

Agent processeur un module logiciel qui effectue des traitements sur les événements. Le traitement peut inclure des opérations comme le filtrage, la corrélation (temporelle, causale ou spatiale), l'agrégation, etc.

Agent source/générateur d'événements un agent qui génère/envoie des événements. Dans le système par abonnement (*publish/subscribe*), il s'occupera de la publication. Il peut assurer d'autres fonction de pré-traitement (e.g. filtrage à la source pour éviter la surcharge).

Agent collecteur/consommateur d'événements un agent qui reçoit des événements. Il est éventuellement abonné «*subscribe*» auprès d'une ou plusieurs sources d'événements.

Canal d'événements un conduit dans lequel les événements partent d'un source à un agent cible.

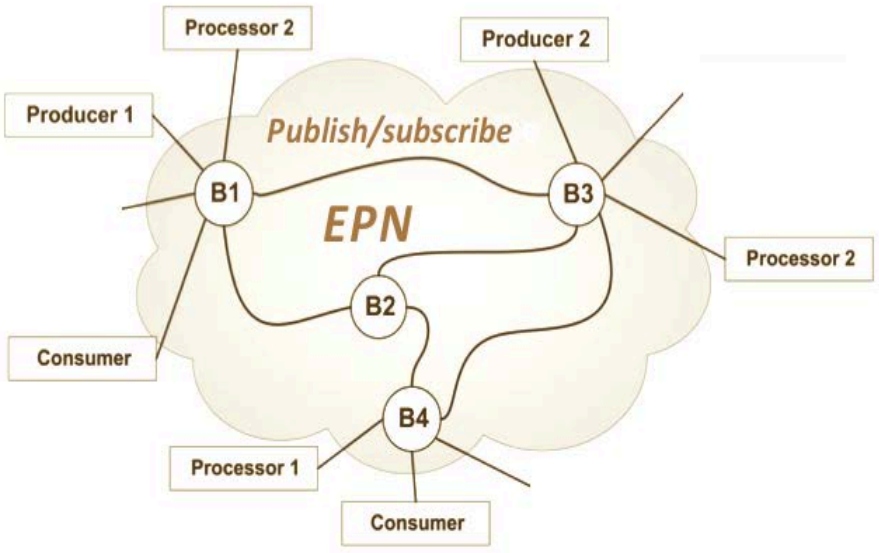


FIGURE 2.9 – Réseaux d'agents de traitement d'événements (EPN)

2.5 Vers une synergie entre SOA, BPM et CEP

Le BPM événementiel (EdBPM) «*Event-Driven Business Process Management*» est aujourd'hui une amélioration de la gestion des processus métier (BPM) par de nouveaux concepts

comme CEP et EDA. En effet, une synergie entre BPM et CEP est possible dans le sens où un processus métier peut jouer le rôle de consommateur et/ou producteur d'événements (cf. *figure* 2.10). Les événements peuvent être produits par un moteur workflow/BPM et/ou par les services IT qui sont associés aux différentes étapes des processus métiers [DGD12]. Aussi des événements provenant de différentes sources, d'un milieu interne comme externe, peuvent déclencher un processus métier ou influencer l'exécution du processus ou un service, qui peut, à son tour, entraîner lui-même un autre événement. De plus, la corrélation d'un ensemble de ces événements peut être traitée comme un événement complexe de niveau plus abstrait, agissant lui-même sur l'exécution d'autres processus ou services.

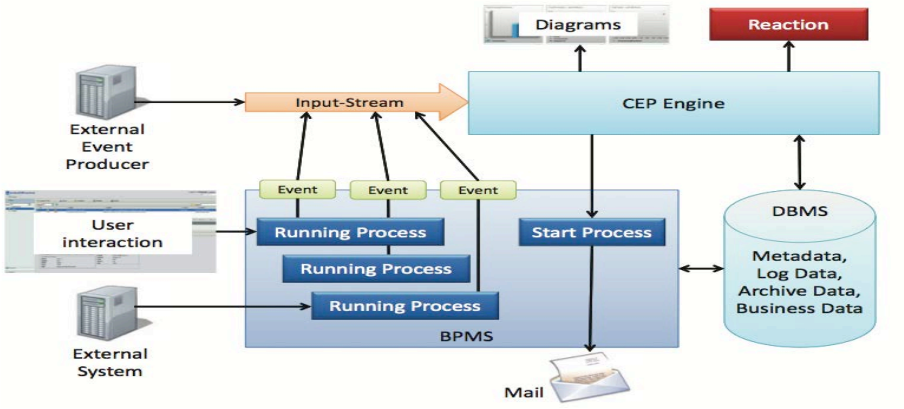


FIGURE 2.10 – Architecture d'un environnement BPM-CEP [DGD12]

"Business event processing for the right-now business is a 21st-century development, and it is growing fast. It is a technology aimed at enabling an enterprise to take action right now, the instant information become available. " [Luc11]

Comme nous avons mentionné précédemment, un processus métier peut être considéré (i.e. exposé et invoqué) comme un service et/ou interagir avec d'autres processus/services appartenant aux différentes entreprises et organisations. Du fait, l'EdBPM permet de combiner à la fois les avantages de SOA, BPM et CEP, et surtout de ce que peut apporter le couplage lâche de l'architecture événementielle :

Réactivité Les événements peuvent se produire à n'importe quel moment à partir de n'importe quelle source et les processus peuvent y répondre immédiatement et à chaque fois qu'ils se produisent.

Agilité De nouveaux processus peuvent être modélisés, mis en œuvre, déployés, et optimisés plus rapidement et en réponse à l'évolution des besoins de l'entreprise.

Flexibilité Les processus peuvent s'étendre sur des plates-formes hétérogènes. Les applications utilisées peuvent être améliorées ou modifiées sans changer le modèle de processus.

2.6 Synthèse

Dans ce chapitre, nous avons introduit les architectures orientées services et leur rôle pour améliorer la compétitivité et l'agilité de l'entreprise. Nous avons aussi présenté un état de l'art sur les différents standards pour la conception et la gestion des processus métiers collaboratifs, et nous avons décrit les différentes approches pour la composition des services, à savoir, l'orchestration et la chorégraphie. Dans ce contexte, nous avons montré comment une plate-forme logicielle BPM permet d'offrir aux entreprises la possibilité de modéliser, gérer et optimiser des processus métier pour un gain significatif. Enfin, nous avons survolé les principaux concepts liés aux architectures orientées événements (EDA) et montré comment elles peuvent compléter les architectures SOA et BPM en offrant un couplage très lâche entre les services. Nous avons aussi montré l'importance des processeurs CEP et comment ils peuvent détecter des patrons/motifs d'événements complexes diffusés massivement dans un système afin d'évaluer avec précision son état.

Le chapitre suivant est dédié aux principaux travaux qui portent sur la modélisation, l'analyse, la sécurité et le suivi des compositions des services web. Nous allons nous intéresser plus particulièrement aux chorégraphies inter-organisationnelles. Nous allons discuter des différentes approches qui traitent le sujet de la supervision des processus métiers inter-organisationnels et la vérification de la conformité des chorégraphies durant la phase d'exécution. Nous enchainons par une étude comparative et les apports de notre travail par rapport à ces approches.

3 Supervision des compositions de services

«It takes imagination to see the world as it is, to understand people as they are, to perceive abstract relations, to find analogies, to view a single truth from many angles, to sort out the essential from the inessential.»

Joseph Sobran

Sommaire

Introduction

L'objectif de ce chapitre est d'explorer les travaux similaires à nos contributions. Nous mettons l'accent, d'une part, sur les différents langages de modélisation des chorégraphies de services

et, d'autre part, sur les différentes approches qui traitent les aspects sécurité, contrôle d'accès et suivi d'exécution des processus métier.

La première partie est consacrée à la présentation des travaux qui portent sur la modélisation et l'analyse des compositions de services. (cf. section 3.1). Dans la deuxième partie de ce chapitre (cf. section 3.2), nous nous focalisons sur les chorégraphies de services. Dans un premier temps, nous présentons un ensemble de langages qui permettent la modélisation des chorégraphies de services, à savoir *BPEL4Chor, WS-CDL, WSCI* et *Let's Dance*. Nous cherchons à faire le lien avec les orchestrations les langages exécutables comme WS-BPEL. Ensuite, nous nous intéressons aux diagrammes d'interactions de la notation BPMN 2.0 que nous allons utiliser dans notre approche. Enfin, nous présentons la règle de réalisabilité que chaque modèle de chorégraphie doit satisfaire.

La troisième partie est consacrée à la présentation des travaux qui gravitent autour de la sécurité et des contrôles d'accès des processus métier (cf. section 3.3). Dans la quatrième partie (cf. section 3.4), nous décrivons les travaux similaires à nos contributions. Par une analyse critique, nous montrons les intérêts et les limites respectifs ainsi que les aspects complémentaires des différents travaux liés. Finalement, la section 3.5 conclut ce chapitre.

3.1 Modélisation et analyse des compositions de services

La modélisation et l'analyse des compositions de services représente un des axes de recherche importants dans le domaine des processus collaboratifs inter et intra-organisationnels [FBD+11]. Une des premières propositions pour l'analyse formelle des réalisations de compositions de services Web a été entreprise par [Nak02]. Dans ce travail, l'auteur voit qu'en raison de la nature des compositions étant déployées sur Internet, l'effet d'une erreur dans une telle composition est beaucoup plus grand que celui dans les déploiements dans les systèmes conventionnels. Il a également proposé des analyses de compositions spécifiées dans le langage de flux de service Web WSFL «*Web Service Flow Language*»[Ley01]. Il effectue, ensuite, une transformation, du langage WSFL vers un autre langage appelé *Promela* (le langage de l'outil SPIN) [Nak02]. Le travail représente ainsi une référence utile pour la transformation des schémas XML vu que ces derniers sont utilisés pour définir la quasi-majorité des spécifications qui tournent autour des services Web.

Parmis les premiers travaux de transformation de WS-BPEL, nous pouvons citer celui dans [KvB04]. L'auteur présente un *calcul de processus* étendu, appelé *BPEL-Calculus* qui vise à décrire, avec concision, des processus BPEL avec une notation semblable à celle du *calcul de processus* CCS [Mil80]. Le processus résultant est ensuite traduit dans un système de transition labellisé (LTS). Les auteurs de *BPEL-Calculus* montrent les inconvénients des autres méthodes de modélisation de BPEL à savoir la difficulté de tracer les résultats rapportés dans l'outil final. En effet, ce dernier point est très utile pour la vérification formelle des compositions.

Dans [Fer04], les spécifications des services Web ont été décrites dans un langage de spécifications temporelles ordonnées (LOTOS). Les auteurs étendent la transformation ordinaire entre

l'algèbre et WS-BPEL en proposant des règles pour des processus bidirectionnels. Ils affirment, cependant, qu'à cause de la structure flexible et expressive de LOTOS, la transformation de LOTOS vers WS-BPEL ne préserve pas la structure du processus. Il y a ainsi besoin d'inclure des ressources additionnelles permettant de remplir les lacunes entre l'algèbre de processus et les spécifications d'exécution. Alternativement, [HB03, YK04] utilisent des modèles basés sur les réseaux de Petri pour représenter les flux des compositions de services Web. Dans [HB03], les auteurs définissent également *une algèbre de service Web.*

D'autres travaux se sont focalisés sur la définition des sémantiques formelles pour les langages de composition de services Web [FRMB+12]. Dans [AA02], les auteurs décrivent les sémantiques pour DAML-S (une autre proposition pour la spécification des compositions de services Web). Ils définissent la notion d'un *service Web sémantique* comme une série de ressources Web qui fournissent des services, effectuant un certain nombre d'actions. Le Web sémantique devrait permettre aux utilisateurs de trouver, choisir, utiliser, composer, et surveiller des services Web, et ce, d'une façon automatisée. Tandis que dans [DBLL04], les processus abstraits de WS-BPEL, généralement utilisés pour la description des interfaces entre les processus, sont analysés et les sémantiques sont données par la construction des réalisations de WS-BPEL. Dans [WPSW04], les auteurs ajoutent une extension aux spécifications WSDL afin de décrire les interactions entre les services Web. Les tâches sont représentées par des processus et l'enchaînement des dépendances entre les tâches est représenté par des canaux. Puisque WS-BPEL étend WSDL avec des processus abstraits, cette transformation vise, d'avantage, le niveau chorégraphie, vu que la logique interne d'un service n'est pas observable directement.

En termes de conversations et chorégraphie de services Web, des travaux sur la communication asynchrone entre les services Web ont été décrites dans [BFHS03]. Un cadre formel des spécifications a été décrit avec objectif d'analyser les conversations produites par les canaux de communication asynchrone utilisés sur Internet. La technique proposée semble être plus utile pour modéliser la communication générale des services, plutôt que les détails de la composition. Des travaux sur la modélisation de communication asynchrone et des interactions de WS-BPEL ont été réalisés à travers des *automates à états finis gardés* (GFSA). Ces automates permettent la modélisation des dépendances entre les données par des transitions de processus. Dans [BCPV04] les auteurs décrivent une approche pour formaliser les conversations, en transformant le standard WSCI en CCS pour la description des chorégraphies de services Web. La technique ressemble à celle de formalisation des compositions, par la transformation de chacune des actions et des paramètres de données entre deux ou plusieurs partenaires. La conversation est modélisée en transformant les activités de réception et des réponse des services partenaires avec celles d'invocations de services Web. Les auteurs présentent une vue commune pour des modèles de chorégraphies. Ils insistent aussi sur le fait que les autres travaux dans ce secteur ne fournissent pas un support pour les adaptateurs de canaux permettant de lier les interactions de services ensemble.

Dans [Gue10], l'auteur présente un étude de l'impact des propriétés temporelles dans une chorégraphie dans laquelle les services Web supportent des communications asynchrones. Il propose une démarche basée sur le «*model checking*» qui permet de détecter les éventuels conflits

temporisés qui peuvent surgir dans une chorégraphie. L'approche proposée est basée sur la génération d'un médiateur pour essayer, quand c'est possible, de contourner les incompatibilités temporisées et non-temporisées qui peuvent surgir lors d'une collaboration.

Dans [KPS06], les auteurs proposent une approche pour la vérification d'un ensemble de compositions de services Web, et ce, en utilisant les systèmes de transition. L'approche proposée suppose que les services échangent les messages selon une topologie de communication pré-définie (*linkage structure*) basée sur les canaux.

Dans [FKMU03, FUMK04], les auteurs modélisent les sémantiques des processus BPEL en transformant les patrons BPEL dans une algèbre FSP et en construisant des modèles de comportement de processus. Ils ont aussi fourni un support pour la modélisation des interactions entre plusieurs compositions de services Web. La modélisation de ces interactions est importante dans le sens où elle permet de vérifier et valider la chorégraphie et voir si l'implémentation est compatible par rapport à la spécification.

La vérification des compositions de systèmes concurrents a été entreprise dans plusieurs articles. Dans la plupart de ces travaux, un modèle du comportement des systèmes concurrents est créé. Cependant, le contexte de vérification est défini de différentes façons. Par exemple, dans [HS02], le contexte consiste à vérifier l'équivalence entre les implémentations et les besoins. Dans [KvB04], les auteurs discutent la vérification des compositions pour contrôler l'existence des interblocages *deadlocks* ou des *livelocks* et ordonnancer les contraintes dans des conversations de services. Ils proposent un exemple comportant un ensemble de propriétés à vérifier dans les compositions et, plus particulièrement, les chorégraphies de services Web. Dans [BFHS03], les auteurs présentent une méthode de vérification, de la communication asynchrone, pour les modèles de conversations de services Web comportant des interactions durables et découplées entre les services. Les auteurs utilisent les dépendances entre les données dans le but de vérifier les valeurs et les chemins alternatifs possibles, de l'exécution, par une analyse de messages (i.e. utilisant les descriptions d'interfaces pour les messages échangés entre les partenaires). Dans [HB03], l'analyse consiste à décrire le processus d'affaires réel et les spécifications correspondantes.

3.2 Les chorégraphies de services : modélisation, analyse et réalisabilité

Dans cette section, nous nous intéressons au cas de chorégraphies de services. Nous commençons par décrire les langages de modélisations spécifiques aux chorégraphies. Nous nous focalisons sur les diagrammes d'interactions de la notation BPMN 2.0 que nous allons utiliser dans notre approche. Ensuite, nous présentons la règle de réalisabilité que chaque modèle de chorégraphie doit satisfaire.

3.2.1 WSCI : Web Services Conversation Language

WSCI «*Web Services Choreography Interface*» [AII+02], lui aussi basé sur XML, propose de se focaliser sur l'aspect chorégraphie de messages par la représentation des services web en tant qu'interfaces décrivant le flot de messages échangés. Il propose ainsi de décrire le comportement externe observable de chaque service. Pour cela, WSCI propose d'exprimer les dépendances temporelles et logiques entre les messages échangés à l'aide de contrôles de séquences, corrélation, gestion de fautes et transactions.

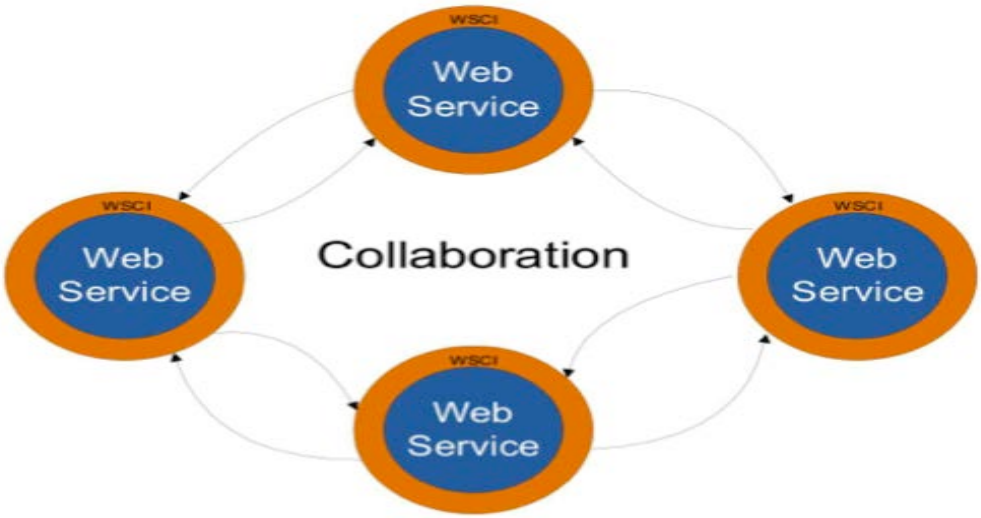


FIGURE 3.1 – WSCI : Web Services Conversation Language.

Le langage WSCI a été proposé par Sun, SAP, BEA, et Intalio, et est devenu une note dans le W3C en 2002 [AII+02]. La collaboration est coordonnée de manière décentralisée sans utiliser de processus principal, mais plutôt plusieurs processus distribués (cf. *figure* 3.1).

3.2.2 BPEL4Chor

Le langage BPEL4Chor [DKLW07] a été conçu dans le but d'enrichir WS-BPEL et de l'adapter à l'aspect chorégraphie. Il permet de fournir des extensions à BPEL pour passer d'un langage d'orchestration à un langage de chorégraphie. BPEL4Chor se compose de trois artéfacts :

- la description du comportement des participants (PBD) «*Participant Behavior Descriptions*» : une vue locale qui décrit les dépendances de flux de contrôle au niveau de chaque participant. Les PBDs sont généralement des processus abstraits (*abstract BPEL*) qui ne sont pas exécutables.
- la description du comportement de la topologie «*Topology Behavior Description*» : une vue locale qui décrit l'ensemble des participant existants et les liens en terme de messages échangés entre eux.
- la configuration technique du participant «*Participant Grounding*» : une configuration concrete des formats de données et des port-types.

3.2.3 WS-CDL : Web Services Choreography Description Language

WS-CDL [KBRL04] est un langage de description de chorégraphie. Il définit une chorégraphie comme un contrat multi-partenaires qui décrit, d'un point de vue global, le comportement commun observable des services qui participent à une collaboration [Bhi05]. En WS-CDL, seules les interactions entre des services web sont considérées. La description qui en découle prend la forme d'un processus dont l'exécution est effectuée de façon décentralisée [DA07]. La description des interactions est indépendante des environnements d'exécution des services impliqués. Une définition globale des conditions et des contraintes selon lesquelles les messages sont échangés est donnée par le biais d'un contrat que chaque partenaire doit respecter au moment de l'exécution. La *table* 3.1 montre les éléments principaux définis par le langage WS-CDL.

Elément	Description
Choreography	définition globale d'une collaboration.
Participant Types	Les entités physiques qui participent à la chorégraphie.
Role Type	le rôle que joue un participant dans une chorégraphie. Plusieurs rôles peuvent être joués par un même participant.
Channel Types	Les points de collaboration définissant où et comment les messages sont envoyés.
Basic Activities (Interactions, perform, assign)	La structure d'échange des messages qui permet de décrire une communication binaire (entre deux rôles). D'autres activités comme *perform* (utilisée pour inclure une sous-chorégraphie) et *assign* (utilisée pour l'attribution de variables) sont aussi considérées comme activités basiques).
Work-units	Les points principaux d'évolution du processus, chacun définissant les contraintes à respecter pour passer au point suivant. Généralement utilisés pour décrire des boucles (des séquences de messages qui peuvent se répéter 0 ou plusieurs fois selon une condition *guard* qui est évaluée au cours de l'exécution).
Structural Activities (Ordering Structures)	Les structures d'ordre imposées sur le séquencement des messages. Il existe trois structures de bases : *Sequence, Parallel*, et *Choice*.
Recovery, Exception, Finalizer	Structures plus avancées utilisées pour le traitement des exceptions.

Table 3.1 – Eléments principaux dans WS-CDL

Bien que BPEL et WS-CDL aient été proposés dans des contextes différents, ils partagent plusieurs points communs. En effet, ces deux propositions adoptent plusieurs concepts des systèmes de workflow pour permettre la spécification de la logique d'exécution des invocations de services. En plus, puisque BPEL est destiné à l'orchestration de services, il ne supporte que des exécutions centralisées. WS-CDL se focalise plus sur une exécution décentralisée multi-partenaires. WS-CDL n'est pas orienté processus métier dans le sens où il ne permet pas la description de processus métier exécutable comme WS-BPEL. Il vise la coordination de transactions longues en gérant le séquencement et les conditions d'échanges entre partenaires. Pour permettre de définir le séquencement des interactions selon un ordre prédéfini, WS-CDL propose quatre patrons

Patron	Description
Sequence	Une activité de chorégraphie est activée juste après la terminaison d'une autre.
Parallel	est un point dans la chorégraphie où un itinéraire unique se sépare en deux ou plusieurs itinéraires différents dans le but de réaliser deux ou plusieurs activités en parallèle.
Choice	un itinéraire s'ouvre sur plusieurs itinéraires possibles et que le cas d'exécution suit un seul de ces itinéraires.
Work-units	Répéter une interaction ou un bloc d'interaction jusqu'à ce que la condition de sortie (*guard condition*) soit évaluée à faux. La condition de répétition est évaluée à la fin de la boucle.

Table 3.2 – Patrons de séquencement de messages dans WS-CDL

de séquencement (i.e. aussi appelés structures d'ordre). La *table* 3.2 résume l'ensemble de ces patrons.

3.2.4 Let's Dance

Let's Dance [ZBDtH06] est un langage visuel qui permet la description de chorégraphies de services. Ce langage est destiné à la modélisation des interactions de services web à deux niveaux d'abstraction :

- les interactions au niveau global pour décrire le modèle global de la chorégraphie
- les interactions au niveau local pour décrire un ou plusieurs modèles locaux appartenant au modèle global.

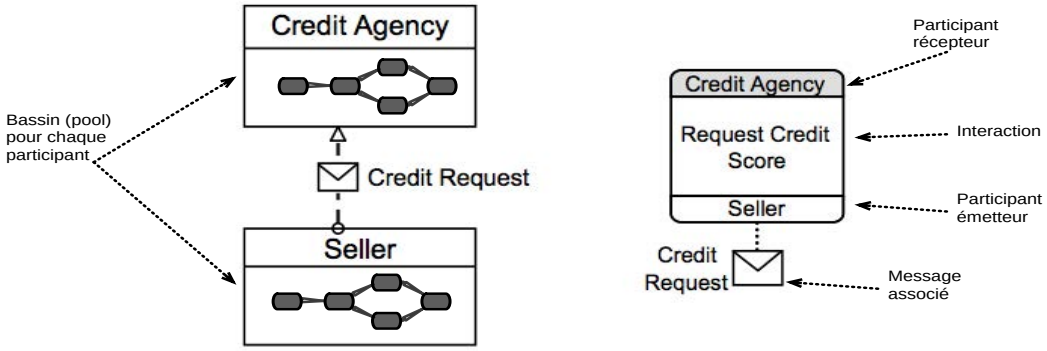
À la différence des langages comme WS-CDL ou WS-BPEL qui ont une structure de langage de programmation basée sur XML, Let's Dance est un langage dédié à l'analyse et à la conception de processus métiers. Une chorégraphie dans Let's Dance est décrite par un ensemble d'interactions qui se traduisent par des échanges de messages entre les services participants. *Maestro* est la mise en oeuvre de ce langage. Il permet la modélisation des interactions des services rentrant dans une chorégraphie et il supporte aussi l'analyse statique des modèles globaux et la génération des modèles locaux à partir du modèle global. Les interactions entre ces modèles peuvent être obtenues par simulation. Ce langage est plus expressif que WS-CDL et permet de surmonter certaines de ses insuffisances (e.g. supporter les vues locales et les interactions un à plusieurs).

3.2.5 Diagrammes d'interactions en BPMN 2.0 (collaboration et chorégraphie)

Un diagramme d'interactions en BPMN 2.0 [Mod11] consiste en une description du comportement normal des participants au sein d'un processus collaboratif, et ce, en se focalisant sur

les interactions coordonnées entre deux ou plusieurs participants. BPMN 2.0 dispose de deux diagrammes pour modéliser les interactions d'une chorégraphie : le diagramme de collaboration et le diagramme de chorégraphie. Le premier est disponible en BPMN 1.x et a été renforcé dans la deuxième version, tandis que le second est apparu dans la nouvelle version.

La *figure* 3.2 montre comment un message échangé entre deux bassins (*pools*) dans un diagramme de collaboration est modélisé sous forme d'interaction dans un diagramme de chorégraphie. Dans un diagramme de collaboration (voir *figure* 3.2.a), les participants sont représentés par des rectangles appelés bassins (*pools*). Les messages sont représentés par des enveloppes avec une étiquette contenant le nom. Le flux de messages apparaissent sous forme de flèches en pointillés indiquant la direction de chaque message.



a) Interaction dans un diagramme de collaboration **b) Interaction dans un diagramme de chorégraphie**

FIGURE 3.2 – Modélisation d'une interaction avec BPMN 2.0

Dans un diagramme de chorégraphie (voir *figure* 3.2.b), les interactions sont affichées en tant qu'activités, avec des messages qui leur sont liés par lignes en pointillés (appelées associations). Chaque interaction (qui représente l'activité de base dans une chorégraphie) est affichée sous forme d'un rectangle arrondi, contentant le nom (ou identifiant) de l'interaction ainsi que les participants sous forme de bandes. La bande en blanc indique l'émetteur du message et la bande en gris indique le récepteur.

Les messages circulent habituellement entre les participants dans un ordre particulier. Par exemple, dans un processus collaboratif d'achat de produits, le paiement est effectué avant que le produit ne soit livré. Ainsi, le message associé à l'interaction de paiement doit être envoyé avant celui de la notification sur la livraison. Dans les diagrammes, l'ordre entre les interaction est matérialisé par des flèches appelées « séquence de flux de messages » (*Message Flow Sequence*). La *figure* 3.3 illustre sur un exemple comment modéliser un ordre sur les interactions selon le type de diagramme utilisé. Dans cet exemple, un message demandant la balance de crédit est envoyé avant celui qui offre le résultat de la requête. Le participant émetteur dans la première interaction (i.e. le *Seller*) est le récepteur dans la seconde.

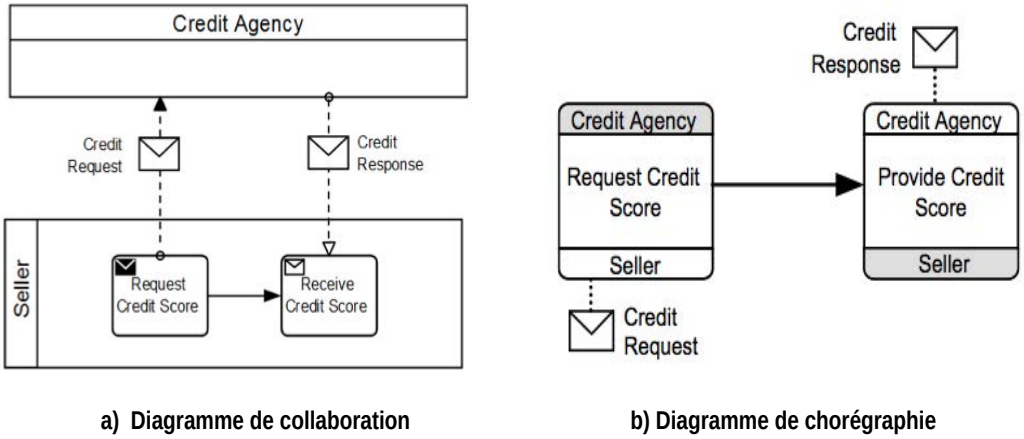


FIGURE 3.3 – Séquencement des interactions avec BPMN 2.0

3.2.6 Réalisabilité d'une chorégraphie

La règle générale qui régit le séquencement du flux des messages est appelée règle de réalisabilité. Cette règle est définie comme suit.

Définition 1 (Chorégraphie réalisable) *Une chorégraphie est dite réalisable si, et seulement si, l'expéditeur dans chaque activité (excepté la première) est également un participant dans l'activité qui la précède directement que ce soit comme expéditeur ou récepteur.*

Concernant la première activité qui n'a pas de prédécesseur, l'expéditeur de cette activité est appelé l'initiateur de la chorégraphie entière.

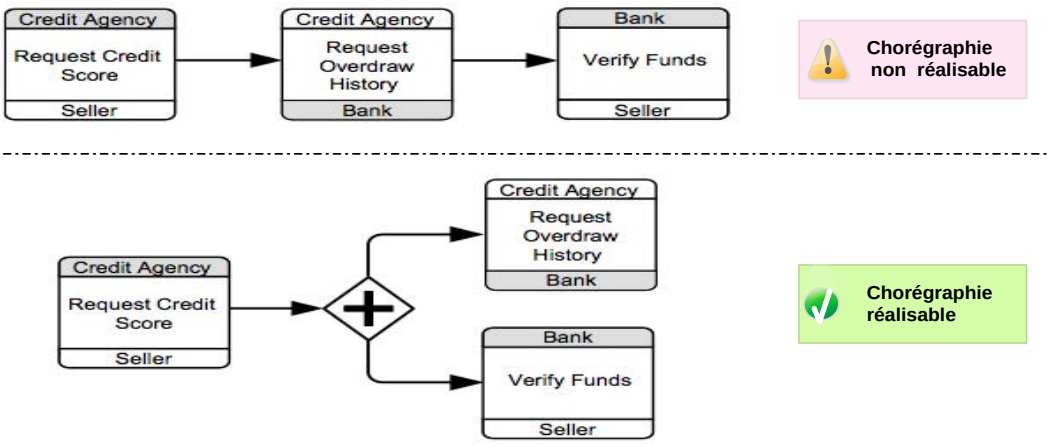


FIGURE 3.4 – Réalisabilité d'une chorégraphie

La *figure* 3.4 montre deux exemples de chorégraphies. La première n'est pas réalisable vu que l'expéditeur de la troisième interaction qui est le *Seller* n'est pas participant dans la deuxième interaction, ni comme expéditeur ni comme récepteur. Cependant, la deuxième chorégraphie est

réalisable vu que les expéditeurs des deux interactions autres que l'interaction initiale, à savoir le *Credit Agency* et le *Seller*, sont tous les deux participants dans la première.

3.3 Contrôle d'accès et sécurité des processus métier

La sécurité des processus métier, particulièrement les processus inter-organisationnels, représente un domaine de recherche très actif durant ces dernières années. Dans cette section, nous présentons quelques approches et solutions techniques qui traitent le contrôle d'accès, l'intégrité et l'exécution sécurisée des compostions de services.

BPEL et RBAC Dans [MSSN04], Mendling et al. présentent une approche qui permet d'intégrer le modèle de contrôle d'accès basé sur les rôles RBAC «*Role-Based Access Control*» dans un moteur WS-BPEL (au niveau méta-modèle). Une telle approche, qui est basée sur l'extension de WS-BPEL avec des transformations (*mappages*) vers RBAC, peut être utilisée pour automatiser les étapes d'ingénierie de rôle.

TBAC Thomas et Sandhu [TS97] ont proposé une extension de RBAC se basant sur les tâches appelée TBAC «*Task-based Authorization Control*». TBAC permet d'accorder ou de révoquer des autorisations en se basant sur le *timing* durant lequel chaque tâche a été programmée, et ce, pour permettre leur accès que pendant une durée bien déterminée. Pour ce faire, la politique d'autorisation a été étendue en ajoutant deux nouveaux champs pour activer et désactiver l'autorisation au cours de l'exécution du processus et suivant l'étape courante (i.e. l'activité en cours d'exécution). Une politique TBAC prend alors la forme suivante : $< s, o, a, usage, authorization - step >$.

OrBAC Avec l'utilisation d'un nouveau type de contrôle d'accès qui basé sur l'organisation et appelé OrBAC «*Organization based Access Control*», Ayed et al. [ACBC08] ont suggéré une approche dynamique et décentralisée pour gérer une politique de sécurité dans les workflows inter-organisationnels qui tient compte du contrôle de flux. Une règle de sécurité OrBAC prend la forme $< type, organization, role, activity, view, context >$ où *type* peut être : *permission*, *interdiction* ou *obligation*. Comme dans TBAC, les règles de sécurité ne s'appliquent pas statiquement et leur activation dépend des conditions contextuelles. L'algorithme défini dans [ACBC08] montre comment utiliser ce modèle dans un environnement WFMS distribué. Cependant, l'approche nécessite une amélioration sur comment gérer les flux d'échange entre les différentes organisations.

Génération automatique de politiques de contrôle d'accès Dans [KR09], le contrôle de flux inter-organisationnel est mis en œuvre avec le mécanisme PEP/PDP «*Policy Enforcement Point / Policy Decision Point*» . En effet, la spécification d'autorisation a été étendue et prend la forme suivante : $< policy - id, s, o, a, l_{enable}, l_{disable}, state >$. L'idée est d'activer et de désactiver les politiques à l'égard de l'étape en cours dans la chorégraphie afin de garantir l'ordre d'exécution (i.e. séquencement des activités inter-organisationnelles). Chaque organisation doit

implanter un nouveau composant appelé Générateur de politique (PGC) qui est responsable de la création d'une politique locale au cours de l'étape d'instanciation de processus (cf. *figure* 3.5). La génération dépend des modèles de processus. Un inconvénient est que le patron *parallel* a besoin de la génération de politiques supplémentaires ($n * 2^{n-1}$ politiques pour n branches parallèles). Vagts [Vag07] étend cette solution pour assurer la sécurité en cas d'exceptions.

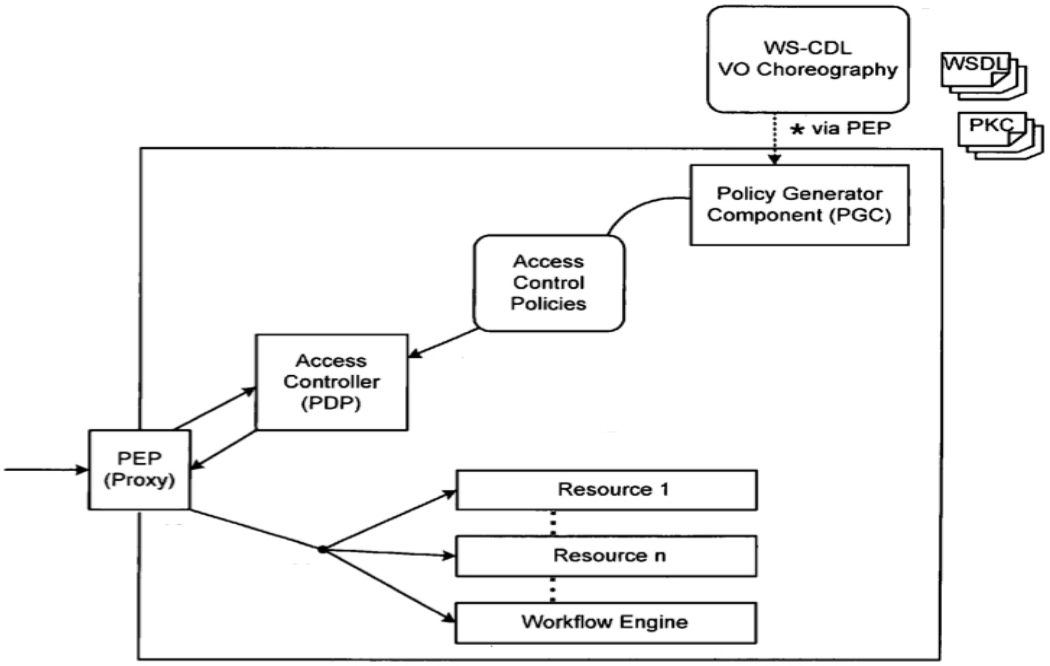


FIGURE 3.5 – Approche de dérivation de politiques de contrôle d'accès [KR09]

Cryptage en oignon Montagut et Molva [MM08] ont utilisé une technique de cryptage en oignon «*onion encryption*» pour renforcer l'intégrité de l'ordre d'exécution des processus distribués. La solution proposée permet de garantir que l'accès aux données de processus est effectué par rapport au plan d'exécution de processus et fournit des preuves d'exécution aux organisations partenaires impliquées dans le processus. Cependant, le cryptage en oignon est très gourmand en temps de calcul, du coup, cette technique risque de diminuer les performances. De plus, l'ajout excessif des clés et des signatures à chaque entête SOAP risque d'augmenter considérablement la taille du message.

Acheminement d'un conteneur La vérification de l'ordre d'exécution d'une manière sécurisée a également été considérée par Biskup et al. [BCF+07] qui ont proposé une structure de conteneur et des mécanismes d'authentification pour l'accès aux données. En d'autres termes, ils introduisent un framework pour le soutien middleware basé sur un conteneur de processus, qui fait appel à des services dédiés qui assurent des besoins non fonctionnels comme la sécurité, la persistance et l'échange fiable (*reliable messaging*). Cette approche supporte aussi le cas d'un système décentralisé. Une approche similaire est proposée dans [RKV09]. Dans ces deux approches, le conteneur est acheminé le long du chemin d'exécution du flux de contrôle afin de vérifier la

bonne exécution des chorégraphies inter-organisationnelles. Cependant, cette technique ne traite pas les exceptions qui peuvent se produire, à savoir la corruption ou la perte du message ou le conteneur lui-même.

Vérification centralisée Dans [RFG+10], des techniques de vérification et de validation pour les conversations de service Web et leurs chorégraphies ont été exposées. Néanmoins, la tâche de vérification est gérée de manière centralisée et n'est donc pas adapté aux chorégraphies inter-organisationnelles, i.e., quand il n'y a pas une unité de confiance commune entre tous les participants.

