AVERTISSEMENT

LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n°92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992
http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

En vue de l'obtention du doctorat de

L'UNIVERSITÉ DE TOULOUSE
et de L'UNIVERSITÉ DE SFAX

Délivré par

L'Université Toulouse 1 Capitole

et de La Faculté des Sciences Economiques et de Gestion de Sfax

Discipline ou spécialité : *Informatique*

Présentée et soutenue par ***Imen BEN SAID***

Le 15 Mai 2017

BPMN4V pour la modélisation de versions de processus intra- et inter-organisationnels

---

JURY

Mme Selmin NURCAN	Maître de conférences HDR, Université Paris 1 Panthéon–Sorbonne	Rapporteur
M. Djamal BENSLIMANE	Professeur, Université Claude Bernard Lyon 1	Rapporteur
M. Mourad Chabane OUSSALAH	Professeur, Faculté des Sciences de Nantes	Examinateur
M. Faiez GARGOURI	Professeur, ISIMS – Université de Sfax	Examinateur
M. Chihab HANACHI	Professeur, Université Toulouse 1–Capitole	Directeur
M. Rafik BOUAZIZ	Professeur, FSEG – Université de Sfax	Directeur
M. Eric ANDONOFF	Maître de conférences, Université Toulouse 1– Capitole	Co-encadrant

---

Ecole doctorale : *Ecole Doctorale Mathématiques Informatique Télécommunication de Toulouse*

Laboratoires de recherche : *Institut de Recherche en Informatique de Toulouse-IRIT-UMR 5505*

Multimedia InfoRmation systems and Advanced Computing Laboratory –MIRACL- Sfax

Equipes d'acceuil : Systèmes Multi-Agents Coopératifs & Equipe des Bases de Données Avancées

Imen BEN SAID

BPMN4V pour la modélisation de versions de processus intra- et inter-organisationnels

Directeurs de thèse :
Chihab HANACHI, Professeur à l'Université Toulouse 1 – Capitole, Toulouse
Rafik BOUAZIZ, Professeur à la Faculté des Sciences Economiques et de Gestion, Sfax

Co-encadrant de thèse :
Eric Andonoff, Maître de conférences Toulouse 1 – Capitole, Toulouse

Résumé

Nos travaux de recherche abordent la problématique de la modélisation des processus intra- et inter-organisationnels flexibles à l'aide des versions. En effet, le concept de version est un concept approprié pour prendre en compte les changements que peuvent subir les schémas de processus. On peut ainsi définir une nouvelle version de processus lorsque des modifications significatives doivent être apportées à son schéma.

Différentes contributions de l'état de l'art ont abordé le versionnement des processus. Chacune de ces contributions a constitué une avancée pour le versionnement des processus. Mais elles ont en commun de principalement considérer le versionnement de la perspective comportementale des processus, sans étudier les impacts du versionnement sur les perspectives organisationnelles et informationnelles, qui sont pourtant fondamentales dans la définition des processus. De plus, il est également nécessaire lorsqu'on aborde la problématique du versionnement des processus de considérer une autre perspective, la perspective contextuelle, qui permet de modéliser les situations d'utilisation des versions. En outre, ces contributions ne s'appuient pas sur les standards existants et notamment sur BPMN (Business Process Model and Notation) qui est reconnu comme le standard pour la modélisation de schémas de processus. Enfin, ces contributions abordent uniquement la flexibilité des processus intra-organisationnels.

Cette thèse a pour objectif de pallier ces deux faiblesses. Plus précisément, elle propose BPMN4V une extension du standard BPMN pour la modélisation de versions de processus tout en considérant les perspectives de modélisation des processus. Ainsi les contributions proposées sont les suivantes. En ce qui concerne la modélisation des versions des processus intra-organisationnels, nous avons introduit BPMN4V-PP (BPMN for Versions of Private Processes), qui est une extension de BPMN pour la modélisation de versions de processus intra-organisationnels, représentées en BPMN par des processus privés. Les aspects statiques et dynamiques du versionnement sont abordés. Puis nous avons étendu BPMN4V-PP pour modéliser le contexte d'utilisation des processus. Cette notion de contexte est importante pour aider le concepteur (i) à définir pourquoi une version de processus a été modélisée et (ii) à sélectionner la version qui convient à une situation particulière. En ce qui concerne la modélisation des versions de processus inter-organisationnels, modélisées comme des collaborations ou des chorégraphies en BPMN, nous avons défini BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies) qui étend BPMN pour la modélisation et la manipulation des versions de collaborations et de chorégraphies. Nous avons aussi défini six patrons d'adaptation qui sont des opérations de haut niveau facilitant la modification des schémas des versions de collaborations. Nous avons également proposé une démarche pour la génération automatique d'une version de chorégraphie à partir d'une version de collaboration. Nous avons finalement implanté ces propositions dans un plug-in Eclipse, appelé BPMN4V-Modeler, permettant d'assister les concepteurs lors de la modélisation des versions de processus selon les méta-modèles BPMN4V-PP et BPMN4V-CC. Nous avons évalué nos contributions en mesurant notamment les performances de l'outil BPMN4V-Modeler.

Institut de Recherche en Informatique de Toulouse – UMR 5505 CNRS
Université Toulouse 1 – Capitole, 2 rue du Doyen-Gabriel-Marty, 31042 Toulouse cedex

Multimedia InfoRmation systems and Advanced Computing Laboratory
Pôle technologique de Sfax : Route de Tunis Km 10 B.P. 242, 3021 Sfax

Imen BEN SAID

BPMN4V for modeling versions of intra- and inter-organisational processes

Supervisors:
Chihab HANACHI, Professor at Toulouse 1– Capitole University, Toulouse
Rafik BOUAZIZ, Professor at Faculty of Economics and Management, Sfax

Co-Supervisor:
Eric Andonoff, Associate Professor at Toulouse 1 – Capitole University, Toulouse

Abstract

Our research deals with modeling flexible intra and inter-organizational processes using versions. Indeed, the notion of version has been recognized as a key notion to keep track of changes on process schemas. Thus it is possible to define a new process version for each significant modification of its schema.

Several contributions addressing process flexibility using versions have been proposed in the literature. Each of these contributions provides solutions for process versioning. However, they mainly focus on the behavioral perspective of processes neglecting the informational and organizational perspectives, which are nevertheless important to have a comprehensive view of processes. Furthermore, the contextual perspective of processes must also be considered to characterize the situations in which versions of processes have to be used. Moreover, the proposed solutions are specific and have no chance to be used by process designers since they do not support standards for process modeling such as BPMN (Business Process Model and Notation). Finally, these contributions focus only on intra- organizational processes.

This thesis aims at overcoming these two weaknesses, introducing BPMN4V, an extension of BPMN with versions to address intra and inter-organizational process flexibility. More precisely, the contributions of this thesis are the following. Regarding flexibility of intra-organizational processes, we have introduced BPMN4V-PP (BPMN for Versions of Private Processes) which is an extension of BPMN for modeling versions of intra-organizational processes, modeled in BPMN as private processes, considering both static and dynamic aspects of process versions. BPMN4V takes into account the three main perspectives of business processes, *i.e.,* the behavioral perspective, the informational perspective and the organizational perspective. In order to consider the contextual perspective, we have also proposed another extension to BPMN allowing the definition of versions context. This perspective is fundamental to help process designers in featuring why a process version is defined and in selecting a particular process version according to a concrete situation. Regarding flexibility of inter-organizational processes, modeled as collaborations and choreographies in BPMN, this thesis has recommended BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies) for modeling and handling versions of collaboration. Secondly, it has introduced six new adaptation patterns, which are high-level operations for collaborative process schema update. Then, it has defined a mapping approach for deducing version of choreography from version of collaboration. Finally, we have implemented our contributions in the BPMN4V-Modeler plug-in, which assists process designer in modeling versions of processes according to BPMN4V-PP and BPMN4V-CC. We have evaluated our contributions mainly measuring BPMN4V-Modeler performances.

Institut de Recherche en Informatique de Toulouse – UMR 5505 CNRS
Université Toulouse 1 – Capitole, 2 rue du Doyen-Gabriel-Marty, 31042 Toulouse cedex

Multimedia InfoRmation systems and Advanced Computing Laboratory
Pôle technologique de Sfax : Route de Tunis Km 10 B.P. 242, 3021 Sfax

Dédicaces

Aux plus chers à mon cœur :
mes parents,
mon mari, mon fils
et mes sœurs

Remerciements

Pour dieu, donneur de vie, de force, et de sagesse…

Je remercie les membres du Jury pour l'honneur qu'ils m'ont fait de s'intéresser à ce travail et d'avoir accepté de l'évaluer. Plus précisément, je remercie :

- Madame Selmin Nurcan et Monsieur Djamal BENSLIMANE pour avoir rapporté sur ma thèse. Leurs travaux constituent des repères scientifiques.
- Messieurs Faiez GARGOURI et Mourad Chabane OUSSALAH pour avoir accepté de participer au Jury.
- Messieurs Rafik BOUAZIZ et Chihab HANACHI pour avoir dirigé et encadré mes recherches, pour leur rigueur scientifique, leur critiques constructives, leur soutien à la fois en recherche mais aussi sur d'autres aspects de la vie.
- Monsieur Eric ANDONOFF, pour avoir encadré et suivi cette thèse, pour ses conseils judicieux, sa disponibilité, son gradeur d'âme, ses lectures minutieuses et ses remarques précieuses, qui m'ont été d'un grand intérêt.

Mes remerciements vont également à mes parents pour les sacrifices consentis à mon égard. Ils ont tout fait pour mon bonheur et ma réussite. Leurs conseils prodigieux et leurs principes dans la vie demeurent toujours présents dans mon esprit. Qu'ils trouvent dans ce modeste travail, le témoignage de ma profonde affection et mon attachement indéfectible. Que Dieu leur réserve bonne santé et longue vie.

Je remercie aussi mon mari, pour son soutien, ses conseils, ses aides et son sacrifice de tant de manières pour parvenir à la réalisation de ce projet. Que Dieu le garde et le protège.

Je remercie également toute ma famille et tous mes amis, qui m'ont encouragé tout au long de ces années de thèse. Merci d'avoir répondu présent aux moments les plus importants.

Je vous apprécie !

Table des matières

Table des figures

Table des Tableaux

Introduction Générale

I. Contexte général

Les travaux de cette thèse s'intéressent à la problématique de la gestion des processus métier, appelés aussi processus d'entreprise (Business Process Mangement BPM).

Dans ce contexte, un processus métier est défini comme *un ensemble d'activités connectées qui réalisent collectivement un objectif particulier. Chaque processus appartient à une seule organisation mais il peut interagir avec d'autres processus appartenant à d'autres organisations* [Weske, 2012]. La littérature distingue deux types de processus métier : les processus intra-organisationnels et les processus inter-organisationnels. Un processus est dit intra-organisationnel s'il est réalisé dans une seule organisation alors qu'un processus est dit inter-organisationnel s'il s'étend sur plusieurs organisations, appelées aussi partenaires. Plus précisément, la définition d'un processus repose sur quatre principales perspectives, à savoir la perspective comportementale (décrivant les activités et leur enchaînement), la perspective organisationnelle (décrivant les ressources impliquées dans l'accomplissement des activités), la perspective informationnelle (décrivant les données produites ou consommées par les activités) et la perspective contextuelle (décrivant le contexte dans lequel les processus opèrent) [Aalst *et al.*, 2003(a)] [Nurcan et Edme, 2005].

Depuis l'apparition des systèmes d'information orientés processus (ProcessAware Information System dans la littérature anglophone) [Reichert et Weber, 2012] [Dumas *et al.,* 2013], la bonne gestion des processus métier est reconnue comme un facteur de succès pour les organisations [Weske, 2012]. En effet, les processus participent grandement à l'amélioration des performances des entreprises, notamment parce qu'ils contribuent à la mise en lumière des objectifs que ces entreprises doivent atteindre, à l'organisation et à la rationalisation du travail des acteurs de l'entreprise, à la définition et à la formalisation de la coopération entre ces acteurs, et à l'exploitation de l'information.

Une bonne gestion des processus métier consiste à traiter le processus tout au long de son cycle de vie, qui inclut les étapes d'analyse, de modélisation, d'optimisation, de contrôle et d'exécution. Nous pensons que l'étape de modélisation des processus est fondamentale puisque les autres en dépendent. La modélisation d'un processus consiste à concevoir une définition de ce processus qui peut, par la suite, être instanciée puis exécutée par un moteur d'exécution. De nombreux paradigmes de modélisation de processus ont été proposés dans la littérature, parmi lesquels nous citons le paradigme guidé par les données, le paradigme guidé par les contraintes, le paradigme guidé par les buts et le paradigme guidé par les activités. Dans le présent travail, nous adoptons le paradigme guidé par les activités puisqu'il est supporté par la majorité des systèmes de gestion de processus (Business Process Management System ou BPMS), issus du monde industriel ou académique (*e.g.*, [Dadam et Reichert, 2009], FLOWer [Aalst *et al.,* 2005], [Aalst et Berens, 2001]). De même, plusieurs langages et notations de processus adoptent le paradigme guidé par les activités, tels que les

réseaux de Petri, le diagramme d'activités d'UML, Yawl et BPMN. Dans cette thèse, nous nous intéressons à la modélisation des processus en utilisant la notation BPMN 2.0 (Business Process Model and Notation) [OMG, 2011]. En fait, BPMN est le standard de modélisation de processus le plus répandu. Il considère les trois premières perspectives de modélisation de processus. De plus, BPMN offre trois types de diagrammes pour prendre en compte les différents types de processus. Les diagrammes des processus privés permettent de modéliser les processus intra-organisationnels. Les diagrammes de collaborations et les diagrammes de chorégraphies permettent de modéliser les processus inter-organisationnels qui appartiennent à des organisations distinctes.

II. Problématique

II.1. La flexibilité des processus

La technologie BPM est devenue, de nos jours, une technologie mûre pour les processus stables qui ne changent pas dans le temps. En effet, plusieurs BPMS ont vu le jour et certains sont commercialisés et utilisés dans l'industrie. Cependant, la technologie BPM n'apporte pas de véritable solution pour traiter les processus qui évoluent dans le temps. De tels processus sont qualifiés de processus flexibles. Le problème de la modélisation des processus flexibles est d'une importance majeure puisque les entreprises et les organisations sont plongées dans un environnement économique qui devient de plus en plus ouvert, dynamique et concurrent, dans lequel les seules à rester compétitives sont celles qui peuvent s'adapter de manière rapide et efficace à cet environnement fortement fluctuant. Ainsi, ces entreprises et organisations sont amenées à réviser et modifier fréquemment leurs processus pour prendre en compte les nouveaux besoins de leurs clients ou pour se conformer à des nouvelles réglementations du travail.

Actuellement, plusieurs travaux de recherche dans le domaine des BPM traitent du problème de la flexibilité des processus, d'où un état de l'art riche. Les travaux de [Chaâbane, 2012] définissent la flexibilité comme étant *la capacité d'un processus à prendre en compte les changements prévisibles et non prévisibles qui se produisent suite à une modification de son schéma d'exécution, les acteurs qu'il invoque et / ou les ressources informationnelles qu'il manipule, tout en gardant l'efficacité et l'intégralité de la définition du processus.* Ainsi un processus est qualifié de flexible s'il est capable de prendre en compte tout type de changement (prévisible ou non) qui touche les principales perspectives impliquées dans sa définition.

Plusieurs techniques de modélisation ont été proposées pour suivre les changements que peut subir un processus. Parmi ces techniques nous citons la modélisation tardive, la liaison tardive, les variantes et le versionnement. Cette dernière consiste à créer une version de processus pour représenter les changements significatifs opérés sur sa définition. Ainsi plusieurs versions de définition pourront être définies pour un même processus. La technique de versionnement, telle qu'elle est utilisée dans

les domaines des systèmes d'information (BD, entrepôts de données, ontologies) et du génie logiciel, est reconnue pour sa capacité à modéliser : (i) l'évolution d'une entité tout en gardant trace des anciennes versions, qui peuvent être réutilisées si la même situation se présente, et (ii) des situations alternatives. C'est pour ces raisons que nous avons choisi, dans ce projet de thèse, de traiter le problème de la flexibilité des processus en adoptant une approche basée sur les versions.

Parmi les travaux utilisant la technique de versionnement pour aborder la flexibilité des processus, nous citons ceux de [Ekanayake *et al.,* 2011], [Chaâbane, 2012] et [Zhao et Liu, 2013]. Ces travaux constituent une avancée pour le versionnement des processus. Ils ont cependant les inconvénients suivants. Les auteurs de [Ekanayake *et al.,* 2011] et [Zhao et Liu, 2013] ont abordé la problématique de la flexibilité des processus en adoptant le paradigme dirigé par les activités et la technique de versionnement, mais ils se sont focalisés sur la perspective comportementale des processus et n'ont pas considéré les trois autres perspectives (organisationnelle, informationnelle et contextuelle). De plus, la notion de version s'applique uniquement sur la notion de processus ou de sous-processus. Or la granularité du versionnement dans le domaine du BPM peut être plus fine. Les travaux de [Chaâbane, 2012] couvrent les quatre perspectives de modélisation des processus. Plus précisément, ils ont donné lieu à la définition d'un méta-modèle, appelé VBP2M (Versioned Business Process MetaModel). Dans ce méta-modèle, le versionnement est appliqué sur les concepts permettant de garder trace des changements relatifs au travail réalisé dans un processus, au mode d'organisation relatif à ce travail, aux ressources impliquées et aux informations manipulées. De plus, ces travaux ont traité les problèmes relatifs à la gestion des versions des processus en proposant un langage, nommé VBPQL (*Versioned Business Process Query Language*), permettant la définition, la manipulation et l'interrogation des versions de processus. Si cette contribution traite la flexibilité des processus en considérant toutes les perspectives de modélisation des processus, elle ne s'appuie pas sur les standards de modélisation des processus en proposant un méta-modèle propre à eux et elle ne traite que la flexibilité dans un cadre intra-organisationnel. C'est pour cette raison que nous avons opté pour une solution basée sur un standard tout en considérant les quatre perspectives de modélisation de processus.

II.2. Problématique abordée

Cette thèse aborde la problématique de la modélisation des versions des processus définis à l'aide de la notation BPMN. Plus précisément, il s'agit de proposer des solutions pour rendre les diagrammes de BPMN capables de supporter la notion de version. Cette thèse se propose d'apporter des réponses aux points suivants :

1) Modélisation des versions des processus intra-organisationnels : BPMN 2.0 modélise ce type de processus par le diagramme de processus privé. Il s'agit (i) d'identifier les concepts du méta-modèle de BPMN 2.0, relatifs au diagramme de processus privé (*e.g., Process*, *Activity*, *Gateway*), qui doivent être étendus

afin qu'ils puissent permettre de modéliser les versions, (ii) d'étendre ces concepts pour le support du versionnement, (iii) d'assurer la manipulation des versions, et (iii) de définir la situation dans laquelle une version peut être utilisée, en décrivant son contexte d'utilisation.

2) Modélisation des versions des processus inter-organisationnels : BPMN 2.0 modélise ce type de processus soit par un diagramme de collaboration soit par un diagramme de chorégraphie. Pour aborder le problème de la modélisation des versions de collaboration, il faut (i) identifier les concepts du méta-modèle de BPMN 2.0, relatifs au diagramme de collaboration (*e.g., Collaboration*, *Message*, *Participant*), qui méritent d'être étendus afin qu'ils puissent permettre de modéliser les versions, (ii) étendre ces concepts pour permettre leur versionnement, et (iii) définir les opérations nécessaires à la manipulation des versions. Quant au diagramme de chorégraphie, qui est considéré comme une abstraction d'un diagramme de collaboration, il s'agit de définir des mécanismes qui permettent de générer automatiquement une version de chorégraphie à partir d'une version de collaboration.

3) Proposition d'un éditeur pour la modélisation et la manipulation des versions de processus (intra- et inter-organisationnels). Il s'agit d'implanter les solutions apportées aux deux points précédents.

En résumé, les contributions de cette thèse traitent de la modélisation des processus flexibles à l'aide de la technique de versionnement, en proposant des extensions au méta-modèle de BPMN pour permettre la modélisation et la gestion de versions des processus et de leurs composants. Cette thèse ne propose ni une démarche méthodologique qui permet la modélisation de ces versions, ni la mise en œuvre de ces versions. Ces aspects seront traités dans des travaux futurs, comme indiqué en conclusion.

III. Contributions proposées

Les contributions de cette thèse sont les suivantes :

1) Le méta-modèle BPMN4V-PP (BPMN for Versions of Private Processes), qui étend le méta-modèle de BPMN 2.0 en intégrant la notion de version afin qu'il puisse instancier les versions des processus privés (*i.e.,* les processus intra-organisationnels). Ce méta-modèle comporte sept classes versionnables (*Process*, *Sub-Process*, *Task*, *Event*, *Resource, ResourceRole*, *ItemAwareElement*) pour lesquelles la gestion du changement sera prise en compte. L'aspect statique permet la modélisation des versions de processus conformément à BPMN4V-PP [Ben Said *et al.,* 2014(a)]. Quant à l'aspect dynamique, il traite la manipulation des versions modélisées comme des instances du méta-modèle BPMN4V-PP. A cet effet, nous avons proposé un diagramme d'états-transitions décrivant les états d'une version et les transitions possibles permettant de la faire passer d'un état à un autre, sachant que les deux principaux états sont « *En travail* » et « *Stable* ».