3.4 Classification des approches de supervision des compositions de services

La supervision temps réel de la composition des services a été un sujet d'intérêt pour plusieurs travaux de recherche [AFG+07, FGP10, MRD11b, BFPG12, BPG12, CCMN04, HV09]. Dans cette section, nous présentons certaines des contributions les plus pertinentes par rapport à notre travail.

3.4.1 Le BAM et les approches commerciales

Les processus métier des entreprises d'aujourd'hui utilisent principalement le traitement des événements complexes (CEP) pour suivre l'exécution d'un processus et signaler tout état incohérent de ses instances individuelles. Au cours des dix dernières années, le domaine CEP a été activement appliqué dans le milieu professionnel. Le BAM «*Business Activity Monitoring*», un concept qui s'intéresse à la supervision des processus et des activités métiers, a été l'une des applications les plus prospères où le CEP a été utilisé. Ce concept propose une exploitation rationnelle des instruments et équipements de l'informatique décisionnelle BI «*Business Intelligence*» afin d'assurer le pilotage des activités et des processus métier les plus critiques. Le BAM cherche ainsi à proposer la meilleure perception de la performance des activités et processus. Du point de vue de l'administrateur, responsable d'activités ou de processus, la solution BAM se concrétise en tableaux de bord de management composés d'indicateurs de performance KPI «*Key Performance Indicators*», rafraîchis en temps réel [19].

S'appuyant sur les KPI, la technologie BAM permet la supervision des processus métier en continu, et ce, en se basant sur la génération et le traitement des événements de contrôle . La plupart des produits logiciels commerciaux de BPM (e.g. Oracle BAM, Nimbus, Tibco, IBM Tivoli, etc.) comprennent des installations de tableau de bord BAM permettant le suivi, les rapports sur les violations des accords de niveau de service (SLA [20]), et l'affichage des résultats et des

19. Temps réel ne signifie pas instantanéité. Le temps réel définit plutôt la mise à disposition suffisamment tôt des informations afin que la décision puisse être prise en toutes connaissances de causes et sans précipitation.

20. Service Level Agreement

métriques sous forme de graphiques. Cependant, vu que les vendeurs de ses applications commerciales sont des concurrents, ces produits ne permettent de surveiller qu'une fraction prédéfinie des événements et des indicateurs de performance, généralement au sein d'une suite d'applications d'un même fournisseur. En outre, ils sont généralement limités à des processus internes qui sont sous contrôle d'une même entité administrative, c'est à dire, ne supportent pas des environnements inter-organisationnels.

3.4.2 Approches académiques centralisées

Subramanian *et al.* [STN+08] ont présenté une approche qui permet d'enrichir les moteurs BPEL en proposant un nouveau moteur dédié appelé «*SelfHealBPEL*» qui met en œuvre des fonctionnalités supplémentaires pour la détection des erreurs d'exécution et le traitement des exceptions. Cette approche peut avoir un impact négatif sur la performance et l'agilité car il n'y a pas de séparation entre la logique de supervision et le moteur BPEL.

Barbon *et al.* [BTPT06] ont proposé une architecture qui sépare la logique métier d'un service Web de celle liée à la supervision et de définir un langage qui permet de spécifier des propriétés statistiques et temporelles. Cependant, leur approche se focalise sur les orchestrations BPEL simples et ne traite pas le problème de la supervision dans un cadre inter-organisationnel.

Ardissono *et al.* [AFG+08] ont présenté un *framework* dédié au suivi de l'avancement de l'exécution d'une chorégraphie étape par étape. Le but est d'assurer la détection précoce des défauts et la notification des participants concernés des exceptions au cours de l'exécution. L'approche consiste en une entité centrale (i.e. un moniteur) qui est notifiée par chaque participant à chaque fois qu'il envoie ou reçoit un message. A partir de ces notifications, cette entité peut suivre l'évolution de l'exécution en regardant l'état actuel par rapport au précédent et comparant ce passage par rapport à celui défini dans le modèle de chorégraphie. Néanmoins, cette approche est centralisée et n'est donc pas bien adaptée aux chorégraphies inter-organisationnelles, spécialement quand il n'est pas possible de trouver une entité de confiance commune entre tous les participants.

3.4.3 Approches académiques de supervision des processus décentralisés

Dans le cas des processus décentralisés au sein de la même organisation (ou dans un cercle de confiance), Chafle *et al.* [CCMN04] ont modélisé une entité centrale en tant que moniteur d'état qui est implanté comme étant un service Web. Sur chaque partition, un agent local de supervision capture l'état local de la partition de service composite et met à jour périodiquement le moniteur d'état centralisé. Le moniteur d'état maintient ainsi l'état de toutes les activités. Cependant, sous des charges élevées, le maintien de l'état global à un seul endroit peut devenir un goulot d'étranglement. Dans [HV09], les auteurs introduisent le concept de moniteur MBM «*Monitor-based Messenger*», qui traite les messages échangés au cours d'exécution. Chaque moniteur affranchit avec une sorte de timbres tous les messages sortants de son organisation avec l'état actuel de l'exécution pour éviter les dé-synchronisations, fournir un ordre total sur les

messages, et offrir une protection en terme de fiabilité de messagerie (*reliable messaging*).

En ce qui concerne les perspectives centrées sur les événements, les solutions de supervision des processus sont principalement basées sur la technologie BAM, et ce, uniquement dans un cadre intra-organisationnel [DWC11]. Au meilleur de notre connaissance, seules deux approches événementielles s'intéressent à la supervision des chorégraphies inter-organisationnelles. La première [KSK07] utilise un format d'audit commun qui permet le traitement et la corrélation d'événements à travers un ensemble de moteurs BPEL différents. La seconde approche [WKK+10] introduit une spécification d'événement complexe et utilise un identificateur d'instance chorégraphie (CIID) pour permettre la corrélation des événements (ce qui n'est pas pris en charge dans [KSK07]). L'inconvénient de cette approche est qu'elle est spécifique au langage BPEL4Chor. De plus, elle se base sur le mécanisme «publish/subscribe» qui doit être centralisé dans unité de confiance.

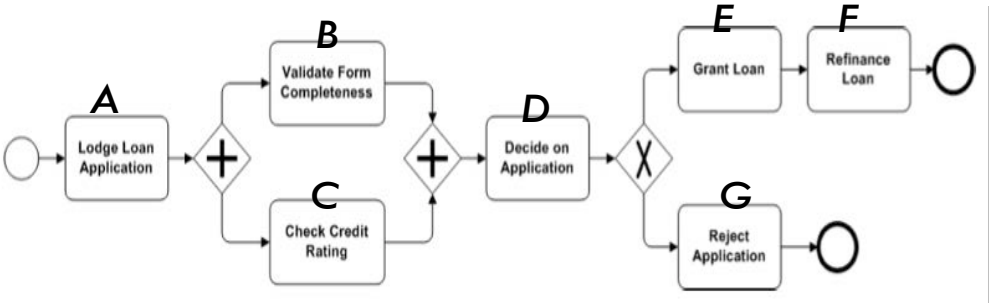
Notre approche de supervision permet le suivi d'une exécution décentralisée d'une chorégraphie de services déployée à travers les frontières organisationnelles. Contrairement aux approches précédemment citées, nous nous sommes plutôt concentrés sur comment fournir un mécanisme automatisé et décentralisé qui permet l'échange de notifications entre les partenaires concernés, et ce, selon des règles spécifiques qui peuvent être dérivées automatiquement à partir de la spécification de la chorégraphie. Nous nous focalisons aussi sur comment optimiser l'ensemble notifications échangées et de règles générées afin d'améliorer les performances. Nous insistons à ce que notre approche soit réalisable et mise en oeuvre dans un environnement CEP pour permettre un bon passage à l'échelle.

3.4.4 Approche événementielle pour la vérification de comportement

Weidlich *et al.* [WZM+11] propose une technique formelle pour générer, à partir d'un modèle de processus, des requêtes CEP qui, une fois exécutées, permettent de détecter les comportements anormaux au cours de l'exécution d'un processus métier. Pour ce faire, ils utilisent le concept de profil de causalité comportemental «*Behavioral Profile*» défini dans [WPMW10]. Les auteurs proposent de générer une règle d'ordre entre chaque couple d'activités d'un processus métier. La *figure* 3.6 montre comment générer l'ensemble de règles d'ordre dans un tableau pour un exemple de processus métier composé de 7 activités A,B,C,D,E,F et G. Quatre types de relations binaires entre deux activités ont été définies :

- «$a_1 \rightarrow a_2$» désigne que a_1 doit précéder a_2
- «$a_1 \leftarrow a_2$» désigne que a_1 doit succéder à a_2
- «$a_1 + a_2$» désigne que a_1 et a_2 sont en exclusion mutuelle. Dans une même instance, on ne peut pas exécuter les deux activités (ou bien l'une ou bien l'autre).
- «$a_1 \,||\, a_2$» désigne qu'il n'y a pas d'ordre entre a_1 et a_2. L'une peut s'exécuter avant l'autre.

Remarque : Pour indiquer qu'une même activité peut se répéter plusieurs fois les auteurs utilisent la relation «$a_1 \,||\, a_1$». Dans le cas inverse, lorsqu'une répétition n'est pas tolérée, on utilise l'exclusion mutuelle sur la même activité «$a_1 + a_1$».



	A	B	C	D	E	F	G
A	+	→	→	→	→	→	→
B	←	+	||	→	→	→	→
C	←	||	+	→	→	→	→
D	←	←	←	+	→	→	→
E	←	←	←	←	||	→	+
F	←	←	←	←	←	+	+
G	←	←	←	←	+	+	+

FIGURE 3.6 – L'approche de profil comportemental «*Behavioral Profile*»

L'approche de supervision basée sur les profils comportementaux permet de générer une requête CEP pour chacune des règles binaires. En effet, la requête CEP correspond à l'inverse de la règle correspondante (appelée aussi anti-patron). Par exemple, si on a une règle qui impose que deux activités A et B soient en exclusivité mutuelle, la requête CEP correspondante permet de chercher tout couple d'activités A et B appartenant à la même instance d'un processus. Cette méthode permet donc de détecter tout comportement qui n'est pas conforme au modèle prédéfini du processus métier.

Limites de l'approche

Nous pouvons remarquer sur le tableau de la *figure* 3.6 que l'ensemble de règles est loin d'être optimal. En effet, il y a plusieurs règles qui se chevauchent et d'autres qui ne sont pas nécessaires vu qu'elles peuvent être inférées (i.e. déduites indirectement à partir d'autres règles). Si on a deux règles «A $\rightarrow$ B» et «B $\rightarrow$ D» il n'est pas nécessaire d'ajouter une troisième règle «A $\rightarrow$ D». Cette dernière peut être déduite par transitivité. De plus, lorsqu'une de ces règles est violée, nous pouvons avoir toute une série de répercussions sur d'autres règles indirectement dépendantes à la violation d'origine. Ainsi, des requêtes supplémentaires doivent être ajoutées afin d'identifier la cause fondamentale de l'ensemble des infractions.

Au lieu de fixer une contrainte entre chaque couple d'interaction, notre approche consiste à fixer des contraintes seulement entre les blocs voisins d'interactions. L'idée est trouver comment dégager, à partir d'un modèle de chorégraphie, les relations de voisinage entre les interactions.

Ceci permettra d'optimiser l'ensemble de règles de supervision, et donc de réduire le nombre de requêtes CEP à générer. De plus, le nombre de répercussions de chaque violation détectée sera considérablement réduit.

3.5 Conclusion

Dans ce chapitre, nous avons proposé un état de l'art sur les différentes approches de modélisation et d'analyse des compositions de services. Ensuite, nous nous sommes focalisés sur l'aspect chorégraphie de services en présentant les différentes langages de modélisation proposées pour la conception des processus métiers collaboratifs. Nous avons commencé par décrire *BPEL4Chor, WS-CDL, WSCI* et *Let's Dance*. Nous nous sommes ensuite focalisés sur les diagrammes d'interactions de la notation BPMN 2.0 que nous allons utiliser dans notre approche et nous avons présenté la règle de réalisabilité que chaque modèle de chorégraphie doit satisfaire.

Nous avons enchaîné par une synthèse sur les approches proposées pour traiter les problèmes de contrôle d'accès et de sécurité dans le domaine des processus métier. Nous avons ainsi remarqué que le problème de sécurité devient plus délicat lorsqu'un processus est étendu sur plusieurs organisation administrativement indépendantes. Des mécanismes de supervision, d'authentification, d'autorisation et gestion de confiance deviennent désormais nécessaires.

La dernière section de ce chapitre a été consacrée à la classification des différentes approches qui gravitent autour de la supervision des compositions de services. Nous avons commencé par décrire les approches commerciales qui utilisent généralement les outils de BAM. Ensuite, nous avons présenté les approches académiques s'appuyant sur des mécanismes centralisés, décentralisés et/ou événementielles. Nous avons plus particulièrement détaillé l'approche basée sur les profils comportementaux et nous avons montré les points faibles qui peuvent être améliorés.

Dans les chapitres suivants, nous allons commencer par présenter un modèle formel et architectural. Ensuite, nous exposons nos contributions sur la supervision des chorégraphies de services web.

Deuxième partie

Contributions

4 Modèle formel et architectural

«La science ne connaît qu'une loi : la contribution scientifique.»

Bertolt Brecht

Sommaire

Introduction

L'objectif de ce chapitre est d'explorer les chorégraphies inter-organisationnelles d'une manière indépendante des langages de spécification et d'offrir un nouveau modèle conceptuel en proposant une architecture générale pour la supervision des collaborations. Nous mettons l'accent, d'une part, sur le besoin d'une méthode agile et flexible qui permet d'assurer la conformité de la séquence des interactions inter-organisationnelles avec un plan prédéfini sous forme de modèle de chorégraphie et, d'autre part, sur le besoin d'un mécanisme décentralisé, dynamique et

efficace qui permet l'échange des données de supervision entre les partenaires tout en assurant la traçabilité d'exécution du processus global.

La section 4.1 propose un modèle formel général, simple et compréhensible qui va permettre la mise en œuvre de notre approche à partir de tout type de langage de spécification. La section 4.2 introduit la notion de vérification événementielle dans le cadre des chorégraphies de services. Ensuite, la section 4.3 décrit l'architecture générale de notre contribution. Finalement, la section 4.4 conclut ce chapitre.

4.1 Modélisation formelle

Notre approche vise à fournir un mécanisme simplifié qui garantit la conformité du flux inter-organisationnel par rapport au schéma de la chorégraphie. Avant de présenter l'approche, nous commençons par modéliser une chorégraphie comme un ensemble d'interactions et de patrons de séquencement. Dans cette section, nous proposons un modèle formel général indépendant de tout langage de chorégraphie, simple et compréhensible et permettant l'application de notre approche à partir de tout type de langage de spécification.

4.1.1 Vue globale d'une chorégraphie

Une chorégraphie définit les règles réutilisables et communes qui régissent l'ordre des messages échangés, et les patrons de provisionnement du comportement coopératif, tel que convenu entre deux ou plusieurs participants qui interagissent [KBR+05]. Dans ce manuscrit, nous considérons une chorégraphie comme une description des séquences admissibles d'envoi ou de réception de messages pour chacune des parties participantes. Notre approche est centrée sur le comportement global de la chorégraphie. Indépendamment de ce qui se passe à l'intérieur de chaque participant (i.e. processus et activités internes), nous regardons uniquement l'aspect collaboratif (i.e. interface d'échange ou processus abstrait de chaque participant). Durant toute notre démarche, nous nous intéressons qu'aux quatre patrons de séquencement « séquence », « parallèle », « exclusivité » et « itération » ainsi que les activités d'interaction (i.e. activités d'échange de messages). En d'autres termes, les activités internes (e.g. les activités silencieuses, les activités d'allocation et d'affectation de variables locales ou globales) sont hors de la portée du présent document car elles ne génèrent pas d'échanges de messages. Nous utilisons « participant », « partie » et « partenaire » de façon interchangeable. Nous formalisons la sémantique d'une chorégraphie comme suit.

Définition 2 (Chorégraphie : Vue globale) *Formellement, une chorégraphie $\mathcal{C}$ est un tuple ($\mathcal{P}$, $\mathcal{I}$, $\mathcal{O}$) avec*

- *$\mathcal{P}$ un ensemble fini de participants,*
- *$\mathcal{I}$ un ensemble fini d'interactions,*
- *$\mathcal{O}$ un ensemble fini de structures d'ordre partiel définissant des contraintes sur le séquencement des interactions.*

La vue globale d'une chorégraphie est une vue de haut niveau de la conversation entre les participants, et peut être considérée comme étant interprétée d'un point de vue d'observateur.

4.1.2 Interaction

Une interaction est la brique de base d'une chorégraphie. Elle se traduit par un échange de message entre les parties. Chaque interaction $I \in \mathcal{I}$ correspond à un certain type de message (e.g. XML Schema) et est associée à un sens de communication, c'est-à-dire une source et une destination du message échangé. Soit $\mathcal{M}_\mathcal{T}$ l'ensemble des types de messages. Formellement, une interaction est définie comme suit.

Définition 3 (Interaction) *Une interaction $I \in \mathcal{I}$ est un tuple (Iid,s,d,m_t) avec :*
- *Iid un identifiant unique de l'interaction,*
- *s,d $\in \mathcal{P}$ respectivement la source et la destination du message associé,*
- *$m_t \in \mathcal{M}_\mathcal{T}$ le type de message associé.*

4.1.3 Contraintes et patrons de séquencement des interactions

L'ensemble $\mathcal{O}$ des contraintes sur le séquencement des interactions permet de régir l'ordre des échanges de messages suivant un schéma voulu et agréé par tous les participants durant la phase de conception. En effet, le séquencement des interactions est généralement capturé par quatre types de structures d'ordre : «Séquence», «Exclusivité», «Parallèle» et «Itération». Ces dernières sont dites «patrons de base» à partir desquels nous pourrons établir d'autres structures plus complexes. La *figure* 4.1 illustre la notation en BPMN 2.0 de ces quatre patrons.

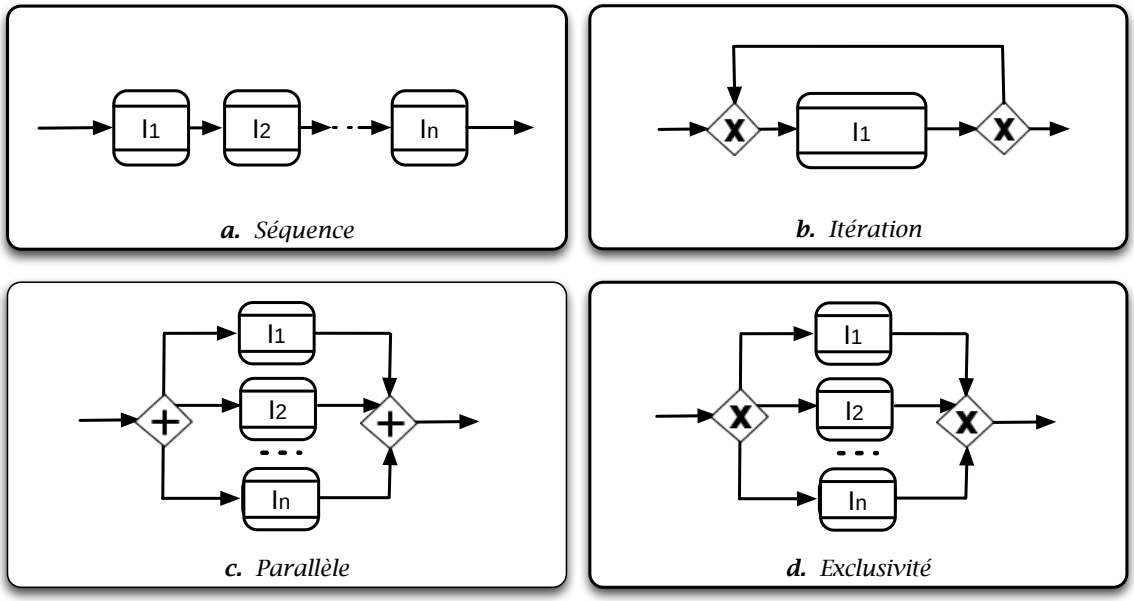


FIGURE 4.1 – Patrons de séquencement des interactions en BPMN 2.0

Séquence Cette structure impose une contrainte d'ordre strict entre les interactions. Elle indique qu'un ensemble de messages doit s'échanger séquentiellement l'un après l'accomplissement de l'autre. La notation en BPMN 2.0 est montrée dans la *figure* 4.1.a.

Nous utilisons la notation $Seq(I_1, I_2)$, pour dire qu'une interaction I_1 doit se produire avant I_2 et que I_2 ne doit jamais se produire avant I_1.

Exclusivité Cette structure permet de spécifier qu'un seul parmi deux ou plusieurs sous-ensembles d'interactions peut être choisi. Le choix se fait par l'un des participants selon les valeurs d'une ou plusieurs variables locales ou globales. La notation en BPMN 2.0 est montrée dans la *figure* 4.1.d. La contrainte d'exclusivité entre deux interactions, notée $Ex(I_1, I_2)$, impose le fait que I_1 et I_2 ne peuvent jamais co-exister dans la même instance d'une chorégraphie.

Parallèle Cette structure indique une contrainte de co-occurrence permettant de spécifier l'activation de plusieurs interactions en même temps. La notation en BPMN 2.0 est montrée dans la *figure* 4.1.c. La contrainte de co-occurrence entre deux interactions, notée par la fonction $And(I_1, I_2)$, impose le fait que I_1 et I_2 doivent toujours co-exister dans la même instance d'une chorégraphie.

Boucle/Itération Cette structure décrit une exécution conditionnelle et répétée d'une ou plusieurs interactions. L'ensemble d'itérations à l'intérieur d'un bloc d'itération peut se répéter autant de fois qu'une condition est vérifiée. Les variables évaluées dans cette condition peuvent dépendre des informations contenues dans les messages échangés et/ou d'une configuration locale.

La contrainte d'itération d'une interaction, notée $Loop(I_1)$, impose le fait que I_1 peut se répéter plusieurs fois dans la même instance d'une chorégraphie. La notation en BPMN 2.0 est montrée dans la *figure* 4.1.b.

Exemple 1 *La figure 4.2 montre la vue globale de la chorégraphie de notre exemple (cf. figure 1.1) en utilisant le diagramme d'interaction de la notation BPMN 2.0. Nous avons dans cet exemple 4 participants* $R, F, T1, T2 \in \mathcal{P}$ *et 6 interactions* $I_1, I_{2a}, I_{2b}, I_{2c}, I_{3a}, I_{3b}, \in \mathcal{I}$. *Chaque interaction est définie par un identifiant, une source, une destination et un type de message. Par exemple, l'interaction* 1 *est définie par* $(1, R, F, 1)$.

Concernant les contraintes sur le séquencement des interactions, l'ensemble de relations binaires est défini comme suit : $\mathcal{O} = \{\ Seq(I_1, I_{2a}),\ Seq(I_1, I_{2b}),\ Seq(I_1, I_{2c}),\ Seq(I_{2a}, I_{3a}),\ Seq(I_{2b}, I_{3b}), Seq(I_1, I_{3a}),\ Seq(I_1, I_{3b}),\ Ex(I_{2a}, I_{2b}),\ Ex(I_{2a}, I_{3b}),\ Ex(I_{2a}, I_{2c}),\ Ex(I_{3a}, I_{2b}),\ Ex(I_{3a}, I_{3b}),\ Ex(I_{3a}, I_{2c}),\ Ex(I_{2b}, I_{2c}),\ Ex(I_{3b}, I_{2c})\ \}$.

4.1.4 Vue locale d'un participant

La vue locale est dérivée de la vue globale et spécifie le comportement pertinent du point de vue de chaque participant. Pour chaque chorégraphie, nous avons une seule vue globale et

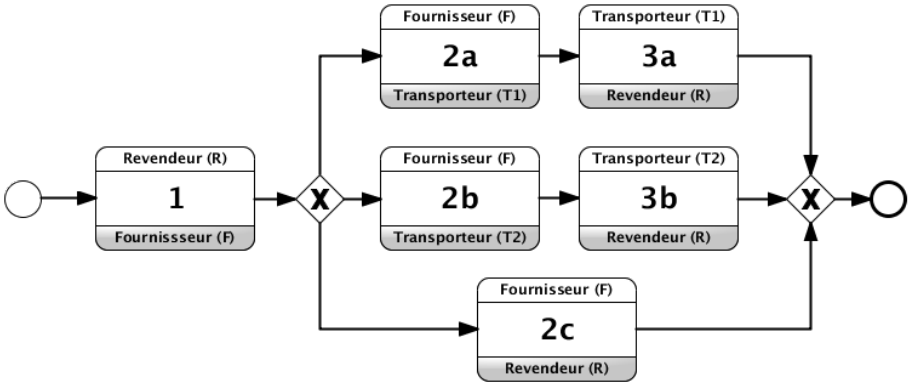


FIGURE 4.2 – Vue globale d'une chorégraphie (Diagramme d'interaction BPMN2.0).

une vue locale par participant et donc autant de vues locales que le nombre de participants. En effet, chaque vue locale inclut uniquement les interactions dont le participant concerné joue un rôle (i.e. émetteur ou récepteur). Elle comprend également l'ensemble des contraintes sur le séquencement de ce sous-ensemble d'interactions. Formellement, nous définissons une vue locale comme suit.

Définition 4 (Chorégraphie : Vue locale) *Une vue locale $\mathcal{C}_i$ d'un participant P_i est un tuple ($\mathcal{I}_i$, $\mathcal{O}_i$) avec*

- *$\mathcal{I}_i \subseteq \mathcal{I}$ l'ensemble des interactions impliquant P_i (i.e. interactions ayant P_i comme source ou destination) :*
  $I_k = (Iid, s, d, m_t) \in \mathcal{I}_i \Leftrightarrow s = P_i$ ou $d = P_i$,
- *$\mathcal{O}_i \subseteq \mathcal{O}$ l'ensemble de contraintes sur $\mathcal{I}_i$.*

Pour contrôler ses propres interactions, un participant n'a pas besoin de toute la vue globale. En effet, chaque participant peut vérifier ses propres interactions avec le monde extérieur par rapport à sa propre vue locale. Comme indiqué dans [KR09], nous pouvons simplement calculer une vue locale d'un participant donné en éliminant toutes les interactions n'ayant pas ce participant comme une source ou destination tout en gardant les patrons de séquencement qui régissent l'ordre des interactions trouvées. La *figure* 5.7 montre les vues locales de chacune des organisations de notre exemple de chorégraphie.

Exemple 2 *Prenons par exemple le cas du revendeur (R). Sa vue locale $\mathcal{C}_R = (\mathcal{I}_R, \mathcal{O}_R)$ est définie comme suit :*

$\mathcal{I}_R = \{ I_1, I_{2c}, I_{3a}, I_{3b} \}$.

$\mathcal{O}_\mathcal{R} = \{ Seq(I_1, I_{3a}), Seq(I_1, I_{3b}), Seq(I_1, I_{2c}), Ex(I_{3a}, I_{3b}), Ex(I_{3a}, I_{2c}), Ex(I_{3b}, I_{2c}) \}$.

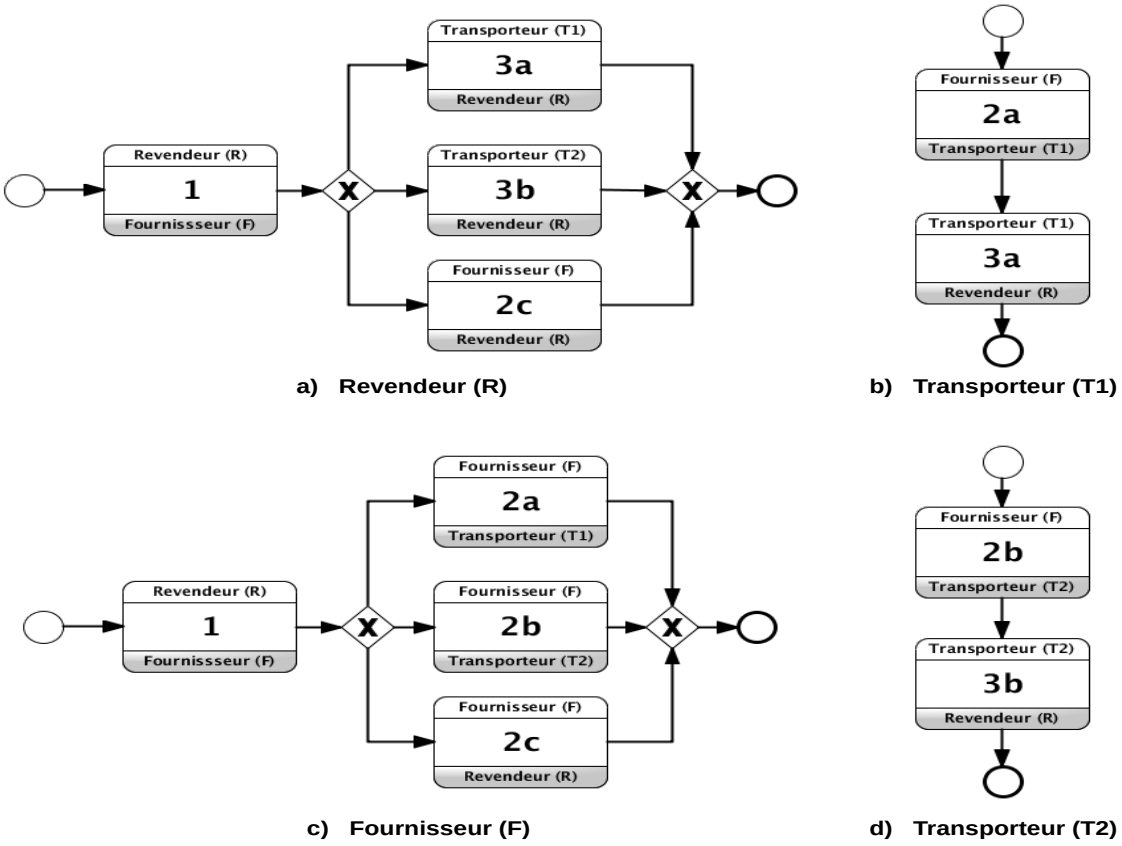


FIGURE 4.3 – Vue locale pour chacun des participants.

4.2 Vérification évènementielle des échanges de messages

Au lieu de vérifier périodiquement (avec une « requête / réponse ») si une situation de violation a été détectée ou non, la vérification de l'exécution d'une chorégraphie est effectuée uniquement quand il y a un nouveau message ou une information intéressante à signaler. Pour permettre cela, nous avons choisi de proposer une approche basée sur les événements dans laquelle chaque nouvelle action ou information peut être associée à un événement. En effet, une approche événementielle permet une meilleure réactivité et moins de trafic réseau que l'approche alternative « requête / réponse ». Cette dernière approche risque d'induire une charge inutile importante lorsque la période (i.e. l'intervalle de temps entre deux vérifications) est relativement très petite. L'approche est beaucoup moins réactive dans le cas inverse. La *figure* 4.4 montre ces inconvénients en prenant deux exemples d'intervalles et six arrivées d'événements $I_{1..6}$. Nous remarquons dans cette figure que le nombre de requêtes inutiles augmente pour une vérification périodique avec un intervalle plus petit. Dans le cas inverse, lorsqu'on augmente l'intervalle entre deux vérifications, l'approche devient moins réactive (i.e. le retard de détection «rt» augmente).

L'adoption d'une approche événementielle pour assurer des besoins non fonctionnels tels que la supervision permet de réduire le couplage d'un système. Tout d'abord, un producteur d'événements doit être configuré pour permettre la génération d'un nouvel événement à chaque

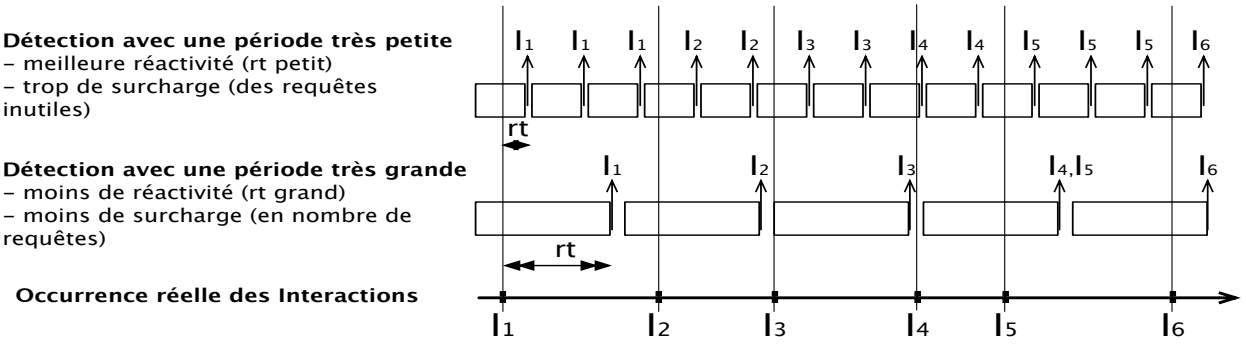


FIGURE 4.4 – Approche requête / réponse périodique

fois qu'un message de chorégraphie a été échangé. Ensuite, les événements générés sont analysés et corrélés aux différentes instances de chorégraphies en cours d'exécution. Pour ce faire, il faut commencer par définir la structure des événements.

4.2.1 Événement d'occurrence d'un échange de message

Dans une architecture événementielle, un événement représente le fait que quelque chose se produit. Dans le contexte d'une chorégraphie de services, un changement d'état se produit lorsque un message est envoyé et/ou reçu. Dans ce contexte, un événement peut être donc associé à une occurrence de message (envoi ou réception). En d'autres termes, un événement correspond à une transition dans le graphe d'états d'une chorégraphie.

Suivant cette logique, nous avons choisi de générer un nouvel événement local pour chaque message échangé (i.e. au niveau du participant émetteur au moment de l'envoi et au niveau du participant destinataire au moment de la réception). Chaque événement est donc corrélé à une interaction définie dans le modèle de la chorégraphie et est généré dans le but d'informer sur l'occurrence d'un message qui est associé. Nous appelons ce type d'événement «événement d'échange» et nous le définissons comme suit.

Définition 5 (Événement d'échange) *Lors de l'exécution d'une chorégraphie, un évènement correspond à toute transition d'un état global à un autre ce qui est exactement le résultat d'un envoi ou d'une réception d'un message par un des participants, ou encore, d'une occurrence d'une exception.*

Formellement, un événement d'échange $e \in \mathcal{E}$ est un tuple :

$$e = (Eid, Cid, Iid, TS)$$

avec Eid un identifiant unique de l'événement, Cid un identifiant d'instance de chorégraphie (utilisé pour la corrélation), $Iid \in \mathcal{I}$ l'identifiant de l'interaction correspondant au message échangé, et TS l'estampille de génération de l'événement.

4.2.2 Corrélation des événements

Au cours de la phase d'exécution, nous pouvons avoir plusieurs instances de différents modèles de chorégraphie. De toute évidence, nous avons besoin de corréler les événements appartenant à la même instance pour permettre l'analyse de l'ensemble d'une manière cohérente.

Dans une chorégraphie, la corrélation des message consiste à associer à chaque message créé un contexte particulier correspondant à l'instance spécifique du processus métier et différent des autres instances. Ce mécanisme permet l'organisation des conversations entre les participants. Par exemple, dans le cadre d'une chorégraphie de réservation de voyages, un client enregistré peut avoir effectué une réservation, mais doit fournir des informations supplémentaires afin de la finaliser. Lorsqu'il revient à sa réservation, un identifiant unique permet de retrouver son billet, de sorte qu'il peut ensuite procéder au paiement. Un tel identifiant unique est aussi utilisé par l'agence de voyages afin d'associer le vol choisi à un séjour. L'identifiant est généralement généré par le participant initiateur de la chorégraphie.

Techniquement parlant, la solution la plus courante pour faire face à ce problème consiste à définir un identificateur qui doit être attaché à chaque message échangé (e.g. inclus dans l'en-tête SOAP). À titre d'exemple, BPMN 2.0 utilise la notion de « clé de corrélation » pour lier les messages avec les instances de processus. Une clé de corrélation est un jeu de variables qui est utilisé pour identifier une instance de processus afin de router les messages qui lui sont appliqués. Elle peut être créée dans un diagramme de chorégraphie avec n'importe quel langage d'orchestration. Ainsi, une conversation BPMN 2.0 représente un ensemble de flux de messages groupés en se basant sur une clé de corrélation [CCDCRD11].

Dans l'approche présentée dans [WKK+10], deux identifiants sont utilisés séparément : l'identifiant de la chorégraphie (i.e. un identifiant unique pour chaque modèle de chorégraphie) et l'identifiant d'instance (i.e. un identifiant unique pour chaque instance de chorégraphie).

Dans notre cas, le champ Cid est un identifiant qui combine les deux ($Cid = ChorID.InsID$ avec $ChorID$ l'identifiant de la chorégraphie et $InsID$ le numéro d'instance). Ce champ nous permet de déterminer pour chaque événement nouvellement généré l'instance de laquelle le message échangé fait partie.

4.2.3 Horodatage des événements

Le champ TS représente l'heure à laquelle le message est envoyé ou reçu. L'horodatage des événements permet de définir un ordre local qui est nécessaire pour analyser les données de surveillance collectées.

Dans une configuration distribuée, il faut définir un temps global cohérent pour tous les processus. Une première solution consiste à synchroniser au mieux les horloges physiques locales entre elles ou avec une horloge de référence (e.g. la source externe fournit un temps de référence et les horloges locales se re-synchronisent régulièrement à partir de cette source) [AP97]. Une autre solution consiste à créer un temps logique (e.g. horloge de Lamport [Lam78]).

Dans ce manuscrit, nous supposons que les communications asynchrones entre deux participants sont fiables dans le sens où une et une seule réception est associée à chaque envoi et que les messages arrivent dans le même ordre dans lequel ils sont envoyés. Cette hypothèse est réalisable en adoptant un des protocoles fiables disponibles par la pile des services Web (e.g. *WS-ReliableMessaging* [FPD+09]).

4.3 Architecture générale

L'architecture que nous présentons dans cette section consiste à fournir de nouveaux composants qui seront intégrés et déployés au sein de chaque organisation participante. Le but de ces composants est quadruple :

1. Interception des messages échangés : étant déployés le long des frontières de chaque organisation participante, nos composants doivent être en mesure d'intercepter tous les messages échangés avec le milieu extérieur. Les messages interceptés sont ensuite filtrés et contrôlés.

2. Vérification de la structure de chaque message : en d'autres termes, les messages sont reçus de la part (ou à destination) de participants connus (i.e. jouant un rôle prédéfini dans la chorégraphie) qui se comportent comme prévu (i.e. structure de message conforme à un schéma prédéfini).

3. Vérification du séquencement des messages : ceci consiste à vérifier que la séquence des messages envoyés et reçus est conforme à un schéma prédéfini. Le but est de s'assurer que les parties participantes ont accompli leurs tâches en respectant les patrons de séquencement utilisés pour le contrôle de flux de la chorégraphie.

4. Échange de notifications entre partenaires : pour permettre une supervision décentralisée, il est nécessaire d'échanger des données sur le suivi des messages échangés pendant l'exécution de la chorégraphie. En plus, il est nécessaire d'assurer la notification sélective du (des) partenaire(s) concerné(s) en cas de détection d'un flux incohérent.