Les opérations permettant la manipulation des versions et assurant leurs transitions entre les états sont les suivantes : opération de création, opération de modification, opération de validation et opération de dérivation. La spécification de chacune de ces opérations varie d'un concept versionnable à un autre [Ben Said *et al.,* 2015(a)].

2) Le méta-modèle BPMN4V-Context (BPMN for Version Context), qui étend le méta-modèle BPMN4V-PP afin de pouvoir prendre en considération le contexte des versions de processus privés. BPMN4V-Context comporte les concepts permettant de prendre en compte les situations qui ont incité la création et l'utilisation d'une version [Ben Said *et al.,* 2014(b)].

3) Le méta-modèle BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies), qui étend le méta-modèle de BPMN 2.0 pour qu'il puisse supporter les versions des processus inter-organisationnels. BPMN4V-CC comporte des classes versionnables permettant de suivre les changements d'une collaboration, des processus qui composent cette collaboration et des messages échangés lors de l'interaction entre ces processus. De plus, nous avons apporté des solutions assurant la dynamique des versions modélisées conformément à BPMN4V-CC. Nous avons spécifié les opérations permettant la création, la modification, la validation et la dérivation des versions de collaboration [Ben Said *et al.,* 2015(b)].

4) Six patrons d'adaptation facilitant la création et la modification des versions de collaboration. Ces patrons, définis comme des primitives de haut niveau, regroupent un ensemble d'actions élémentaires qui correspondent à des primitives de bas niveau [Ben Said *et al.,* 2015(b)].

5) Une démarche pour la génération automatique d'une version de chorégraphie à partir d'une version de collaboration [Ben Said *et al.,* 2016].

6) BPMN4V-Modeler, qui est un outil dédié à la modélisation et la manipulation des versions des processus privés et des collaborations et à la génération automatique des versions de chorégraphie conformément aux méta-modèles BPMN4V-PP et BPMN4V-CC [Ben Said *et al.,* 2015(a)].

IV. Organisation du mémoire

Ce mémoire est organisé en cinq chapitres.

Le premier chapitre rappelle en premier lieu les concepts de base du domaine des processus métier. Il présente ensuite les différentes perspectives à prendre en compte lors de la modélisation des processus. Il détaille également les types de coordination et les formes d'interopérabilité des processus inter-organisationnels. Finalement, il présente la notation BPMN dédiée à la modélisation graphique des processus métier.

Le deuxième chapitre motive tout d'abord le problème de la flexibilité. Il présente ensuite les deux taxonomies les plus représentatives de l'état de l'art, puis

effectue un survol des travaux traitant de la problématique de la flexibilité des processus. Il défend ensuite l'utilisation de la technique de versionnement pour aborder cette problématique et montre les capacités de cette technique pour supporter les différents types de flexibilité. Puis il détaille les travaux les plus représentatifs de la littérature traitant du problème de la flexibilité à l'aide des versions. Finalement, il présente la perspective contextuelle des processus permettant la définition de la situation dans laquelle une version de processus sera utilisée.

Le troisième chapitre présente tout d'abord le méta-modèle BPMN4V-PP qui est une extension du méta-modèle de BPMN 2.0 permettant la prise en compte des processus intra-organisationnels flexibles. Il détaille par la suite l'aspect dynamique des versions de processus qui sont des instances du méta-modèle BPMN4V-PP. Finalement, il présente les extensions apportées à BPMN pour la modélisation du contexte des versions de processus.

Le quatrième chapitre traite de la flexibilité des processus inter-organisationnels modélisés par des collaborations et des chorégraphies dans BPMN 2.0. Il présente en premier lieu le méta-modèle BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies) pour la modélisation des versions de collaborations. Ensuite, Il détaille les opérations permettant la manipulation des versions de collaborations modélisées conformément à BPMN4V-CC. Il présente également six patrons d'adaptation facilitant la création et la modification des versions de collaboration. Finalement, il présente une démarche de déduction d'une version de chorégraphie à partir d'une version de collaboration.

Le cinquième chapitre explique la mise œuvre de nos propositions à travers l'implémentation de l'éditeur BPMN4V-Modeler dédié à la modélisation et la gestion des versions des processus intra- et inter-organisationnels.

Enfin, nous concluons ce mémoire en rappelant nos contributions et les originalités de notre travail, en précisant également ses limites. Nous terminons en indiquant les perspectives de recherche envisageables.

Chapitre I – Contexte d'étude

Résumé. *Ce premier chapitre présente d'abord les concepts de base des processus intra- et inter-organisationnels. Il fixe ainsi la terminologie qui sera utilisée dans ce mémoire. Il distingue ensuite les différentes perspectives à considérer pour modéliser des processus. Il détaille, aussi, les types de coordination et les formes d'interopérabilité des processus inter-organisationnels. Finalement, il présente le standard BPMN dédié à la modélisation graphique des processus.*

Sommaire du Chapitre I.

De nos jours, le fonctionnement des organisations est basé sur le concept de processus métier, tels que les processus de production ou de recrutement. Ainsi, afin d'améliorer leurs performances, réduire leurs coûts de fonctionnement, garantir la qualité de leurs services, les organisations doivent veiller au bon fonctionnement et à l'optimisation de leurs processus. En effet, les processus jouent un rôle fondamental dans les systèmes d'information des entreprises : ils servent notamment de support à l'alignement entre ces systèmes et les stratégies fixées par les organisations [Rolland *et al.,* 2010] [Simonin *et al.,* 2013].

L'objectif de ce chapitre est de présenter le cadre dans lequel s'inscrit notre travail. La section 1 fixe tout d'abord la terminologie du domaine en présentant les concepts de base des processus métier. Elle présente également les étapes du cycle de vie d'un processus. La section 2 explique les différentes perspectives à prendre en compte lors de la modélisation d'un processus afin d'obtenir un modèle complet et rigoureux. La section 3 se focalise sur les processus inter-organisationnels, *i.e.,* les processus traversant les frontières des organisations. Elle détaille les types de coordination et les formes d'interopérabilité de ces processus. Finalement, la section 4 détaille la notation BPMN (Business Process Model and Notation) dédiée à la modélisation graphique des processus métier.

I. Concepts de base

La gestion et l'automatisation des processus métier étaient la vocation de plusieurs technologies durant les deux dernières décennies. En effet, les années 90 ont vu l'émergence des technologies de workflow, d'IAE (Intégration d'Applications d'Entreprise) et de SGD (Système de Gestion de Documents) offrant des outils de transmission de données et des services de gestion du mode d'accomplissement de processus individuels répétables et moins complexes. Le début des années 2000 marque l'apparition de la technologie BPM (Business Process Managment). Cette technologie permet d'aboutir à une vue, meilleure et plus globale, de l'ensemble des processus métier d'une entreprise et de leurs interactions afin d'être en mesure de les automatiser et optimiser. Cette section introduit la technologie BPM en donnant, dans un premier temps, quelques définitions de base et en expliquant, dans un second temps, le cycle de vie d'un processus.

I.1. Définitions

Définition 1.1 – Processus d'entreprise ou processus métier (Business process)

> *Un processus métier est un ensemble coordonné d'activités permettant la réalisation d'un objectif. Chaque processus appartient à une seule organisation mais il peut interagir avec d'autres processus appartenant à d'autres organisations [Weske, 2012].*

La littérature distingue deux types de processus métier : les processus intra-organisationnels et les processus inter-organisationnels. Un processus intra-organisationnel est un processus appartenant à une seule organisation. Comme exemple de processus intra-organisationnel, on peut citer *le processus d'intervention chirurgicale* qui comprend les activités suivantes : *Admission du patient, Préparation du patient, Réalisation de l'intervention, Suivi du patient et Rédaction du rapport d'intervention*. Ce processus fait remplir et circuler des documents, tels que *la lettre d'admission* et *le rapport de l'intervention*, et implique des ressources organisationnelles, telles que *la secrétaire d'accueil des patients*, *le médecin* et *la salle d'opération*. A l'opposé, un processus inter-organisationnel est un processus qui s'étend sur plusieurs organisations appelées aussi partenaires. Comme exemple de processus inter-organisationnel, on peut citer *le processus d'organisation de voyage* impliquant les organisations suivantes : *une agence de voyage, une compagnie aérienne et un client*.

Définition 1.2 – Activité ou Tâche (Activity or Task)

> *Une activité est une unité de travail représentant une étape d'un processus. Une activité fait appel à des ressources humaines logicielles et / ou matérielles pour son accomplissement. [Weske, 2012]*

La littérature distingue les activités privées des activités coopératives. Une activité privée est une activité interne à un processus intra-organisationnel alors qu'une activité coopérative concerne un processus inter-organisationnel. Cette dernière représente un point d'un partenaire où une information est consommée et / ou produite permettant ainsi la synchronisation et l'échange de données avec les autres partenaires du processus [Chebbi *et al.,* 2006]. Les activités coopératives sont de deux types : les activités productrices et les activités consommatrices :

- Une activité coopérative productrice est une activité qui produit des informations pour une activité externe appartenant à un autre partenaire.
- Une activité coopérative consommatrice est une activité qui consomme des informations provenant d'une activité externe appartenant à un autre partenaire.

Définition 1.3 – Définition de processus ou schéma de processus (Process definition or Process schema)

> *Une définition de processus décrit (i) les activités qui composent le processus, (ii) la structure d'enchaînement des activités, i.e., la dépendance d'exécution entre les tâches, (iii) les critères de lancement et d'achèvement du processus et (iv) les informations relatives aux activités (participants, ressources, applications appelées, données spécifiques, etc.) [Reichert et weber, 2012].*

La définition de processus est explicitée au travers des langages de modélisation des processus. Certains de ces langages permettent une représentation graphique de la définition de processus tandis que d'autres offrent une spécification textuelle de cette définition. Dans la section IV de ce chapitre, nous détaillons la notation BPMN qui est

connue comme un standard de modélisation graphique des définitions des processus. La Figure I.1(a) illustre la définition d'un *processus de réalisation d'une intervention chirurgicale* modélisée conformément à la notation BPMN. Cette définition est interprétée comme suit : à l'arrivée d'un patient accompagné par une lettre d'admission, une secrétaire procède à son admission. Une fois admis, une infirmière prépare le patient. Par la suite, un médecin chirurgien effectue l'intervention chirurgicale. Après l'intervention, le suivi post-opératoire se fait dans le service de réanimation, dans le cas où l'état de santé du patient est grave, sinon il s'effectue dans la chambre d'hospitalisation du patient. Finalement, le médecin rédige un rapport détaillant l'état de santé du patient et le résultat de l'intervention.

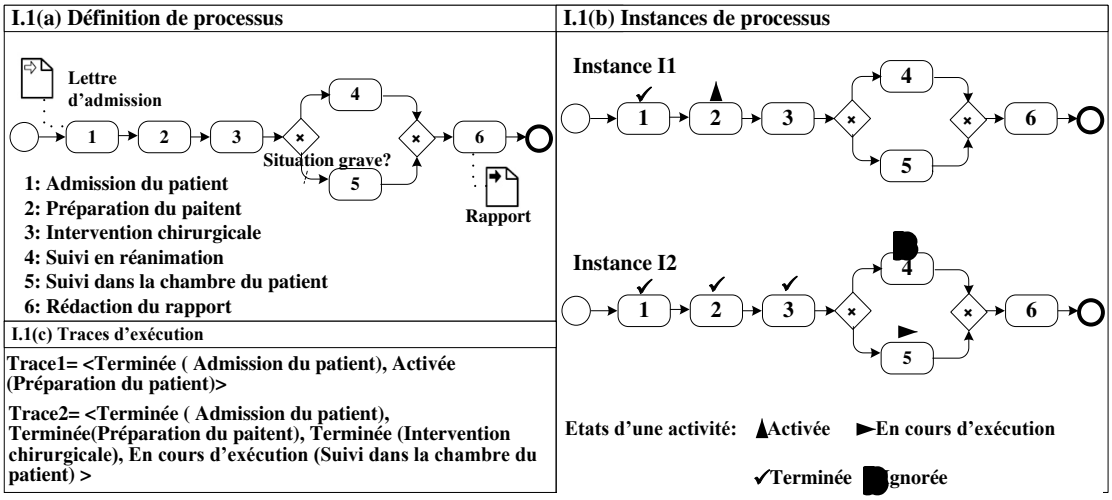


Figure I.1. Exemple d'une définition et des instances d'un processus

Définition 1.4 – Instance de processus ou Cas de processus (Process instance)

> *« Une instance de processus correspond à un cas concret d'une exécution de la définition de ce processus. Une définition peut être instanciée plusieurs fois et différentes instances peuvent s'exécuter concurremment » [Reichert et Weber, 2012].*

Une instance d'un processus est la combinaison de la définition du processus et d'une trace d'exécution comportant des événements qui reflètent l'état d'exécution des activités. Par exemple, la Figure I.1(b) illustre deux instances I1 et I2 de la définition du processus présentée dans la Figure I.1(a). Ces instances correspondent à deux exécutions différentes de la même définition. La Figure I.1(c) montre les traces d'exécution relative aux instances I1 et I2. L'ensemble des traces d'exécution d'un processus formant toutes ses instances est sauvegardé dans un fichier "log[1]".

[1] Un fichier log connu sous le vocable anglais « execution logs » comporte les informations se rapportant à l'exécution des instances (ou cas) d'un processus [Dumas et *al.,* 2013].

Une instance de processus exécute des instances d'activités. Une instance d'activité possède un ensemble d'états par lesquels elle peut transiter, depuis son activation jusqu'à sa terminaison. Parmi ces états nous citons :

- Activée : une instance d'activité a été créée.
- En cours : l'activité est en cours d'exécution.
- Terminée : l'exécution de l'activité est terminée.
- Ignorée : L'exécution de l'activité est ignorée.

Définition 1.5 – Gestion des processus métier (Business Process Managment BPM)

> *La gestion des processus métier inclut les concepts, les méthodes et les techniques qui permettent la modélisation, l'administration, la configuration, la mise en œuvre et l'analyse des processus métier [Weske, 2012].*

La gestion des processus se base essentiellement sur une représentation explicite du processus et de ses activités et sur des contraintes d'exécution entre ces activités. Une fois modélisé, un processus peut subir des analyses et des améliorations avant d'être mis en œuvre. Nous reviendrons sur les aspects de gestion de processus dans la section I.2.

Définition 1.6 – Système de gestion de processus (Business Process Management System BPMS)

> *Un **S**ystème de **G**estion de **P**rocessus est un système qui définit, implémente et gère l'exécution d'un ou plusieurs processus à l'aide d'un environnement logiciel fonctionnant avec un ou plusieurs moteurs d'exécution. Ce système est capable d'interpréter la définition d'un processus, de gérer la coordination des participants et d'appeler des applications externes [Dumas et al., 2013].*

Avant l'émergence des BPMS, plusieurs outils permettant l'automatisation des processus ont été proposés. Ces outils, connus sous le nom de Workflow Management Systems (WfMS), regroupent des composants logiciels qui interprètent les définitions de processus, puis les exécutent en utilisant un moteur de workflow. La Workflow Management Coalition (WfMC) a proposé une architecture de référence d'un WfMS [WfMC, 1994]. Cette architecture illustre les différents composants d'un WfMS ainsi que les interfaces assurant la liaison entre ces composants. Les BPMS, considérés comme la nouvelle génération des WfMS, couvrent mieux le cycle de vie des processus tels qu'il est défini dans le domaine du BPM (*cf.* section I.2) que celui couvert par l'architecture de la WfMC. Plus précisément, les BPMS se distinguent par les fonctionnalités suivantes :

- La découverte des processus : l'adjonction d'une base contenant les définitions des processus et d'une relation entre cette base et le composant de modélisation rend la modélisation des processus plus puissante car il est possible de réutiliser les

définitions existantes. De plus, cette base de définition de processus favorise la tâche d'analyse et de découverte basée sur la description des processus.

- L'analyse des traces d'exécution afin d'assurer la réingénierie des processus : l'adjonction d'une base contenant les traces d'exécution et d'une relation entre cette base et le composant d'administration et de pilotage permet de faire des analyses sur les exécutions antérieures afin de déduire de nouvelles connaissances assurant la réingénierie des processus.

L'architecture d'un BPMS est donnée dans la Figure I.2. Cette architecture comporte les composants suivants : un moteur d'exécution, un outil de modélisation, des outils d'administration et de pilotage et un gestionnaire de la liste des tâches (Worklist). Nous détaillons, dans ce qui suit, chacun de ces composants.

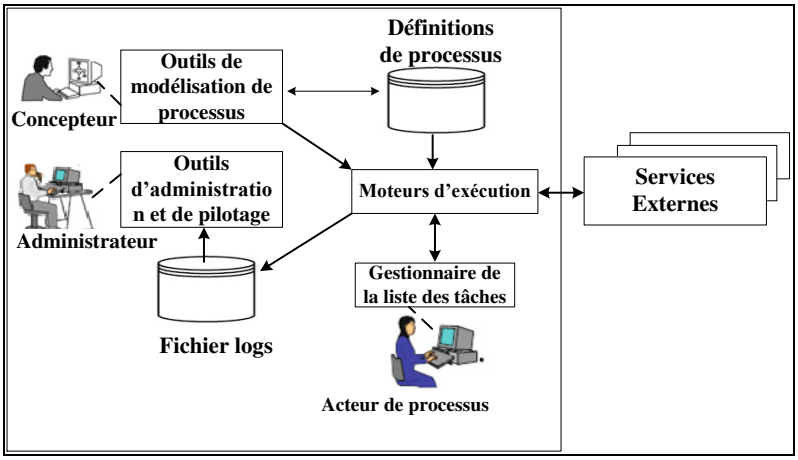


Figure I.2. Architecture d'un BPMS [Dumas *et al.*, 2013]

1) Le moteur d'exécution : c'est le composant central d'un BPMS. Il correspond à l'environnement capable d'exécuter une ou plusieurs définition(s) de(s) processus. Les principales fonctionnalités assurées par un moteur d'exécution sont les suivantes : (i) la création des instances exécutables, (ii) la distribution du travail sur les différents ressources du processus, (iii) la recherche automatique des informations nécessaires à l'exécution du processus et (iv) le stockage des instances des processus (les traces d'exécution) dans un fichier log.
2) Les outils de modélisation et de définition de processus : Ces outils ont pour objectif d'aider le concepteur à saisir la définition des processus en des schémas de processus qu'il pourra ensuite valider. Ils permettent, aussi, la sauvegarde, le partage et la recherche des processus modélisés à partir d'un répertoire de définitions de processus (process model repository). Ces outils permettent, aussi, le déploiement de la définition du processus pour être exécutée par le moteur d'exécution.
3) Le gestionnaire de la liste des tâches (Worklist) : Un worklist est le composant d'un BPMS qui comporte le travail à effectuer par les ressources du processus. Il est similaire à une boite de réception d'un compte de messagerie électronique. Au

travers de cette boite, les acteurs de processus peuvent connaître le travail qu'ils sont censés effectuer.

4) Les services externes : Un BPMS doit s'intégrer dans le système d'information des organisations dans lesquelles il est déployé. Ainsi, il doit pouvoir communiquer avec d'autres applications, nécessaires à l'accomplissement des activités des processus, tels que l'accès à un SGBD pour recueillir de l'information et l'appel au service de messagerie et à divers autres outils bureautiques. Ces applications, considérées comme des services externes à un BPMS, doivent inclure des services d'interfaçage assurant l'interaction avec le moteur d'exécution.
5) Les outils d'administration et de pilotage : ces outils, sous la responsabilité d'un administrateur, doivent supporter les fonctions suivantes : (i) gestion des concepteurs chargés de produire des définitions de processus, (ii) gestion des participants associés à l'exécution d'une instance de processus (autorisations, mots de passe, révocation), (iii) supervision et contrôle des cas en cours d'exécution et (iv) audit et analyse du fonctionnement du système. A travers cette dernière fonction, les outils d'administration offrent des indicateurs de performance utilisés pour analyser les fichiers log, ce qui permet d'assurer la réingénierie des processus.

I.2. Cycle de vie d'un processus

Le cycle de vie d'un processus (*cf.* Figure I.3) est composé essentiellement de six étapes :

- L'identification : Cette étape pose, dans un premier temps, le problème à résoudre et identifie, dans un second temps, les besoins que le processus doit satisfaire.
- La modélisation / la découverte : Les besoins spécifiés dans l'étape d'identification peuvent être concrétisées par la modélisation d'un schéma de processus reflétant ces besoins. L'étape de modélisation peut être remplacée par une étape de découverte dans le cas où il existe des processus modélisés préalablement répondant aux besoins identifiés précédemment.
- L'analyse : Cette étape permet l'identification et la documentation des problèmes relatifs au processus modélisé. Le résultat de cette phase est un ensemble structuré de problèmes classifiés par ordre de priorité en fonction de l'impact de chaque problème ou bien de l'effort nécessaire pour le résoudre.
- La refonte : L'objectif de cette étape est la prise en compte des problèmes recensés dans la phase d'analyse. Cette phase consiste à faire les ajustements nécessaires de la définition de processus afin d'obtenir une définition plus adéquate. Ainsi, le résultat de cette phase est une définition de processus prête à être instanciée.
- L'implémentation : Cette étape est consacrée à l'instanciation d'un schéma de processus (*i.e.,* création des cas), puis, au déploiement de cette instance dans un BPMS pour son exécution. L'exécution est assurée par le moteur d'exécution et peut faire appel à un acteur (*i.e.,* une ressource humaine) et/ou à des applications externes.