Pour atteindre ces objectifs d'une manière agile et flexible, nous proposons une architecture dans laquelle les échanges inter-organisationnels sont perçus comme des événements. En effet, traiter les points mentionnés ci-dessus de manière événementielle favorise le passage à l'échelle tout en bénéficiant des avantages d'un couplage lâche des architectures orientées événements (EDA). Ces dernières ont montré leurs avantages en réduisant le couplage entre les différents composants tout en proposant une infrastructure hautement re-configurable. De plus, l'architecture EDA complète l'architecture orientée services (SOA) car les services peuvent être activés par les « triggers » que sont les événements [Cha06, TYPM09]. Suivant le principe de *Complex Event Processing* (CEP) [Luc02b], nous utilisons des patrons d'événements (*event patterns*) pour filtrer, vérifier et traiter les événements correspondant à des messages entrants et sortants.

4.3.1 Séparation des préoccupations (aspects)

Selon le paradigme de séparation des préoccupations (*Separation of Concerns*, SoC), un système est un ensemble de préoccupations fonctionnelles et non-fonctionnelles (aspects). Les préoccupations fonctionnelles sont les fonctionnalités métiers que le système doit assurer, alors que les préoccupations extra-fonctionnelles sont des services dont le système a besoin pour effectuer ses fonctionnalités métiers (e.g. sécurité, la synchronisation, la gestion de la persistance, etc.). La séparation des préoccupations permet de séparer les parties non-fonctionnelles des parties fonctionnelles d'un système. L'objectif escompté est d'offrir une meilleure réutilisation, réduire la complexité des systèmes, augmenter leur évolutivité et faciliter ainsi la maintenance [HM08]. Parmi les approches les plus utilisées, nous pouvons citer la programmation orientée aspect (*Aspect Oriented Programming*, AOP) [KLM+97].

Dans notre cas, pour permettre de gérer l'aspect non-fonctionnel auquel nous nous intéressons, en l'occurrence celui de la supervision, la conception et l'intégration de composants supplémentaires s'avèrent donc nécessaires. En outre, la séparation de l'aspect supervision de la logique métier (i.e. processus métier) est également motivée par le fait que la collecte des données de suivi et leur évaluation peut avoir des effets négatifs sur l'exécution d'une tâche et la performance de l'ensemble du processus [BGG].

4.3.2 Organisation des composants

Afin de présenter le concept d'une manière indépendante des technologies existantes, nous préférons définir une architecture générale plutôt que de décrire les détails techniques liés à un langage spécifique de chorégraphie de services. Ainsi, nous proposons d'étendre l'architecture de chaque organisation avec les trois composants : «Politique de flux externe (EFP)», «Contrôleur de flux externe (EFC)» et «Superviseur de flux externe (EFM)» (*Figure* 4.5).

4.3.3 Politique de flux externe (EFP)

Après la création de la vue locale de chaque chorégraphie, une politique locale de contrôle de flux externe est automatiquement créée au sein de chaque participant. Cette politique spécifie des contraintes sur la structure des messages reçus et envoyés en se basant sur le schéma des interactions spécifiées dans la vue locale. Ces contraintes vont permettre par la suite de vérifier en cours d'exécution que la source, la destination et le type de chaque message sont conformes à au moins une des interactions déjà définies.

Nous définissons ces contraintes sous forme de tuples formant ce que nous appelons une «politique de flux externe» (de l'anglais External Flow Policy, ou EFP). Dans chaque organisation, une politique EFP est définie comme suit.

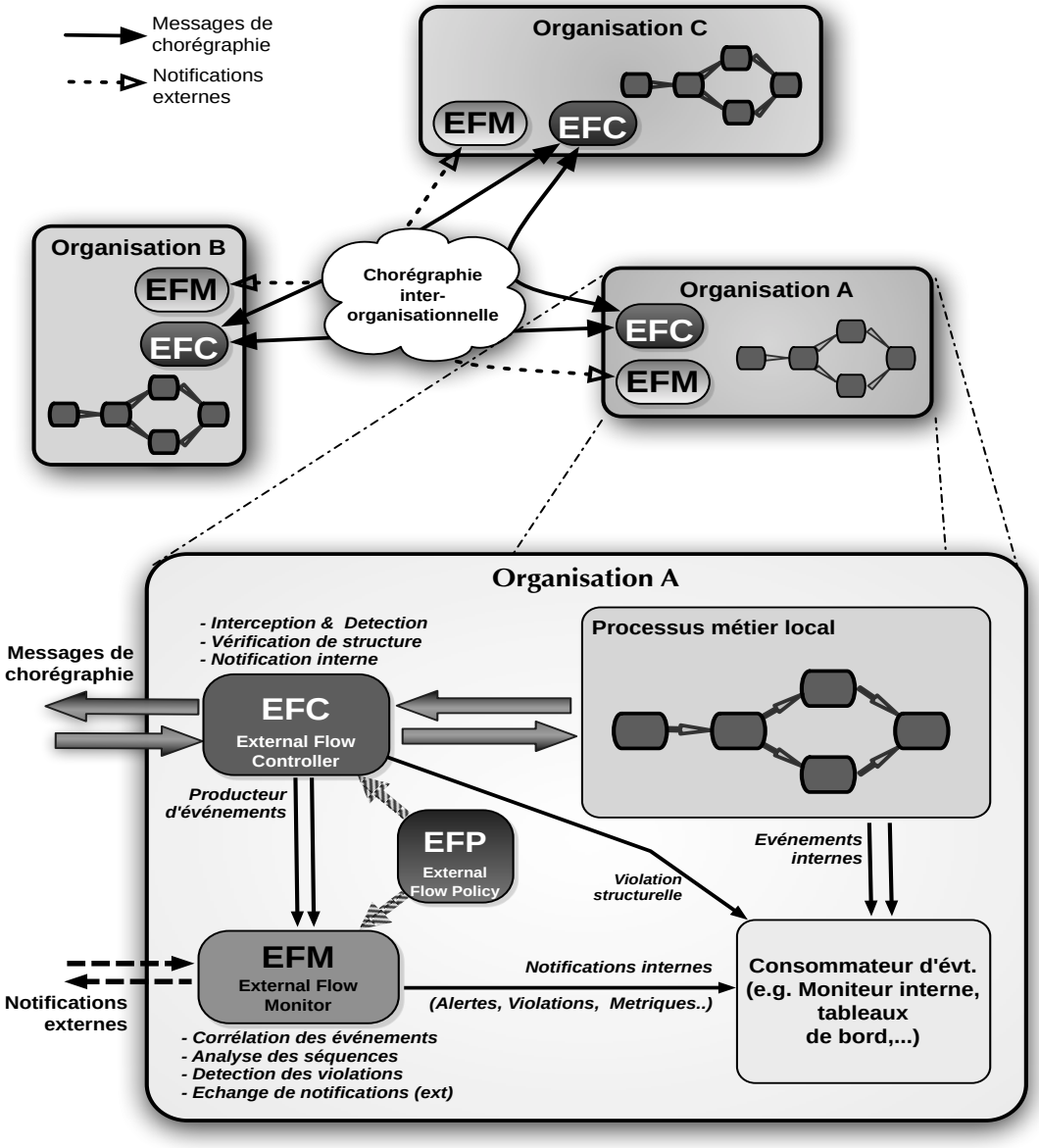


FIGURE 4.5 – Architecture générale

Définition 6 (Politique de flux externe (EFP)) *Une politique EFP d'un participant P_i est un ensemble de tuples de cette forme :*

$$< ChorID, Iid, Dir, M_{type}, Src/Dst >$$

avec $ChorID$ est un identifiant unique attribué à chaque modèle de chorégraphie, Iid est un identifiant d'une interaction, Dir est la direction du message associé (envoi ou réception), M_{type} est le type du message (e.g. $XMLSchema$), Src/Dst est le partenaire source ou destination du message.

Pour chaque modèle de chorégraphie, chaque participant est en charge de générer automatiquement sa propre table de politique EFP à partir de sa vue locale. Ainsi, chaque entrée de la politique EFP correspond à une des interactions définies dans la vue locale. Vérifier chaque message par rapport à cette politique consiste à détecter toute non conformité structurelle. En d'autres termes, chaque participant peut vérifier que chaque message échangé est une instance correcte d'une des interactions définies dans le schéma de la chorégraphie. Les messages qui ne correspondent à aucune des entrées de l'EFP doivent être signalés.

Par opposition aux politiques dynamiques, telles que celles proposées dans [KR09] et [Vag07] et qui souffrent de problèmes de passage à l'échelle (i.e. pour chaque nouvelle instance une nouvelle politique doit être instanciée), notre politique est statique pour un modèle de chorégraphie donné et partagée par toutes les instances de ce modèle. En effet, une et une seule politique est générée automatiquement après chaque création d'un nouveau modèle de chorégraphie. Mis à part le problème de passage à l'échelle, il est à noter également que les politiques statiques ont aussi l'avantage d'être moins sujettes aux erreurs. Par contre, l'inconvénient de notre politique statique est qu'elle ne permet pas de reconnaître «l'activité en cours» de chaque instance. Ceci ne présente pas de problème puisque c'est traité par le composant de supervision présenté ci dessous (cf. *section* 4.3.5).

Exemple 3 *La table 4.1 montre la politique EFP générée localement au sein de chaque organisation pour notre exemple de chorégraphie précédemment présenté. Notons que chaque participant génère sa politique localement indépendamment des autres. Ainsi, les identifiants $ChorID$ et Iid sont locaux et donc uniques à l'intérieur d'une même organisation.*

4.3.4 Contrôleur de flux externe (EFC)

Ce composant est chargé du contrôle et de la vérification de la structure des messages échangés conformément à ce qui a été défini dans la politique EFP. En d'autres termes, ceci consiste à filtrer les messages qui ne correspondent pas à une interaction autorisée par la politique. Deux types de contrôle sont définis :

(i) le contrôle des messages entrants afin de restreindre l'accès à des ressources internes en tenant compte du contexte de l'appel et des interactions autorisées par la politique,

(ii) le contrôle des messages sortants afin d'empêcher de divulguer des informations sensibles de façon involontaire en dehors d'un cadre prédéfini de collaboration.

En outre, le contrôleur EFC génère une notification et l'envoie à l'EFM chaque fois qu'un message est autorisé à être échangé avec le milieu extérieur, et ce, en vue de permettre le suivi et la supervision de l'évolution de l'exécution. Cette notification est définie et générée localement (i.e. au sein d'une même organisation). Dans le cas contraire, quand le message n'a pas été autorisé (i.e. violation de la politique), une alerte est envoyée au moniteur local.

La *figure* 4.6 illustre les étapes à suivre à chaque fois qu'un message d'une instance de chorégraphie est échangé au niveau d'un des participants. Il s'agit des trois étapes définies comme suit :

EFP du Revendeur (R)					EFP du Fournisseur (F)				
ChorID	Iid	Dir	M_{type}	Src/Dst	ChorID	Iid	Dir	M_{type}	Src/Dst
1	1	Send	1	F	1	1	Rec	1	R
1	2	Rec	2c	F	1	2	Send	2a	T1
1	3	Rec	3a	T1	1	3	Send	2b	T2
1	4	Rec	3b	T2	1	4	Send	2c	F
EFP du Transporteur (T1)					EFP du Transporteur (T2)				
ChorID	Iid	Dir	M_{type}	Src/Dst	ChorID	Iid	Dir	M_{type}	Src/Dst
1	1	Rec	2a	F	1	1	Rec	2b	F
1	2	Send	3a	R	1	2	Send	3b	R

TABLE 4.1 – Politiques de flux externe (EFP)

1. L'EFC commence par analyser le contenu du message afin de déterminer l'identifiant de l'instance à laquelle il appartient, la destination s'il s'agit d'un message sortant, la source s'il s'agit d'un message entrant, l'identifiant de l'interaction associée et le type de message. L'ensemble de ces données représente la structure du message.

2. Ensuite, l'EFC vérifie par rapport à la politique EFP de la chorégraphie associée si cette structure fait partie de ce qui a été défini.

3. Si la structure est conforme, l'EFC génère un nouvel événement associé à ce message et l'envoie à l'EFM. Dans l'autre cas (i.e. message non conforme), une alerte est envoyée à destination du moniteur local. Il s'agit ici d'une violation de structure.

L'EFC se comporte comme un pare-feu de niveau application permettant de filtrer et vérifier les messages par rapport à un ensemble de structures acceptées. Cependant, la vérification est statique dans le sens où elle ne permet pas de prendre en compte l'étape actuelle dans chacune des instances auxquelles l'organisation participe. Ceci dit, un message ayant une structure définie dans le schéma d'une chorégraphie mais qui n'est pas conforme à celui de l'étape en cours sera tout de même accepté par l'EFC. Pour permettre le suivi dynamique de l'exécution de chaque instance, une vérification plus sophistiquée s'impose. C'est l'objectif du composant EFM détaillé ci-dessous.

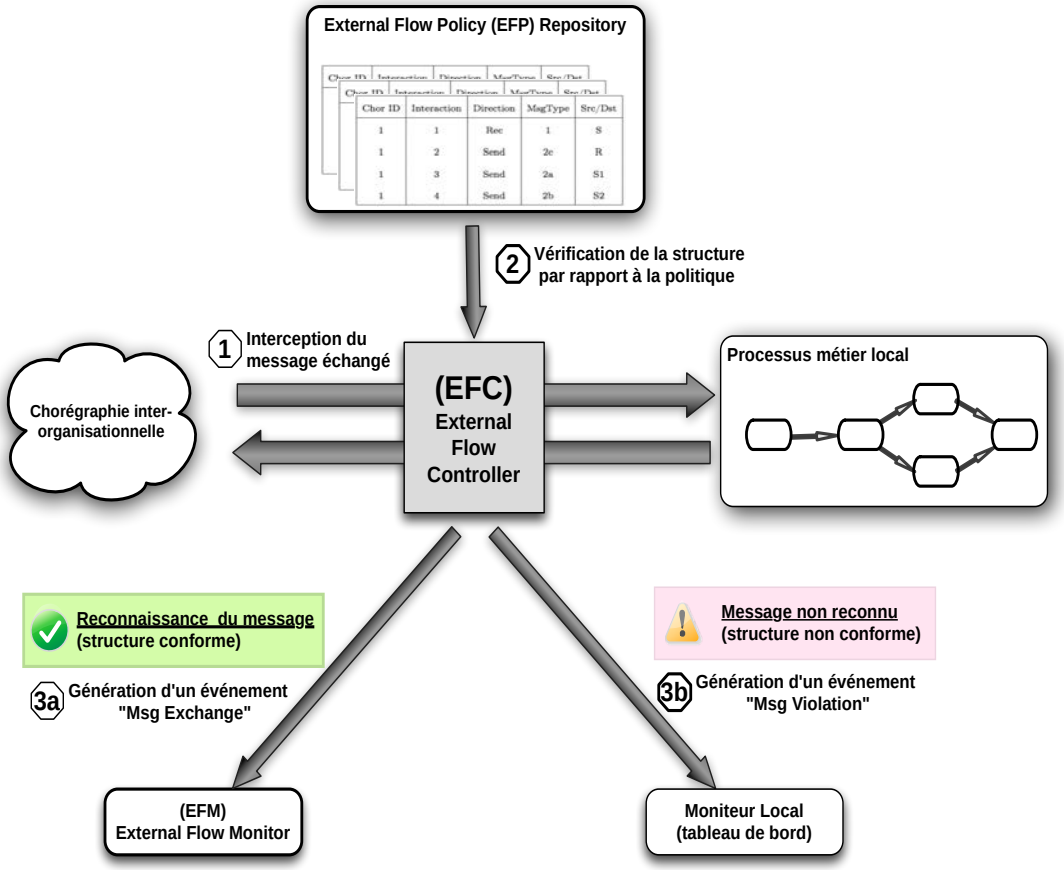


FIGURE 4.6 – Le contrôleur de flux externe (EFC)

4.3.5 Superviseur de flux externe (EFM)

Ce composant représente l'élément principal de l'approche de supervision décentralisée. Il permet le suivi de l'évolution de l'exécution de chaque chorégraphie à laquelle l'organisation (dont il fait partie) participe. Pour accomplir sa tâche et pour élargir son champ de vision, il échange en permanence des notifications avec le monde extérieur, plus précisément les EFMs des autres participants en respectant un mécanisme d'échange spécifique. À partir des notifications reçues du milieu extérieur et celles générées localement par l'EFC, l'EFM maintient l'état actuel pour chaque instance de chorégraphie en vue de vérifier son comportement par rapport à sa vue de supervision (*EFM-view*). L'EFM peut être configuré pour fournir un suivi réactif aux indicateurs de tableaux de bord internes. Un non-respect des règles relatives à la co-occurrence, l'exclusivité, et l'ordre des interactions peut également être détecté par des techniques de traitement d'événements complexes.

La *figure* 4.7 illustre le mode de fonctionnement de l'EFM. Ce dernier permet le traitement instantané des événements provenant de l'EFC en utilisant la technologie CEP. Nous distinguons deux fonctionnalités majeures :

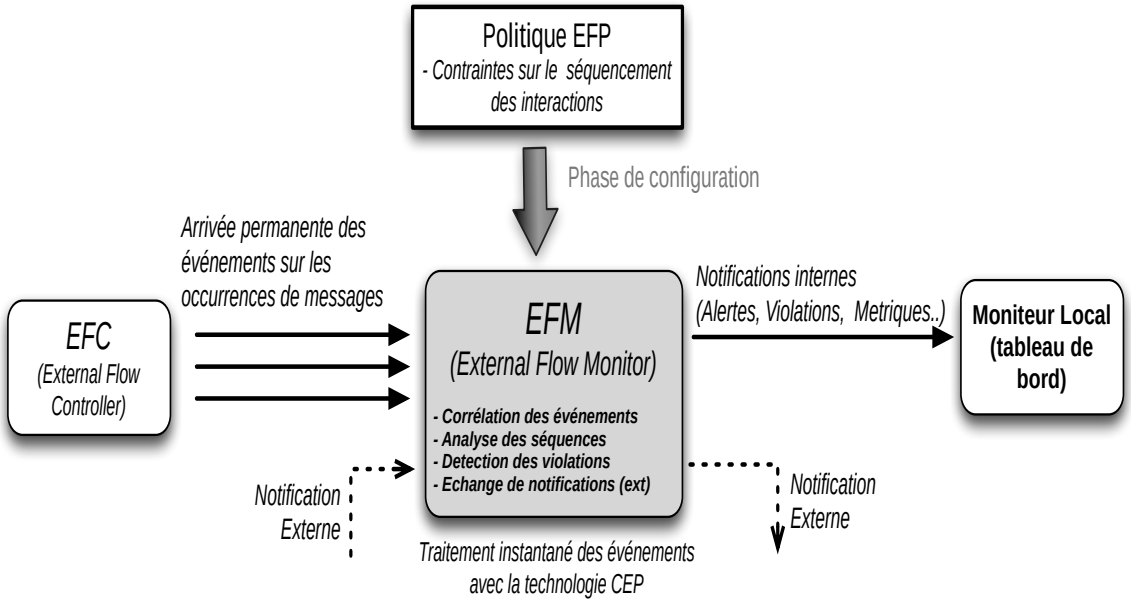


FIGURE 4.7 – Le superviseur de flux externe (EFM).

Échange de notifications entre les organisations pour construire sa vue de supervision : Après la vérification de chaque message par rapport au schéma de la chorégraphie, une notification est automatiquement générée, envoyée et propagée aux EFMs d'un sous ensemble pré-calculé de participants. Il s'agit d'un mécanisme automatisé et décentralisé pour l'échange de données de supervision entre les participants. Nous définissons de nouveaux canaux pour l'échange de notifications entre les différents EFMs. Notre approche n'est pas intrusive, c'est-à-dire que ces canaux nouvellement définis sont indépendants des canaux dans lesquels les messages de la chorégraphie sont échangés. L'EFM peut voir «passivement» ce qui est échangé sans rien modifier dans le modèle de chorégraphie.

Cette fonctionnalité est détaillée dans le *chapitre* 5.

Traitement événementiel complexe pour détecter les violations : Cette fonctionnalité consiste à vérifier la conformité des séquences d'interaction par rapport à sa vue de supervision. Cette vérification repose sur les évènements générés et reçus de la part de l'EFC à chaque échange de message détecté. Les contraintes sur le séquencement des messages doivent être transformées sous forme de requêtes événementielles. La trace d'exécution (i.e. le flux d'événements généré par l'EFC) est vérifiée par rapport à ces requêtes, et ce, à chaque arrivée d'un nouvel événement au cours de l'exécution. Les violations détectées seront instantanément envoyées au moniteur local.

Cette fonctionnalité est détaillée dans le *chapitre* 6.

4.3.6 Notification (interne/externe)

Dans notre approche, nous distinguons deux types de notifications : les notifications internes (i.e. au sein de la même organisation) qui sont générées par l'EFC à destination de l'EFM ou d'un moniteur interne et les notifications externes qui sont échangés par les EFMs de différentes organisations (i.e. dépassant les frontières d'une même organisation).

Notifications internes En cas de violation structurelle au cours d'un échange inter-organisationnel de message, le moniteur local d'une organisation reçoit une notification de la part de son EFC qui est en charge de vérifier la structure de chaque message entrant ou sortant. En cas de violation de l'une des contraintes sur le séquencement des interactions, le moniteur est notifié par l'EFM. Comme expliqué précédemment, l'EFM analyse le flux des événements indiquant les messages échangés qui est généré en permanence par l'EFC. Le moniteur peut être aussi configuré pour recevoir des notifications de part des processus locaux (e.g. suivi d'exécution, erreur interne). La *figure* 4.8 schématise les échanges de notifications entre les différents composants d'une même organisation.

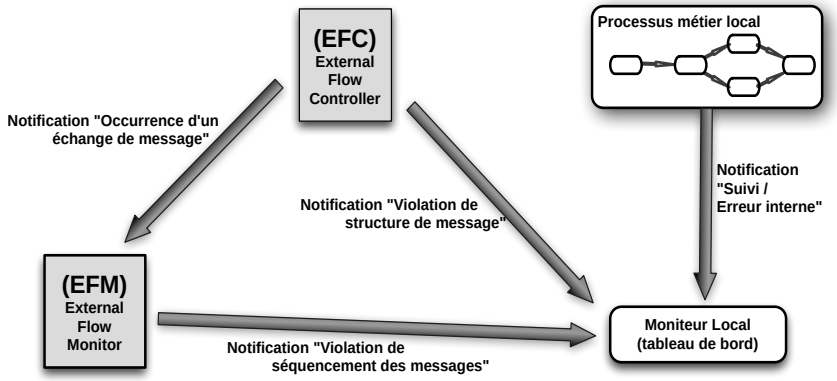


FIGURE 4.8 – Les notifications entre les composants au sein d'une même organisation.

Notifications externes Ces notifications sont échangées entre les EFMs des différentes organisations participantes à une même chorégraphie. Cet échange de notifications permet d'avoir une vue plus large que la vue locale de chaque participant et de recueillir des données supplémentaires qui sont utilisées pour l'analyse de l'exécution de la chorégraphie. Ces données pourraient aussi servir a posteriori pour mesurer la performance globale du processus et le suivi de la réalisation des objectifs de l'entreprise.

D'un point de vue technique, ce flux inter-organisationnel de données de supervision peut être considéré comme un nouveau protocole entre les partenaires. Ce protocole doit être agréé par tous les participants durant la phase de conception de la chorégraphie.

4.4 Synthèse et Conclusion

Dans ce chapitre, nous avons souligné le besoin d'une méthode agile et flexible qui permette d'assurer la conformité de la séquence des interactions inter-organisationnelles avec un plan prédéfini sous forme de modèle de chorégraphie. Nous avons commencé par définir les chorégraphies inter-organisationnelles d'une manière indépendante des langages de spécification. Nous avons ainsi proposé un modèle formel général, simple et compréhensible qui va servir de base pour notre approche de supervision. Nous avons ensuite introduit la notion de vérification événementielle dans le cadre des chorégraphies de services.

Nous avons, d'autre part, discuté du besoin d'un mécanisme décentralisé, dynamique et efficace qui permette l'échange des données de supervision entre les partenaires tout en assurant la traçabilité d'exécution du processus global. Nous avons ainsi proposé un nouveau modèle conceptuel en proposant une architecture générale pour la supervision décentralisée des collaborations.

Après avoir présenté les composants de notre architecture, nous avons détaillé l'aspect vérification structurelle des messages réalisé par l'EFC. Cependant, nous n'avons pas encore détaillé les deux fonctionnalités centrales de notre approche, celles réalisées par l'EFM, à savoir la supervision décentralisée avec échange inter-organisationnel de notifications et la génération automatique des règles au sein de chaque organisation. Dans le chapitre suivant, nous allons présenter notre approche d'échange de notifications entre les différents participants d'une chorégraphie. Nous allons montrer comment un tel échange va permettre d'offrir plus de maîtrise au niveau de chaque organisation participante et de réduire les délais et les coûts en cas d'occurrence d'exceptions.

5 Supervision décentralisée et échange de notifications entre partenaires

«Prouver que j'ai raison serait admettre que je puisse avoir tort.»

P-A Caron de Beaumarchais

Sommaire

5.1 Introduction

Dans ce chapitre, nous proposons une approche de suivi décentralisé de l'exécution d'une chorégraphie dans laquelle les échanges inter-organisationels de messages sont perçus comme des évènements. Les évènements sont ensuite traités par l'EFM de chaque participant. Ces derniers, étant implantés sous forme de services déployés le long des frontières de chaque organisation, ils sont invoqués à chaque détection d'un nouveau message reçu de (ou envoyé à) une organisation partenaire. Après la vérification de chaque message par rapport au schéma de la chorégraphie, une notification est automatiquement générée, envoyée et propagée à un sous ensemble pré-calculé de participants.

Échanger des notifications entre les partenaires risque d'induire un trafic supplémentaire très important. Afin de réduire cette charge supplémentaire du réseau, nous proposons une approche d'envoi ciblé au lieu de « broadcaster » toutes les notifications à tous les participants. En d'autres termes, nous essayons de réduire le nombre de partenaires qui doivent être notifiés. Pour ce faire, nous introduisons un nouveau type de relation entre les participants de la chorégraphie. Avec une telle relation, nous définissons une direction particulière pour chaque notification.

La section 5.2 montre l'intérêt et l'applicabilité de notre approche, notamment en étudiant le cas des chorégraphies de chaînes d'approvisionnement inter-entreprises. La section 5.3 donne un aperçu sur l'approche. La section 5.4 présente l'approche de classification hiérarchique des participants et définit les notions de *super/sous partenaires* ainsi que la vue de supervision (*EFM-view*). Ensuite, la section 5.5 décrit les algorithmes de configuration et d'exécution. La section 5.6 montre comment appliquer notre approche sur un cas de chaîne d'approvisionnement. Finalement, la section 5.7 décrit comment généraliser notre approche et l'étendre à tout type de chorégraphie inter-organisationnelle, et la section 5.8 conclut ce chapitre.

5.2 Chaînes d'approvisionnement et suivi des processus inter-entreprises

Comme nous l'avons mentionné antérieurement, la croissance continuelle des entreprises incite de plus en plus l'externalisation et la sous traitance de certaines activités. Cela a conduit à une forte augmentation du nombre d'interactions entre les différents partenaires en collaboration. Pour être capable d'externaliser leurs opérations ou vendre une partie de leurs processus métiers, certaines entreprises éprouvent le besoin de partitionner leurs processus et de séparer les éléments qui les constituent [WKK+10]. Une chorégraphie inter-organisationnelle peut être mise en oeuvre afin de permettre la coordination et la synchronisation entre ces différentes partitions. Ainsi, chaque organisation est en charge d'exécuter une partie du processus global.

Parmi les inconvénients majeurs de la sous-traitance, nous pouvons citer la perte de maîtrise, le manque de flexibilité de la part du prestataire, le risque de délais trop longs et le manque d'information et de transparence. En plus, le coût peut être parfois plus élevé que la production interne. Afin de pallier à ces problèmes, nous proposons une approche de supervision inter-

organisationnelle des fragments externalisés, et ce, de façon abstraite sans pour autant dévoiler la logique métier de chaque entreprise. En d'autres termes, l'approche proposée consiste à faire propager des données de supervision entre partenaires pour permettre une gestion décentralisée des informations recueillies, et ce, dans le but d'offrir plus de maitrise au niveau de chaque organisation, et de réduire les délais et les coûts en cas d'occurrence d'exceptions.

Pour illustrer les concepts et la démarche de l'approche présentée dans ce chapitre et pour bien comprendre les problèmes qui peuvent exister, nous adoptons le scénario d'une chorégraphie inter-organisationnelle qui englobe trois types de participants (si on ne compte pas le client final) jouant les trois rôles : Revendeur, Fournisseur et Constructeur dans une chaine d'approvisionnement. Dans ce type de chaine inter-organisationnelle, un revendeur peut interagir avec plusieurs fournisseurs qui, eux à leurs tours, interagissent avec plusieurs constructeurs dans le but d'obtenir un devis pour des biens ou des services. La structure d'un tel réseau est présentée dans la *figure* 5.1.

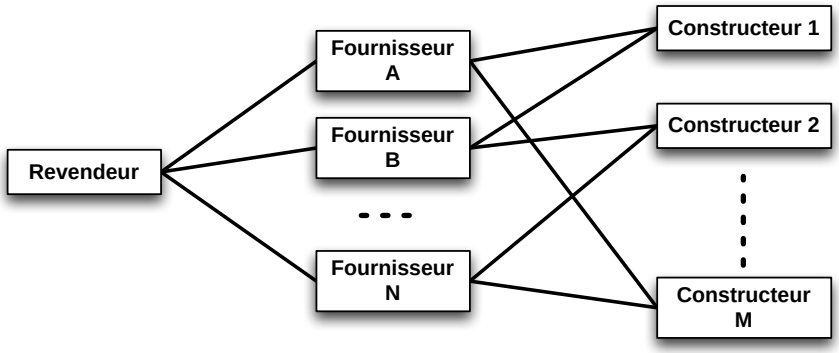


FIGURE 5.1 – Structure d'une chaîne d'approvisionnement

5.2.1 Scénario d'une chorégraphie de chaîne d'approvisionnement

La *figure* 5.2 montre un exemple de chorégraphie impliquant sept organisations principales : un client (C), un revendeur (R), deux fournisseurs (SA et SB), et trois constructeurs ($A1$, $A2$, et $A3$).

Dans cette chorégraphie, 14 interactions bi-partenaires sont définies ($I_{1,...,14}$). Le revendeur reçoit, en premier, une demande d'un devis de la part du client (I_1). Ensuite, il sélectionne un des deux fournisseurs (suivant le contenu de la demande) et élabore une nouvelle demande. En cas de sélection du fournisseur SA (à travers I_2), les constructeurs $A1$ et $A2$ seront invoqués en séquence ($I_{3,...,6}$). Dans l'autre cas (i.e. SB sélectionné), les constructeurs $A2$ et $A3$ seront invoqués en parallèle ($I_{9,...,12}$). Après avoir retourné les résultats finaux (I_7 ou bien I_{13}), le revendeur pourra enfin répondre le client par un devis complet (I_{14}).

La chorégraphie ainsi présentée permet d'offrir au client final un service de demande de devis au préalable. Dans le cas d'une chaine de fabrication sous mesure de pièces originales, l'exécution d'une telle chorégraphie peut prendre des jours (voire des semaines pour des produits complexes

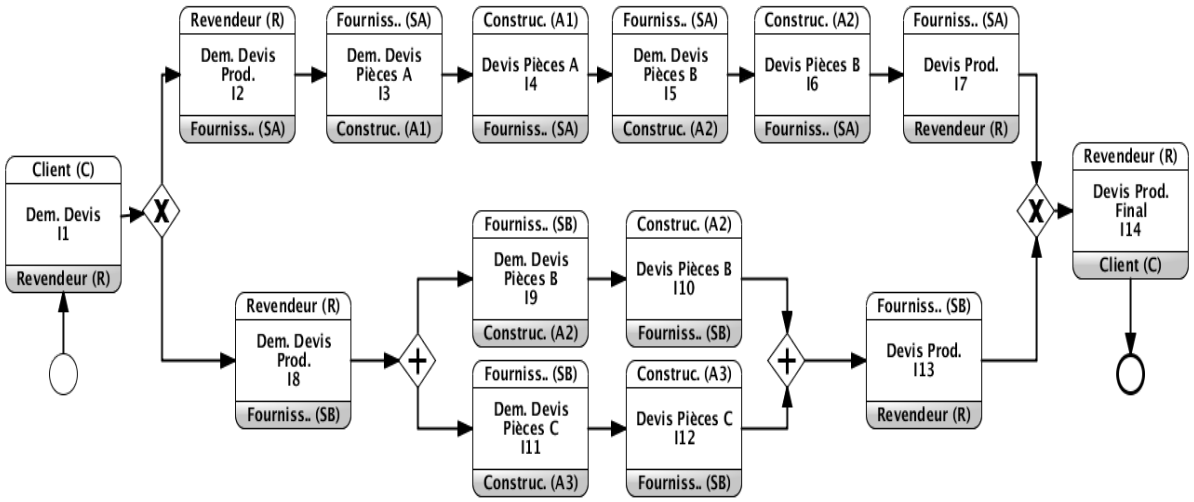


FIGURE 5.2 – Exemple de chaîne d'approvisionnement (Modélisation BPMN 2.0 : diagramme d'interaction).

nécessitant des pièces si particulières).

5.2.2 Suivi temps réel d'un processus global

Nous avons vu que dans un cadre inter-organisationnel, chaque organisation n'est en relation qu'avec les partenaires qui lui sont immédiatement connectés (i.e. avec lesquels il interagit directement) et ne peut donc pas voir au-delà. Techniquement, un service (ou un sous-processus exposé en tant que service) interagit en invoquant ses fournisseurs et en étant invoqué par ses consommateurs avec lesquels il a déjà conclu des accords de niveau de service. Lors de l'exécution, il n'a donc aucune information sur le reste des participants de la chorégraphie. Toutefois, un service dépend de tous ses services inférieurs, que ce soit par un lien direct ou transitif. Toute organisation peut donc s'intéresser à la surveillance de tous les fragments de processus externalisés et/ou exécutés par ses sous traitants.

Dans notre exemple, le client C interagit uniquement avec le revendeur R et ne peut donc voir ce qui se passe au-delà. Après avoir envoyé sa demande à R, C restera en attente de la réponse sans avoir aucune information sur l'état actuel de sa demande. Ainsi, il serait intéressant que C reste informé de l'évolution de l'exécution de la chorégraphie. Un tel exemple montre la nécessité d'échanger des notifications afin de permettre le suivi d'une chorégraphie. En effet, dans une configuration centralisée (e.g. orchestration), ce problème ne se pose pas puisque tout est coordonné par une unité centrale qui peut connaître à tout moment l'état actuel de l'exécution. Par contre, lorsqu'il s'agit d'une chorégraphie qui franchit les frontières de plusieurs organisations administrativement indépendantes, la tâche risque de devenir délicate.

En outre, l'échange des notifications permet d'avoir une vue plus large que la vue locale de chaque participant et de recueillir des données supplémentaires qui pourraient être utilisées

a posteriori pour mesurer la performance globale du processus et le suivi de la réalisation des objectifs de l'entreprise.

5.2.3 Exceptions

Des exceptions peuvent survenir à tout moment pendant l'exécution d'une chorégraphie. La raison peut être un problème interne d'un participant ou un message incorrect qui diffère de celui qui est spécifié dans la description du processus. Dans le cas d'une erreur interne qui ne peut être transmise au niveau de la chorégraphie pour être traitée comme une exception globale (en utilisant par exemple les exceptions WS-CDL), les participants concernés ne seront jamais notifiés et resteront toujours en attente d'un message. Traditionnellement, ce genre de situation est géré par l'ajout des appels de confirmation après la fin de chaque sous-processus et/ou la fixation des délais d'attente à l'échelle de la chorégraphie. Cependant, les appels supplémentaires peuvent compliquer la chorégraphie de flot de contrôle. Aussi, les délais d'attente ne sont pas bien adaptés aux collaborations de longue durée. Pour résoudre ce problème, il est donc préférable de garantir que tous les partenaires concernés soient informés de toutes les erreurs qui se produisent dès qu'un sous-processus interne souffre d'une exception.

5.2.4 Délais d'attente (Timeouts)

Le délai d'attente correspond à la durée maximale pendant laquelle un service (au sens large) peut continuer à attendre un certain événement. Si cet événement ne se produit pas pendant cet intervalle de temps, une exception est levée et le service quitte son traitement ordinaire pour exécuter un morceau de code prédéfini qui servira à traiter ce genre d'exception [Tos09].

Dans le cadre de processus métiers, lors de l'utilisation du langage BPEL par exemple, un délai d'attente sur chacune des activités de réception de messages peut être défini. Ce délai permet au processus de basculer vers un autre chemin pour ne pas attendre indéfiniment l'arrivée d'un message externe et éviter ainsi un blocage potentiel. Dans ce cas, la valeur de temporisation est fixée selon le temps maximal prévu pour accomplir la tache réalisée par chaque service invoqué, et ce, dans les conditions normales. Généralement, cette valeur maximale est définie par un SLA (Service Level Agreement) qui a pour but de garantir un certain niveau de qualité de service, en particulier un délai maximal de réponse.

Dans notre exemple de chaîne d'approvisionnement (*figure* 5.2), chaque participant peut fixer un délai pour chaque interaction afin d'indiquer combien de temps il pourra attendre un message avant de signaler une erreur de type "timeout" (délai expiré).

La *Figure* 5.3 montre deux scénarios d'exécution. Entre chacun de ces participants, des délais d'attente ont été fixés au préalable. Chaque message reçu d'un autre partenaire ne doit pas arriver après l'expiration de son délai d'attente correspondant. Par exemple, nous supposons que le délai t_1 est fixé par le client C à deux jours. Les délais t_2 et t_3, fixés respectivement par le revendeur R et le fournisseur SB, auront nécessairement une valeur plus petite (par exemple, 12 heures et 8 heures, respectivement). Dans le cas d'une erreur interne au sein du constructeur $A2$ (figure de

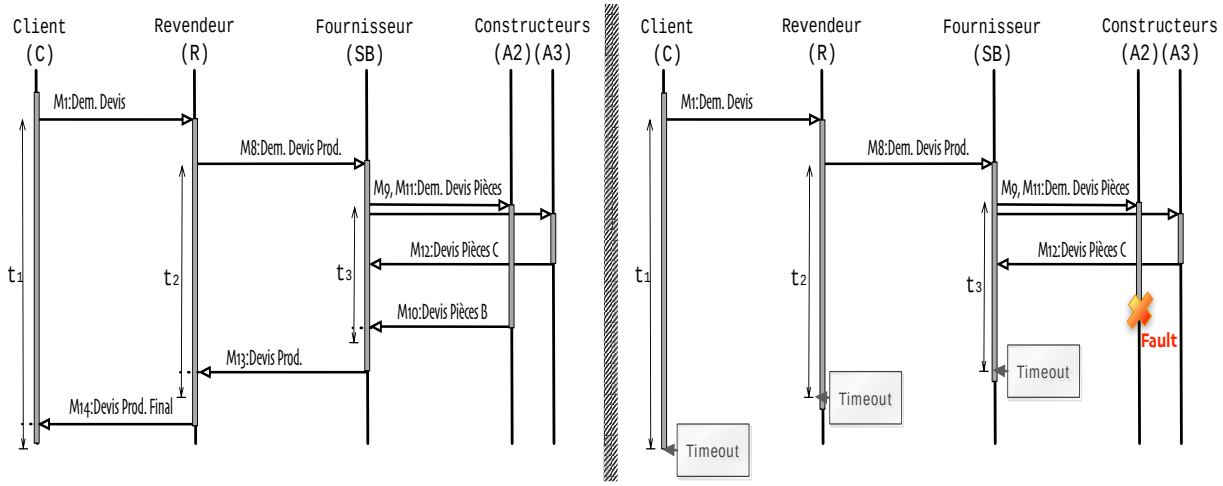


FIGURE 5.3 – Scénarios d'exécution qui termine avec succès (gauche) avec délai expiré (droite).

droite) ou d'une non réception d'un message d'un autre participant dans le délai fixé localement, le fournisseur SB, et donc le revendeur R et le client C, restent non informés de cette exception et continuent donc à attendre une réponse jusqu'à ce que le délai d'attente local de chacun expire. Nous remarquons que ceci peut causer une grosse perte de temps, et ne peut donc être adapté pour une telle collaboration de longue durée.