▪ La surveillance et le contrôle : Une fois l'instance de processus est en cours d'exécution, des données pertinentes issues de cette exécution sont collectées et analysées. Ces données permettent de vérifier si le processus obéit à ses objectifs et ses attendus. Plus précisément, cette étape consiste à détecter les erreurs récurrentes et les écarts par rapport au traitement prévu afin de prendre des mesures correctives. Ainsi, de nouveaux problèmes peuvent être recensés nécessitant la répétition du cycle.

Les étapes du cycle de vie d'un processus peuvent être classées en deux phases. La phase de conception (connue sous le vocabulaire anglais « Build Time ») regroupe les étapes identification, modélisation, analyse et refonte. La phase d'exécution (on la retrouve également sous le vocabulaire anglais « Run Time ») comporte les étapes implémentation et Surveillance et contrôle.

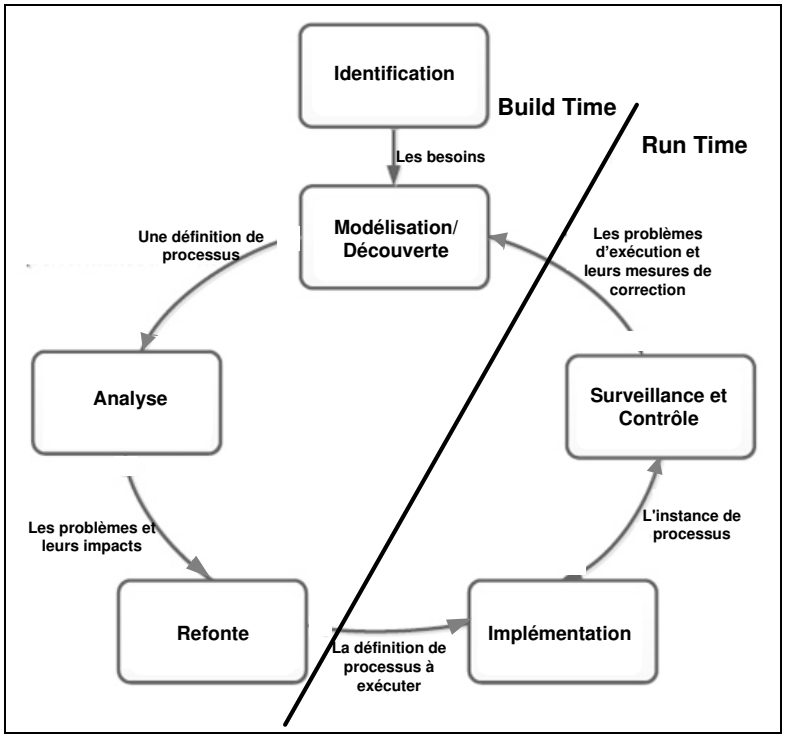


Figure I.3. Cycle de vie d'un modèle de processus [Dumas et al., 2013]

II. Les perspectives de modélisation des processus

La modélisation d'un processus repose essentiellement sur trois perspectives en interaction [Aalst *et al.*, 2003(a)] [Aalst, 2004], décrivant chacune un point de vue différent de ce processus. Il s'agit de la *perspective comportementale*, la *perspective organisationnelle*, et la *perspective informationnelle*. La *perspective comportementale* décrit « le *quoi* faire » d'un processus, *i.e.,* les activités et les opérations qui le composent, et « *comment* » ces activités sont coordonnées entre elles. La *perspective organisationnelle* structure les ressources en rôles et en unités organisationnelles et leur attribue des autorisations pour exécuter les activités du processus. Elle permet de décrire le « *qui* », c'est-à-dire les acteurs dans le processus. Finalement, la *perspective informationnelle* décrit la structure des documents et des données consommées et / ou

produites par chacune des activités du processus. Généralement les événements de déclenchement et d'achèvement d'une activité sont décrits par l'existence d'informations et / ou des valeurs prises par ces informations.

Ces perspectives ont été, par la suite, enrichies par une quatrième, la *perspective intentionnelle* [Saidani et Nurcan, 2009] qui permet de décrire « le *pourquoi* » d'un processus, ceci en mettant l'accent sur les objectifs attendus par ce processus.

Un modèle de processus est considéré comme complet s'il est capable de prendre en compte toutes ces perspectives afin de pouvoir décrire toutes les dimensions d'un processus qui sont « *qui fait quoi, comment, quand et pourquoi* ? » [Aalst *et al.*, 2003(a)], [Chaâbane, 2012]. Dans le reste de cette section, nous reprenons en détail chacune de ces perspectives.

II.1. La perspective organisationnelle

La perspective organisationnelle s'intéresse à l'organisation des ressources. Une ressource peut être un humain, une machine ou un logiciel, voire un logiciel agissant pour le compte d'un humain et capable de réaliser des activités d'un processus. La perspective organisationnelle a un double objectif. Le premier est de structurer les ressources en classes partageant les mêmes caractéristiques fonctionnelles. Une classe correspond à un rôle lorsqu'elle comporte des ressources ayant les mêmes caractéristiques, ou bien à une unité organisationnelle lorsque les ressources sont des participants appartenant à une même structure organisationnelle [Aalst et Kumar, 2000]. Le deuxième objectif de cette perspective est d'attribuer des autorisations aux ressources pour exécuter un certain nombre d'activités dans le processus.

II.2. La perspective informationnelle

La perspective informationnelle décrit la structure des formulaires, des documents et des données utilisés et / ou produits par le processus. En particulier, cette perspective détermine l'existence et les valeurs des informations déterminent si une activité satisfait les conditions nécessaires à son exécution ou non (pré-condition d'une activité). Les informations nécessaires à la réalisation des activités correspondent à leur condition de déclenchement, c'est-à-dire la condition qui doit être vérifiée pour exécuter une activité. Par conséquent, la présence de ces informations et la disponibilité des ressources peuvent être considérées comme des événements définissant *quand* une activité peut être réalisée.

II.3. La perspective comportementale

La perspective comportementale est considérée comme étant la perspective pivot dans la définition d'un processus. Sa définition englobe deux aspects, à savoir l'aspect fonctionnel et l'aspect opérationnel, et elle fait référence aux perspectives informationnelle et organisationnelle. L'aspect fonctionnel décrit les activités à réaliser lors de l'exécution d'un processus. Il définit aussi comment les activités complexes sont décomposées en activités atomiques. Quant à l'aspect opérationnel, il décrit les

opérations élémentaires à exécuter dans chacune des activités atomiques d'un processus. Ces opérations manipulent des ressources informationnelles et invoquent des ressources organisationnelles dans leur exécution.

En outre, la perspective comportementale décrit les conditions de déclenchement et d'achèvement des activités composant un processus, leur coordination et les informations et les ressources impliquées dans chaque activité. La coordination entre les activités d'un processus est décrite à l'aide de patrons de coordination [Aalst *et al.*, 2003(b)].

Un patron de coordination permet de représenter l'ordre dans lequel des activités sont exécutées dans un processus. Les travaux de la **W**orflow **P**atterns **I**nitiative (WPI) [Aalst *et al.*, 2003(b)] distinguent une quarantaine de patrons de coordination classés en deux catégories :

- Les patrons de coordination de base : séquence, branchement multiple (AND-split), synchronisation (AND-join), choix exclusif (Xor-Split), jonction simple (Xor-join) et itération.
- Les patrons de coordination avancés : choix multiple, jonction synchronisée, multi-merge et discriminateur. Par exemple, le choix multiple permet de décider, à un point précis d'un processus, de poursuivre plusieurs voies plutôt qu'une seule.

II.4. La perspective intentionnelle

La perspective intentionnelle vient enrichir la description multi-dimensionnelle des processus par une nouvelle dimension, traitant du *pourquoi* du processus [Nurcan et Edme, 2005]. Dans notre travail nous considérons, en plus des intentions des processus, leur contexte (situation) d'utilisation. De ce fait, nous appelons cette perspective, dans le reste de ce document, la perspective contextuelle.

La notion de contexte se retrouve dans plusieurs domaines de l'informatique, tels que ceux des langages naturels [Chan et Franklin, 2003], de l'intelligence artificielle [Theodorakis et Spyratos, 2002] et de l'ingénierie des systèmes web [Keidl et Kemper, 2004]. Dans la littérature, il existe plusieurs techniques pour exprimer un contexte. Parmi ces techniques, nous citons la technique paire/triplet, la technique orientée modèle et la technique orientée ontologies. Nous reviendrons en détail sur ces techniques dans le chapitre II.

III. Les processus inter-organisationnels : scénarii de coopération et formes d'interopérabilité

Un Processus Inter-Organisationnel (PIO) envisage la coopération de plusieurs processus issus de plusieurs organisations réparties, autonomes et hétérogènes. Il se distingue d'un processus intra-organisationnel par trois aspects essentiels relatifs à l'implication de plusieurs organisations:

- La répartition des organisations : les processus participant à un PIO appartiennent à des organisations distantes et géographiquement éloignées.

- L'autonomie des organisations : chaque organisation prend individuellement les décisions concernant les conditions de coopération, c'est-à-dire quand, comment et avec qui elle doit coopérer.

- L'hétérogénéité des organisations à faire coopérer : cela concerne les différences en termes de modèles et de systèmes.

En dépit de ces aspects, une coopération des processus d'un PIO doit être bien établie afin de garantir le bon déroulement du PIO. De plus, ces processus peuvent être exécutés sur plusieurs BPMS, d'où la nécessité d'une certaine interopérabilité entre ces outils. Dans ce qui suit nous détaillons les scénarii de coopération ainsi que les formes d'interopérabilité des processus d'un PIO.

III.1. Les scénarii de coopération dans les processus inter-organisationnels

Dans le cadre des processus inter-organisationnels, la coopération des partenaires réparties et hétérogènes peut être étudiée selon deux scénarios : une coopération lâche, dite aussi PIO lâche, et une coopération serrée, dite aussi PIO serré [Divitini *et al.,* 2001] [Bouzguenda, 2006].