Dans un tel cas de situation, nous pouvons remarquer l'utilité d'un mécanisme d'échange de notifications entre partenaires qui permettra de relayer les occurrences d'exceptions vers les participants en attente d'un message. Un tel mécanisme permettra un gain énorme de temps surtout dans le cas des collaborations de longue durée (des processus inter-entreprises qui durent des heures et des jours).

N.B. Dans le cas d'un message « one way » (envoi sans réponse), le participant ne reste pas en attente (il n'y a pas de blocage), mais il envoie tout simplement sans avoir besoin de fixer un délai d'attente « fire and forget ».

5.3 Mécanisme décentralisé pour l'échange de notifications entre partenaires

Un superviseur traditionnel est censé superviser la conformité de l'exécution des processus locaux par rapport aux modèles définis et aux règles métiers de sa propre organisation administrativement indépendante (i.e. en se référant uniquement à ce qu'il peut voir localement). Ce type de supervision ne permet donc pas le suivi global d'une chorégraphie inter-organisationnelle puisqu'il n'est pas possible de voir ce qui se passe au-delà des frontières de l'organisation (e.g. les communications entre d'autres partenaires de collaboration). Pour superviser l'ensemble de la chorégraphie, des solutions ont été proposées [AFG+08, CCMN04, WKK+10]. Ces approches sont centralisées et se basent donc sur un superviseur central implanté dans une organisation à laquelle tous les autres participants doivent faire confiance. Cette entité centrale est notifiée

par chaque participant chaque fois qu'un événement se produit (en utilisant, par exemple, le mécanisme de « *publish/subscribe* » ou publier/ s'abonner [WKK+10]) . Néanmoins, elle peut présenter un point de défaillance unique et peut ne pas être adaptée lorsqu'il n'y a pas d'entité de confiance commune à tous les participants. En plus, on pourra aussi citer tous les autres inconvénients d'une configuration centralisée (e.g. goulot d'étranglement,...).

5.3.1 Aperçu sur notre approche

Pour faire face à ce genre de problème, nous proposons un mécanisme automatisé et décentralisé pour l'échange de données de supervision entre les participants. Nous définissons de nouveaux canaux pour l'échange de notifications entre les différents EFM (défini au Chapitre 4). Notre approche n'est pas intrusive, c'est-à-dire que ces canaux nouvellement définis sont indépendants des canaux dans lesquels les messages de la chorégraphie sont échangés. L'EFM peut voir « passivement » ce qui est échangé sans rien modifier dans le modèle de chorégraphie. La *Figure* 5.4 montre un aperçu sur notre approche.

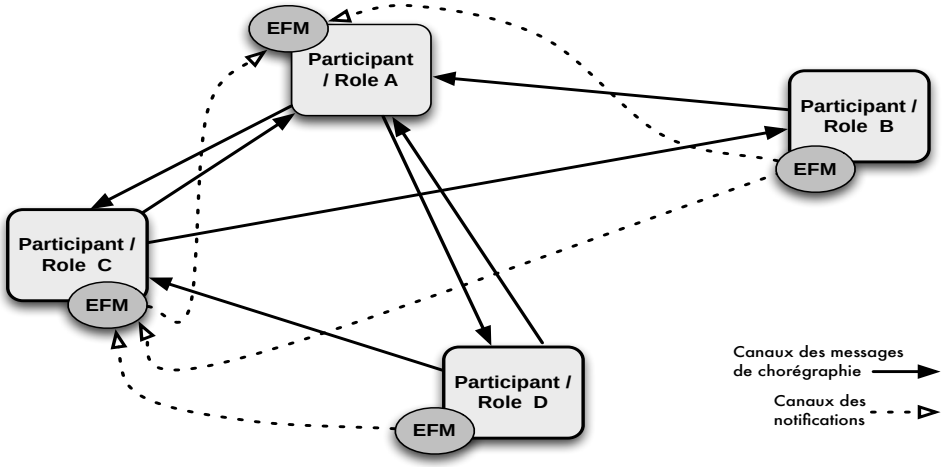


FIGURE 5.4 – Mécanisme décentralisé pour l'échange de notifications entre partenaires.

5.3.2 Définition des notifications externes

Lors de l'exécution, le modèle chorégraphie est utilisé comme une base pour observer "en transparence" le comportement des participants. Notre approche repose uniquement sur les changements d'états de la chorégraphie (i.e. quand un message est envoyé ou reçu) permettant ainsi le suivi des échanges de messages d'une manière non intrusive (i.e. les messages échangés ne sont pas modifiés par les EFMs et les services composant la chorégraphie de base ne sont pas conscients des EFMs et des notifications créées et échangées). Chaque notification est définie en se basant sur le modèle de chorégraphie en spécifiant les données qu'elle doit contenir. Le but de cette notification est d'informer les autres partenaires de l'occurrence d'un évènement global. Un évènement global dans une chorégraphie est défini comme dans la *Définition* 5.

Pour chaque occurrence d'un évènement, l'EFM responsable de ce dernier (i.e. l'EFM du participant émetteur ou récepteur du message) peut générer une nouvelle notification qui sera envoyée à des partenaires spécifiques. Chaque notification est corrélée à un message défini dans le modèle de la chorégraphie. Nous définissons une notification comme suit.

Définition 7 (Notification) *Une notification $n \in \mathcal{N}$ est un tuple (c_i,t,s,d,m_t) avec*
- *c_i l'identifiant de l'instance (utilisé pour la corrélation),*
- *t le timestamp de la génération de l'évènement,*
- *s,$d \in \mathcal{P}$ (respectivement, la source et la destination du message associé),*
- *et $m_t \in \mathcal{M}_\mathcal{T}$ (le type/structure de message).*

Dans un tel environnement distribué par nature, le temps représente un souci majeur du fait que les participants n'ont pas accès à une horloge globale. En effet, chaque notification échangée est horodatée par le participant qu'il l'a créée. Cependant, pour permettre une analyse correcte des données de surveillance acquises par chacun des participants, les événements de notifications générés doivent être horodatés par l'intermédiaire d'une horloge commune. Pour faire face à ce problème, les algorithmes de synchronisation d'horloge (*Clock Synchronization Algorithms* [AP97]) peuvent être adoptés. Sans avoir besoin de régler une horloge globale, une solution alternative consiste à envoyer le temps séparant deux événements consécutifs au lieu d'envoyer le temps absolu de génération de chaque événement.

5.4 Classification hiérarchique des partenaires

Afin de permettre l'envoi sélectif des notifications, il faut commencer par déterminer le sous-ensemble des partenaires potentiellement intéressés par l'événement correspondant. Dans le cas des chaînes d'approvisionnement, les partenaires intéressés sont ceux qui sont considérés comme des consommateurs ou clients de ce participant (e.g. une entreprise est considérée comme cliente chez ses sous-traitants et est donc intéressée par l'accomplissement de toutes ses parties métiers externalisées).

Nous proposons donc de classer hiérarchiquement les participants suivant la relation producteur/consommateur et l'évolution de la valeur ajoutée dans une chorégraphie de services donnée.

5.4.1 Super / sous partenaire

Nous classifions les partenaires en plusieurs niveaux hiérarchiques. Nous commençons par mettre le client (c'est-à-dire l'initiateur de la chorégraphie) au plus haut niveau de la hiérarchie (sur la couche supérieure). Les couches les plus basses sont constituées d'un ensemble de fournisseurs de services. Ainsi, chaque participant d'une couche donnée fournit un service à un service de la couche qui est juste au dessus (voir *figure* 5.5). Comme chaque couche dépend (directement ou transitivement) de toutes les couches inférieures, tout participant qui attend un service d'un autre de la couche du dessous sera ainsi intéressé par le suivi de son exécution. Pour

permettre cela, nous autorisons la propagation des notifications uniquement du bas vers le haut de la hiérarchie. Une telle hiérarchie indique en quelque sorte l'organisation des partenaires de la chorégraphie de services en vision consommateur/ fournisseur de service. Elle représente aussi les niveaux de visibilité des participants.

Pour permettre une telle classification, nous introduisons la notion de *super / sous-partenaire* comme suit.

Définition 8 (Super / sous partenaire (direct)) *Un participant $P_i \in \mathcal{P}$ est appelé le super− partenaire direct d'un participant P_j ssi*

P_i est l'émetteur dans la première interaction définie dans la vue locale de P_j

ou encore

l'instance du sous-processus de P_j est créée suite à un message venant de P_i.

En d'autres termes, P_i est responsable de la participation de P_j dans la chorégraphie. Ainsi, P_j est appelé $sous - partenaire$ de P_i.

Formellement, on note :

$$Super(P_i) = P_j \Leftrightarrow P_i \in Sub(P_j)$$

Exemple 4 *Dans notre exemple de chaîne d'approvisionnement (présenté précédemment dans la Figure 5.2) :*

$Super(SA) = R$; $Super(SB) = R = Super(Super(A2))$;

$SA, SB \in Sub(R)$; $A1, A2 \in Sub(SA)$; $A2, A3 \in Sub(SB)$

Il faut remarquer que notre approche fait l'hypothèse que la chorégraphie est réalisable [LW10] (*cf. Définition* 1) et avec un seul initiateur. Par exemple, dans une chaîne d'approvisionnement, un fournisseur de services peut avoir lui même un ou plusieurs fournisseurs, qui eux aussi, peuvent avoir, à leur tour, d'autres fournisseurs et/ou sous-traitants, etc. Ceci fait qu'une telle structure est hiérarchique [uHHS09]. De cette façon, chaque participant aura exactement un *super-partenaire* (sauf celui qui a initié la chorégraphie) et peut avoir un ou plusieurs *sous-partenaires*. En tant que tel, les participants peuvent être classés dans un arbre avec l'initiateur comme racine. Nous appelons cet arbre CPT «*Choreography Participant Tree*». La *figure* 5.6 montre l'arbre CPT de notre exemple de chaîne d'approvisionnement précédemment défini.

Chaque participant notifie son *super-partenaire* en générant une nouvelle notification chaque fois qu'il échange un message (envoi ou réception). En plus des notifications qu'il génère, il lui transmet aussi toutes celles reçues de la part de ses *sous-partenaires*, et ce, directement dès qu'il les reçoit. Ainsi, les notifications se propagent toujours du bas vers le haut de l'arborescence des partenaires (i.e. de chaque *sous-partenaire* vers son *super-partenaire*).

L'utilisation d'une telle architecture hiérarchique permet de réduire le nombre de notifications échangées et donc la surcharge du réseau. Ceci permet aussi de recevoir chaque notification au maximum une fois et élimine donc toute redondance suite à une réception multiple des notifications sur le même évènement de la part de deux sources différentes.

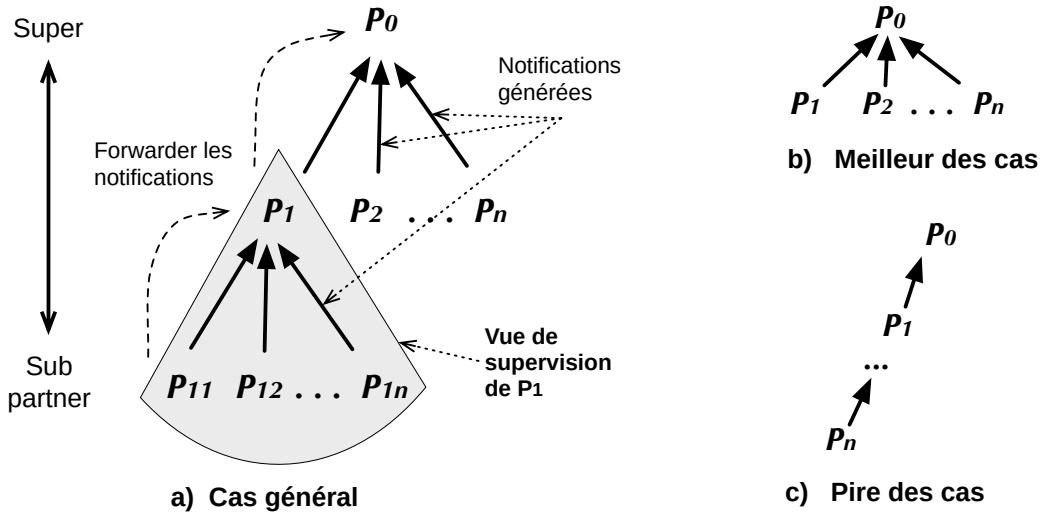


FIGURE 5.5 – Classification hiérarchique des partenaires (Arbre CPT).

Complexité de l'approche Nous sommes en mesure de calculer la complexité de notre approche par rapport au nombre total de notifications échangées. En effet, ce nombre dépend du nombre de messages de la chorégraphie et du nombre de participants.

Complexité en fonction du nombre de messages

Soit d la profondeur de l'arbre généré. Pour chaque message, il y a au maximum $d-1$ notifications qui doivent être transmises (une génération + $d-2$ transferts). Ainsi, le nombre de notifications est majoré par $O((d-1)m)$ et donc $O(m)$ (avec m le nombre total de messages échangés dans la chorégraphie).

Complexité en fonction du nombre de participants Soit p le nombre de participants, la complexité, dans le meilleur des cas (lorsque la profondeur est égale à 1), est évidemment linéaire (voir *figure* 5.5.b). Dans la moyenne des cas (la quasi-totalité des cas), notre approche nécessite $O(p.Log(p))$ notifications. Dans le pire des cas et lorsque l'arbre est totalement déséquilibré (voir *figure* 5.5.c.), nous avons besoin de $O(p^2)$ notifications. Ici, l'arbre ressemble à une liste chaînée (aussi appelé arbre dégénéré).

Preuve 1 *Soit :*

- *c le nombre moyen de fils pour chaque nœud (i.e. le nombre moyen de* sous-partenaires *pour chaque participant),*
- *m le nombre maximum de messages envoyés par chaque participant,*

– *et I_i (avec $i \in [0..d]$) l'étage numéro i de l'arbre (en commençant à partir du bas de l'arbre).*

Pour chaque noeud (participant) dans le premier niveau I_1, il y aura au maximum m notifications à envoyer. Pour I_2, il y aura $2 * m$, et pour I_{d-1}, il y aura $(d-1) * m$, et ce pour chaque participant. Donc, pour chaque participant nous aurons au maximum $(d-1) * m$ notifications. Dans un arbre quelconque, le nombre moyen de noeuds du niveau i est égal à c^i qui est majoré par c^d. Ainsi, le nombre total de notifications n est majoré par $d * m * c^d$

$$n = O(d * c^d)$$

Nous savons que la profondeur d'un c-arbre (arbre avec c fils en moyenne par noeud) est $d = O(log_c(p))$. Nous trouvons finalement :

$$n = O(log_c(p) * c^{log_c(p)}) = O(p * log_c(p)),\ si\ c > 1.$$

En effet, le cas particulier $c = 1$ décrit le pire des cas représenté dans la figure 5.5.c.

Exemple 5 *Figure 5.6 montre l'arbre généré pour notre exemple de chaîne d'approvisionnement.*

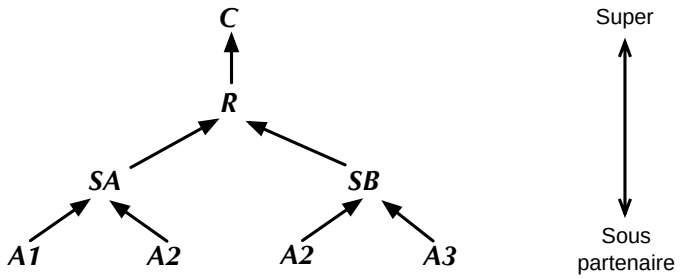


FIGURE 5.6 – Arborescence des partenaires (CPT) pour l'exemple de chaîne d'approvisionnement.

Nous remarquons que le participant $A2$ figure deux fois dans l'arbre. En effet, suivant le chemin emprunté dans le schéma de la chorégraphie (i.e. le choix du fournisseur entre SA et SB fait par le revendeur R), le constructeur $A2$ pourra avoir un des deux fournisseurs comme *super-partenaire*. Nous rappelons que l'affectation des *super-partenaires* se fait au cours de l'exécution de la chorégraphie au moment de la réception du premier message pour chaque participant et ce de façon indépendante pour chaque instance. Donc, ici $A2$ aura, à chaque instance, SA ou bien SB comme *super-partenaire* et non pas les deux. En d'autre termes, le *super-partenaire* de chaque participant est toujours unique. Ceci permet, entre autres, de garantir à l'organisation des partenaires de garder sa structure hiérarchique. De plus, ça permet aussi de ne pas avoir des duplicatas de notifications (e.g. dans cet exemple, une notification venant de $A2$ ne sera reçue qu'une seule fois par R).

5.4.2 Super / sous partenaire transitif

Nous appelons *super-partenaire* transitif tout partenaire avec un lien de parenté indirect dans l'arborescence, c'est-à-dire transitivement (e.g. *super-partenaire* du *super-partenaire*, etc.). Formellement, nous définissons *super-partenaire* transitif comme suit.

Définition 9 (Super-partenaire transitif) *Un participant $P_i \in \mathcal{P}$ est appelé un super − partenaire transitif d'un participant P_j ssi il existe un ensemble de partenaires $\{P_k$, avec $k \in [1..n]\}$ et $n > 0$, tel que $P_i = Super(P_1)$, $P_k = Super(P_{k+1})$ pour tout $k \in [1..n]$ (si $n > 1$), et $P_n = Super(P_j)$.*

Formellement, on note :

$$P_i \in Super_t(P_j)$$

De la même façon, nous définissons *sous-partenaire* transitif comme suit.

Définition 10 (Sous-partenaire transitif) *Un participant $P_i \in \mathcal{P}$ est appelé un sous−partenaire transitif d'un participant P_j ssi il existe un ensemble de partenaires $\{P_k$, avec $k \in [1..n]\}$ et $n > 0$, tel que $P_i \in Sub(P_1)$, $P_k \in Sub(P_{k+1})$ pour tout $k \in [1..n]$ (si $n > 1$), et $P_n \in Sub(P_j)$.*

Formellement, on note :

$$P_i \in Sub_t(P_j)$$

Exemple 6 *Dans notre exemple de chaîne d'approvisionnement (Figure 5.6) :*

$R \in Super_t(A_1)$; $C \in Super_t(A_3)$;

$A_1, A_2, A_3 \in Sub_t(R)$; $SA, SB \in Sub_t(C)$

5.4.3 Vue de supervision externe (EFM-View)

Une vue locale dans une chorégraphie représente la visibilité du point de vue d'un des participants. Elle fait référence au modèle restreint à toutes les interactions d'un seul partenaire. Par défaut, chaque participant est limité seulement à sa propre vue.

En utilisant notre approche hiérarchique d'échange de notifications, chaque EFM reçoit des notifications sur tout changement d'état relatif à l'ensemble de ses *sous-partenaires*. Ainsi, l'EFM de chaque participant aura une vue plus large que la vue locale de ce dernier. Nous appelons cette vue nouvellement créée « vue de supervision externe » (*EFM-View*).

L'*EFM-View* d'un participant P_i inclut toutes les interactions ayant comme émetteur ou receveur un des *sous-partenaires* (direct ou transitif) de P_i. Il comprend également l'ensemble des contraintes sur le séquencement de ces interactions. Formellement, nous définissons la vue de supervision (*EFM-view*) comme suit.

Définition 11 (Vue de supervision (EFM-View)) *Une vue de supervision $\mathcal{V}_i$ d'un participant P_i est un tuple ($\mathcal{IS}_i$, $\mathcal{LS}_i$) avec*
- $\mathcal{IS}_i = \cup_{j \in Sub(P_i)} \mathcal{IS}_j \cup \mathcal{I}_i$
- $\mathcal{LS}_i \subseteq \mathcal{L}$ *est l'ensemble de contraintes sur* $\mathcal{IS}_i$.

Note. Si un participant P_i n'a pas de *sous-partenaires* ($Sub(P_i) = \emptyset$), alors nous avons $\mathcal{IS}_i = \mathcal{I}_i$, ce qui veut dire que sa vue de supervision *EFM-view* est restreinte à sa vue locale ($\mathcal{V}_i = \mathcal{C}_i$).

5.5 Algorithmes de configuration et d'échange de notifications

Pour permettre la mise en oeuvre de notre approche d'échange de notification entre partenaires, nous commençons par distinguer deux phases : la phase de configuration et la phase d'exécution.

5.5.1 Phase de configuration

La phase de configuration consiste à calculer le *super-partenaire* et les *sous-partenaires* directs de chaque participant et la définition des notifications nécessaires pour chaque modèle de chorégraphie (i.e. quelles sont les interactions qui doivent être notifiées et à quel partenaire). L'*algorithme* 5.1 montre un pseudo-code de l'algorithme de configuration.

Algorithme 5.1: Algorithme d'échange de notification (Phase de configuration)

```
1 Require : - C : un modèle de choreographie (une vue globale)
2 - P : un ensemble de participants
3 for (each P_i in P) do
      /* Identification of the Super of P_i                    */
4     I_i0 ← GetFirstInteraction(P_i)
5     Super_i ← GetSource(I_i0)
6     SubPartners(Super_i) ← SubPartners(Super_i) ∪ {P_i}
      /* Définition de l'ensemble des notifications nécessaires */
7     V_i ← ConstructNotificationSet(P_i)
8 End
```

Soit C est une chorégraphie et $\mathcal{P}$ l'ensemble des participants. Pour chacun des participants P_i, nous cherchons la première interaction dans sa vue locale et nous regardons dans le message associé l'émetteur . Ce dernier est considéré comme responsable de l'introduction de P_i dans la collaboration (i.e. son *super-partenaire*). Ensuite, nous définissons, à partir du modèle défini de la chorégraphie, l'ensemble des notifications nécessaires (i.e. celles qui doivent être générées pendant la phase d'exécution).

5.5.2 Phase d'exécution

Un EFM peut être vu comme un cas particulier de machine à états finis dont les transitions sont étiquetées avec des événements d'occurrence de message (envoi ou réception). Commençant par l'état initial de *EFM-view* (quand la chorégraphie est localement instanciée), l'état actuel est mis à jour chaque fois qu'un évènement correspondant à un nouveau message échangé est détecté ou qu'une notification externe est reçue. En tant que tel, chaque EFM suit l'exécution de chaque instance de chorégraphie à laquelle son organisation participe, et ce de façon non intrusive (i.e. écoute passive sans modification dans les messages).

Algorithme 5.2: Algorithme d'échange de notifications (Phase d'exécution)

```
1  Require : - S : Le super partenaire du participant en question
2  - V : La vue de supervision EFM-view
3  for (each event occurrence e_i ) do
4      if (e_i.type ≠ Exception and CheckConformance(e_i)) then
5          UpdateMonitorState(e_i)
6          if (e_i.msrc ≠ Super and e_i.mdst ≠ Super) then /* Generate a new notification if
           e_i is not coming from/going to the Super */
7              if (e_i.type = message exchange) then
8                  n ← GenerateNotification(e_i)
9                  SendNotification(n, Super)
10             else
                   /* e_i.type = notification exchange                                  */
11                 ForwardNotification(e_i, Super)
12     else
13         AlertInternalMonitor()
14         ReportExceptionTo(Super, Sub(P_i))
15 End
```

L'*algorithme* 5.2 décrit le processus d'échange de notifications qui est exécuté au niveau de chaque participant. Cet algorithme montre comment et quand générer, envoyer et transférer les notifications pour chacun des EFMs. Nous soulignons le fait que chaque EFM traite trois types d'événements : (i) les événements liés aux messages échangés avec d'autres partenaires (messages de chorégraphie), (ii) les événements liés au transfert de notifications vers le sommet de l'hiérarchie (i.e. les événements liés aux messages échangés entre des sous-partenaires transitifs), et (iii) les événements indiquant des exceptions.

Selon le type d'événement, l'EFM se comporte différemment :

– **Si le type d'événement est un échange de message**, l'EFM vérifie si c'est conforme à sa vue de supervision ($EFM - view$). Ce contrôle consiste à vérifier si le message reçu ou envoyé est conforme à celui qui est attendu, et ce, en se basant sur les contraintes spécifiées dans le modèle de chorégraphie. Ceci permettra, entre autres, d'éviter l'envoi de notifications inappropriées ainsi que la duplication des notifications. Si le message est

conforme, l'EFM met à jour son statut (noeud actuel dans le graphe de vue de supervision), génère une nouvelle notification indiquant l'occurrence du message et l'envoie à son *super-partenaire*. En effet, nous n'avons pas besoin de notifier le *super-partenaire* si ce dernier figure dans le champ émetteur ou receveur du message car il est déjà au courant de son occurrence.

- **Si le type d'événement est une réception de notification** (i.e. un des *sous-partenaires* nous a envoyé une notification sur une occurrence de message), l'EFM vérifie si c'est conforme à sa vue et met à jour son statut. Ensuite, il transfère la notification à son *super-partenaire*. Dans ce cas aussi, nous optimisons le nombre de notifications envoyées sur le réseau en éliminant celles qui ne sont pas nécessaires (e.g. quand le *super-partenaire* est déjà au courant de l'évènement). L'EFM ne transfère donc pas les évènements ayant son *super-partenaire* comme émetteur ou receveur du message notifié.
- **Si le type d'événement est une exception générée par un autre partenaire**, l'EFM traite localement cette exception et la transmet tout de suite à son *super-partenaire* et à tous ses *sous-partenaires*. Les autres feront de même afin que l'exception soit propagée à tout le monde.

Dans les deux premiers cas, si l'événement n'est pas conforme à celui attendu, une exception est générée, le moniteur interne est alerté. Le *super-partenaire* et tous les *sous-partenaires* sont ensuite informés.

Il est à noter que les *super-partenaires* directs et transitifs peuvent déceler l'exception automatiquement sans avoir à être notifiés par leurs *sous-partenaires* car ils seront bloqués après un certain temps. Une telle propagation rapide des exceptions dans tout l'arbre CPT permettra d'accélérer la détection des situations de blocage.

5.6 Application : Cas d'une chorégraphie d'une chaîne d'approvisionnement

Afin d'illustrer le fonctionnement de notre algorithme d'échange de notifications, nous avons appliqué notre approche sur la chorégraphie chaîne d'approvisionnement introduite dans *Section* 5.2.1.

La *figure* 5.7 montre le diagramme de collaboration complet (modélisé en BPMN 2.0). Ce diagramme montre les vues locales de tous les participants ($C, R, SA, SB, A1, A2$ et $A3$) ainsi que les messages échangés. Nous ajoutons, en bleu, les canaux nouvellement créés pour les notifications, et en vert la définition des *super-partenaires*.

Phase de configuration Après l'exécution de l'algorithme de configuration (*cf. Algorithme* 5.1), on définit l'ensemble des relations entre les participants (i.e. *super/sous-partenaires*). Par la suite, l'ensemble des notifications qui seront envoyées par chaque participant à son *super-*

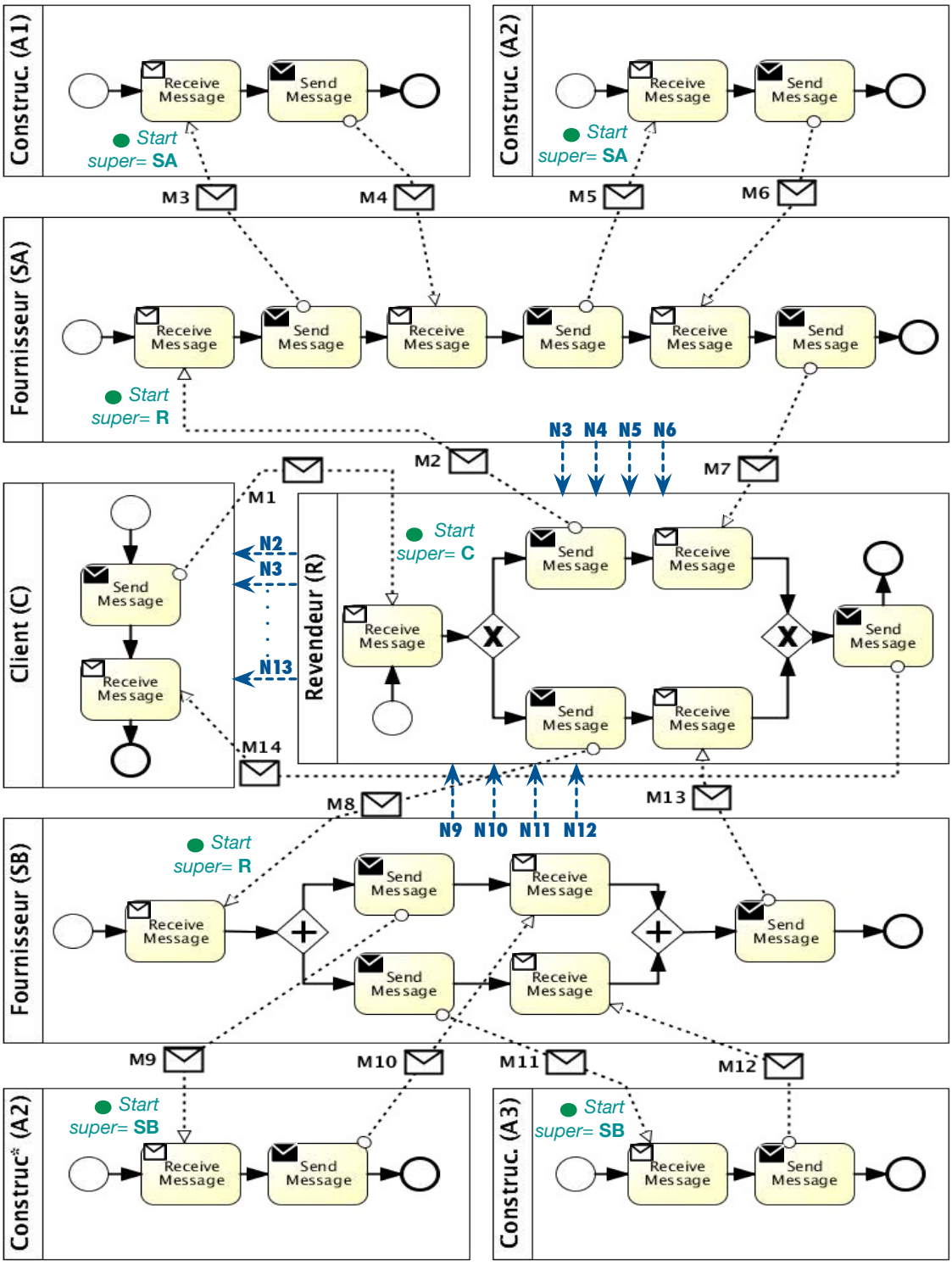


FIGURE 5.7 – Vues locales et échange de notifications : Client (C), Revendeur (R), Fournisseurs (SA,SB) et Constructeurs (A1,A2,A3).

partenaire est automatiquement défini à partir du schéma de la chorégraphie. Dans cet exemple, nous discernons douze types de notifications $N_{2,...,13}$ respectivement associées aux messages $M_{2,...,13}$ définis dans la chorégraphie (e.g. le fournisseur SA définit les quatre notifications $N_{3,...,6}$).

Cela permet de construire, pour chaque participant, une vue de supervision (*EFM-view*) qui va inclure la vue locale et les différentes notifications à recevoir. La *figure* 5.8 montre l'*EFM-view* du Revendeur (R). Dans cette figure, nous pouvons remarquer l'ajout de huit notifications ($N_{3,...,6}$ et $N_{9,...,12}$) qui correspondent à tous les messages échangés par les *sous-partenaires* (directs et transitifs) du Revendeur. Ces notifications lui sont envoyées de la part des *sous-partenaires* directs (i.e. les fournisseurs SA et SB).

Nous remarquons que les constructeurs $A1$, $A2$, et $A3$ n'ont pas de *sous-partenaires* ($Sub(Ai) = \emptyset$). Ainsi, leurs vues de supervision sont égales à leurs vues locales ($\mathcal{V}_{Ai} = \mathcal{C}_{Ai}$).

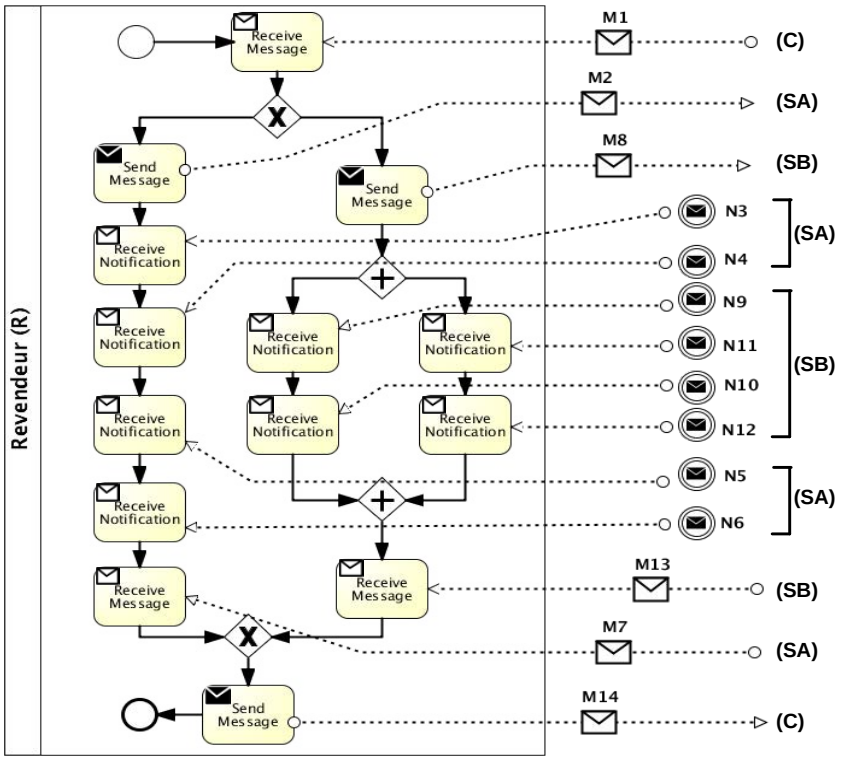


FIGURE 5.8 – Vue de supervision (*EFM-view*) du Revendeur (R).

Phase d'exécution En phase d'exécution et à la réception du premier message, chaque participant fixe son *super-partenaire* (e.g. $Super(R) = C$, $Super(SA) = R$ et $Super(SB) = R$) à l'exception du client C qui n'a pas de *super-partenaire* puisqu'il est l'initiateur de la chorégraphie (racine de l'arborescence).

Ensuite, à chaque réception ou envoi de message par un participant, ce dernier envoie la notification associée à son *super-partenaire* (e.g. quand le fournisseur SA envoie le message M_3

il notifie son *super-partenaire* R avec la notification N_3).

En plus, à chaque réception de notification de l'un des *sous-partenaires*, le participant transfère cette notification à son *super-partenaire* (e.g. quand le revendeur R reçoit la notification N_3, il la transfère à son *super-partenaire* C).

Afin de réduire le trafic et supprimer les transferts des notifications non nécessaires, les notifications associées aux messages ayant le *super-partenaire* comme émetteur ou receveur ne sont pas transférées au *super-partenaire* (il est déjà au courant).

La *figure* 5.9 montre, à l'aide d'un diagramme de séquence, un scénario d'exécution réussie. Les messages de la chorégraphie sont modélisés par des flèches continues. Les notifications échangées, quant à elles, sont modélisées par des flèches en pointillés. *ci*1 est l'identificateur de l'instance. τ_i représente le *timestamp* de l'occurrence du message M_i (temps de réception ou d'envoi). En fait, l'attribut *timestamp* est nécessaire pour la vérification de l'ordre des messages. Les trois derniers attributs de la notification correspondent à l'émetteur, receveur et type du message associé.

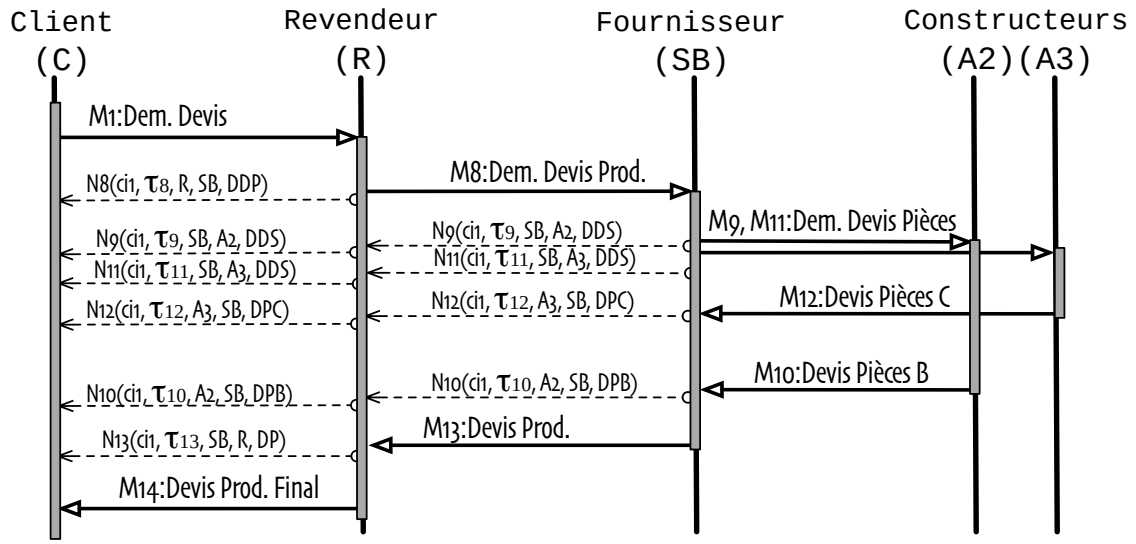


FIGURE 5.9 – Exemple d'une exécution correcte (Diagramme de séquence).

Renforcement des contraintes au niveau de chaque participant Notre approche permet de renforcer les contraintes locales de chaque participant (relatives au séquencement des messages) au niveau de son *super-partenaire* ainsi que tous ses *super-partenaires* transitifs. Par exemple, la contrainte de co-occurrence entre les messages M_9 et M_{11}, vérifiée localement par le fournisseur SB, peut être aussi contrôlée par l'EFM du revendeur R en vérifiant la co-occurrence des deux notifications N_9 et N_{11}. Cette dernière co-occurrence peut pareillement être contrôlée par le client C puisqu'elle figure aussi dans sa vue de supervision.