1) le PIO lâche correspond à une coopération occasionnelle, sans contraintes structurelles, dans laquelle les partenaires impliqués et leur nombre ne sont pas prédéfinis. La coopération dans le cas d'un PIO lâche intervient dans les étapes suivantes : la recherche de partenaires, la négociation entre les partenaires, la spécification et la signature des contrats entre partenaires et l'exécution de services. Un exemple de PIO lâche est le commerce électronique [Bouaziz, 2010].

2) le PIO serré correspond à une coopération structurelle entre des organisations, c'est-à-dire basée sur une infrastructure bien établie entre des partenaires bien identifiés. De ce fait, ces derniers constituent une fédération et sont plus ou moins dépendants les uns des autres. Ce type de coopération est celui qui convient lorsque les partenaires sont engagés dans des coopérations à long terme et / ou suffisamment intenses pour nécessiter des interactions formalisées. Des exemples de PIO serré peuvent être concrétisés dans le concept de l'entreprise virtuelle (EV[2]). Par exemple, dans une EV, les entreprises intègrent des collaborateurs (fournisseurs, clients) dans leur système de façon permanente, selon une hiérarchie claire. Le système de gestion de processus doit traverser les limites des entreprises

[2]Le concept de l'entreprise virtuelle (EV) est basé sur un échange intensif d'informations entre les différentes entreprises participantes. Dans une entreprise virtuelle, un fabricant ne crée plus un produit complet de façon autonome : il s'intègre dans un réseau de fournisseurs, clients, ingénieurs, etc.

afin d'assurer une intégration étroite de tous les participants, notamment pour la gestion de la chaîne logistique [Kanzow, 2004].

III.2. Les formes d'interopérabilité des processus inter-organisationnels

La WfMC [WfMC, 1999] a défini l'interopérabilité des processus formant un processus Inter-Organisationnel par la capacité des différents systèmes impliqués dans l'exécution du PIO à communiquer entre eux afin de supporter l'exécution coordonnée des instances des processus partenaires. D'autre part, Aalst [1999] propose cinq architectures conceptuelles favorisant l'interopérabilité des PIO, appelées aussi formes d'interopérabilité des PIO. Dans ce qui suit, nous citons les différentes formes d'interopérabilité des PIO proposées dans [Aalst, 1999].

III.2.1. Le partage de capacité

Dans cette forme d'interopérabilité, un ensemble de partenaires se mettent d'accord, dès la phase de conception, sur un processus global mettant en service leurs capacités et savoir-faire mutuels. Le contrôle, quant à lui, est centralisé et le routage des activités est assuré et géré par un seul BPMS, alors que l'exécution des activités est distribuée sur les différents participants. En effet, à chaque fois que le BPMS gérant le processus global rencontre une activité, il fait appel au partenaire adéquat ainsi qu'aux ressources appropriées permettant l'exécution de cette activité. Le partage de capacité est simple et utilise une infrastructure commune ainsi qu'une gestion centralisée d'un processus unifié [Aalst, 1999]. La Figure I.4 illustre un exemple de la forme d'interopérabilité « Partage de capacité ».

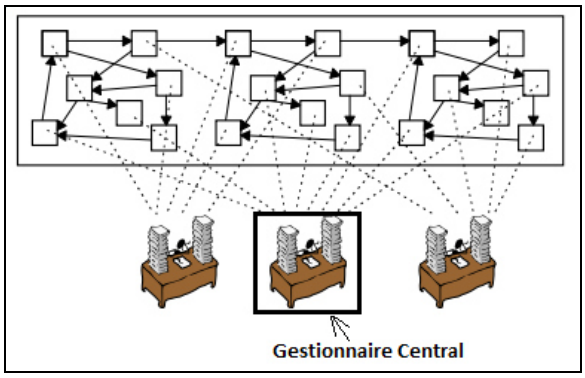


Figure I.4 Exemple de forme « Partage de capacité »

III.2.2. L'exécution chainée

L'exécution chaînée consiste à modéliser un processus global en plusieurs sous processus disjoints exécutés séquentiellement par les différents partenaires. En effet, chaque partenaire se charge d'une partie de processus. Une fois sa partie exécutée, le partenaire transfère le flot au partenaire suivant. Cette forme d'interopérabilité ne requiert pas d'exécution parallèle et se base sur un contrôle distribué de processus. L'exécution chaînée est illustrée dans l'exemple de la Figure I.5 [Aalst, 1999]

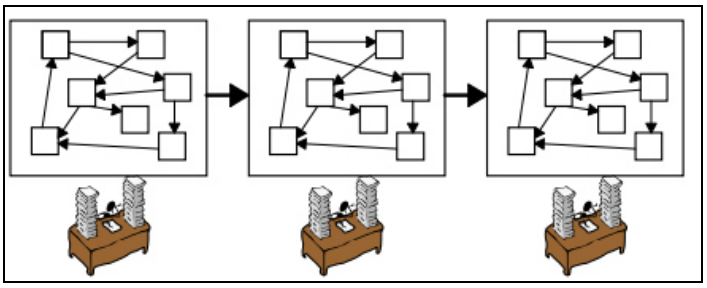


Figure I.5 Exemple de forme « Exécution chainée »

III.2.3. La sous-traitance

La sous-traitance est une forme d'interopérabilité permettant à un partenaire principal de déléguer l'exécution et la coordination d'une partie de son processus à d'autres partenaires. Le contrôle des processus étant hiérarchique, le partenaire principal voit les processus sous-traités comme des processus atomiques, alors que ceux-ci peuvent avoir des structures complexes au niveau des partenaires les exécutant. La Figure I.6 est un exemple de sous-traitance [Aalst, 1999].

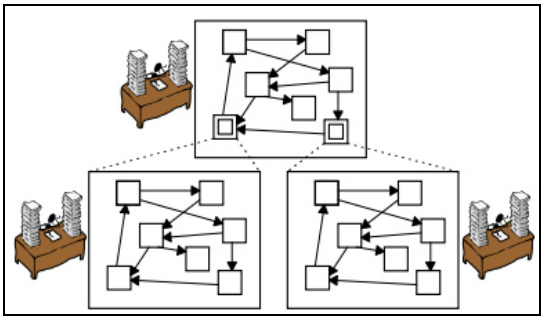


Figure I.6 Exemple de forme « Sous-traitance »

III.2.4. Le transfert de cas

Dans cette forme d'interopérabilité, les différents partenaires impliqués dans l'interopérabilité partagent une description commune de la définition d'un processus, *i.e.,* la spécification de ce processus est répliquée au niveau de chaque participant. Chaque partenaire peut exécuter des activités spécifiques du processus. En outre, à un instant donné, l'instance (cas) de processus est en possession d'exactement un seul partenaire. Cette instance passe d'un partenaire à l'autre pour exécution [Aalst, 1999]. Le transfert de cas est illustré en Figure I.7.

Par ailleurs, un « Transfert de cas étendu » est une forme dérivée de la forme « Transfert de cas » dans laquelle les partenaires impliqués dans le processus inter-organisationnel peuvent personnaliser leur version du processus et entrer des variations locales (adjonction d'activités, par exemple) selon leurs besoins avec la contrainte que le comportement des processus personnalisés soit identique à celui du processus global [Aalst, 1999].

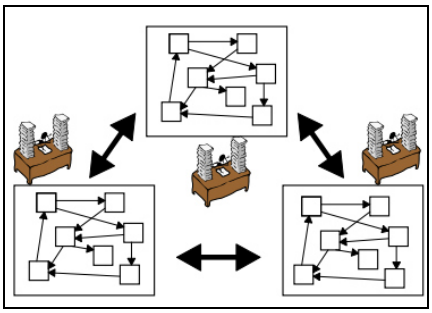


Figure I.7 Exemple de forme « Transfert de cas »

III.2.5. Le couplage faible

Dans cette forme d'interopérabilité, le processus est divisé en sous-processus disjoints. Typiquement, il existe *n* partenaires impliqués dans un seul processus global chacun disposant de son propre processus local privé. Les partenaires ne connaissent pas toujours le schéma et l'état des instances de processus des autres partenaires. La communication entre les instances se fait par un protocole commun [Aalst, 1999]. La Figure I.8 donne un exemple de couplage faible.

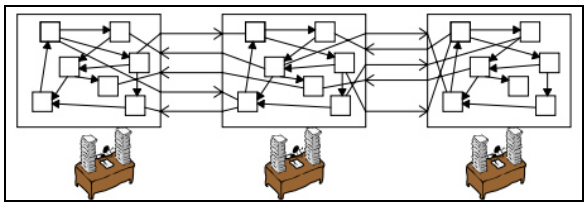


Figure I.8 Exemple de forme « Couplage faible »

III.2.6. Synthèse des formes d'interopérabilité

Le tableau I-1 présente une synthèse des formes d'interopérabilité en se basant sur les critères suivants :

- Le type de contrôle (Centralisé / Distribué) : Ce critère indique le type de contrôle utilisé pour synchroniser les activités du processus inter-organisationnel. Le contrôle est considéré centralisé dans le cas où un seul partenaire pilote l'exécution. Il est considéré distribué dans le cas où le pilotage est réparti sur les différents partenaires.

- Technique d'interopérabilité : Ce critère indique la technique utilisée pour assurer l'interopérabilité. Nous distinguons deux techniques différentes : la première permet la duplication du schéma de processus chez les différents partenaires alors que la deuxième technique consiste à répartir le schéma de processus sur les différents partenaires. En effet, dans la première technique, tous les partenaires ont le même schéma de processus qu'ils exécutent de façon indépendante. Dans la deuxième technique, chaque partenaire possède une portion du schéma de processus. D'où la nécessité de protocoles assurant la communication et l'échange de messages entre les différents partenaires.

▪ Type d'exécution (Séquentielle / Parallèle) : Ce critère spécifie si l'exécution des activités appartenant aux différents partenaires peut se faire en parallèle ou bien uniquement en séquence.

Tableau I-1. Tableau récapitulatif des capacités des formes d'interopérabilité

Critères / Formes d'interopérabilité	Type de contrôle	Types d'exécution	Technique d'interopérabilité
Partage de capacité	Centralisé	Séquentielle ou parallèle	Répartition du schéma de processus
Exécution chainée	Distribué	Séquentielle	Répartition du schéma de processus
Sous-Traitance	Centralisé	Séquentielle ou parallèle	Répartition du schéma de processus
Transfert de cas	Distribué	Séquentielle ou parallèle	Duplication du schéma de processus
Couplage faible	Distribué	Séquentielle/ ou parallèle	Répartition du schéma de processus

D'après ce tableau, nous remarquons que les formes d'interopérabilité « Partage de capacité » et « Sous-traitance » ont un contrôle centralisé, alors que les formes « Exécution chainée », « Transfert de cas » et « Couplage faible » adoptent un contrôle distribué. En outre, seule la forme « Exécution chainée » exige une exécution séquentielle des activités, tandis que toutes les autres formes permettent, en plus de l'exécution séquentielle, l'exécution parallèle. Finalement, la forme d'interopérabilité « Transfert de cas » utilise la technique de duplication du schéma de processus dans les différents partenaires, alors que toutes les autres formes adoptent la technique de répartition du schéma de processus.

IV. Le langage de modélisation des processus

Dans la description de l'architecture d'un BPMS que nous avons présentée précédemment (*cf.* Figure I.2), le composant « Outils de modélisation de processus » d'un BPMS offre au concepteur la possibilité de définir des schémas de processus. Il s'agit donc d'une étape de modélisation des processus durant laquelle le concepteur peut utiliser soit une notation graphique, soit un langage textuel :

▪ Une notation (langage) graphique pour définir un processus, a pour objectif une visualisation graphique de la définition d'un processus pour éventuellement la valider avant qu'elle ne soit exécutée. En effet, au travers d'une représentation graphique, le concepteur d'une application peut visualiser le schéma du processus modélisé et simuler son exécution (à l'aide d'un jeu de jetons). Parmi ces langages, nous citons : BPMN, les réseaux de Petri et le diagramme d'activité d'UML.

▪ Un langage de spécification textuel (non graphique), vise à donner une spécification du processus pouvant être interprétée et exécutée par un moteur d'exécution (*e.g.,* les langages XPDL et BPEL).

Dans cette section nous nous intéressons à la présentation de la notation BPMN qui est reconnue comme le standard le plus répandu pour la modélisation graphique des processus.

IV.1. BPMN : Besoin, naissance et évolution

Business Process Model and Notation (BPMN) est une notation standard utilisée pour représenter graphiquement un processus métier en séparant les informations métiers des informations techniques. Il a été développé par la **B**usiness **P**rocess **M**anagement **I**nitiative (BPMI[3]), et est actuellement maintenu par l'**O**bject **M**angement **G**roup (OMG[4]). Cette notation permet la modélisation des processus en utilisant **des éléments simples et compréhensibles** par tous les utilisateurs professionnels : les analystes métier qui créent la version initiale du processus, les développeurs techniques chargés de l'application de la technologie qui va exécuter ces processus, et finalement les personnes qui permettront de gérer et de contrôler ces processus. L'idée des fondateurs était de créer un **mécanisme simple pour modéliser la complexité inhérente à un processus**. BPMN essaie de réunir ces deux exigences paradoxales. En effet, le développement de BPMN est une étape importante dans la réduction des différences de notation entre les divers outils du marché. C'est que de nombreuses notations (*e.g.,* diagramme d'activité UML, RosettaNet et Event-Process Chain), offrant des représentations graphiques différentes pour les processus, ont vu le jour. Une notation commune permettra sans doute une interopérabilité sémantique entre les différentes applications et une facilité dans la compréhension et l'analyse des processus par les différents utilisateurs (analystes, concepteurs, etc.). Depuis la standardisation de sa première version BPMN 1.0, BPMN a connu plusieurs améliorations pour étendre ses fonctionnalités. BPMN 1.1 [OMG, 2008] vient pour rendre la notation donnée dans la première version plus explicite. La version BPMN 1.2 [OMG, 2009] apporte des améliorations relatives à l'exécution des modèles BPMN. Ainsi, des solutions de conversion vers d'autres langages exécutables ont été développées afin de pouvoir utiliser une spécification BPMN pour les services web en définissant des règles de passage d'un modèle BPMN à une spécification exécutable pour les services web sous le langage **Business Process Execution Language for Web Service (BPEL4WS)**. La version BPMN 2.0 [OMG, 2011] vient remédier aux problèmes d'exécution des modèle BPMN. Dans cette version BPMN évolue vers un schéma d'échange standard basé sur XML permettant l'échange de modèles exécutables. BPMN 2.0 a vocation de

[3]BPMI est un consortium de standardisation dans le domaine des processus d'entreprise. En juin 2005, BPMI est fusionné avec la branche BPM (Business Process Modeling) de l'OMG pour donner lieu à BMI/DTF (Business Modeling and Integration/Domain Task Force).

[4]OMG est une association américaine à but non lucratif, créée en 1989, dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. Il est notamment à la base de plusieurs standards tels qu'UML (Unified Modelling Language) et MOF (Meta-Object Facility).

devenir un langage de modélisation exécutable en remplacement de BPEL. Plus précisément, BPMN 2.0 ajoute par rapport à la version 1.2 les points suivants :

- Un méta-modèle normalisé et un format de sérialisation pour BPMN, qui permet aux concepteurs d'échanger des modèles BPMN entre les outils de différents fournisseurs.
- Une sémantique d'exécution normalisée pour BPMN, qui va permettre aux fournisseurs logiciels d'implémenter des moteurs d'exécution interopérables pour les processus métier.
- Un processus de transformation détaillé de BPMN pour WS-BPEL, montrant l'alignement de BPMN avec les outils et les normes existants.
- Une définition d'un nouveau type de diagramme, nommé *chorégraphie,* permettant de se focaliser sur les interactions entre les partenaires d'un processus inter-organisationnel.
- Certains éléments de modélisation supplémentaires pour des processus, telles que les tâches manuelles et les tâches humaines.

De plus, BPMN 2.0 est une notation ouverte et extensible par les concepteurs et les outils de modélisation. En effet, BPMN 2.0 offre un mécanisme d'extension permettant l'adjonction de nouveaux éléments (ayant une représentation graphique particulière) et / ou attributs (informations caractérisant un élément). Cependant, ces extensions ne doivent pas changer les représentations (les formes) définies en standard. Parmi les mécanismes d'extension offerts par BPMN, nous pouvons citer :

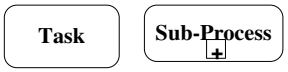
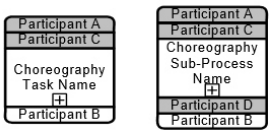
- l'utilisation de marqueurs (indicateurs) pour décrire un type particulier ou bien une information spécifique d'un élément existant dans le standard.
- l'adjonction de nouvelles formes graphiques pour satisfaire un besoin spécifique. Les éléments ajoutés sont considérés comme des artefacts.
- la coloration des éléments graphiques ou bien le changement de leurs styles de trait pour introduire une sémantique particulière qu'on associe à un élément de la notation standard. Par exemple, elle permet de distinguer les informations produites des informations consommées en utilisant deux couleurs différentes.

IV.2. Les éléments de base de BPMN

IV.2.1. Les éléments de Flux

Le tableau I-2 illustre les éléments de flux pouvant exister dans un diagramme BPMN. Ces éléments représentent les principaux concepts graphiques permettant la définition du comportement d'un processus ou d'une chorégraphie.

Tableau I-2. Les éléments de flux de BPMN 2.0

Elément	Description	Notation
Activité (Activity)	Une activité est un travail à accomplir dans le processus. Elle peut être atomique (Task) ou composite (Sub-Proces), *i.e.,* une activité décomposable en plusieurs autres activités.	
Activité de Chorégraphie (Choreography activity)	Une activité de chorégraphie représente une interaction entre deux ou plusieurs participants. Elle peut être atomique (une tâche de chorégraphie impliquant un seul message) ou composite (un sous processus de chorégraphie impliquant plusieurs messages)	
Evénement (Event)	Un événement peut être utilisé pour débuter (Start Event) ou finir (End Event) un processus et gérer des actions spécifiques au cours de celui-ci (Intermediate Event)	
Branchements (Gateway)	Un branchement est utilisé pour dissocier ou réunir des flux.	

Chaque élément de flux est enrichi par un certain nombre de symboles qui précisent leur sens. La Figure I.9 montre des détails additionnels qui permettent la distinction de certains types de tâches et d'événements. Plus précisément, nous illustrons une tâche automatique (*Service Task*), une tâche d'envoi de message (*Send Task*), une tâche de réception de message (*Receive Task*), une tâche manuelle (*Manual Task*), une tâche semi-automatique (*User Task*), des événements d'envoi et de réception de message (Message Events), des événements temporels (Timer Events) et des événements conditionnels (Conditional Events).

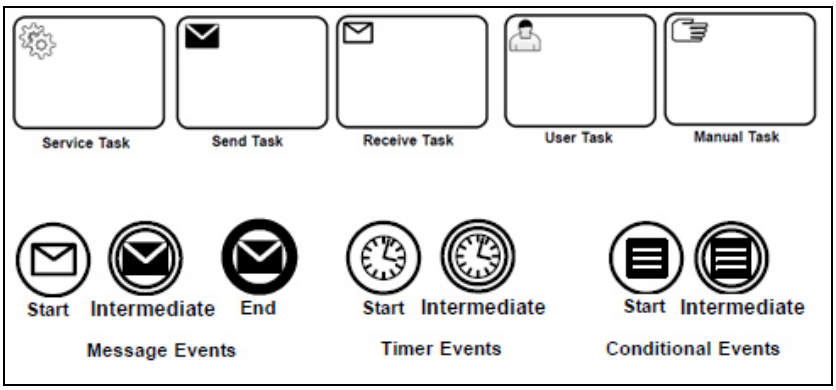


Figure I.9 Exemple de tâches et d'événement de BPMN 2.0

Concernant les branchements, le symbole à l'intérieur du losange sert à identifier leur nature. La Figure I.10 illustre les principaux branchements offerts par BPMN 2.0. Ces branchements sont les suivants :

- Le branchement exclusif : il peut être utilisé lors d'une division pour faire passer le flux de séquence exclusivement à un embranchement, ou bien lors d'une convergence pour déclencher le flux de séquence sortant à l'arrivée d'un seul flux de séquence entrant.
- Le branchement parallèle : il peut être utilisé lors de la division d'un flux de séquence pour activer simultanément tous les embranchements. Il peut aussi être utilisé lors d'une convergence de flux parallèles. Dans ce cas, le branchement parallèle attend que tous les flux entrants soient complétés pour déclencher le flux sortant.
- Le branchement complexe : dans un branchement complexe, une division ou une convergence est utilisée lorsque le comportement ne peut être capturé par les autres types de branchement.
- Le branchement inclusif : lors d'une division, un branchement inclusif permet l'activation d'un ou de plusieurs embranchements. Lors d'une convergence, tous les embranchements doivent être complétés avant de finaliser cette convergence.