Détection, gestion et transfert d'exception La *figure* 5.10 montre, en utilisant un diagramme de séquence, un scénario d'exécution avec exception. Après avoir reçu le message M_9, le

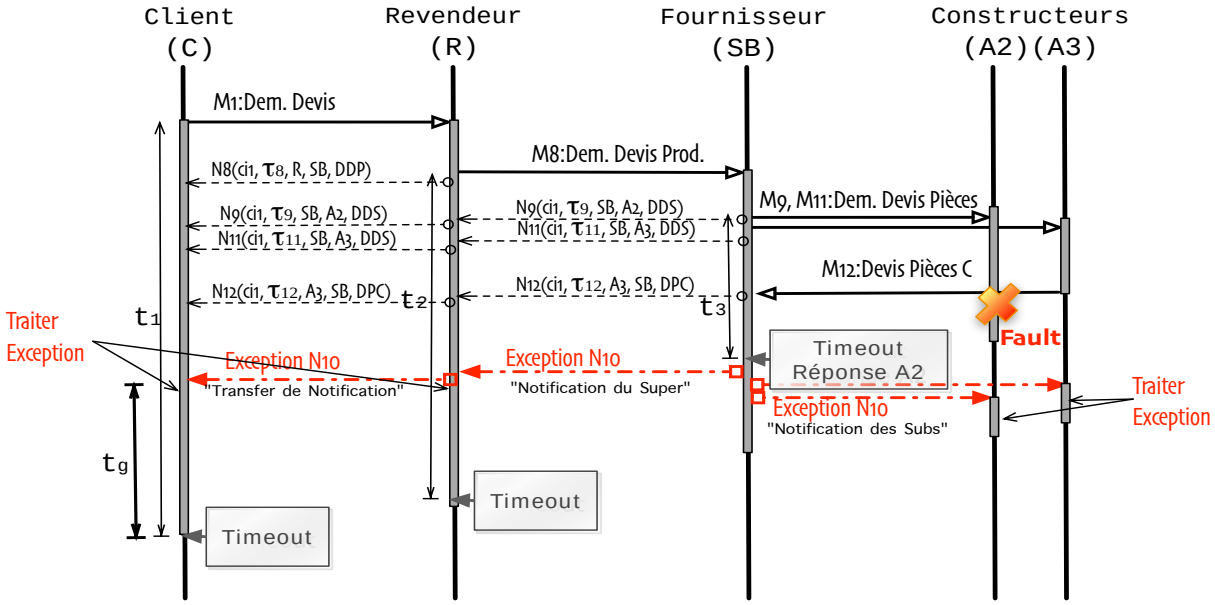


FIGURE 5.10 – Détection, gestion et transfert d'exception (Diagramme de séquence).

constructeur $A2$ échoue à cause d'une erreur interne qui ne peut être remontée au niveau de la chorégraphie. Ainsi, le fournisseur SB reste en attente de sa réponse jusqu'à ce que son timeout expire. A ce moment, SB génère une nouvelle exception et notifie son *super-partenaire* R et ses deux *sous-partenaires* $A2$ et $A3$. A la réception de l'exception, les autres feront de même jusqu'à ce que l'exception se propage dans toute l'arborescence des participants. Ainsi, chacun pourra traiter localement cette exception en faisant ce qu'il a à faire.

Nous pouvons remarquer qu'avec un tel mécanisme, il est possible de détecter chaque exception beaucoup plus rapidement. Le temps t_g montre par exemple le gain en temps au niveau du Client C.

5.7 Limitation et généralisation de l'approche

Dans un cas plus général et en dehors du cadre des chaînes d'approvisionnement, la classification hiérarchique des participants peut paraître peu évidente. En effet, les chorégraphies inter-organisationnelles peuvent comporter des cycles dans les interactions (e.g. trois participants A, B, C avec A qui invoque B, B invoque C qui, lui même, invoque A). Dans de pareils cas, l'affectation des *super-partenaires* peut se faire de façon dynamique (i.e. affectation au premier message reçu). Reste que dans certains cas, nous pouvons avoir des notifications non nécessaires, voire redondantes parfois. Pour pallier ce problème, nous proposons les deux extensions suivantes.

5.7.1 Extension 1 : éliminer les redondances

Les redondances peuvent se produire lorsqu'un participant reçoit plusieurs notifications sur le même message échangé, de la part d'au moins deux de ses *sous-partenaires*. Une telle situation peut se présenter par exemple si deux *sous-partenaires* B et C, d'un même participant A, échangent plusieurs messages entre-eux $M_3, ..., M_n$ (voir *figure* 5.11). A chaque message, chacun des deux participants va donc envoyer une notification à A. Ce dernier va donc recevoir deux notifications pour chaque message échangé et va à son tour les transférer à son *super-partenaire* S.

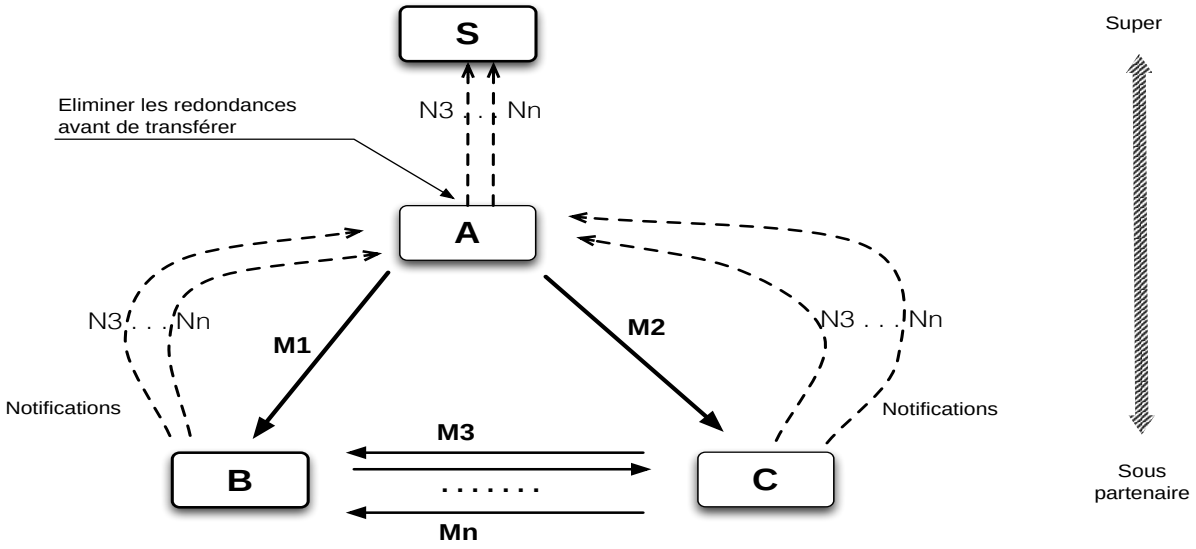


FIGURE 5.11 – Extension de l'approche (Elimination de la redondance des notifications).

Ce type de problème peut être détecté au niveau du participant A (i.e. le point où converge les notifications). Une amélioration possible de l'algorithme de routage des notifications consiste donc à filtrer les notifications sur le même message en éliminant les redondances et en ne transférant qu'un seul exemplaire (ce qui est possible grâce à CEP par exemple).

5.7.2 Extension 2 : éliminer les cycles

Dans le cas d'une chorégraphie qui comporte des cycles, notifier son *super-partenaire* peut parfois être non nécessaire. Il arrive que ce *super-partenaire* soit déjà au courant de l'occurrence de l'événement.

La *figure* 5.12 montre, en utilisant un diagramme de séquence, un scénario d'exécution d'une chorégraphie de transfert d'argent par courrier électronique. Dans cet exemple, nous avons quatre participants : l'expéditeur S, sa banque SB, le bénéficiaire R et sa banque RB. Après avoir appliqué notre mécanisme de classification hiérarchique, nous pouvons obtenir facilement les relations suivantes : $Super(SB) = S$, $Super(R) = SB$ et $Super(RB) = R$.

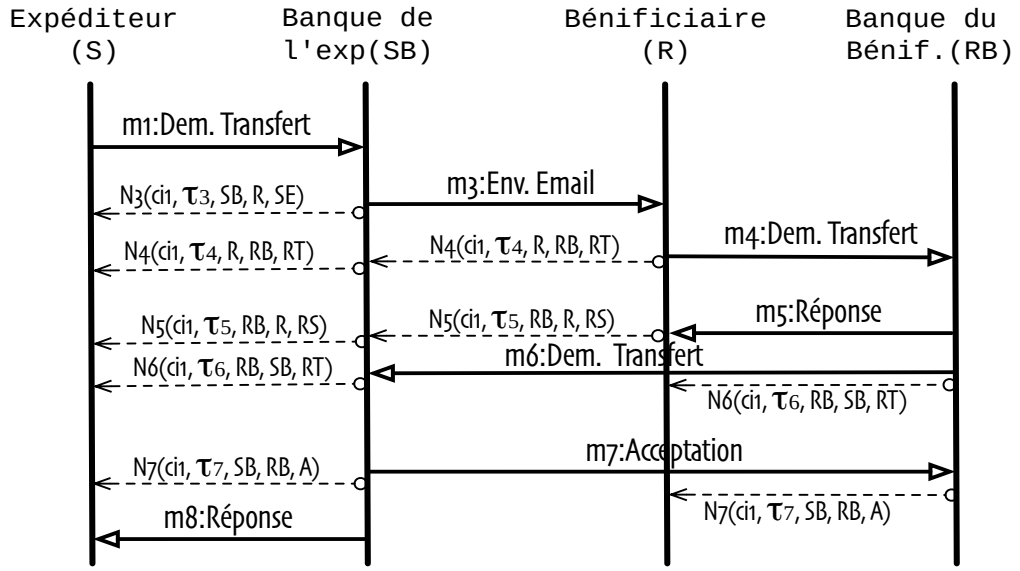


FIGURE 5.12 – Extension de l'approche (Elimination des notifications non nécessaires).

Nous remarquons sur la figure que R n'a pas transféré les deux notifications N_6 et N_7 à son *super-partenaire* SB. En effet, notifier SB n'est pas nécessaire puisqu'il est déjà au courant de l'occurrence du message M_6 (c'est lui le destinataire) et du message M_7 (c'est lui l'émetteur).

5.8 Conclusion

Dans ce chapitre, nous avons présenté un mécanisme d'échange de notifications entre les différents participants d'une chorégraphie. Nous avons montré comment une telle approche pourra permettre une supervision décentralisée de l'exécution d'une chorégraphie, et ce, dans le but d'offrir plus de maîtrise au niveau de chaque organisation participante et de réduire les délais et les coûts en cas d'occurrence d'exceptions. Elle permet aussi d'offrir une traçabilité de l'exécution des processus. En effet, les informations recueillies pourraient être utilisées a posteriori pour mesurer la performance globale du processus et le suivi de la réalisation des objectifs de chaque organisation.

La supervision se fait de façon abstraite (i.e. uniquement sur les échanges de messages inter-organisationnels) sans pour autant dévoiler la logique métier de chaque entreprise. La propagation hiérarchique des données de supervision permet de limiter l'exposition des données et d'éviter une surcharge du réseau (notification sélective).

Nous ajoutons aussi que notre approche est non intrusive dans le sens où elle se base sur l'écoute passive des messages de la chorégraphie sans aucune modification de la structure ou du contenu des messages (i.e. les processus ne sont pas altérés et ne sont pas conscients de la supervision et l'échange de notifications). L'implantation des EFMs est modulable et respecte bien le principe de séparation des préoccupations "*Separation of concerns*" (i.e. séparation des aspects non fonctionnels des aspects fonctionnels).

Dans le chapitre suivant, nous allons montrer comment détecter, au niveau de chaque participant, les violations pendant la phase d'exécution. Pour cela, nous proposons une approche qui permettra de générer automatiquement un ensemble optimisé de règles qui doivent être vérifiées à chaque détection d'un nouvel événement.

6 Génération automatique et optimisée de requêtes de supervision

«Plus on pédale moins fort, moins on avance plus vite.»
Coluche

Sommaire

Introduction

Ce chapitre est dédié à l'analyse et l'évaluation des événements générés par les échanges de messages entre les partenaires. Le but est de vérifier si le comportement réel des entités en interaction adhère efficacement aux contraintes métier imposées. Pour y parvenir, notre approche évènementielle permet la génération automatique d'un ensemble optimisé de requêtes à partir d'un schéma d'interaction défini. Cette approche permet de vérifier ensuite la conformité des séquences d'interaction par rapport au modèle de la chorégraphie, et ce, à chaque échange de message avec ses partenaires. La vérification se fait au niveau de chaque participant par rapport à sa vue de supervision *EFM-view* (cf. chapitre précédent). Nous visons à réduire au maximum le nombre de règles sur le séquencement des interactions en ne fixant que des contraintes entre les interactions voisines.

La section 6.1 donne un aperçu sur l'organisation des composants et les différentes étapes de l'approche. La section 6.2 explique comment CEP peut être utilisé pour la supervision des chorégraphies. La section 6.3 présente le mécanisme de fragmentation structurelle et son utilité. La section 6.4 illustre le mécanisme de génération de règles. Une évaluation de notre approche en terme de nombre de règles générées est également présentée dans cette section. La section 6.5 montre comment détecter les violations avec les anti-patrons. La section 6.6 propose un mécanisme d'agrégation de violations et la section 6.7 conclut ce chapitre.

6.1 Aperçu sur l'approche

L'approche proposée dans ce chapitre repose uniquement sur les changements d'état d'une chorégraphie, i.e. quand un message de chorégraphie est envoyé ou reçu. Elle permet le suivi de la conformité des interactions entre les participants d'une manière discrète. En d'autres termes, les messages échangés ne sont pas modifiés et les pairs ne sont pas conscients des moniteurs (i.e. écoute passive). Elle repose donc sur les évènements générés pour chaque échange de message détecté. La *figure* 6.1 donne un aperçu sur l'approche proposée. Nous distinguons deux phases : une phase de configuration et une phase d'exécution.

Phase de configuration (pré-exécution) : se déroule une fois par modèle de chorégraphie (i.e. pour chaque EFM-*view*).

1. **Génération de l'arbre de structure (CST) :** La fragmentation du modèle de chorégraphie en blocs SESE (single entry / single exit) [PVV11] nous permet de construire ce que nous appelons un arbre de structure de chorégraphie CST (Choreography Structure Tree). Cet arbre représente l'arborescence des blocs et est unique pour chaque modèle de chorégraphie.

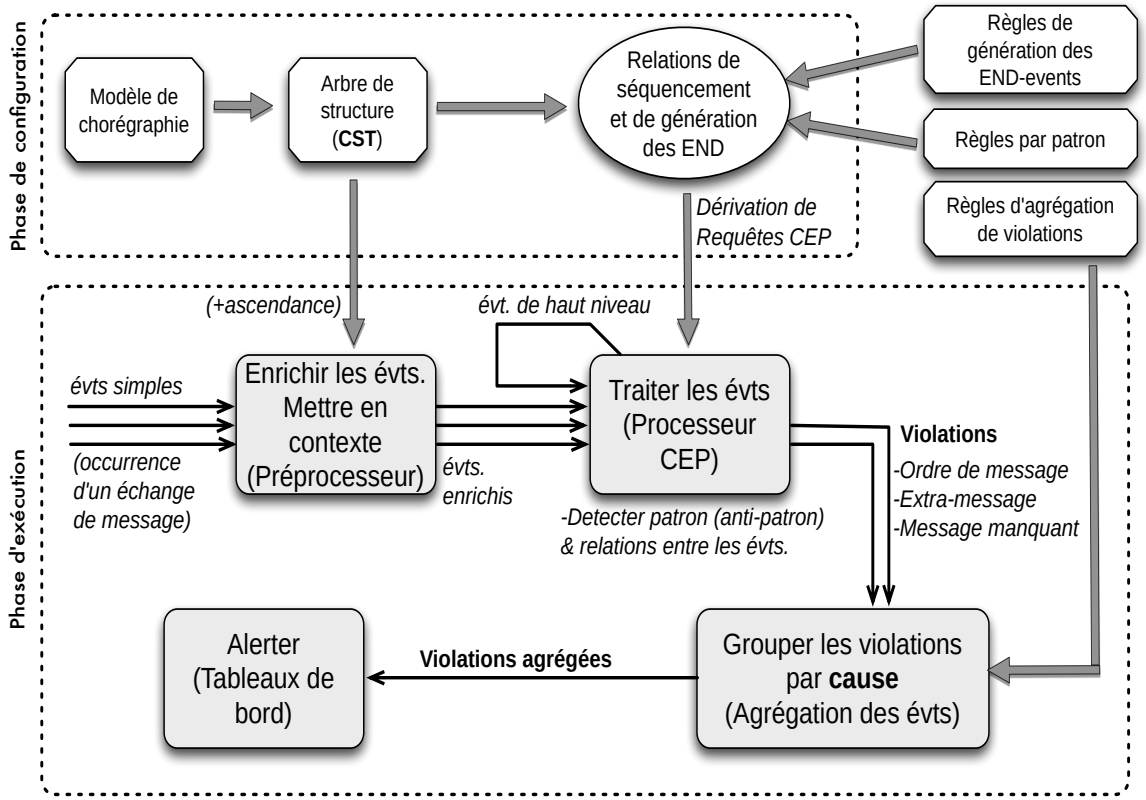


FIGURE 6.1 – Aperçu sur l'approche proposée pour la génération de requêtes et détection de violations dans un environnement CEP.

2. **Génération des requêtes :** A partir de l'arbre généré et en fonction des règles de génération par patron, un ensemble optimal de requêtes CEP spécifiques au modèle en question est automatiquement généré. Nous distinguons deux types de requêtes générées : requêtes pour la vérification du séquencement des événements et requêtes de génération des évènements de haut niveau (i.e. les évènements de fin d'exécution d'un bloc *END-events*).

Phase d'exécution : se déroule pour chaque instance de chorégraphie.

1. **Production d'événements de base :** Les événements sont générés à chaque envoi ou réception d'un message ou collectés par l'EFM (e.g. reçus de la part des *sous-partenaires*).
2. **Enrichissement des évènements :** Avant d'être transmis à un moteur CEP, ces événements de base sont enrichis par un pré-processeur en ajoutant un nouveau champ « ascendancy » qui contient la liste des blocs ascendants (i.e. contenant l'interaction associée) dans l'arbre CST qui est dérivé du modèle de chorégraphie.
3. **Traitement des évènements enrichis :** le traitement se fait par un processeur CEP qui va détecter, à partir des requêtes générées dans la phase de configuration, des patrons de violations (i.e. des anti-patrons). Le processeur exécute aussi en parallèle des requêtes prédéfinies pour la génération des événements de fin de blocs (END-

events). Ces événements complexes (ou de haut niveau d'abstraction) nouvellement générés sont aussi traités et évalués par rapport aux requêtes CEP.

4. **Agrégation des violations :** Afin de minimiser au maximum le nombre d'alertes, les violations détectées sont regroupées suivant leurs causes principales. En d'autres termes, toutes les violations ayant la même cause principale sont agrégées en une seule violation contenant la date, le type et les différentes répercussions. Les violations agrégées sont ensuite envoyées pour être affichées dans les tableaux de bord de supervision.

6.2 Supervision événementielle dans un environnement CEP

Pour repérer les violations de conformité au cours de l'exécution d'une chorégraphie dans un environnement CEP, les contraintes de séquencement des messages doivent être transformées sous forme de requêtes événementielles. La trace d'exécution (i.e. le flux d'événements générés) est vérifiée par rapport à ces requêtes, et ce, à chaque arrivée d'un nouvel événement au cours de l'exécution. Cependant, la construction de l'ensemble de requêtes n'est pas une tâche facile surtout lorsqu'il s'agit de processus complexes et de grande taille. En fait, l'augmentation du nombre de requêtes augmente le risque d'erreurs (humaine), alourdit le moteur CEP et ralentit donc la vitesse de détection de violations. En plus, avec l'évolution et les mises à jour des processus, la maintenance aussi devient difficile et les requêtes risquent de se superposer, dupliquer, voire d'être conflictuelles. Pour faire face à ces problèmes, il faut trouver comment générer ces requêtes d'une manière automatique et efficace.

Dans le domaine des processus métier, une approche se basant sur le concept appelé profil de causalité comportementale (*causal behavioral profiles*) montre comment résoudre le problème en proposant de générer une règle entre chaque couple d'activités [WZM+11, WPMW10]. Cette approche fournit une abstraction du comportement défini par un modèle de processus en capturant les caractéristiques comportementales par des relations binaires. Autrement dit, une relation est fixée pour chaque couple d'activités indiquant si les deux activités doivent s'exécuter dans un ordre strict, exclusivement l'un de l'autre ou sans aucune contrainte entre elles. Ensuite, chaque relation d'ordre entre deux activités est transformée en une requête CEP qui permet de détecter si l'inverse (appelé anti-patron) de cette relation se produit. Par exemple, une relation d'exclusion entre deux activités est violée si et seulement si une requête détecte l'exécution de deux activités ensemble dans la même instance.

Dans une chorégraphie, une interaction peut être considérée comme l'activité de base. Ainsi, le même type de relation peut être utilisé. Par exemple, dans le modèle présenté dans la *figure* 6.2, les interactions (I_1) et (I_3) sont dans un ordre strict. Les interactions (I_4) et (I_7) sont exclusives l'une à l'autre. Cependant, il n'y a pas de contrainte de séquencement entre les interactions (I_4) et (I_8) par exemple (i.e. l'une peut se produire avant l'autre). Lorsqu'il n'y a pas de contrainte de séquencement entre deux interactions, il n'est pas nécessaire de définir une relation qui symbolise

l'absence d'une contrainte d'ordre. Par conséquent, cette relation n'est pas considérée lors de la surveillance d'exécution de chorégraphie.

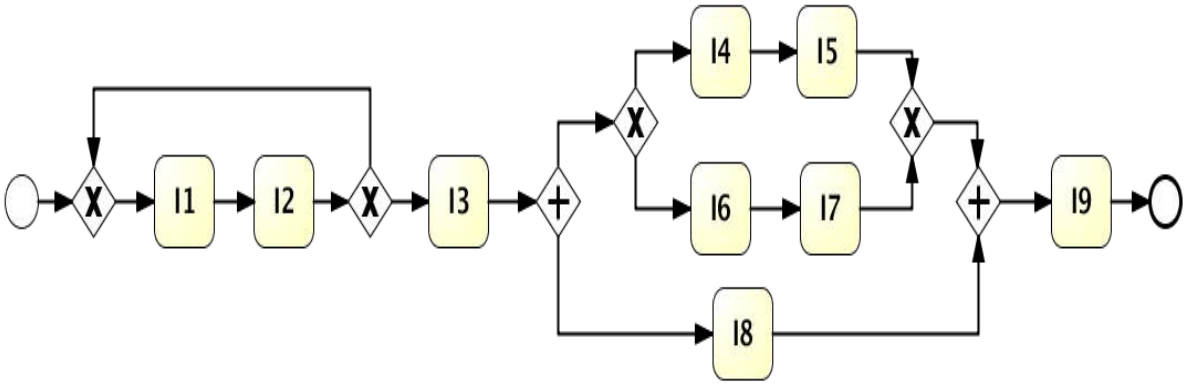


FIGURE 6.2 – Exemple de vue de supervision

Suivant cette approche, le nombre de relations (i.e. une relation pour toute combinaison de couple de d'interaction), et donc de requêtes, est en $O(n^2)$ pour toute chorégraphie ayant n interactions (précisément $n(n+1)/2$ si on ne prend pas les règles qu'on peut déduire symétriquement). La *table* 6.1 montre l'ensemble des relations d'ordre binaires de notre exemple de chorégraphie (nécessairement sous forme d'une matrice symétrique). Nous utilisons les notations =>, <=, + et II pour désigner, respectivement, les relations «précède», «suit», «exclusif à» et «parallèle».

Nous remarquons qu'il est impossible de fixer directement les relations entre les interactions I_1 et I_2. Ces deux relations sont contenues dans une branche qui peut se répéter en boucle. Ainsi, nous pouvons avoir dans la trace d'une même instance l'occurrence de I_1 après celle de I_2, et inversement. Cependant, après chaque I_1 il doit y avoir un I_2. Le nombre d'itérations, quant à lui, peut ne pas être connu qu'au moment de l'exécution.

Nous pouvons également remarquer qu'il y a beaucoup de relations non nécessaires (i.e. qui peuvent être déduites indirectement). Par exemple, puisque «(I_1) précède (I_3)» et «(I_3) précède (I_4)», nous pouvons directement déduire que «(I_1) précède (I_4)». Nous raisonnons de manière identique pour (I_1) par rapport à (I_5),..., (I_9). Cela est imputable du fait que certaines contraintes de séquencement sont transitives (e.g. les relations «précède» et «suit»).

Au moment de la génération des requêtes CEP, nous risquons d'avoir un nombre important de requêtes qui se chevauchent. En plus, lorsqu'une interaction est effectuée à un stade inattendu (i.e. violation), les requêtes générées peuvent entraîner de multiples alertes redondantes. En effet, lorsqu'il y a violation, nous pouvons avoir plusieurs requêtes qui lèvent la même exception et plusieurs alertes seront générées et envoyées pour la même violation. Par exemple, si un message M_9 correspondant à l'interaction I_9 est envoyé avant tous les autres messages, nous aurons 8 violations (I_9 par rapport I_1, I_9 par rapport I_2, etc.) alors qu'il s'agit d'une seule violation avec 7 répercussions.

Ce problème a été traité a posteriori dans [WZM+11]. Des requêtes supplémentaires ont été

R()	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
I_1	?	?	=>	=>	=>	=>	=>	=>	=>
I_2	?	?	=>	=>	=>	=>	=>	=>	=>
I_3	<=	<=	+	=>	=>	=>	=>	=>	=>
I_4	<=	<=	<=	+	=>	+	+	II	=>
I_5	<=	<=	<=	<=	+	+	+	II	=>
I_6	<=	<=	<=	+	+	+	=>	II	=>
I_7	<=	<=	<=	+	+	<=	+	II	=>
I_8	<=	<=	<=	II	II	II	II	+	=>
I_9	<=	<=	<=	<=	<=	<=	<=	<=	+

TABLE 6.1 – Relations binaires d'ordre entre les interactions

ajoutées afin d'identifier la cause principale de tout un ensemble de violations. Cependant, même avec cette amélioration, la détermination de l'ensemble de violations en relation reste toujours une phase délicate et varie pour chaque patron de séquencement. De plus, le problème des boucles reste toujours non traité.

Pour tenter de limiter ces faiblesses, nous proposons une nouvelle approche. Au lieu de fixer une contrainte entre chaque couple d'interaction, notre approche consiste à fixer des contraintes seulement entre les interactions de voisinage. Pour ce faire, une fragmentation structurelle du modèle de chorégraphie est nécessaire.

6.3 Fragmentation structurelle d'une chorégraphie et événements de blocs

Afin de réduire le nombre de contraintes, et donc le nombre de requêtes sur les événements, nous proposons de décomposer le modèle de chorégraphie en nous inspirant de la décomposition des processus métier de l'arbre raffiné de structure de processus (R-PST : *Refined Process Structure Tree*) défini dans [PVV11]. Cette technique consiste à faire l'analyse et la découverte de la structure du graphe de flux de contrôle d'un processus donné. Elle permet une décomposition

hiérarchique d'un modèle de processus en un ensemble de blocs *canoniques* avec un seul flux d'entrée et un seul flux de sortie (SESE : *Single Entry/ Single Exit*). Un bloc *canonique* est défini comme suit [VVL07].

Définition 12 (Bloc canonique) *Un bloc B est dit* non-canonique *si et seulement s'il existe trois blocs X, Y et Z avec ces trois conditions réunies :*

- *X et Y sont en séquence,*
- *$B = X \cup Y$,*
- *B et Z sont en séquence.*

Sinon, B est dit canonique.

Il a été prouvé que la décomposition R-PST est unique, modulaire et peut être calculée en temps linéaire [PVV11]. En effet, les blocs *canoniques* ne se chevauchent jamais. Prenant deux blocs, soit un bloc est entièrement contenu dans l'autre, soit les deux blocs sont totalement disjoints.

6.3.1 Arbre de structure de chorégraphie (CST)

Suivant l'idée de décomposition R-PST, nous analysons le graphe de chorégraphie et nous le fragmentons en une hiérarchie de blocs *canoniques* SESE. La *figure* 6.3 illustre le résultat de la décomposition de notre exemple motivation.

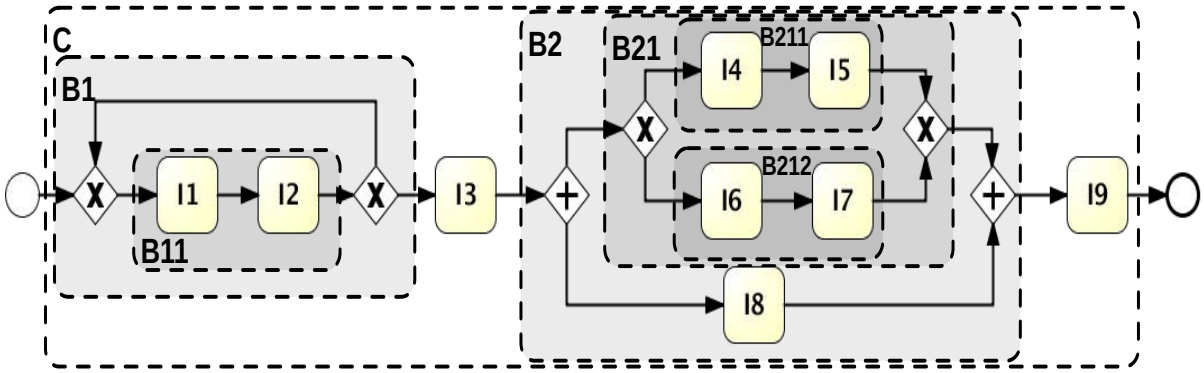


FIGURE 6.3 – Décomposition en fragments

Le résultat d'une telle décomposition peut être représenté sous forme d'un arbre que nous appelons arbre de structure de chorégraphie (CST : *Choreography Structure Tree*). La racine de l'arbre représente le plus grand bloc qui contient l'ensemble du graphe d'interactions. Les blocs descendants d'une séquence sont classés de gauche à droite (i.e. de l'entrée vers la sortie). La *figure* 6.4 montre l'arbre CST généré de notre exemple de la *figure* 6.2. Nous concrétisons les noeuds internes de l'arbre en les annotant avec le type de patron de séquencement entre

les descendants directs, à savoir «Seq» pour la séquence, «And» pour le patron parallèle, «Ex» pour l'exclusion et «Loop» pour désigner une boucle. Ainsi, nous avons explicitement établi les relations structurelles de voisinage entre les blocs.

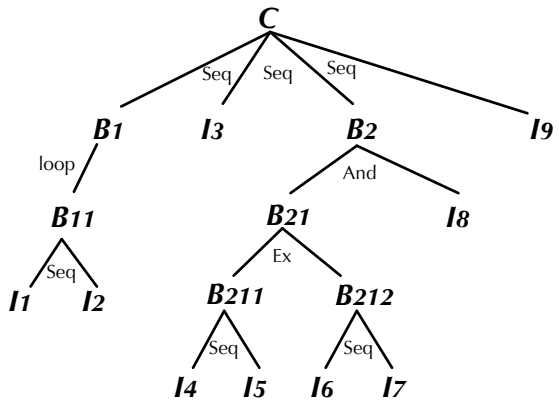


FIGURE 6.4 – Arbre de structure de chorégraphie (CST)

6.3.2 Enrichissement des événements

Après avoir généré l'arbre CST, nous proposons d'enrichir chaque événement en ajoutant un nouveau champ appelé *ascendancy* contenant la liste de tous les blocs supérieurs de l'interaction observée. Par exemple, l'événement lié à l'interaction (I_4) dans notre exemple de motivation (Figure 6.3) est marqué par une séquence de blocs $< B_{211}, B_{21}, B_2 >$, car il appartient, respectivement, aux blocs B_{211}, B_{21} et B_2. En fait, il s'agit d'une sorte de marquage (*event tagging*) de chaque événement afin d'indiquer son emplacement supposé dans l'arbre CST.

Après l'étape d'enrichissement, chaque événement « simple » (*cf. Définition* 5) est transformé sous la forme suivante :

$$E = (Eid, Cid, \text{Iid}, < ascendancy >, TS)$$

Cette étape consiste à préparer les évènements et à les mettre en contexte avant qu'ils soient traités par le processeur CEP principal. Elle peut être réalisée par l'intermédiaire d'un pré-processeur (i.e. un composant placé en avant du processeur principal) accomplissant à la fois le filtrage et l'enrichissement des événements. En effet, à chaque réception d'un nouvel événement, le pré-processeur cherche dans l'arbre CST la liste des blocs qui contiennent l'interaction correspondante à l'événement arrivé et attache le résultat trouvé dans le champ <ascendancy> nouvellement créé. La *figure* 6.5 illustre comment enrichir les événements à partir de l'arbre CST de la *figure* 6.4. Le flux des événements enrichis produit par le pré-processeur servira d'entrée au processeur principal d'événement.

$$
\begin{aligned}
&I_1 \rightarrow generate(E_{id}, C_{id}, I_1, < \mathbf{B_{11}}, \mathbf{B_1} >, TS)\\
&I_2 \rightarrow generate(E_{id}, C_{id}, I_2, < \mathbf{B_{11}}, \mathbf{B_1} >, TS)\\
&I_3 \rightarrow generate(E_{id}, C_{id}, I_3, <>, TS)\\
&I_4 \rightarrow generate(E_{id}, C_{id}, I_4, < \mathbf{B_{211}}, \mathbf{B_{21}}, \mathbf{B_2} >, TS)\\
&I_5 \rightarrow generate(E_{id}, C_{id}, I_5, < \mathbf{B_{211}}, \mathbf{B_{21}}, \mathbf{B_2} >, TS)\\
&I_6 \rightarrow generate(E_{id}, C_{id}, I_6, < \mathbf{B_{212}}, \mathbf{B_{21}}, \mathbf{B_2} >, TS)\\
&I_7 \rightarrow generate(E_{id}, C_{id}, I_7, < \mathbf{B_{212}}, \mathbf{B_{21}}, \mathbf{B_2} >, TS)\\
&I_8 \rightarrow generate(E_{id}, C_{id}, I_8, < \mathbf{B_2} >, TS)\\
&I_9 \rightarrow generate(E_{id}, C_{id}, I_9, <>, TS)
\end{aligned}
$$

FIGURE 6.5 – Enrichissement des événements

6.3.3 Événements de haut niveau (END-events)

Après avoir enrichi les événements de base avec leurs blocs supérieurs, les contraintes de séquencement peuvent désormais être appliquées non seulement sur les événements mais aussi sur les blocs. Nous pouvons ainsi fixer des relations binaires entre deux événements, entre deux blocs, ou encore entre un événement et un bloc.

En agissant ainsi, nous fixons uniquement des relations entre les voisins directs et faisons donc décroitre donc le nombre de requêtes de façon exponentielle du fait de la structure arborescente des blocs. Ceci représente un des avantages majeurs de notre approche.

Pour permettre d'établir des relations entre blocs, nous avons besoin de générer des événements de plus haut niveau qui symbolisent l'exécution de tout un bloc. Pour ce faire, nous avons choisi de définir une nouvelle structure d'événement que nous appelons *évènement de fin de bloc* (*END-events*). Nous notons ainsi $End(B)$ l'évènement de fin d'exécution du bloc B.

Définition 13 (Événements de haut niveau) *Dans un environnement CEP, un événement est dit de haut niveau s'il représente une abstraction d'autres événements, appelés ses membres [Luc02b]. Dans notre cas, les membres d'un événement $End(B)$ sont tous les événements liés aux interactions et aux blocs contenus dans le bloc B.*

Exemple 7 *Dans la figure 6.3, les événements associés à I_1, I_2 et $End(B_{11})$ sont les membres de l'événement de plus haut niveau $End(B_1)$.*

Remarque : Un événement de haut niveau ne peut avoir lieu que lorsque tous ses membres ont été détectés. Il prouve donc l'occurrence de ses membres. Ainsi, une fois qu'un événement de fin de bloc a été généré, tous ses membres (i.e. tous les événements intérieurs au bloc) peuvent être supprimés de la trace d'exécution.

6.4 Mécanisme de génération automatique

Dans cette section, nous illustrons le mécanisme de génération de règles. Nous commençons par spécifier les contraintes de séquencement sous forme de relations binaires. Ensuite, nous

définissons les règles de voisinage sur les blocs ainsi que les règles de base pour la génération pour chaque type de patron. Enfin, nous évaluons notre approche en terme de nombre de règles générées.

6.4.1 Contraintes sous forme de relations binaires et règles de voisinage de blocs

Pour spécifier les contraintes de séquencement des événements, nous définissons deux types de relations : « séquence » et « exclusion ».

Remarque : La relation « parallèle » n'est pas nécessaire dans notre approche. Nous allons voir par la suite comment une violation d'une telle relation peut être détectée à l'aide des événements de fin de bloc.

Relation « séquence »

Cette relation permet de définir un ordre strict entre deux événements, deux blocs ou encore un bloc et un événement. La *table* 6.2 montre les différentes combinaisons possibles.

$Seq(I_i, I_j)$	Cette relation stipule que pour deux événements I_i et I_j, I_j ne se produit jamais avant I_i.
$Seq(B_i, B_j)$	Cette relation stipule que pour deux événements I_i et I_j, respectivement marqués avec B_i et B_j, I_j ne se produit jamais avant la fin de B_i ($End(B_i)$). (voir *Figure* 6.6)
$Seq(I_i, B_j)$	Cette relation stipule que pour deux événements I_i et I_j, avec I_j marqué par B_j, I_j ne se produit jamais avant I_i.
$Seq(B_i, I_j)$	Cette relation stipule que pour deux événements I_i et I_j, avec I_i marqué par B_i, I_j ne se produit jamais avant la fin de B_i ($End(B_i)$).

TABLE 6.2 – La relation « séquence »

Propriété 1 *La relation $Seq(I_i, End(B_i))$, avec I_i marqué par B_i, est toujours valable. En d'autres termes, I_i ne se produit jamais après la fin de bloc B_i.*

Propriété 2 *La relation « séquence » est transitive.*

$$\underline{Si}\ Seq(I_i, I_j)\ et\ Seq(I_j, I_k)\ \underline{alors}\ Seq(I_i, I_k)$$

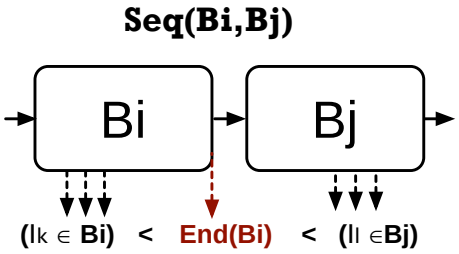


FIGURE 6.6 – La relation « séquence » entre deux blocs voisins.

Relation « exclusion » :

Une relation d'exclusion entre deux événements consiste à spécifier qu'ils ne se produisent jamais ensemble au sein d'une même instance d'une chorégraphie. Cette relation est valable aussi entre deux blocs ou encore entre un bloc et un événement. La *table* 6.3 montre les différentes combinaisons possibles.

$Ex(I_i, I_j)$	Cette relation stipule que pour deux événements I_i et I_j, I_i ne se produit jamais avec I_j dans la même instance de chorégraphie.
$Ex(B_i, B_j)$	Cette relation stipule que pour deux événements I_i et I_j, respectivement marqués avec B_i et B_j, I_i ne se produit jamais avec I_j dans la même instance de chorégraphie.
$Ex(I_i, B_j)$ $Ex(B_j, I_i)$	Cette relation stipule que pour deux événements I_i et I_j, avec I_j marqué par B_j, I_i ne se produit jamais avec I_j dans la même instance de chorégraphie.

TABLE 6.3 – La relation « exclusion »

Propriété 3 *La relation $Ex(I_i, I_i)$) est toujours valable (en dehors des boucles atomiques). En d'autres termes, I_i ne se produit qu'une seule fois au sein de la même instance sauf si l'interaction est en boucle sur elle même (i.e. boucle atomique).*

Propriété 4 *La relation « exclusion » est commutative.*

$$\underline{Si}\ Ex(I_i, I_j)\ \underline{alors}\ Ex(I_j, I_i)$$

6.4.2 Règles de génération par patron

Selon les patrons de séquencement dans un modèle de chorégraphie (i.e. séquence, parallèle, exclusion et itération), nous définissons les règles de génération automatique des relations binaires ainsi que les événements de fin de bloc. La *figure* 6.7 illustre les règles pour chaque type de patron.

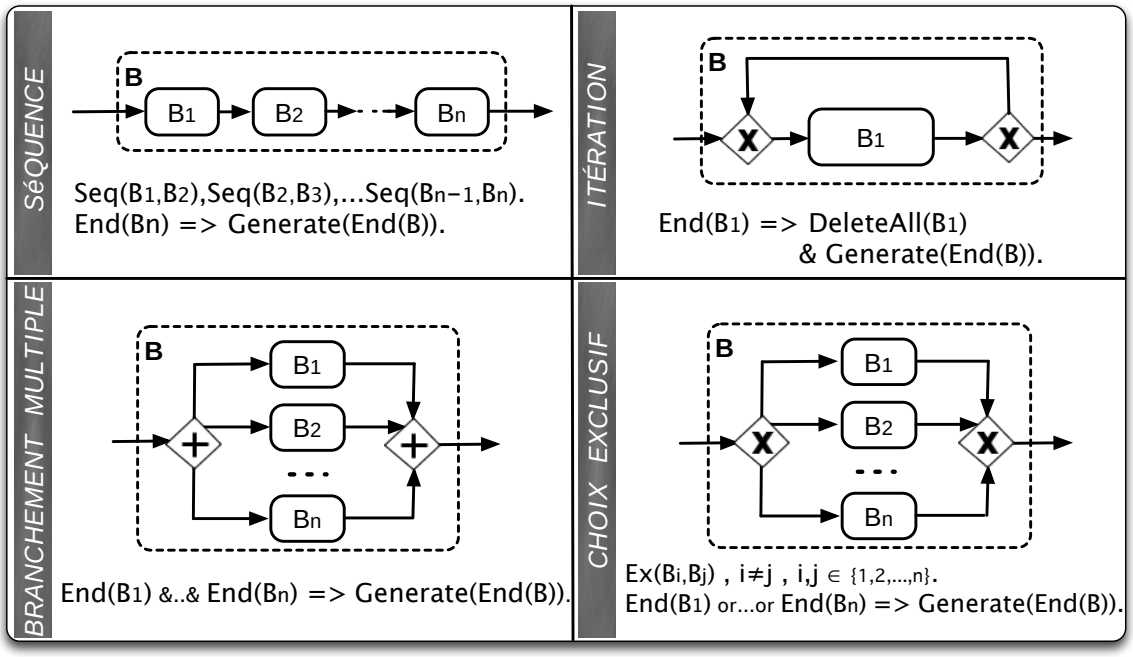


FIGURE 6.7 – Règles par patron

Bloc séquence

Lorsque nous avons un bloc B qui contient n blocs d'interaction en séquence $B_1, ..., B_n$, nous générons uniquement $n-1$ relations binaires entre chaque deux blocs consécutifs, c'est-à-dire,

$$Seq(B_i, B_{i+1}),\ i \in \{1..n-1\}.$$

L'achèvement du dernier bloc dans la liste provoque la génération de l'événement de fin du bloc B, c'est-à-dire,

$$End(B_n) => Generate(End(B)).$$

Bloc parallèle

Dans le cas d'un bloc B qui contient n blocs d'interaction en parallèle $B_1, ..., B_n$, une violation se produit lorsqu'on découvre la non exécution ou le non achèvement d'au moins un des blocs internes. Ainsi, une telle violation peut être détectée seulement à l'achèvement de l'ensemble du bloc B et au commencement du bloc qui le suit.

L'événement de fin de bloc B n'est généré qu'après la génération des événements de fin de bloc de tous les blocs fils, c'est-à-dire,

$$End(B_1)\&...\&End(B_n) => Generate(End(B)).$$

Bloc de choix exclusif

Dans le cas d'un bloc B de choix exclusif entre n blocs d'interaction $B_1, ..., B_n$, un et un seul chemin doit être emprunté. La décision sur le choix de la branche sélectionnée est prise par le processus interne d'un des participants de la chorégraphie, et ce, au cours de son exécution. Ici, une violation se produit lorsqu'on détecte, dans la même instance, au moins deux événements E_p et E_q appartenants à deux branches différentes (i.e. marqués respectivement par deux blocs B_i et B_j, avec $i \neq j$). Pour détecter une telle violation, nous sommes donc contraints de générer une relation d'exclusion entre chaque couple de blocs, c'est-à-dire,

$$Ex(B_i, B_j),\ i \neq j\ ,\ i, j \in \{1..n\}.$$

L'événement de fin de bloc B est généré après la génération d'un des événements de fin de bloc des blocs fils, c'est-à-dire,

$$End(B_1)\ or ... or\ End(B_n) => Generate(End(B)).$$

Bloc d'itération

Dans une chorégraphie, un bloc d'itération (aussi appelé *boucle*) B désigne qu'un fragment peut s'exécuter une ou plusieurs fois selon un choix fait par un des participants à chaque itération. Ce fragment est appelé corps de l'itération (ici, B_1). Afin de permettre la répétition des interactions sans soulever des violations inappropriées, nous avons besoin d'effacer, à la fin de chaque itération, la trace des tous événements de l'itération accomplie et de générer un événement de fin de bloc $End(B_1)$ pour permettre au bloc suivant d'être exécuté, c'est-à-dire,

$$End(B_1) => DeleteAll(B_1)\ \&\ Generate(End(B)).$$

Ici, la fonction $DeleteAll(B_1)$ supprime de la trace tous les événements marqués par B_1 (i.e. contenant B_1 dans leurs champ « ascendancy »).

6.4.3 Génération automatique des règles

En appliquant les quatre règles définies dans la section précédente, un ensemble de relations optimisé et spécifique à chaque modèle de chorégraphie peut être généré automatiquement en explorant l'arbre CST niveau par niveau. Dans chaque niveau de l'arbre, les relations binaires entre les noeuds voisins sont générées selon l'annotation du noeud parent qui définit le type de patron de séquencement. Ensuite, les règles de génération des événements de fin de bloc sont à leur tour générées.

La *table* 6.4 affiche l'ensemble des relations et des règles générées à partir de notre exemple de modèle de chorégraphie (revoir l'arbre CST *figure* 6.4). Nous classifions les règles par leur niveau (i.e. profondeur dans l'arbre CST). Nous séparons en deux colonnes les relations binaires et les règles de génération.

Niveau	Relations	Règles de génération
1	$Seq(B_1, I_3)$ $Seq(I_3, B_2)$ $Seq(B_2, I_9)$	$I_9 \Rightarrow generate(End(C))$
2		$End(B_{11}) \Rightarrow deleteAll(B_{11})$ & $generate(End(B_1))$ $End(B_{21})$ & $I_8 \Rightarrow generate(End(B_2))$
3	$Seq(I_1, I_2)$ $Ex(B_{211}, B_{212})$	$I_2 \Rightarrow generate(End(B_{11}))$ $End(B_{211})$ or $End(B_{212}) \Rightarrow generate(End(B_{21}))$
4	$Seq(I_4, I_5)$ $Seq(I_6, I_7)$	$I_5 \Rightarrow generate(End(B_{211}))$ $I_7 \Rightarrow generate(End(B_{212}))$

TABLE 6.4 – Relations et règles de fin de bloc générées

Remarque : Nous pouvons ajouter la relation $Ex(I_i, I_i)$ pour imposer le fait que chaque interaction ne peut pas se produire plusieurs fois. Cette relation n'affecte pas les événements associés à des interactions d'un bloc de boucle car ces événements sont supprimés à la fin de chaque itération.

6.4.4 Évaluation du nombre de requêtes générées

Nous pouvons remarquer une large différence en terme de nombre de règles et relations générées par rapport à l'approche classique présentée dans la section 6.2. En effet, rien que sur cet exemple, utiliser notre approche permet de réduire le nombre de relations binaires à 7 (voir 2ème colonne de la *table* 6.4). En plus de ces 7 relations, nous avons 7 règles de génération d'événements de haut niveau (voir 3ème colonne du tableau) ce qui nous ramène à un total de 15 (au lieu de $9^2 = 81$ relations avec l'approche classique [WZM+11]).

De toute évidence, le gain apporté par notre approche dépend de la topologie de l'arbre CST (i.e. le nombre moyen d'interactions par bloc, les types de patrons de séquencement et leurs combinaisons). Nous illustrons dans ce qui suit quelques cas.

Séquence de N interactions : Dans le cas de N interactions en séquence, nous avons à générer $N - 1$ relations binaires entre chaque deux blocs consécutifs. À la détection de l'événement associé à la dernière interaction I_N, l'événement de fin de bloc $End(B)$ sera généré. Ceci ramène le nombre total de règles à N au lieu de N^2 avec l'approche classique (voir

figure 6.8).

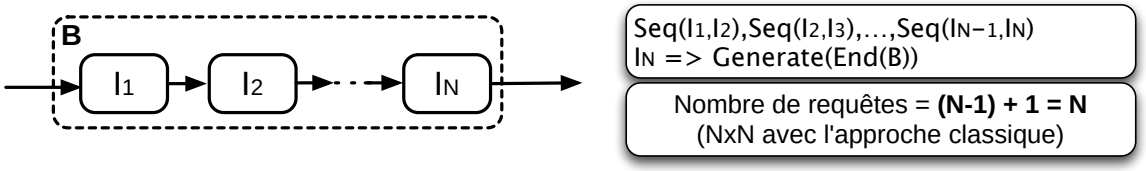


FIGURE 6.8 – Cas 1 : Séquence de N interactions

Deux blocs en parallèle : Dans le cas de deux blocs B_1 et B_2 en parallèle contenant respectivement N et M interactions en séquence, nous avons à générer $N - 1$ relations binaires de séquence et un événement de fin de bloc pour chacun des deux blocs. La génération des deux événements de fin de bloc va permettre au bloc suivant de s'exécuter. La *figure* 6.9 illustre l'exemple avec l'ensemble des règles générées.

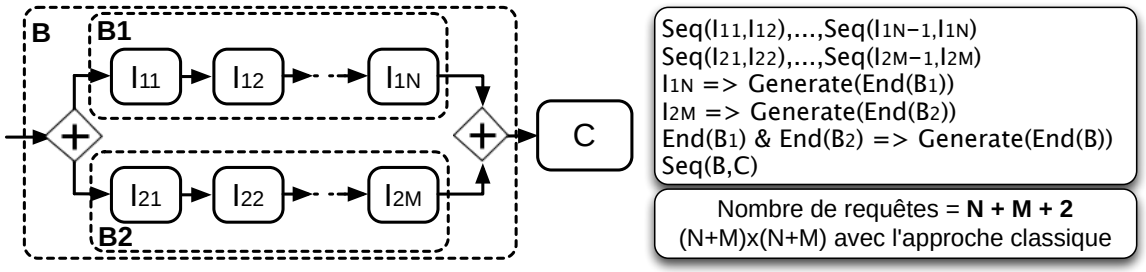


FIGURE 6.9 – Cas 3 : Deux blocs en parallèle

Choix multiple entre deux blocs : Comme nous avons pu le remarquer, le seul cas dans lequel nous sommes contraints de générer toutes les combinaisons possibles de relations est celui de plusieurs blocs en exclusion mutuelle (voir le cas choix exclusif dans la *figure* 6.7). Dans ce cas unique, nous allons avoir le même nombre de règles que celui de l'approche classique. En revanche, dans le cas particulier de plusieurs blocs dans deux branches en exclusion, nous n'aurons qu'une seule relation d'exclusion à générer. La *figure* 6.10 montre l'exemple de deux blocs B_1 et B_2 exclusifs l'un à l'autre contenant respectivement N et M interactions en séquence.

Dans chacun des deux derniers exemples, le nombre total de règles générées est égal à $N + M + 2$ au lieu de $(N + M)^2$ avec l'approche classique. De toute évidence, lorsque le nombre de bloc N ou M augmente, le gain en terme de nombre de règles augmente également. Par exemple, lorsque $N = M = 5$, le nombre de règles est égal à 12 au lieu de 100 (88 % de gain). Cependant, lorsque $N = M = 10$, le nombre de règles est égal à 22 au lieu de 400 (≈ 95 % de gain).

Il s'agit de quelques cas, parmi d'autres, qui illustre comment notre approche permet de réduire considérablement le nombre de règles nécessaires pour la supervision et la détection des violations. Cette optimisation va permettre non seulement une meilleure lisibilité, mais aussi une meilleure réactivité vu que le processeur de règles mettra moins de temps pour le traitement des requêtes.

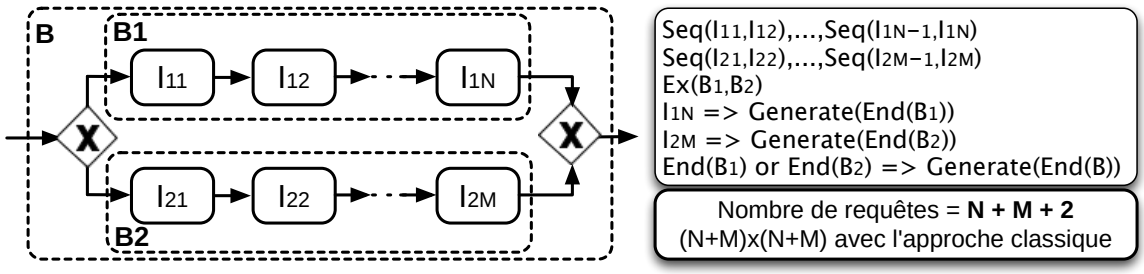


FIGURE 6.10 – Cas 2 : Choix multiple entre deux blocs

6.5 Reconnaissance des patrons et détection événementielle des violations

6.5.1 Anti-patrons et dérivation de requêtes CEP

Afin de détecter les violations, nous allons transformer les relations d'ordre ainsi que les règles de génération de fin de bloc sous forme de requêtes CEP. La transformation se fait de façon automatique, et ce, règle par règle. Sachant que les requêtes générées vont être exécutées sur le flux des évènements reçus par le processeur CEP, les résultats de ces requêtes doivent correspondre à des violations détectées. Pour ce faire, il faut inverser les relations générées avant de les transformer en requêtes. En effet, chaque requête CEP générée doit correspondre à une négation d'une relation d'ordre. Par exemple, la relation $Seq(B_1, B_2)$ est violée, si le processeur CEP détecte un événement marqué par B_2 qui est suivi par un autre événement marqué par B_1 ou simplement suivi par l'événement de fin de bloc $End(B_1)$. Cependant, la relation $Ex(B_1, B_2)$ est violée lorsqu'il détecte deux événements, respectivement marqués avec B_1 et B_2, ayant le même identifiant d'instance.

```
 // Détecter les violations de la relation Seq(I1,I2) :
"@Name('Seq I1 I2') select * from pattern "
    + "[ (e2=MsgEvent(e2.iid=2) and not e1=MsgEvent(e1.iid=1))] where e1.cid=e2.cid" ,

// Détecter les violations de la relation Ex(B211,B212) :
"@Name('Ex B211 B212') select * from pattern "
    + "[ e1=MsgEvent(e1.ascendancy like '%B211,%') and "
    + "e2=MsgEvent(e2.ascendancy like '%B212,%')] where e1.cid=e2.cid",

// Détecter les violations de la relation Seq(B1,I3) :
"@Name('Seq B1 I3') select * from pattern "
    + "[ e3=MsgEvent(e3.iid=3) and not b1=MsgEvent(b1.endOf like 'B1')] "
    + "where e3.cid=b1.cid",
/*...*/
```

FIGURE 6.11 – Formulation de requêtes CEP avec le langage Esper

La *Figure* 6.11 montre trois requêtes de notre exemple de motivation formulées avec le langage Esper (un langage de traitement des événements CEP [Esp11]).

- La première requête correspond à la négation de la relation $Seq(I_1, I_2)$. Elle permet de détecter si un événement $e1$ de type I_1 n'est pas précédé par un autre événement $e2$ de type I_2 appartenant tous les deux à la même instance de chorégraphie ($e1.Cid = e2.Cid$).
- La seconde requête correspond à la négation de la relation $Ex(B_{211}, B_{212})$. Elle permet de détecter si deux événements $e1$ et $e2$ marqués respectivement par B_{211} et B_{212} appartiennent à la même instance de chorégraphie ($e1.Cid = e2.Cid$).
- La troisième requête montre le cas d'une séquence entre un bloc et un message. La requête permet de détecter si la relation $Seq(B_1, I_3)$ a été violée. En d'autres termes, elle détecte tout événement de type I_3 qui n'est pas précédé par l'événement de fin de bloc $End(B_1)$.

6.5.2 Classification des violations par type

Lors de l'exécution des requêtes d'événements, trois types de violation peuvent être détectés : violation d'ordre de messages, présence d'un message hors-séquence et absence de message.

Désordre dans le séquencement des messages (Violation d'ordre)

Cette violation se produit lorsque l'ordre des messages n'est pas cohérent avec le comportement défini. Considérons, à titre d'exemple, notre exemple de chorégraphie avec la séquence des événements $< I_1, I_2, I_4, I_8, I_3, I_5, I_9 >$ reçus par le processeur CEP. En se référant aux règles relations générées dans la *table* 6.4, nous pouvons remarquer que la relation $Seq(I_3, B_2)$ est violée par chacun des deux événements I_4 et I_8, qui sont étiquetés avec B_2, car ils ont eu lieu avant l'événement I_3. La *figure* 6.12 schématise les violations sur le modèle.

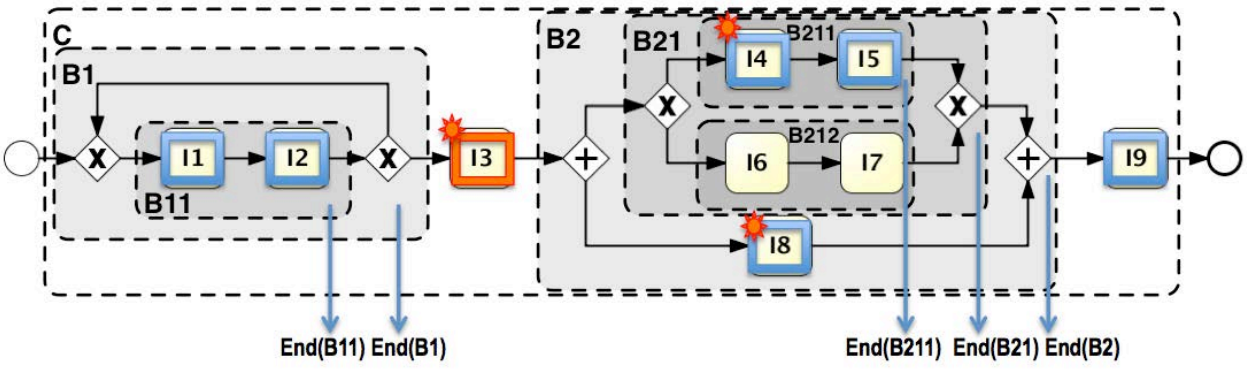


FIGURE 6.12 – Violation d'ordre de messages : $< I_1, I_2, I_4, I_8, I_3, I_5, I_9 >$

Présence d'un message non attendu (hors-séquence)

Cette violation est détectée en cas d'occurrence d'un message supplémentaire. Cela peut correspondre, par exemple, à une présence conjointe de deux événements supposés être exclusifs. Prenons le cas de la séquence $< I_1, I_2, I_3, I_4, I_8, I_6, I_5, I_9 >$. Nous remarquons ici que l'événement I_6 correspond à un message qui est hors séquence. En effet, la relation $Ex(B_{211}, B_{212})$ est violée deux fois parce que deux événements (I_4 et I_5 qui sont étiquetés avec le bloc B_{211}) ont eu lieu avec l'événement I_6 (qui est étiqueté avec le bloc B_{212}) dans la même instance. La *figure* 6.13 schématise les violations sur le modèle.

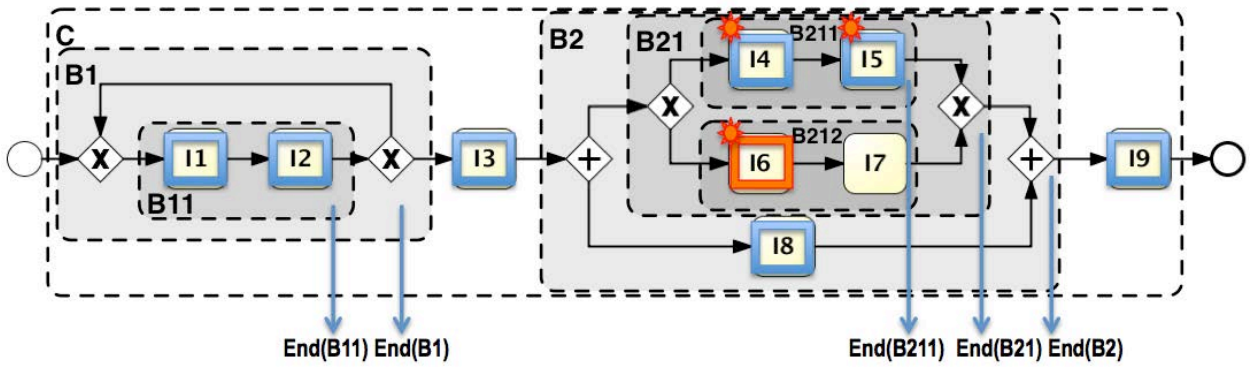


FIGURE 6.13 – Message supplémentaire : $< I_1, I_2, I_3, I_4, I_8, I_6, I_5, I_9 >$

Message manquant dans la séquence

Cette violation est détectée en cas d'une non occurrence d'un message. Les violations de ce type ne peuvent être matérialisées qu'à la fin d'exécution du plus petit bloc contenant l'interaction en question. En fait, lorsque l'événement de fin de ce bloc n'est pas généré, la séquence suivante est nécessairement violée.

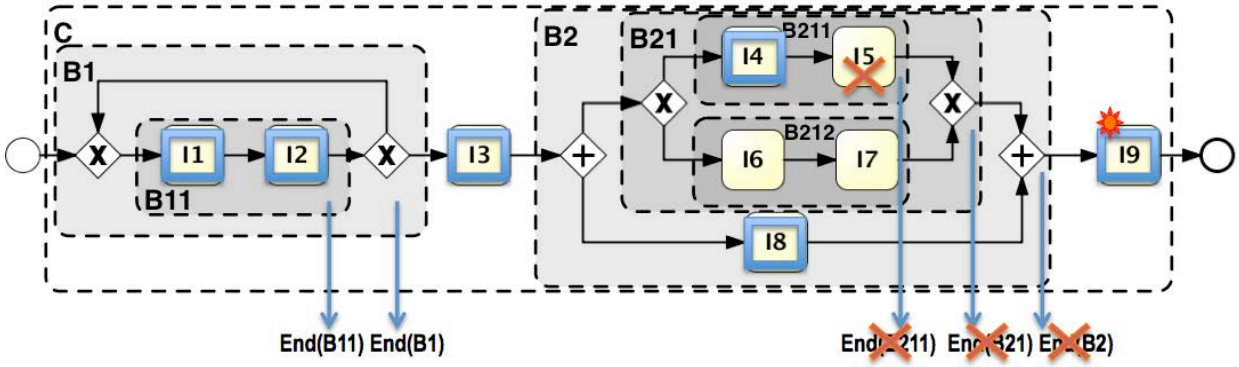


FIGURE 6.14 – Message manquant : $< I_1, I_2, I_3, I_8, I_4, I_9 >$

Par exemple, soit $< I_1, I_2, I_3, I_8, I_4, I_9 >$ la séquence des événements générés. Comme nous pouvons le remarquer, l'événement associé à I_5 est manquant dans la séquence. Dans ce cas, l'événement de fin de bloc $End(B_{211})$ ne sera pas généré (règle de génération : $I_5 \Rightarrow generate(End(B_{211}))$). Par conséquent, les événements de fin de blocs $End(B_{21})$ et $End(B_2)$ ne seront non plus générés. Comme l'événement I_9 a eu lieu avant l'événement de fin de bloc $End(B_2)$, la relation $Seq(B_2, I_9)$ est alors violée. La *figure* 6.14 schématise les violations sur le modèle.

6.6 Agrégation des violations

Cette section présente une méthode qui permet la collecte, le tri et l'agrégation des violations détectées au cours de l'exécution. En effet, les violations générant des événements simples sont recueillies à partir de l'environnement CEP vers un composant dédié à leur post-traitement, et ce, avant de les envoyer au tableau de bord. Le but est d'ajouter une nouvelle couche de niveau supérieur ayant plus de valeur sémantique permettant ainsi d'offrir à l'administrateur une meilleure visibilité sur l'origine et les relations qui existent entre les violations détectées. Cette étape permet aussi de minimiser le nombre d'alertes inutiles et de faciliter ainsi la tâche pour discerner ce qui s'est passé réellement.

6.6.1 Violation atomique et événement de violation

Lorsqu'une violation d'une des relations binaires définies est détectée par le processeur d'événement, ce dernier génère un nouveau événement de type violation dite « atomique». Ce nouveau type d'événement est défini comme suit.

Définition 14 (Violation atomique) *Un événement de violation atomique $v \in \mathcal{V}$ est un tuple*

$$v = (Vid,\ Cid,\ VR,\ Iid_c\ ,Iid_r\ ,TS)$$

avec Vid un identificateur unique de l'événement, Cid un identificateur unique pour l'instance de chorégraphie, VR la relation binaire violée, $Iid_c \in \mathcal{I}$ l'identifiant de l'interaction qui a causé la violation, $Iid_r \in \mathcal{I}$ l'identifiant de l'interaction avec laquelle Iid_c est en conflit, et TS le temps de génération de la violation.

Exemple 8 *Dans notre exemple de motivation, $v_1 = (1, 14, Seq(I_3, B_2), I_5\ , I_3\ , 20130112095023)$ correspond à une violation de la relation $Seq(I_3, B_2)$ suite à une occurrence de I_5 avant I_3.*

6.6.2 Origine et classement des violations

Lors de l'exécution d'une instance de chorégraphie, plusieurs violations peuvent survenir. Dans notre approche, une violation est détectée à chaque fois qu'une des relations binaires n'est pas satisfaite. Une relation est violée quand deux interactions ne se produisent pas dans l'ordre

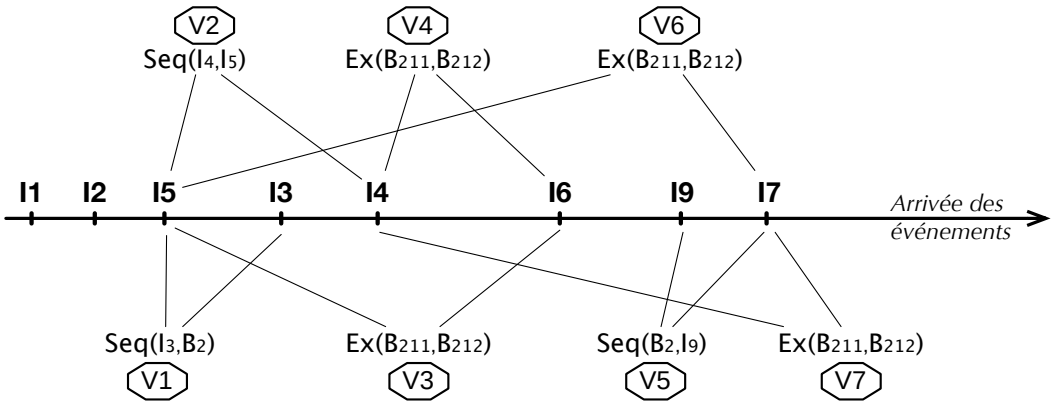


FIGURE 6.15 – Exemple de violations enregistrées dans une instance de chorégraphie

(cas de la relation Seq) ou bien quand elles ne sont pas exclusives l'une à l'autre (cas de la relation Ex). Cependant, il arrive qu'une même interaction soit à l'origine de deux ou plusieurs violations. Ainsi, des violations multiples peuvent être causées par un même message. Ajoutons à cela le fait qu'une même relation peut être violée plusieurs fois par différents messages (e.g. la violation d'une relation d'exclusion entre deux blocs A et B peut se produire plusieurs fois dans la même instance lorsque plusieurs interactions marquées avec le bloc A se produisent avec d'autres marquées avec le bloc B).

Revenons à notre exemple de modèle de chorégraphie présenté précédemment en considérant la trace d'événements $< I_1, I_2, I_5, I_3, I_4, I_6, I_9, I_7 >$. L'ensemble des violations détectées est représenté dans la *figure* 6.15. Nous remarquons ici que la relation $Ex(B_{211}, B_{212})$ a été violée quatre fois par différentes combinaisons de couples d'interactions (I_6 avec I_5, I_6 avec I_4, I_7 avec I_5 et enfin I_7 avec I_4).

D'un autre coté, nous pouvons aussi remarquer que certaines interactions causent la violation de plus qu'une relation. Par exemple, l'interaction I_5 a causé la violation de la relation $Seq(I_3, B_2)$ puisqu'elle est marquée avec le bloc B_2 et s'est produite avant l'interaction I_3. Cette même interaction entraine aussi la violation de la relation $Seq(I_4, I_5)$ en se produisant avant l'interaction I_4.

Afin de faciliter la tâche de l'administrateur qui va recueillir toutes les violations et/ou adopter une compensation adéquate, nous avons décidé d'identifier la cause principale de toute violation et de n'en remonter que cette cause principale.

Pour accomplir cette tâche, nous avons choisi de concevoir un nouveau composant en aval du processeur principal qui va recueillir toutes les violations détectées, les classifier en fonction de la cause principale et les agréger pour ne remonter qu'une seule alerte pour chacune des causes (voir *figure* 6.16). Ceci permet d'éliminer ainsi le nombre d'alertes inutiles qui risquent d'inonder les tableaux de bord.

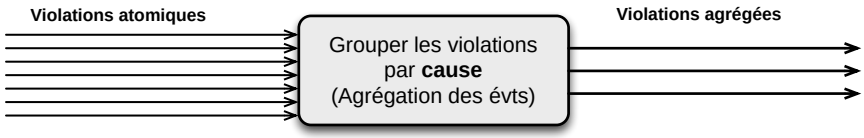


FIGURE 6.16 – Agrégation des violations

6.6.3 Méthode d'agrégation

Afin de permettre l'agrégation des violation, nous commençons par définir la structure d'une violation dite « agrégée » comme suit.

Définition 15 (Violation agrégée) *Un événement de violation agrégée $v \in \mathcal{V}_{agr}$ est un tuple*

$$v = (Vid,\ Cid,\ VR[],\ Iid_c\ ,Iid_r[]\ ,TS)$$

avec Vid un identificateur unique de l'événement, Cid l'identificateur de l'instance de chorégraphie, $VR[]$ l'ensemble des relations binaires violées, $Iid_c \in \mathcal{I}$ l'identifiant de l'interaction qui a causé les violations, $Iid_r[] \subset \mathcal{I}$ les identifiants des interactions associées aux messages avec qui Iid_c est en conflit, et TS le temps de génération de la violation.

Remarque : D'un point de vue technique, il est nécessaire de spécifier une fenêtre coulissante (*time window*) durant laquelle le processeur CEP doit rester en attente avant de générer un événement agrégé. Cette fenêtre (mobile dans le temps) permet de limiter le nombre d'événements considérés par une requête CEP comme expliqué dans [Esp11].

Après avoir défini la structure d'une violation agrégée, nous avons besoin de définir une fonction qui permet l'agrégation de plusieurs violations atomiques en une seule agrégée. Une telle fonction est appelée « fonction d'agrégation » et est définie comme suit.

Définition 16 (Fonction d'agrégation des violations)

$$\begin{aligned} f_{agr} : \quad & \mathcal{V}^n \longrightarrow \mathcal{V}_{agr} \\ & v_1, v_2, ..., v_n \longmapsto v_{agr} \end{aligned}$$

avec :

- $v_{agr}.Vid = GenerateNewID()$
- $v_{agr}.Cid = v_1.Cid$
- $v_{agr}.VR[] = \cup_{i\in[1..n]} v_i.VR$
- $v_{agr}.Iid_c = v_1.Iid_c$
- $v_{agr}.Iid_r[] = \cup_{i\in[1..n]} v_i.Iid_r$
- $v_{agr}.TS = Timestamp()$

Algorithme 6.1: Algorithme d'agrégation de violations

```
1  Require : - v_n : La dernière violation arrivée
2  - T : La table des violations atomiques
3  - T_agr : La table des violations agrégées
4  Insert(T, v_n)
5  original ← true
6  for (each v_i ∈ T_agr ) do
7      if (v_i.Iid_c = v_n.Iid_c) then  /* Si la cause de la violation existe comme cause dans
       la table */
8          Aggregate(v_i, v_n)
9          original ← false
10         Break ;                                          /* Sortir de la boucle */
11 if (original = true) then          /* Violation originale non trouvée dans la table */
12     SendAlert(v_n)
13     GenerateNewAgr(v_n)
14     SendLastaggregation()
15 End
```

L'*algorithme* 6.1 décrit comment traiter chaque violation nouvellement arrivée. Cet algorithme permet de mettre à jour la table de violations atomiques et celle des violations agrégées qui sont deux tables créées pour stocker temporairement les violations en cours. En effet, chaque nouvelle violation est d'abord stockée dans la table des violations atomiques $\mathcal{T}$ (ligne 4) en initialisant la variable booléenne *original* à «true» (ligne 5). Ensuite, nous parcourons la table des violations agrégées (ligne 6) pour chercher si l'interaction qui a causé la violation en cours existe comme cause d'une autre violation. Si c'est le cas, nous fusionnons cette violation atomique avec la violation trouvée et la variable *original* devient «false» afin d'indiquer que la violation n'est pas la cause initiale (lignes 7 à 10). Enfin, si l'interaction n'est pas trouvée comme cause, la violation atomique est envoyée au tableau de bord et une nouvelle entrée est ajoutée dans la table des violations agrégées (lignes 11 à 14).

Les deux tables 6.5a et 6.5b montrent respectivement la table des violations atomiques et celle des violations agrégées pour la séquence d'événements présentée dans la *figure* 6.15. Nous remarquons ici par exemple que l'arrivée de l'événement correspondant à I_5 a causé deux violations atomiques (les deux premières lignes de la table des violations atomiques) qui sont ensuite regroupées en une seule violation agrégée (1ère ligne de la table des violations agrégées).

Vid	Relation violée (VR)	(Iid_c)	(Iid_r)
1	$Seq(I_3, B_2)$	I_5	I_3
2	$Seq(I_4, I_5)$	I_5	I_4
3	$Ex(B_{211}, B_{212})$	I_6	I_5
4	$Ex(B_{211}, B_{212})$	I_6	I_4
5	$Seq(B_2, I_9)$	I_7	I_9
6	$Ex(B_{211}, B_{212})$	I_7	I_5
7	$Ex(B_{211}, B_{212})$	I_7	I_4

(a) Table des violations atomiques

Vid	Relations violées ($VR[]$)	(Iid_c)	($Iid_r[]$)
1	$Seq(I_3, B_2)$, $Seq(I_4, I_5)$	I_5	I_3, I_4
2	$Ex(B_{211}, B_{212})$	I_6	I_5, I_4
3	$Seq(B_2, I_9)$, $Ex(B_{211}, B_{212})$	I_7	I_9, I_5, I_4

(b) Table des violations agrégées

TABLE 6.5 – Table de violations enregistrées dans une instance de chorégraphie

6.7 Conclusion

Dans ce chapitre, nous avons proposé une approche évènementielle permettant de générer un ensemble optimal de requêtes afin de permettre la surveillance de la conformité des séquences d'interaction. Au lieu de fixer une contrainte entre chaque couple d'interaction, notre approche consiste à ne fixer que des contraintes entre les interactions de voisinage. Pour ce faire, nous avons montré comment transformer le diagramme d'une chorégraphie en une hiérarchie de blocs canoniques et enrichir chaque évènement par ses blocs ascendants dans l'arbre de structure CST. Selon les patrons de séquencement (i.e. séquence, parallèle, exclusion et itération), nous avons défini des règles de génération automatique des relations binaires ainsi que les événements de fin de bloc. Ensuite, nous avons illustré comment dériver de façon automatique les relations et

les règles trouvées sous formes de requêtes CEP. Nous avons aussi montré comment ces requêtes sont directement utilisées dans un environnement CEP en fournissant des directives de mise en œuvre.

Nous avons évalué notre approche par rapport à un ensemble de scénarios tout en montrant comment le nombre de requêtes peut être considérablement réduit. Nous avons aussi proposé un mécanisme pour l'agrégation des violations tout en suivant le principe de génération des événements de haut niveau dans un environnement CEP.

Pour faire le lien avec les deux chapitres précédents, nous rappelons que l'approche de génération présentée dans ce chapitre se fait au niveau de chaque participant par rapport à sa vue de supervision *EFM-view*. Cette vérification repose sur les évènements générés et reçus de la part de l'EFC (à chaque message échangé avec un participant), ainsi que les notifications externes (sur des messages échangés entre partenaires) reçues de la part des autres EFMs. D'un autre coté, après chaque vérification, l'EFM génère une notification qui sera envoyée et propagée aux EFMs d'un sous ensemble pré-calculé de participants (cf. chapitre précédent).

Dans le chapitre suivant, nous étudions l'efficacité de notre approche pour différents types de chorégraphies.

Troisième partie

Expérimentations, bilan et perspectives

7 Implémentation et expérimentations

«Ce n'est pas le doute qui rend fou : c'est la certitude.»
Friedrich Nietzsche

Sommaire

7.1 Introduction

Ce chapitre présente les aspects et les choix technologiques liés à l'implémentation de notre approche de supervision. Le but derrière cette implémentation est de tester l'efficacité de notre approche et d'évaluer la surcharge qu'elle génère afin d'étudier, ensuite, son passage à l'échelle. Pour accomplir cette tâche, nous nous basons sur une simulation de génération de séquences d'échanges de messages et nous observons et analysons les résultats par la suite.

Ce chapitre est organisé en trois parties principales. Dans la première, nous présentons notre environnement de développement tout en introduisant les langages de traitement de requêtes CEP. Dans la deuxième, nous parlons de notre projet ChorEM et nous nous focalisons sur les détails d'implémentation des méthodes d'enrichissement, de traitement et d'agrégation des événements. Plus particulièrement, nous présentons l'architecture générale de la conception ainsi que le diagramme de classes en UML. Dans la troisième, nous commençons par un test d'exécution de nos algorithmes. Nous exposons par la suite les différentes expériences que nous avons effectuées grâce à un mécanisme de génération massive et aléatoire d'événements d'échange de messages. Ensuite, nous introduisons les résultats obtenus et donnons une évaluation globale de l'approche. Nous expliquons ainsi les résultats de la simulation et le gain apporté par notre méthode de génération de requêtes. Nous soulignons aussi l'aspect passage à l'échelle qui représente notre objectif majeur. Enfin, nous concluons ce chapitre par une petite synthèse.

7.2 Environnement de développement CEP : Java + Esper

Dans cette section, nous décrivons notre environnement de développement. La section 7.2.1 introduit les langages de traitement de flux « Stream Query Languages », la section 7.2.2 introduit le langage *Esper* et la section 7.2.3 décrit brièvement comment intégrer notre approche dans une architecture SOA existante.

7.2.1 CEP et les langages de traitement de flux

Les solutions CEP peuvent être intégrées soit dans un périmètre applicatif (e.g. utilisation de frameworks) soit positionnées comme un module transverse recueillant l'ensemble des événements provenant des applications diverses d'un système d'information d'une entreprise (e.g., intégrées à un ESB qui représente une plateforme d'échanges transverse d'une entreprise).

Depuis quelques années, nous assistons à l'intérêt grandissant des acteurs majeurs des systèmes d'information et de l'édition logicielle pour la technologie CEP. En effet, les solutions actuelles (Progress APAMA, StreamBase, Tibco BusinessEvents, Oracle CEP, IBM WSBE, Drools, Esper) du marché proposent l'outillage nécessaire pour permettre l'implémentation des concepts inhérents à CEP et/ou l'ESP « Event Stream Processing ». De telles solutions CEP se caractérisent par le traitement continu d'une masse considérable d'événements provenant de sources d'information différentes. Le nombre d'événements traités peut atteindre plusieurs centaines de milliers par seconde. Il y a ainsi besoin d'une prise de décision en temps réel par rapport à un tel flux d'événements surgissant dans une fenêtre temporelle définie. La taille de la fenêtre est de l'ordre de quelques secondes, à quelques heures, voire quelques jours. Les solutions logicielles qui existent sur le marché sont positionnées comme des solutions complémentaires aux solutions existantes de gestion des processus (BPM), d'optimisation du business (BI) et d'infrastructure logicielle et physique (ESB/EAI).

Un flux d'événements « Event stream » est une séquence linéaire et ordonnée (par rapport à un axe temps) d'événements provenant d'une source appelée producteur d'événement « Event

producer ». Cette séquence peut être bornée et contenir des événements de différents types. Un nuage d'événements « Event Cloud » est une séquence d'événements partiellement ordonnée dans le temps et peut être engendrée par des relations autres que le temps. Un nuage est généralement créé par des événements produits par un ou plusieurs systèmes distribués (sources) et peut inclure aussi plusieurs types d'événements et plusieurs flux d'événements. Ceci dit, un flux d'événement est un nuage bien particulier et pas l'inverse.

Les langages de traitement de flux « Stream query languages » [ABW06, GWC+07, WDR06, Str10] ont une syntaxe similaire à celle du langage SQL « Structured Query Language » qui est utilisé pour interroger les bases de données. Cette famille de langages appelée EPL « Event Processing Languages » est utilisée pour écrire des requêtes sur les événements. Elle permet de décrire un filtre d'alerte, d'agréger des alertes et de travailler sur des fenêtres temporelles.

Au lieu de stocker des données, et ensuite les analyser, ce qui sera lent, l'idée est de stocker des filtres, des alertes, puis d'appliquer ces filtres sur les données surveillées en temps réel. Avec la syntaxe EPL, nous pouvons écrire des requêtes et définir les actions qui vont être déclenchées lorsqu'un événement, ou plusieurs événements se produisent. Ceci permet de résoudre un ensemble complexe de problèmes que les architectures de type « requête / réponse » n'ont pas pu résoudre à savoir comment faire signification de tous les événements qui coulent à travers un SI tout en préservant la flexibilité et comment gérer massivement des volumes de données.

7.2.2 Implémentation avec Java et Esper

Pour implémenter les différentes fonctionnalités de notre approche, nous avons choisi un environnement Java pour la simulation et la génération de séquences d'échange de messages entre les participants d'une chorégraphie. Nous avons aussi utilisé le langage *Esper* [Esp11] pour permettre la génération et le traitement des événements, et ce, en se basant sur un modèle de chorégraphie.

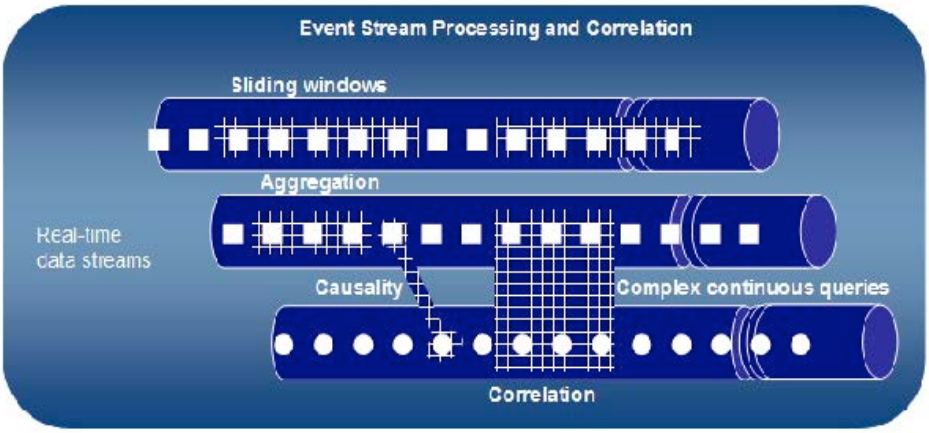


FIGURE 7.1 – Fenêtres coulissantes, causalité et corrélation avec *Esper* [Esp11]

Esper est un moteur de gestion de flux d'événement ESP ainsi qu'un moteur de corrélation

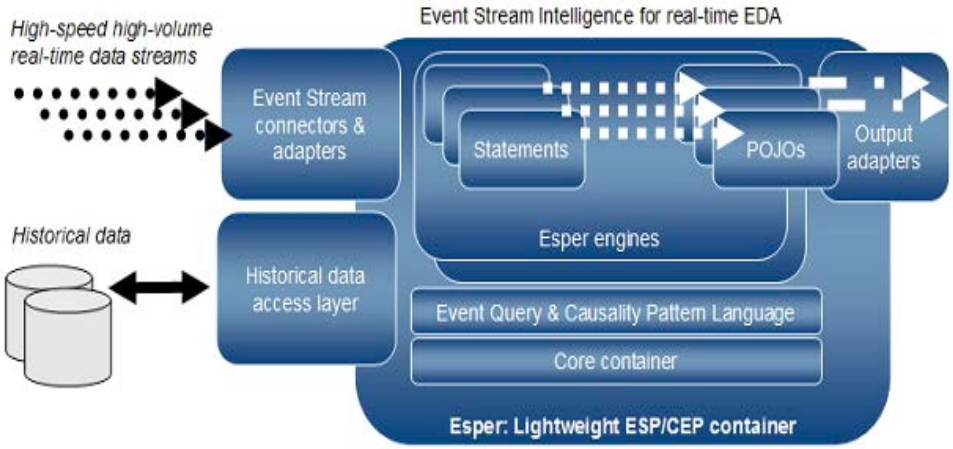


FIGURE 7.2 – Architecture du moteur d'événements *Esper* [Esp11]

d'événements. *Esper* est disponible gratuitement et fournit un langage open source qui fait partie de la famille des langages de traitement de flux d'événement «*Stream Query Languages*». Il permet la gestion et la corrélation des événements (cf. *figure* 7.1). Il permet aussi de réaliser d'autres opérations sur les événements (e.g. filtrage, projection, sélection, division, fusion de flux, etc.). L'idée d'*Esper* est de détecter des événements en temps réel et de déclencher, ensuite, des actions lorsqu'un ou plusieurs événements se produisent, et ce, suivant des règles prédéfinies (cf. *figure* 7.2).

Les bibliothèques Java fournies dans le package permettent d'effectuer plusieurs opérations sur les flux d'événements :

- déclarer un flux et son schéma ;
- créer et exécuter des requêtes en utilisant le langage d'interrogation EPL ;
- instancier un serveur Esper, dont la tâche est de compiler, enregistrer et exécuter les requêtes EPL ;
- contrôler l'instance serveur (e.g. démarrer, arrêter, etc.) ;
- gérer les processus légers et les accès concurrents [Gab11].

La syntaxe du langage *Esper* s'approche de celle du SQL. La principale différence est que dans SQL les données (i.e. les tables) sont fixes et les requêtes changent alors qu'en CEP c'est l'inverse : les requêtes sont fixées à l'avance et les données (i.e. le flux d'événements) se mettent à jour au cours de l'exécution. La *figure* 7.3 montre la structure générale d'une requête sous *Esper*. L'utilisation des expressions entre crochets est facultative.

7.2.3 Intégration dans une architecture SOA existante

Afin de permettre l'implantation de notre approche dans une architecture réelle, il est nécessaire que certains services combinent le mode « requête / réponse » utilisé dans une architecture SOA (dans notre cas les messages de chorégraphie) avec le mode axé sur les événements (génération et consommation des notifications par exemple). Le standard « Web Services Notification

```
[annotations]
[expression_declarations]
[context context_name]
[insert into insert_into_def]
select select_list
from stream_def [as name] [, stream_def [as name]] [,...]
[where search_conditions]
[group by grouping_expression_list]
[having grouping_search_conditions]
[output output_specification]
[order by order_by_expression_list]
[limit num_rows]
```

FIGURE 7.3 – Structure d'une requête avec *Esper*

(WSN) » [Kno10] appartenant à la famille des spécifications d'OASIS illustre comment la programmation orientée événement peut être introduite dans une architecture SOA de manière standardisée. La spécification « *WS-Base Notification specification* », qui représente le document de base de WSN et qui est souvent désignée comme le patron de notification dans SOA (*SOA notification pattern*), unifie les principes et les concepts de SOA avec ceux de la programmation orientée événements. En suivant ce patron de spécification, nous pouvons configurer et déployer les différentes entités comme des services orientés événements. Par exemple, un producteur d'événement (notifications) peut être déployé au sein de notre composant EFC. En fonction de la granularité de ce dernier, ce service peut être chargé de la génération des événements d'échange de messages tout en assumant le rôle de publication d'événements. De l'autre côté, l'EFM peut être configuré comme un consommateur de ces événements en souscrivant à l'EFC comme source.

Un ESB (Enterprise Service Bus) peut être utilisé pour servir de médiateur entre les invocations « requête / réponse » des services et les services orientés événement nouvellement créés et permettre la livraison des notifications depuis les producteurs aux consommateurs. De nos jours, les ESB offrent des techniques d'analyse et de contrôle des événements complexe dans le but d'intégrer à la fois les concepts d'une architecture SOA et EDA. L'ESB peut également fournir d'autres fonctionnalités à valeur ajoutée telles que la transformation du contenu d'un message de notification ou la sauvegarde dans des fichiers logs ou dans une base de données relationnelle à travers un SGBD.

En outre, le patron de passerelle ESB « *ESB gateway pattern* » fournit un pare-feu XML en plus des fonctions de passerelle souhaitées. Selon les exigences de sécurité, ce ESB peut être implanté à l'intérieur de la zone démilitarisée de chaque organisation (DMZ). Une telle solution est bien adaptée pour servir sur les frontières de chaque organisation car elle permet plus d'agilité et de flexibilité. De plus, elle permet un déploiement facile de nos composants exposés comme étant des services. Toutefois, cette étape n'a pas été approfondie et fait partie de nos travaux futurs.

7.3 Le projet ChorEM

Pour mettre en œuvre notre approche de supervision, nous avons conçu le projet ChorEM [21]. Le projet consiste à implanter un réseau de traitement des événements (EPN, « Event Processing Network ») composé de trois agents de traitement CEP comme indiqué dans la *figure* 7.4. Chaque agent CEP permet d'assurer une des fonctions décrites dans notre approche. Le premier s'occupe de l'enrichissement des événements, le deuxième du traitement principal (i.e. exécution des requêtes de détection de violations) et le troisième assure la fonction d'agrégation.

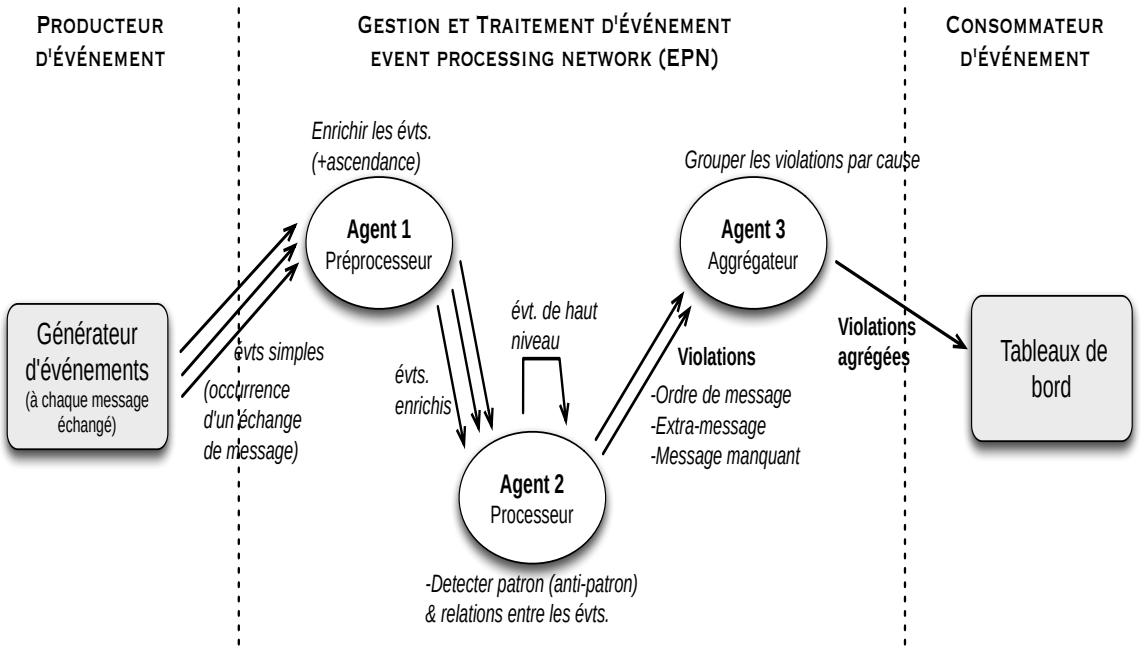


FIGURE 7.4 – Réseau d'agents de traitement implantés

7.3.1 Diagramme de classes

La *figure* 7.5 montre le diagramme de classe de l'application réalisée.

Classe Message : Cette classe définit la structure de chaque événement de type échange de message de chorégraphie et les opérations de manipulations des différents champs du message (i.e. identifiant de l'interaction, expéditeur, récepteur, type de message). Cette classe contient aussi la logique de l'enrichissement des événements.

Classe CEPlistener : Cette classe contient le code qui va s'exécuter à chaque occurrence de violation (i.e. chaque fois qu'un des anti-patrons est détecté). C'est cette classe qui permet de compter le nombre de violations et de les classer par type (i.e. violation d'ordre, message manquant ou extra, cause ou répercussion).

21. Choreography Event Monitoring Project. Documentation technique et code source sont disponibles sur GForge INRIA https ://gforge.inria.fr/projects/chorem/

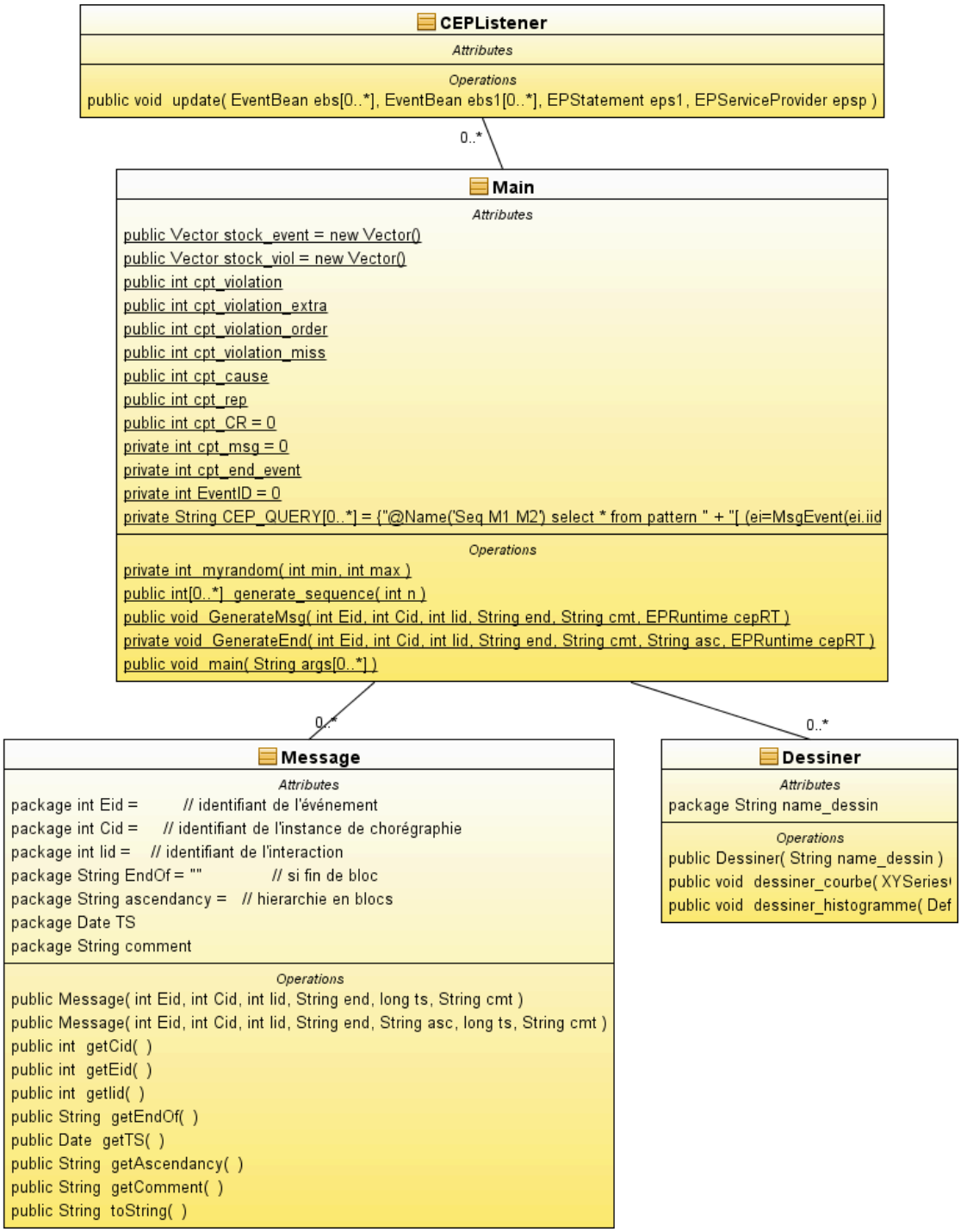


FIGURE 7.5 – Diagramme de classes

Classe Dessiner : Cette classe permet de dessiner les courbes et les histogrammes. Elle fait appel aux fonctions de la bibliothèque *jFreeChart* de Java.

Classe Main : c'est la classe principale qui permet de définir les requêtes *Esper* (i.e. des anti-patrons sur les couples d'événements) et les règles de génération des événements de fin de

bloc (*ENDevents*). Elle contient aussi les méthodes «GenerateMsg()» et «GenerateSeq()» qui permettent de générer, respectivement, les messages et les séquences de messages de façon aléatoire tout en s'approchant des séquences correctes afin de simuler des exécutions proches de celles du monde réel.

7.3.2 Phases

Tout d'abord, un flux d'entrée et une fenêtre pour stocker les événements entrants sont définis et créés. Ensuite, les requêtes définies sont enregistrées dans le moteur *Esper* sous forme de partons d'événements «*event pattern*». En phase d'exécution, les événements entrants sont constamment analysés et traités contre ces patrons. Lorsqu'un couple d'événements correspondant à l'un des patrons est trouvé, les actions prédéfinies dans la classe *CEPlistener* sont exécutées.

```
"@Name('Seq M1 M2') select * from pattern "
+ "[ (ei=MsgEvent(ei.iid=2) and not ej=MsgEvent(ej.iid=1))] ",

"@Name('Loop B11 was not finished correctly') select * from pattern "
+ "[every(ei=MsgEvent(ei.ascendancy like '%B11,%') -> (ej=MsgEvent(ej.ascendancy not like '%B1,%')
      + "and  not b1=MsgEvent(b1.endOf  like 'B11'))))]",

"@Name('Seq B1 M3') select * from pattern "
+ "[every ei=MsgEvent(ei.iid=3) and not ej=MsgEvent(ej.endOf like 'B1')] ",

"@Name('Seq M3 B2') select * from pattern "
+ "[   ei=MsgEvent(ei.ascendancy like '%B2,%') and not ej=MsgEvent(ej.iid=3)] ",

"@Name('Seq B2 M9') select * from pattern "
+ "[   ei=MsgEvent(ei.iid=9) and not ej=MsgEvent(ej.endOf like 'B2')] ",

"@Name('Ex B211 B212') select * from pattern "
+ "[every ei=MsgEvent(ei.ascendancy like '%B211,%') and "
+ "ej=MsgEvent(ej.ascendancy like '%B212,%')] where ei.cid=ej.cid",

"@Name('Seq M4 M5') select * from pattern  "
+ "[  ei=MsgEvent(ei.iid=5) and not ej=MsgEvent(ej.iid=4)] ",

"@Name('Seq M6 M7') select * from pattern  "
+ "[  ei=MsgEvent(ei.iid=7) and not ej=MsgEvent(ej.iid=6)] ",

"@Name('Ex Mi Mi (Repeated interaction !)') select * from pattern "
+ "[every ei=MsgEvent(ei.iid!=0,ei.ascendancy not like '%B1,%') -> ej=MsgEvent(ej.iid=ei.iid)] ",
```

a) Requêtes injectées dans ESPER

```
switch (Iid) {
    case 1:
        asc = "B11,B1,C";
        break;
    case 2:
        asc = "B11,B1,C";
        break;
    case 3:
        asc = "C";
        break;
    case 4:
        asc = "B211,B21,B2,C";
        break;
    case 5:
        asc = "B211,B21,B2,C";
        break;
    case 6:
        asc = "B212,B21,B2,C";
        break;
    case 7:
        asc = "B212,B21,B2,C";
        break;
    case 8:
        asc = "B2,C";
        break;
    case 9:
        asc = "C";
        break;
    default:
        asc = "";
}
```

b) Enrichissement des événements

FIGURE 7.6 – Génération des requêtes CEP et enrichissement des événements avec *Esper*

Phase 1 : Enrichissement des événements

La *figure* 7.6.b montre l'exemple de l'enrichissement des événements associés aux interactions de notre modèle de chorégraphie précédemment présenté dans le chapitre 6. En effet, le code permet d'affecter la liste des blocs ascendants au champ « ascendancy » en fonction du modèle de chorégraphie et de l'identifiant de l'interaction Iid.

Phase 2 : Exécution des requêtes CEP

La *figure* 7.6.a montre la liste des requêtes formulées sous forme d'anti-patrons en langage *Esper* permettant de détecter toutes les violations possibles pour notre exemple de chorégraphie. Chaque requête représente un anti-patron de séquence «$Seq()$» ou d'exclusion «$Ex()$» entre deux événements.

Phase 3 : Agrégation des violations

Nous utilisons un tableau pour stocker temporairement chaque violation détectée. Pour chaque violation, nous enregistrons les deux événements responsables. Avant de générer un événement de violation, nous vérifions si un des deux événements apparaît comme cause dans une des violations précédemment enregistrées. Si c'est le cas, la violation est enregistrée comme répercussion.

Remarque : Les requêtes de génération des événements de fin de bloc ainsi que la fonction de génération des séquences de messages sont montrées dans l'*annexe* 8.3.

7.4 Expérimentations et simulations

Dans cette section, nous allons présenter les différentes expérimentations réalisées en vue de tester l'efficacité de notre approche et la surcharge qu'elle génère afin d'étudier ensuite son passage à l'échelle. Pour ce faire, nous allons commencer par générer aléatoirement une séquence de messages échangés et simuler l'exécution de notre algorithme. Ensuite, nous allons étudier l'évolution du nombre d'événements et mesurer d'autres métriques en générant un flux important de séquences de messages.

7.4.1 Exécution d'une séquence

La *figure* 7.7 montre l'exécution de la séquence <1,2,3,6,7,4,5,8,9> composée de 9 messages associés à des interactions toutes différentes. Au total, 17 événements sont générés : 9 événements simples (un pour chaque message envoyé), 6 événements de fin de bloc « END-events » et deux événements générés suite à une occurrence de violation (lignes 12 et 14). Ces deux violations

correspondent à la même règle d'exclusion entre les blocs B_{211} et B_{212}, à savoir, $Ex(B_{211}, B_{212})$. Chaque violation montre les deux événements résultants de la requête correspondante à la règle violée. À la fin de la ligne 12 par exemple, nous pouvons voir que les deux événements responsables sont I_4 (qui est enrichi par le bloc B_{211}) et I_6 (qui est enrichi par le bloc B_{212}).

La deuxième violation (ligne 14) est détectée suite à l'occurrence de I_5 (enrichi par le bloc B_{211}) qui présente une incohérence par rapport à l'occurrence de I_6. Comme I_6 est déjà responsable d'une violation précédente, cette violation est considérée comme une répercussion de la première.

```
1  Exécution de la séquence : 1, 2, 3, 6, 7, 4, 5, 8, 9,
2  :Eid:(0) Cid:(C0) *** New chor Mesg:lid:=[(M1)]= Asc:(B11,B1,C) time:(Fri Feb 15 15:50:21 CET 2013)
3  :Eid:(1) Cid:(C0) *** New chor Mesg:lid:=[(M2)]= Asc:(B11,B1,C) time:(Fri Feb 15 15:50:21 CET 2013)
4  :Eid:(2) Cid:(C0) ====> End of Block : (END(B11)) ====
5  :Eid:(3) Cid:(C0) ====> End of Block : (END(B1)) ====
6  :Eid:(4) Cid:(C0) *** New chor Mesg:lid:=[(M3)]= Asc:(C) time:(Fri Feb 15 15:50:21 CET 2013)
7  :Eid:(5) Cid:(C0) *** New chor Mesg:lid:=[(M6)]= Asc:(B212,B21,B2,C) time:(Fri Feb 15 15:50:21 CET 2013)
8  :Eid:(6) Cid:(C0) *** New chor Mesg:lid:=[(M7)]= Asc:(B212,B21,B2,C) time:(Fri Feb 15 15:50:21 CET 2013)
9  :Eid:(7) Cid:(C0) ====> End of Block : (END(B212)) ====
10 :Eid:(8) Cid:(C0) ====> End of Block : (END(B21)) ====
11 :Eid:(9) Cid:(C0) *** New chor Mesg:lid:=[(M4)]= Asc:(B211,B21,B2,C) time:(Fri Feb 15 15:50:21 CET 2013)
12 ****=(!)=Violated Rule !: !!!!!! Ex B211 B212*!*!*!*!*!*!*!* Involved interactions: I4 and I6
13 :Eid:(10) Cid:(C0) *** New chor Mesg:lid:=[(M5)]= Asc:(B211,B21,B2,C) time:(Fri Feb 15 15:50:21 CET 2013)
14 ****=(!)=Violated Rule !: !!!!!! Ex B211 B212*!*!*!*!*!*!*!* Involved interactions: I5 and I6
15 :Eid:(11) Cid:(C0) ====> End of Block : (END(B211)) ====
16 :Eid:(12) Cid:(C0) *** New chor Mesg:lid:=[(M8)]= Asc:(B2,C) time:(Fri Feb 15 15:50:21 CET 2013)
17 :Eid:(13) Cid:(C0) ====> End of Block : (END(B2)) ====
18 :Eid:(14) Cid:(C0) *** New chor Mesg:lid:=[(M9)]= Asc:(C) time:(Fri Feb 15 15:50:21 CET 2013)
```

FIGURE 7.7 – Exemple de trace d'exécution

7.4.2 Génération aléatoire de plusieurs séquences

Afin de tester les performances de notre approche à large échelle, nous allons générer aléatoirement plusieurs séquences de messages (allant jusqu'à 70000 messages) durant une très petite période (quelques secondes). Nous allons ensuite étudier l'évolution du nombre des événements générés et leur répartition par type (e.g. événements d'échange de messages, événements de fin de blocs, événements de violations). Les résultats sont détaillés dans ce qui suit.

Étude de la variation du nombre d'événements d'échange, de fin de bloc et de violation

Dans cette section, nous nous intéressons à l'étude de la variation du nombre d'événements générés par rapport au nombre de messages échangés, et ce, en comptant le nombre de chaque type d'événement (i.e. événement de fin de bloc, événement de violation et le total des événements). Nous avons donc généré aléatoirement plusieurs séquences de messages. Nous avons

ensuite compté à chaque fois le nombre des différents types d'événements générés.

FIGURE 7.8 – Variation du nombre de violations, des événements de fin de bloc et de l'ensemble de tous les événements en fonction du nombre de messages

La *figure* 7.8 montre trois courbes. La première (en vert) représente la variation du nombre des événements de violation. La deuxième (en bleu) représente la variation du nombre des événements de fin de bloc et la troisième représente la variation du nombre total des événements. Ces courbes ont été tracées en utilisant la bibliothèque $JfreeChart$ de Java.

Nous pouvons remarquer que l'évolution est quasi-linéaire pour les trois courbes ce qui permet un bon passage à l'échelle. En d'autres termes, les résultats montrent qu'il n'y a pas trop de surcharge en nombre d'événements générés.

Ainsi pour 70000 messages générés :
- $\approx$ 15000 événements de violation (i.e. $\approx$ 0.2 du nombre de messages générés).
- $\approx$ 50000 événements de fin de bloc END-$events$ (i.e. $\approx$ 0.7 du nombre de messages générés).
- $\approx$ 70000 événements d'échange de messages.
- $\approx$ 120000 événements au total (i.e. $\approx$ 1.8 du nombre de messages générés).

Remarque : Le coefficient du nombre d'événements de fin de bloc par rapport au nombre de messages (évalué à $\approx$ 0.7) est spécifique à notre modèle de chorégraphie choisi. En effet, ce coefficient varie d'un modèle à un autre et dépend de la structure de l'arbre CST. Nous avons tout de même essayé de prendre un modèle qui combine le maximum de types de patrons de séquencement (bloc séquence, parallèle, exclusion et itération). Avec un modèle structurellement

plus simple, nous pourrons avoir un coefficient nettement inférieur. Dans tous les cas, l'évolution reste quasi-linéaire à un coefficient près.

Comparaison entre les différents types de violations

La *figure* 7.9 représente l'histogramme de comparaison des trois types de violations (les violations d'ordre, de message manquant et d'extra-message).

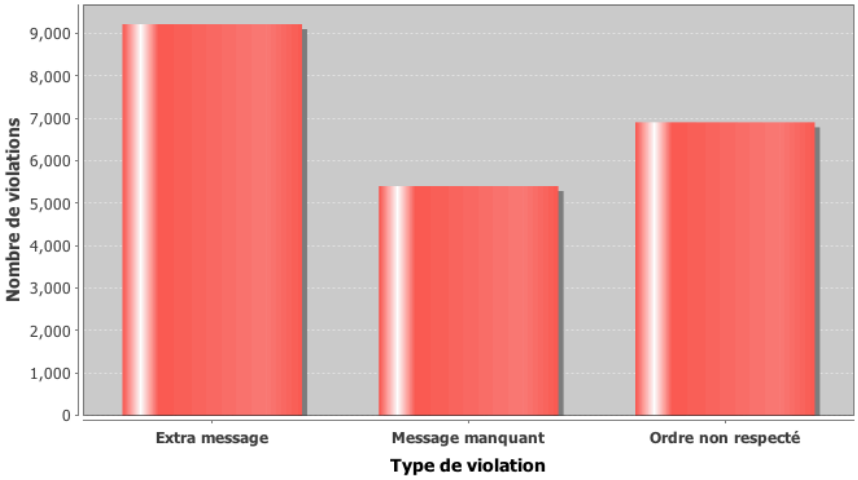


FIGURE 7.9 – Répartition des violations

Nous pouvons remarquer que notre algorithme de génération aléatoire permet d'avoir une bonne répartition entre les différents types de violations. En effet, ces résultats nous ont permis de sélectionner les meilleurs scénarii qui vont nous permettre par la suite d'étudier les causes de violations de façon homogène.

Variation des violations de type «cause» et «répercussion»

Dans cette section, nous évaluons notre mécanisme d'agrégation de violations. Nous étudions le nombre de violations de type «cause» et celles de type «répercussion». Nous classifions donc les violations détectées pendant notre simulation et nous comptons, en continu, le nombre de chaque type afin de dessiner leur évolution en fonction du nombre total de violations. Les résultats sont présentés dans la *figure* 7.10.

D'après les courbes, nous pouvons remarquer que pour cet exemple de chorégraphie le nombre de répercussions constitue le 1/3 du nombre de violations de type «cause». Le nombre de violations de type «cause» constitue 3/4 du nombre total de violations, tandis que le nombre de répercussions en constitue le 1/4. Avec notre mécanisme d'agrégation de violations de type répercussion, nous pouvons ainsi réduire considérablement ($\approx 1/3$ pour ce modèle de chorégraphie) le nombre d'alertes. Ainsi, en arrivant à 7000 violations détectées, seulement 5000 sont envoyées directement aux tableaux de bord et nous comptons donc 2000 alertes en moins (envoyées après avoir été agrégées).

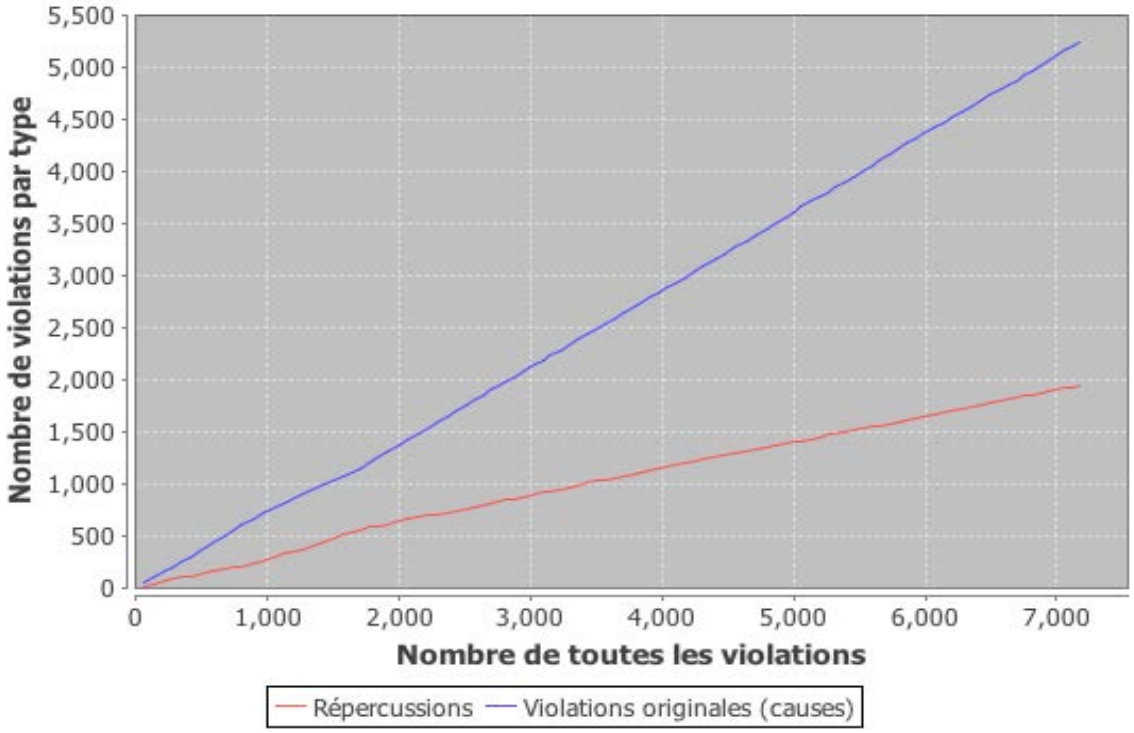


FIGURE 7.10 – Cause et répercussion de violation en fonction du nombre total de violations

7.5 Synthèse

Dans ce chapitre, nous avons présenté les détails d'implémentation de l'approche de génération de requêtes que nous avons proposée. Nous avons ainsi illustré quelques exemples et simulations qui permettent de tester et évaluer les performances. Nous avons expliqué les modules principaux pour la réalisation de notre approche et discuté les résultats obtenus. Nous avons aussi, présenté les résultats des différents tests sur la performance et la surcharge générée au moment de l'exécution de notre mécanisme de supervision.

Tout au long de ce chapitre nous nous sommes appuyés sur l'exemple de la chorégraphie présentée dans la *figure* 6.2 du chapitre précédent, et ce, pour montrer la démarche suivie. D'ailleurs, les résultats sont similaires pour d'autres modèles de chorégraphie à un coefficient près. Ce coefficient dépend du nombre de blocs dans l'arbre de structure CST. Pour notre modèle, ce coefficient a été évalué à 0.7.

A travers le projet *ChorEM*, nous avons pu évaluer aussi le mécanisme de classification des violations par leurs causes ainsi que l'opération d'agrégation. Les résultats nous semblent intéressants du fait qu'ils ont démontré que la solution proposée a permis de détecter instantanément les différents types de violations tout en minimisant la surcharge en terme de nombres d'événements générés.

Pour conclure, l'évolution quasi-linéaire du nombre d'événements générés par rapport au nombre de messages échangés au moment de l'exécution d'une chorégraphie montre clairement

que notre approche induit une surcharge acceptable et permet ainsi un bon passage à l'échelle.

8 Bilan et perspectives

8.1 Rappel du contexte et des objectifs de la thèse

De nos jours, la croissance continuelle des entreprises incite celles-ci à externaliser et à sous traiter certaines de leurs activités. Cela a conduit à une forte augmentation du nombre d'interactions entre les différents partenaires en collaboration. Par opposition aux processus centralisés intra-organisationnels (i.e. au sein d'une même organisation), la configuration décentralisée de ceux qui sont déployés à travers les frontières organisationnelles pose de nouvelles exigences en matière de contrôle. Sans coordonnateur central qui risque d'anéantir les performances dans certains scénarios (i.e. goulot d'étranglement), les organisations collaboratrices doivent être en mesure de mettre en place un processus métier inter-organisationnel, et ce, en divulguant, les unes aux autres, uniquement le stricte nécessaire.

Parmi les inconvénients majeurs de la sous-traitance et de l'externalisation des processus métier, nous pouvons citer la perte de maîtrise, le manque de flexibilité de la part du prestataire, le risque de délais trop longs et le manque d'information et de transparence. Une chorégraphie inter-organisationnelle peut être mise en oeuvre afin de permettre la coordination et la synchronisation de plusieurs processus. Ainsi, chaque organisation est en charge d'exécuter une partie du processus global. Toutefois, cela ne garantit pas nécessairement que des situations erronées puissent se produire suite, par exemple, à des interactions "mal" spécifiées ou encore des comportements malhonnêtes d'un des partenaires. Afin de pallier ces problèmes, il est nécessaire pour chaque organisation de superviser ses fragments externalisés, et ce, de façon abstraite sans pour autant dévoiler la logique métier de chaque partenaire. Le but est de vérifier si le comportement réel (en phase d'exécution) des entités en interaction adhère efficacement aux contraintes métier modélisées (en phase conception). De plus, la propagation des données de supervision entre les partenaires permet une gestion décentralisée des informations recueillies, et ce, dans le but d'offrir plus de maîtrise au niveau de chaque organisation, et de réduire les délais et les coûts en cas d'occurrence d'exceptions.

8.2 Bilan des contributions

Dans ce manuscrit, nous avons souligné le besoin d'une méthode agile et flexible qui permet d'assurer la conformité de la séquence des interactions inter-organisationnelles avec un plan pré-

défini sous forme de modèle de chorégraphie. Nous avons commencé par définir les chorégraphies inter-organisationnelles d'une manière indépendante des langages de spécification. Nous avons ainsi proposé un modèle formel général, simple et compréhensible qui va servir de base pour notre approche de supervision. Nous avons ensuite introduit la notion de vérification événementielle dans le cadre des chorégraphies de services. Nous avons, d'autre part, discuté du besoin d'un mécanisme décentralisé, dynamique et efficace qui permet l'échange des données de supervision entre les partenaires tout en assurant la traçabilité d'exécution du processus global.

Comme première contribution, nous avons proposé un nouveau modèle d'architecture pour la supervision décentralisée des collaborations (*cf. chapitre* 4). Après avoir présenté les trois composants de base de notre architecture (i.e. EFC, EFM et EFP), nous avons détaillé l'aspect vérification structurelle des messages réalisé par l'EFC. Ensuite, nous avons détaillé les deux fonctionnalités centrales de notre approche, celles réalisées par l'EFM, à savoir la supervision décentralisée avec échange inter-organisationnel de notifications et la génération automatique des règles au sein de chaque organisation.

La deuxième contribution de cette thèse consiste à présenter un mécanisme d'échange de notifications entre les différents participants d'une chorégraphie (*cf. chapitre* 5). Nous avons montré comment une telle approche peut permettre une supervision décentralisée de l'exécution d'une chorégraphie, et ce, dans le but d'offrir plus de maîtrise au niveau de chaque organisation participante et de réduire les délais et les coûts en cas d'occurrence d'exceptions. Elle permet aussi d'offrir une traçabilité de l'exécution des processus. En effet, les informations recueillies pourraient être utilisées a posteriori pour mesurer la performance globale du processus et le suivi de la réalisation des objectifs de chaque organisation.

La supervision se fait de façon abstraite (i.e. uniquement sur les échanges de messages inter-organisationnels) sans pour autant dévoiler la logique métier de chaque entreprise. La propagation hiérarchique des données de supervision permet de limiter l'exposition des données et d'éviter une surcharge du réseau (notification sélective). Nous ajoutons aussi que notre approche est non intrusive dans le sens où elle se base sur l'écoute passive des messages de la chorégraphie sans aucune modification de la structure ou du contenu des messages (i.e. les processus ne sont pas altérés et ne sont pas conscients de la supervision et l'échange de notifications). L'implantation des EFMs est modulable et respecte bien le principe de séparation des préoccupations "*Separation of concerns*" (i.e. séparation des aspects non fonctionnels des aspects fonctionnels).

La troisième contribution est dédiée à l'analyse et l'évaluation des événements générés par les échanges de messages entre les partenaires (*cf. chapitre* 6). Nous avons proposé une approche évènementielle permettant de générer un ensemble optimal de requêtes afin de permettre la surveillance de la conformité des séquences d'interaction. Au lieu de fixer une contrainte entre chaque couple d'interaction, notre approche consiste à ne fixer que des contraintes entre les interactions de voisinage. Pour ce faire, nous avons montré comment transformer le diagramme d'une chorégraphie en une hiérarchie de blocs canoniques et enrichir chaque évènement par ses blocs ascendants dans l'arbre de structure CST. Selon les patrons de séquencement d'interactions (i.e. séquence, parallèle, exclusion et itération), nous avons défini des règles de génération automatique des relations binaires ainsi que les événements de fin de bloc. Ensuite, nous avons

illustré comment dériver de façon automatique les relations et les règles trouvées sous formes de requêtes CEP. Nous avons aussi montré comment ces requêtes sont directement utilisées dans un environnement CEP en fournissant des directives de mise en œuvre. Nous avons évalué notre approche par rapport à un ensemble de scénarios tout en montrant comment le nombre de requêtes peut être considérablement réduit. Nous avons aussi proposé un mécanisme pour l'agrégation des violations tout en suivant le principe de génération des événements de haut niveau dans un environnement CEP.

À travers le projet *ChorEM*, nous avons pu évaluer les performances de notre approche de supervision. Nous avons commencé par un test d'exécution de nos algorithmes. Nous avons exposé par la suite les différentes expériences que nous avons effectuées grâce à un mécanisme de génération massive et aléatoire d'événements d'échange de messages. Ensuite, nous avons introduit les résultats obtenus ainsi qu'une évaluation globale de la performance et la surcharge générée par notre approche. Nous avons montré le gain apporté par notre méthode de génération de requêtes tout en soulignant l'aspect passage à l'échelle (*cf. chapitre* 7).

8.3 Perspectives

La conception de notre approche d'échange de notifications entre participants, ainsi que les expérimentations préliminaires du modèle de génération de requêtes de supervision proposé dans cette thèse, ont donné lieu à des propositions et à des réflexions aussi bien sur des aspects techniques que sur des aspects plus généraux. En perspectives, nous envisageons d'étendre l'approche que nous proposons, et ce, à plusieurs niveaux :

Plus de tests Pour tester notre approche et montrer la démarche suivie, nous avons choisi un exemple de chorégraphie qui contient tous les types de patrons de séquencement (i.e. séquence, parallèle, exclusion et boucle), et ce, afin d'essayer de traiter toutes les combinaisons possibles. Théoriquement, les résultats trouvés pour ce modèle sont similaires aux autres modèles de chorégraphie à un coefficient près. Ce coefficient dépend du nombre de blocs dans l'arbre de structure CST. Pour le confirmer, nous avons l'intention d'étudier l'efficacité de notre approche pour différents types de chorégraphies.

Améliorer l'outil ChorEM Nous envisageons la mise en œuvre d'un générateur de code permettant l'automatisation de la génération des règles (e.g. à partir d'un fichier WS-CDL ou d'une notation BPMN 2.0) ainsi que les requêtes CEP en langage *Esper*. L'approche de génération automatique peut être, éventuellement, appliquée à d'autres langages cibles.

Gestion des exceptions Nous proposons l'extension de notre approche afin de gérer les actions à entreprendre si une exception (e.g. violation de contrainte, redondance, blocage, etc.) est détectée. Les travaux futurs viseront à étendre l'architecture proposée par un nouveau composant pour gérer les actions à entreprendre si une exception est détectée.

Confidentialité et vie privée Nous prévoyons aussi d'aborder les questions de confidentialité, et ce, afin d'éviter la divulgation des informations critiques. Pour traiter ces aspects, nous envisageons une supervision basée sur les rôles avec laquelle la construction de la vue de supervision de chaque participant peut tenir compte d'autres contraintes de sécurité.

Chorégraphies non structurées Pour pouvoir diviser le modèle de chorégraphie en blocs, il faut que ce modèle soit bien structuré. A long terme nous envisageons de traiter les chorégraphies qui ne respectent pas les contraintes structurelles (e.g. des modèles dans lesquels il n'y a pas un seul initiateur). Plusieurs travaux portent sur la structuration des processus métier et le passage d'un modèle non structuré à un modèle structuré. Ces travaux sont aussi applicables aux chorégraphies (modèle d'interaction) et peuvent donc être utilisés dans notre cas.

Aspects de qualité de service Notre approche permet, essentiellement, de contrôler la structure des messages échangés et de vérifier leur ordre de séquencement. Des fonctionnalités de contrôle supplémentaires peuvent ainsi être ajoutées. Par ailleurs, nous cherchons à améliorer notre approche pour répondre à certaines des préoccupations de qualité de service. Par exemple, il serait intéressant de faire face à des retards dans les échanges de messages. À cette fin, les violations de contraintes de temps pourraient être détectées en fixant les délais d'attente et/ou le délai prévu pour s'écouler entre deux messages.

Bibliographie

[AA02] M. Burstein A. Ankolekar. Daml-s : Web service description for the semantic web, 2002.

[ABHK00] Wil M. P. van der Aalst, Alistair P. Barros, Arthur H. M. ter Hofstede, and Bartek Kiepuszewski. Advanced workflow patterns. In *Proceedings of the 7th International Conference on Cooperative Information Systems*, CooplS '02, pages 18–29, London, UK, 2000. Springer-Verlag.

[ABW06] A. Arasu, S. Babu, and J. Widom. The cql continuous query language : semantic foundations and query execution. *VLDB Journal 15(2) : 121-142*, 2006.

[ACBC08] S. Ayed, N. Cuppens-Boulahia, and F. Cuppens. Managing access and flow control requirements in distributed workflows. *Computer Systems and Applications*, 2008.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1.* IBM, 2003.

[ACKM03] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services : concepts, architectures and applications.* Springer, 2003.

[AF01] L. Andrade and J. Fiadeiro. Coordination technologies for web-services, 2001.

[AFG+07] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan. Monitoring choreographed services. In Tarek Sobh, editor, *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pages 283–288. Springer Netherlands, 2007.

[AFG+08] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. An event-based model for the management of choreographed services. In Giuseppe Psaila and Roland Wagner, editors, *E-Commerce and Web Technologies*, volume 5183 of *Lecture Notes in Computer Science*, pages 51–60. Springer Berlin Heidelberg, 2008.

[AII+02] Assaf, Assaf Arkin Intalio, Sid Askary Intalio, Scott Fordin, Wolfgang Jekeli Sap, Kohsuke Kawaguchi, David Orchard, Bea Systems, Stefano Pogliani, Karsten Riemer, Susan Struble, Sun Microsystems, Sun Microsystems, Sun Microsystems, Sun Microsystems, and Sun Microsystems. Web service choreography interface 1.0, 2002.

[AP97] E. Anceaume and I. Puaut. A taxonomy of clock synchronization algorithms. 1997.

[BAC+07] Ramón F. Brena, Jose L. Aguirre, Carlos I. Chesñevar, Eduardo Ramírez, and Leonardo Garrido. Knowledge and information distribution leveraged by intelligent agents. *Knowledge and Information Systems (KAIS), Springer Verlag*, 2007.

[BCF+07] J. Biskup, B. Carminati, E. Ferrari, F. Muller, and S. Wortmann. Towards secure execution orders for compositeweb services. *Web Services, 2007. ICWS*, 2007.

[BCPV04] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105 :73–94, 2004.

[BEA05] BEA. Domain model for SOA : Realizing the business benefit of service-oriented architecture. white paper, 2005.

[BFHS03] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification : a new approach to design and analysis of e-service composition. In *WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pages 403–410, New York, NY, USA, 2003. ACM Press.

[BFPG12] Aymen Baouab, Walid Fdhila, Olivier Perrin, and Claude Godart. Towards decentralized monitoring of supply chains. In *19th IEEE International Conference on Web Services (ICWS'12)*, 2012.

[BGG] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart monitors for composed services. *Proceedings of the 2nd international conference on Service oriented computing, ICSOC '04.*

[BH06] Peter F Brown and Rebekah Metz Booz Allen Hamilton. Reference model for service oriented architecture 1.0, 2006.

[Bhi05] Sami Bhiri. *Approche Transactionnelle pour Assurer des Compositions Fiables de Services Web.* PhD thesis, Université Henri Poincaré, Nancy 1, Octobre 2005.

[BIM03] BEA, IBM, and Microsoft. Business process execution language for web services (bpel4ws). 2003.

[Bon05] Pierre Bonnet. Cadre de référence web services : Meilleures pratiques. *Orchestra Networks*, 2005.

[BPBG09] Aymen Baouab, Olivier Perrin, Nicolas Biri, and Claude Godart. Security meta-services orchestration architecture. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, 2009.

[BPG11] Aymen Baouab, Olivier Perrin, and Claude Godart. An event-driven approach for runtime verification of inter-organizational choreographies. In *2011 IEEE International Conference on Services Computing (SCC)*, pages 640 –647, july 2011.

[BPG12] Aymen Baouab, Olivier Perrin, and Claude Godart. An optimized derivation of event queries to monitor choreography violations. In Chengfei Liu, Heiko

Ludwig, Farouk Toumani, and Qi Yu, editors, *Service-Oriented Computing (ICSOC'12*, volume 7636 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 2012.

[BPG13] Aymen Baouab, Olivier Perrin, and Claude Godart. Supervision décentralisée des chorégraphies de services. In *31ème Congrès INFormatique des ORganisations et Systèmes d'Information et de Décision (INFORSID 2013), Paris, France*, 2013.

[BPSM+08] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0 (fifth edition). World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

[BSD03] Boualem Benatallah, Quan Z. Sheng, and Marlon Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1) :40–48, 2003.

[BTPT06] Fabio Barbon, Paolo Traverso, Marco Pistore, and Michele Trainotti. Run-time monitoring of instances and classes of web service compositions. *Web Services, IEEE International Conference on*, 0 :63–71, 2006.

[BV05] Paul A. Buhler and José M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Information Technology and Management*, 6(1) :61–87, jan 2005.

[CCDCRD11] Mario Cortes-Cornax, Sophie Dupuy-Chessa, Dominique Rieu, and Marlon Dumas. Evaluating choreographies in bpmnÊ2.0 using an extended quality framework. In *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 103–117. Springer Berlin Heidelberg, 2011.

[CCMN04] Girish B. Chafle, Sunil Chandra, Vijay Mann, and Mangala Gowri Nanda. Decentralized orchestration of composite web services. In *Proceedings of the 13th international World Wide Web conference*, WWW Alt. '04, pages 134–143, New York, NY, USA, 2004. ACM.

[Cha06] K. Mani Chandy. *Event-Driven Applications : Costs, Benefits and Design Approaches*. California Institute of Technology, 2006.

[CNW01] F. Curbera, W. Nagy, and S. Weerawarana. Web services : Why and how, 2001.

[Cor02] Microsoft Corporation. Microsoft biztalk server 2002 enterprise edition. http ://www.microsoft.com, 2002.

[Cru03] Tanguy Crusson. BPM : De la modélisation à l'exécution. positionnement par rapport aux architectures orientées services. *white paper*, 2003.

[CSE+00] Aaron G. Cass, Barbara Staudt, Lerner Eric, K. Mccall, Leon J. Osterweil, and Er Wise. Little-jil/juliette : A process definition language and interpreter. In *in Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000*, pages 754–757, 2000.

[DA07] Helga DUARTE-AMAYA. *Canevas pour la composition de services web avec propriétés transactionnelles*. PhD thesis, Université Joseph Fourier, Grenoble I, Novembre 2007.

[Dav93] Thomas H. Davenport. *Process innovation : reengineering work through information technology.* Harvard Business School Press, Boston, MA, USA, 1993.

[Dav08] Soto David. Augmenter la flexibilité de l'entreprise grâce à l'architecture orientée service (soa). `http://www-935.ibm.com/services/fr/cio/pdf/soa_ibm_gartner_french_issue_final.pdf`, 2008.

[Dav09] Jeff Davis. *Open Source Soa.* Manning Publications Co., Greenwich, CT, USA, 1st edition, 2009.

[DBLL04] Ziyang Duan, Arthur Bernstein, Philip Lewis, and Shiyong Lu. A model for abstract process specification, verification and composition. In *ICSOC '04 : Proceedings of the 2nd international conference on Service oriented computing*, pages 232–241, New York, NY, USA, 2004. ACM Press.

[DGBD09] Marlon Dumas, Luciano García-Bañuelos, and Remco M. Dijkman. Similarity search of business process models. *IEEE Data Eng. Bull.*, 32(3) :23–28, 2009.

[DGD12] Michael Daum, Manuel Götz, and Jörg Domaschka. Integrating cep and bpm– how cep realizes functional requirements of bpm applications. 2012.

[DKLW07] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor : Extending bpel for modeling choreographies. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 296–303. IEEE, 2007.

[DtH01] Marlon Dumas and Arthur H.M. ter Hofstede. Uml activity diagrams as a workflow specification language. pages 76–90. Springer Verlag, 2001.

[DWC11] Ajantha Dahanayake, Richard J. Welke, and Gabriel Cavalheiro. Improving the understanding of bam technology for real-time decision support. *Int. J. Bus. Inf. Syst.*, 7 :1–26, 2011.

[Esp11] EsperTech. Esper - Complex Event Processing, http ://esper.codehaus.org, 2011.

[FBD+11] Walid Fdhila, Aymen Baouab, Karim Dahman, Claude Godart, Olivier Perrin, and François Charoy. Change propagation in decentralized composite web services. In *CollaborateCom*, pages 508–511, 2011.

[Fdh11] Walid Fdhila. *Décentralisation optimisée et synchronisation des procédés métiers inter-organisationnels.* PhD thesis, Université Henri Poincaré, Nancy 1, Octobre 2011.

[Fer04] Andrea Ferrara. Web services : a process algebra approach. In *ICSOC '04 : Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.

[FFBL+96] R. Fielding, H. Frystyk, Tim Berners-Lee, J. Gettys, and J. C. Mogul. Hypertext transfer protocol - http/1.1, 1996.

[FGP10] Adrian Francalanza, Andrew Gauci, and Gordon Pace. Runtime monitoring of distributed systems (extended abstract). Technical report, University of Malta, 2010. WICT.

[FKMU03] Howard Foster, Jeff Kramer, Jeff Magee, and Sebastian Uchitel. Model-based verification of web service compositions. In *18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[FPD+09] Paul Fremantle, Sanjay Patil, Doug Davis, Anish Karmarkar, Gilbert Pilz, Steve Winkler, and Umit Yalçinalp. Web Services Reliable Messaging (WS-ReliableMessaging). *OASIS*, 2009.

[FRMB+12] Walid Fdhila, Stefanie Rinderle-Ma, Aymen Baouab, Olivier Perrin, and Claude Godart. On evolving partitioned web service orchestrations. In *SOCA*, pages 1–6, 2012.

[FUMK04] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Compatibility verification for web service choreography. In *ICWS '04 : Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 738, Washington, DC, USA, 2004. IEEE Computer Society.

[FW04] David C Fallside and Priscilla Walmsley. Xml schema part 0 : primer second edition. *W3C recommendation*, 2004.

[Gaa06] Walid Gaaloul. *La Découverte de Workflow Transactionnel pour la Fiabilisation des Exécutions*. PhD thesis, Université Henri Poincaré, Nancy 1, novembre 2006.

[Gab11] Nesrine Gabsi. *Extension et interrogation de résumés de flux de données*. PhD thesis, Télécom ParisTech, 2011.

[Gas08] Sébastien Gassmann. 8 août 2008. 2008.

[GPB+09] Claude Godart, Olivier Perrin, Karim Baina, Sami Bhiri, Francois Charoy, Walid Gaaloul, Daniela Grigori, and Samir Tata. *Les processus métiers : Concepts, modèles et systèmes*. Lavoisier, 05 2009.

[Gre06] Paul Grefen. Towards dynamic interorganizational business process management. *Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE*, 2006.

[Gro99] Object Management Group. Unified modeling language specification, june 1999.

[Gue10] Nawal Guermouche. *Etude des interactions temporisées dans la composition de services Web*. PhD thesis, IAEM Lorraine, en partenariat avec LORIA, 2010.

[GWC+07] D. Gyllstrom, E. Wu, H-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. Sase : Complex event processing over streams. *CIDR*, 2007.

[HB03] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *ADC '03 : Proceedings of the 14th Australasian database conference*, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[HM08] M. Hariati and D. Meslati. Les composants logiciels et la séparation avancée des préoccupations : Vers une nouvelle approche de combinaison. 2008.

[HS02] Brent Hailpern and Padmanabhan Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1) :4–12, 2002.

[HV09] S. Halle and R. Villemaire. Flexible and reliable messaging using runtime monitoring. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pages 116 –125, sept. 2009.

[JJDM06] Sally St. Amand Jean-Jacques Dubray and Monica J. Martin. *ebXML Business Process Specification Schema Technical Specification v2.0.4. OASIS*. 2006.

[KBP00] A. Kalnins, J. Barzdins, and K. Podnieks. Modeling languages and tools : state of the art, 2000.

[KBR+05] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. *W3C. Available from :*, 2005.

[KBRL04] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web services choreography description language version 1.0. http ://www.w3.org/TR/ws-cdl-10, October 2004.

[KLL09] Ryan K.L. Ko, Stephen S.G. Lee, and Eng Wah Lee. Business process management (bpm) standards : A survey. *Business Process Management journal*, 15(5), 2009.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP'97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer Berlin Heidelberg, 1997.

[Kno10] Jenny Knowdell. The benefits and disadvantages of contract manufacturing. In *IQS Newsroom. Industrial Quick Search, Inc,*, 2010.

[KPS06] Raman Kazhamiakin, Marco Pistore, and Luca Santuari. Analysis of communication models in web service compositions. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *WWW*, pages 267–276. ACM, 2006.

[KR09] Florian Kerschbaum and Philip Robinson. Security architecture for virtual organizations of business web services. *Journal of Systems Architecture, Volume 55, Issue 4, Secure Service-Oriented Architectures (Special Issue on Secure SOA)*, April 2009.

[KSK07] S. Kikuchi, H. Shimamura, and Y. Kanna. Monitoring method of cross-sites' processes executed by multiple ws-bpel processors. In *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services. CEC/EEE 2007*, pages 55 –64, 2007.

[KtHB00] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In Benkt Wangler and Lars Bergman, editors, *CAiSE*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2000.

[KvB04] Mariya Koshkina and Franck van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *SIGSOFT Softw. Eng. Notes*, 29(5) :1–10, 2004.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[Ley01] F. Leymann. Web services flow language. version 1.0. In *Technical Report, International Business Machines Corporation (IBM)*, May 2001.

[LOP05] Denivaldo Cicero Pavão LOPES. *Étude et applications de l ?approche MDA pour des plates-formes de Services Web.* PhD thesis, UFR Sciences et Techniques, Université de Nantes, july 2005.

[LS08] David Luckham and Roy Schulte. Event processing glossary-version 1.1. *Event Processing Technical Society*, 2, 2008.

[Luc02a] David C Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.

[Luc02b] David C. Luckham. *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Luc08] David Luckham. A brief overview of the concepts of cep1. 2008.

[Luc11] David C Luckham. *Event Processing for Business : Organizing the Real-time Enterprise.* Wiley, 2011.

[LW10] Niels Lohmann and Karsten Wolf. *Realizability Is Controllability*, volume 6194 of *Lecture Notes in Computer Science.* Springer Berlin Heidelberg, 2010.

[MAM+95] Alonso Mohan, G. Alonso, C. Mohan, R. Gunthor, D. Agrawal, A. El Abbadi, and M. Kamath. Exotica/fmqm : A persistent message-based architecture for distributed workflow management. pages 1–18, 1995.

[Mei06] Régis Meissonier. Externaliser le système d'information : Décider et manager. In *Economica*, 2006.

[Mil80] R. Milner. A calculus of communicating systems. pages 205–228, 1980.

[MM08] Frederic Montagut and Refik Molva. Bridging security and fault management within distributed workflow management systems. *IEEE Transactions on Services Computing, pp. 33-48*, January-March, 2008.

[Mod11] Business Process Model. Notation (bpmn), v. 2.0,. *OMG : www. omg. org/spec/BPMN/2.0*, 2011.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100 :1–40, September 1992.

[MRD11a] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Event driven monitoring for service composition infrastructures. In *WISE*, pages 38–51. Springer, 2011.

[MRD11b] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Event driven monitoring for service composition infrastructures. In *WISE.* Springer, 2011.

[MSSN04] Jan Mendling, Mark Strembeck, Gerald Stermsek, and Gustaf Neumann. An approach to extract rbac models from bpel4ws processes. *Enabling Technologies, IEEE International Workshops on, pp. 81-86, 13th IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE'04)*, 2004.

[Nak02] S. Nakajima. Verification of web service flows with model-checking techniques. In *CW*, pages 378–385, 2002.

[NRFJ07] Eric Newcomer, Ian Robinson, Max Feingold, and Ram Jeyaraman. Web services coordination (ws-coordination) version 1.1. Technical report, OASIS, 2007.

[omg92] Object management group. `http://www.omg.org/`, 1992.

[Pap03] Michael P. Papazoglou. Web services and business transactions. *World Wide Web*, 6 :49–91, 2003. 10.1023/A :1022308532661.

[PCR85] J. Postel, Request Comments, and J. Reynolds. File transfer protocol (ftp), 1985.

[Pel03] Chris Peltz. Web services orchestration and choreography. *IEEE Computer*, 28(10) :46–52, 2003.

[PGBD10] Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. In *BPM*, pages 276–293, 2010.

[Phi07] Mouricou Philippe. Stratégie d'entreprise : copier ou innover. In *Sciences Humaines, n¡ 183*, pages 38–40, 2007.

[Pla99] D.S. Platt. *Understanding COM+ : the architecture for enterprise development using Microsoft technologies*. Developer technology series. Microsoft Press, 1999.

[PVV11] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In *Proceedings of the 7th international conference on Web services and formal methods*, WS-FM'10, 2011.

[Ram06] Sylvain Rampacek. *Sémantique, interactions et langages de description des services web complexes*. PhD thesis, Université de Reims Champagne-Ardenne, Novembre 2006.

[RFG+10] Mohsen Rouached, Walid Fdhila, Claude Godart, et al. Web services compositions modelling and choreographies analysis. *International Journal of Web Services Research (IJWSR)*, 7(2) :78–110, 2010.

[RKV09] Carsten Rudolph, Nicolai Kuntze, and Zaharina Velikova. Secure web service workflow execution. *Electronic Notes in Theoretical Computer Science, Volume 236, Proceedings of the 3rd International Workshop on Views On Designing Complex Architectures*, 2 April 2009.

[RR07] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Incorporated, 2007.

[SO00a] Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25 :117–134, April 2000.

[SO00b] Wasim Sadiq and Maria E Orlowska. On business process model transformations. *Lecture Notes in Computer Science*, pages 267–280, 2000.

[SS75] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *In : Proceedings of the IEEE 63, Nr. 9, 1278Ð1308*, 1975.

[SS09] Nenad Stefanovic and Dusan Stefanovic. Supply chain business intelligence : Technologies, issues and trends. In Max Bramer, editor, *Artificial Intelligence An International Perspective*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009.

[Ste93] W. Richard Stevens. *TCP/IP illustrated (vol. 1) : the protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[STN+08] S. Subramanian, P. Thiran, N.C. Narendra, G.K. Mostefaoui, and Z. Maamar. On the enhancement of bpel engines for self-healing composite web services. In *Applications and the Internet. SAINT 2008. International Symposium on*, pages 33 –39, 2008.

[Str10] StreamSQL. http ://www.sqlstream.com, 2010.

[Sys08] `http://www.syged.com/gestion-processus-metier/`, 2008.

[TIB06] TIBCO. Cep - complex event processing ou traitement des événements complexes, 2006.

[Tos09] A. Tost. Planning and handling timeouts in service-oriented environments. *in IBM WebSphere Developer Technical Journal*, 2009.

[TS97] R. Thomas and R. Sandhu. Task-based authorization controls (tbac) : A family of models for active and enterprise-oriented authorization management. 1997.

[TYPM09] Hugh Taylor, Angela Yochem, Les Phillips, and Frank Martinez. *Event-Driven Architecture : How SOA Enables the Real-Time Enterprise*. Pearson Education, 2009.

[uHHS09] Irfan ul Haq, Altaf Huqqani, and Erich Schikuta. Aggregating hierarchical service level agreements in business value networks. In *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 176–192. Springer Berlin / Heidelberg, 2009.

[Vag07] Hauke-H. Vagts. Control flow enforcement in workflows in the presence of exceptions. *MasterÕs thesis, TU Darmstadt*, December 2007.

[vdABCC05] Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors. *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings*, volume 3649, 2005.

[vdADtH03] Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. Web service composition languages : Old wine in new bottles ? In *EUROMICRO '03 : Proceedings of the 29th Conference on EUROMICRO*, page 298, Washington, DC, USA, 2003. IEEE Computer Society.

[VDATHKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14 :5–51, July 2003.

[vdAvH04] Wil van der Aalst and Kees van Hee. *Workflow Management : Models, Methods, and Systems (Cooperative Information Systems)*. The MIT Press, March 2004.

[VVL07] Jussi Vanhatalo, Hagen Volzer, and Frank Leymann. Faster and more focused control-flow analysis for business process models through sese decomposition. In *Service-Oriented Computing Ð ICSOC 2007*, volume 4749. 2007.

[W3C92] World wide web consortium. `http://www.w3.org/2002/ws/`, 1992.

[W3C03a] W3C. Simple object access protocol (soap). http ://www.w3.org/TR/soap, 2003.

[W3C03b] W3C. Universal description, discovery, and integration (uddi). http ://www.uddi.org, 2003.

[W3C03c] W3C. Web services description language (wsdl). http ://www.w3.org/TR/wsdl, 2003.

[WDR06] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. *SIGMOD, pp. 407-418*, 2006.

[Wes07] Mathias Weske. *Business Process Management : Concepts, Languages, Architectures*. Springer Verlag, first edition, November 2007.

[wfm98] Workflow management coalition : process definition interchange version 1.0. `http://www.wfmc.org`, 1998.

[Win10] Phil Windley. Static queries, dynamic data : Enabling the real time web, 2010.

[WKK+10] Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, and Daniel Zwink. Cross-organizational process monitoring based on service choreographies. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2485–2490, New York, NY, USA, 2010. ACM.

[WPMW10] Matthias Weidlich, Artem Polyvyanyy, Jan Mendling, and Mathias Weske. Efficient computation of causal behavioural profiles using structural decomposition. volume 6128 of *Lecture Notes in Computer Science*. 2010.

[WPSW04] S. J. Woodman, D. J. Palmer, S. K. Shrivastava, and S. M. Wheater. Notations for the specification and verification of composite web services. In *EDOC '04 : Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04)*, pages 35–46, Washington, DC, USA, 2004. IEEE Computer Society.

[WS-08] Oasis web services transaction (ws-tx) tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx`, 14 Novembre 2008.

[WvdAP+03] Petia Wohed, Wil M.P. van der Aalst, Wil M. P, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of web services composition languages : The case of bpel4ws. In *Proc. of ER ?03, LNCS 2813*, pages 200–215. Springer Verlag, 2003.

[WZM+11] Matthias Weidlich, Holger Ziekow, Jan Mendling, Oliver Günther, Mathias Weske, and Nirmit Desai. Event-based monitoring of process execution violations. In *Proceedings of the 9th international conference on Business process management*, BPM'11, 2011.

[Yil08] Ustun Yildiz. *Décentralisation des procédés métiers : qualité de services et confidentialité*. PhD thesis, Université Henri Poincaré, Nancy 1, Septembre 2008.

[YK04] Xiaochuan Yi and Krys Kochut. Towards efficient integration of complex web services using a unified model for protocol and process. In *International Conference on Internet Computing*, pages 467–474, 2004.

[YM08] Jiankang Yao and Wei Mao. Smtp extension for internationalized email addresses. Internet RFC 5336, September 2008.

[ZBDtH06] Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, and Arthur H. M. ter Hofstede. Let's dance : A language for service behavior modeling. In *OTM Conferences (1)*, pages 145–162, 2006.

[Zim10] Esteban Zimányi. Prototypage d'une application basée sur les architectures orientées évènements : Application à la gestion d'essais cliniques. 2010.

Annexe A

1 Génération des événements de fin de bloc avec ESPER

```
//initialization

Configuration cepConfig = new Configuration();
cepConfig.getEngineDefaults().getEventMeta().setDefaultEventRepresentation
        (Configuration.EventRepresentation.OBJECTARRAY);

//System.out.println("" + Message.class.getName());
cepConfig.addEventType("MsgEvent", Message.class.getName());
EPServiceProvider cep = EPServiceProviderManager.getProvider("myCEPEngine", cepConfig);
final EPRuntime cepRT = cep.getEPRuntime();
StatementAwareUpdateListener myListener = new CEPListener();
EPAdministrator cepAdm = cep.getEPAdministrator();

EPStatement cepStat0 = cepAdm.createEPL(
        "create window EventWin.win:keepall() as MsgEvent");
cepStat0.addListener(myListener);

for (int i = 0; i < CEP_QUERY.length; i++) {

    EPStatement cepStatement = cepAdm.createEPL(CEP_QUERY[i]);

    cepStatement.addListener(myListener);
}


// generate End(C)
EPStatement rule1 = cepAdm.createEPL("select * from "
+ "pattern "
+ "[every e1=MsgEvent(e1.iid=9)]");

rule1.addListener(new UpdateListener() {

public void update(EventBean[] newData, EventBean[] oldData) {
GenerateEnd(EventID++, 0, 0, "C", "End of Block","C", cepRT);

}
});
```

```
// generate End(B211)
EPStatement rule2 = cepAdm.createEPL("select * from "
        + "pattern "
        + "[every e5=MsgEvent(e5.iid=5)]");

rule2.addListener(new UpdateListener() {

    public void update(EventBean[] newData, EventBean[] oldData) {
        GenerateEnd(EventID++, 0, 0, "B211", "End of Block", "B21,B2,C", cepRT);

    }
});

// generate End(B212)
EPStatement rule3 = cepAdm.createEPL("select * from "
        + "pattern "
        + "[every e7=MsgEvent(e7.iid=7)]");

rule3.addListener(new UpdateListener() {

    public void update(EventBean[] newData, EventBean[] oldData) {
        GenerateEnd(EventID++, 0, 0, "B212", "End of Block", "B21,B2,C", cepRT);

    }
});

// generate End(B21)
EPStatement rule4 = cepAdm.createEPL("select * from "
        + "pattern "
        + "[every e5=MsgEvent(e5.endOf like 'B211') or e7=MsgEvent(e7.endOf like 'B212')]");

rule4.addListener(new UpdateListener() {

    public void update(EventBean[] newData, EventBean[] oldData) {
        GenerateEnd(EventID++, 0, 0, "B21", "End of Block", "B21,B2,C", cepRT);

    }
});
```

```
// generate End(B2)
EPStatement rule6 = cepAdm.createEPL("select * from "
+ "pattern "
+ "[every e8=MsgEvent(e8.iid=8) and e7=MsgEvent(e7.endOf like 'B21')]");

rule6.addListener(new UpdateListener() {

public void update(EventBean[] newData, EventBean[] oldData) {
GenerateEnd(EventID++, 0, 0, "B2", "End of Block", "B2,C", cepRT);
    }
});

// generate End(B11)
EPStatement rule5 = cepAdm.createEPL("select * from "
        + "pattern "
        + "[every e2=MsgEvent(e2.iid=2)]");

rule5.addListener(new UpdateListener() {

    public void update(EventBean[] newData, EventBean[] oldData) {

        GenerateEnd(EventID++, 0, 0, "B11", "End of Block", "B11,B1,C", cepRT);

    }
});

// generate End(B1)
EPStatement rule7 = cepAdm.createEPL("select * from "
        + "pattern "
        + "[every b11=MsgEvent(b11.endOf like 'B11')]");

rule7.addListener(new UpdateListener() {

    public void update(EventBean[] newData, EventBean[] oldData) {
        GenerateEnd(EventID++, 0, 0, "B1", "End of Block", "B1,C", cepRT);
    }
});
```

2 Génération aléatoire de séquences de messages

```
private static int myrandom(int min, int max) {
    return (int) (min + Math.random() * (max - min + 1));
}
public static int[] generate_sequence(int n) {
    int seq1[] = {1, 2, 3, 4, 5, 8, 9};  // 7 msg
    int seq2[] = {1, 2, 3, 4, 5, 8, 9};    // 7 msg
    int seq3[] = {1, 2, 1, 2, 3, 6, 8, 7, 9}; // 9 msg

    int[] seq;

    int r = myrandom(1, 9);
    int i = myrandom(1, 9);

    //System.out.println("Random : " + r);
    //System.out.println("Random : " + i);

    n /= 5;
    if (n < 3) {
        if ((r > 2) && (r < 7)) {
            seq1[r] = i;
        }

        cpt_msg += 7;
        seq = seq1;
    } else if (n > 3) {
        if ((r > 2) && (r < 7)) {
            seq2[r] = i;
        }
        cpt_msg += 7;
        seq = seq2;
    } else {
        if ((r > 1) && (r < 9)) {
            seq3[r] = i;
        }
        cpt_msg += 9;
        seq = seq3;
    }
    return seq;
}
// Generate simple msg event notifications
public static void GenerateMsg(int Eid, int Cid, int Iid, String end, String cmt, EPRuntime cepRT) {
    //int Iid = (int) generator.nextInt(9)+1;
    long timeStamp = System.currentTimeMillis();
    //String Cid = "C1";
    Message event = new Message(Eid, Cid, Iid, end, timeStamp, cmt);
    System.out.println(":" + event);
    cepRT.sendEvent(event); //// envoi d'event dans la file

}
```

3 Configuration du CEPListener

```
/**
 *
 * @author aymenbaouab
 */
 public class CEPListener implements StatementAwareUpdateListener {


    public void update(EventBean[] ebs, EventBean[] ebs1, EPStatement eps1, EPServiceProvider epsp) {


        Message ei = (Message) ebs[0].get("ei");
        Message ej;
        int ii1, ii2; // Les deux événements responsables
        ii1 = ei.getIid();
        try {
            ej = (Message) ebs[0].get("ej");
            ii2 = ej.getIid();
            // System.out.println("ej = " + ej.toString());
        } catch (Exception e) {
            ii2 = 0;
            //System.out.println("ej = null ");
        }


        Main.stock_viol.add(ii1);
        Main.stock_viol.add(ii2);

        /*....

         *
         *
```

Résumé

Durant la dernière décennie, les architectures orientées services (SOA) d'une part et la gestion des processus business (BPM) d'autre part ont beaucoup évolué et semblent maintenant en train de converger vers un but commun qui est de permettre à des organisations complètement hétérogènes de partager de manière flexible leurs ressources dans le but d'atteindre des objectifs communs, et ce, à travers des schémas de collaboration avancée. Ces derniers permettent de spécifier l'interconnexion des processus métier de différentes organisations. La nature dynamique et la complexité de ces processus posent des défis majeurs quant à leur bonne exécution. Certes, les langages de description de chorégraphie aident à réduire cette complexité en fournissant des moyens pour décrire des systèmes complexes à un niveau abstrait. Toutefois, rien ne garantit que des situations erronées ne se produisent pas suite, par exemple, à des interactions "mal" spécifiées ou encore des comportements malhonnêtes d'un des partenaires.

Dans ce manuscrit, nous proposons une approche décentralisée qui permet la supervision de chorégraphies au moment de leur exécution et la détection instantanée de violations de séquences d'interaction. Nous définissons un modèle de propagation hiérarchique pour l'échange de notifications externes entre les partenaires. Notre approche permet une génération optimisée de requêtes de supervision dans un environnement événementiel, et ce, d'une façon automatique et à partir de tout modèle de chorégraphie.

Mots-clés: Chorégraphie de services, processus inter-organisationnel, processus métier, BPM, service web, supervision, décentralisation, CEP.

Abstract

Cross-organizational service-based processes are increasingly adopted by different companies when they can not achieve goals on their own. The dynamic nature of these processes poses various challenges to their successful execution. In order to guarantee that all involved partners are informed about errors that may happen in the collaboration, it is necessary to monitor the execution process by continuously observing and checking message exchanges during runtime. This allows a global process tracking and evaluation of process metrics. Complex event processing can address this concern by analyzing and evaluating message exchange events, to the aim of checking if the actual behavior of the interacting entities effectively adheres to the modeled business constraints.

In this thesis, we present an approach for decentralized monitoring of cross-organizational choreographies. We define a hierarchical propagation model for exchanging external notifications between the collaborating parties. We also propose a runtime event-based approach to deal with the problem of monitoring conformance of interaction sequences. Our approach allows for an automatic and optimized generation of rules. After parsing the choreography graph into a hierarchy of canonical blocks, tagging each event by its block ascendancy, an optimized set of monitoring queries is generated. We evaluate the concepts based on a scenario showing how much the number of queries can be significantly reduced.

Keywords: Choreography, Monitoring, Web service, Business Process Management (BPM), decentralization, Complex Event Processing (CEP).