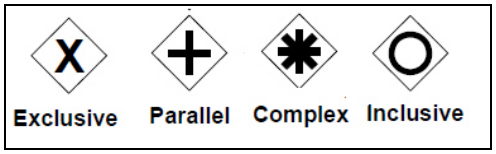


Figure I.10 Les branchements de BPMN 2.0

IV.2.2. Les éléments de connexion

Le tableau I-3 illustre les éléments de connexion offerts par BPMN 2.0 permettant la liaison des éléments. En particulier, les flux de séquence connectent les éléments de flux appartenant à un même processus, les flux de message permettent la liaison de deux activités (ou événements) appartenant à deux processus différents et les associations joignent des informations supplémentaires (modélisées par des artefacts *cf.* §IV.2.5) aux éléments de flux.

Tableau I-3. Les éléments de connexion de BPMN 2.0

Elément	Description	Notation
Flux de message (Message Flow)	Un flux de message est utilisé pour représenter un message envoyé entre deux participants.	
Flux de séquence (Sequence Flow)	Un flux de séquence est utilisé pour illustrer l'ordre des activités et des événements dans un processus.	

Association	Une association sert à lier des informations additionnelles facultatives à des éléments du processus.	

IV.2.3. Les données

Le tableau I-4 présente les éléments de BPMN 2.0 permettant la modélisation des données consommées et / ou produites par les activités d'un processus. Ces éléments n'ont aucun effet sur le flux de processus. Ils sont reliés aux activités par des associations.

Tableau I-4. Les éléments de modélisation des données de BPMN 2.0

Elément	Description	Notation
Data Object	Un data Object fournit les informations consommés et/ou produites par une activité	
Data Input	Un data Input fournit les informations nécessaires pour qu'une activité puisse être exécutée.	
Data Output	Un data Output fournit les informations qui résultent de l'exécution d'une activité.	
Data Store	Un data store fournit un mécanisme permettant aux activités de récupérer et/ou de mettre à jour les informations enregistrées.	

IV.2.4. Les conteneurs

BPMN distingue deux sortes de conteneurs : les Bassins et les Couloirs. Un bassin représente l'entité dans laquelle se déroule un processus alors que les couloirs permettent la classification des éléments de flux figurant dans un bassin en catégories. Cette classification peut se faire selon différents critères, comme par exemple la répartition par département ou par rôle. Le tableau I-5 apporte plus de détails sur les conteneurs.

Tableau I-5. Les éléments de modélisation des données de BPMN 2.0

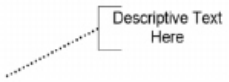
Elément	Description	Notation
Bassin (Pool)	Un Pool contient un processus unique et complet. Le flux d'activité d'un processus ne peut pas sortir d'un	

	bassin : il convient de transférer les actions d'un bassin à une autre en utilisant les flux de messages et les événements.	
Couloir (Lane)	Un couloir représente un acteur, un rôle ou une unité organisationnelle à l'intérieur d'un bassin. Le flux d'activité peut traverser les couloirs pour représenter l'enchainement des tâches entre les différents acteurs d'un processus.	

IV.2.5. Les artefacts

Les artefacts permettent d'ajouter des informations supplémentaires pertinentes facilitant la lecture des diagrammes. Ces éléments n'influencent pas le déroulement du processus et sont inutiles à l'exécution par un moteur d'exécution. Les artefacts sont toujours associés via un lien de type association à un élément de flux. Des exemples d'artefacts sont présentés dans le tableau I-6.

Tableau I-6. Les artecfat de BPMN 2.0

Elément	Description	Notation
Message	Un message est utilisé pour représenter le contenu d'une communication entre deux participants.	
Regroupement (Group)	Un regroupement permet la classification des éléments graphiques en catégories. L'utilisation des groupes dans un diagramme BPMN sert à documenter les diagrammes ou bien à déduire des analyses.	
Text Annotation	Un text annotation est un mécanisme qui permet au concepteur d'ajouter des informations complémentaires pour faciliter la compréhension des diagrammes.	

IV.3. Les diagrammes de BPMN 2.0

BPMN 2.0 offre trois types de diagrammes (modèles) pour prendre en compte les différents types de processus : les processus (privés ou publics), les collaborations et les chorégraphies [OMG, 2011]. Nous détaillons ci-dessous ces trois types de diagrammes.

IV.3.1. Les processus

Avec BPMN 2.0 un processus peut être privé ou public. Un processus privé, dit aussi processus intra-organisationnel, est un processus interne à une organisation. La Figure I.11 donne un exemple de processus privé qui correspond à un processus d'examen radiologique. Il s'exécute au sein d'une seule organisation qui est « le centre de radiologie ». Ce processus comporte quatre activités en séquence : *Réception d'une demande d'examen, Attribution de rendez-vous, Examen et Rédaction de rapport*.

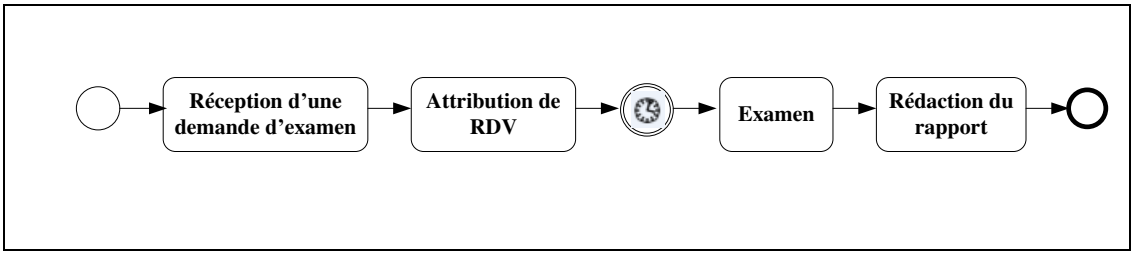


Figure I.11. Exemple de processus privé

Ce type de diagramme met l'accent sur l'enchaînement des activités qui composent le processus sans montrer les acteurs externes et leurs interactions.

Un processus public, connu sous le nom de processus abstrait dans BPMN 1.2, s'intéresse uniquement aux interactions entre un diagramme privé et un autre participant. Dans un tel processus, seules les activités qui participent aux interactions, avec les flux de messages entre ces activités (*i.e.,* les activités qui sont utilisées pour communiquer avec un participant externe), sont représentées. La Figure I.12 montre un exemple de processus public, où nous représentons les interactions entre un processus privé (le processus d'examen radiologique) et un participant (le patient). Ce type de processus se concentre plus sur les interactions du processus avec les acteurs externes. C'est pour cette raison que ce type de processus ne montre que les activités qui participent aux interactions.

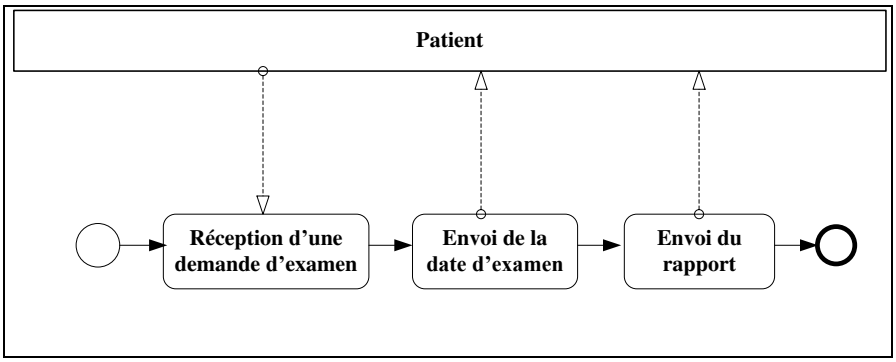


Figure I.12. Exemple de processus public

IV.3.2. Les collaborations

Une collaboration représente les interactions entre deux ou plusieurs organisations, nommées participants. Ces interactions sont définies en utilisant un ensemble d'activités qui communiquent en échangeant des messages. Un diagramme

de collaboration peut être représenté par deux ou plusieurs diagrammes publics en interaction. Un diagramme de collaboration explicite les interactions entre les activités sources de l'interaction (*i.e.,* les activités émettrices de messages) et les activités cibles de l'interaction (*i.e.,* les activités réceptrices de messages). La Figure I.13 reprend l'exemple du processus d'examen radiologique et illustre, au travers d'une collaboration, les interactions entre les deux participants *Centre de radiologie* et *Patient*, dont chacune est représentée par un processus public.

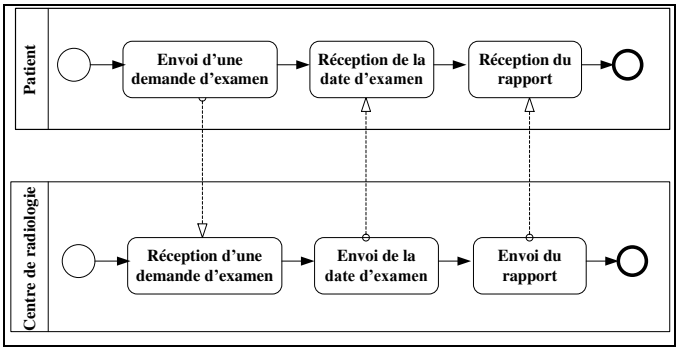


Figure I.13. Exemple d'une collaboration

En se référant au forme d'interopérabilité présentées dans la section III.2, un diagramme de collaboration permet la prise ne compte des formes suivantes :

- Le partage de capacité : une collaboration comporte un seul partenaire qui pilote et synchronise la communication entre les autres partenaires.
- La sous-traitance : dans une collaboration, un partenaire peut déléguer une partie de son processus à un autre partenaire. Dans ce cas, le partenaire délégant envoie les inputs nécessaires à l'accomplissement de la partie du processus au partenaire délégué. A la fin de l'exécution, le délégué envoie le résultat de son travail (*e.g.,* un produit, un document) au partenaire délégant.
- Le couplage faible : une collaboration est réalisée par un ensemble de partenaires qui s'échangent les informations par envoi de message.

IV.3.3. Les chorégraphies

Une chorégraphie est la modélisation d'un comportement attendu entre des participants qui interagissent les uns avec les autres et qui veulent coordonner leurs activités ou leurs tâches à l'aide de messages. Dans ce type de modélisation, la focalisation n'est pas sur l'orchestration (processus privé), c'est-à-dire sur la manière dont est accompli le travail selon le point de vue des participants, mais sur les échanges de messages entre les participants. Une chorégraphie ressemble à un processus privé dans la mesure où elle se présente sous la forme d'une connexion d'activités, d'événements et de gateways. Cependant, une chorégraphie en diffère puisque les activités en interaction représentent des échanges de messages impliquant deux ou plusieurs participants. La Figure I.14 reprend l'exemple du processus d'examen radiologique et illustre, au travers d'une chorégraphie, les interactions entre le centre de

radiologie et le patient. Cette chorégraphie comporte une séquence de trois activités de chorégraphies (Choreography activity) dont chacune définit le participant émetteur de message, le participant récepteur de message et le message transmis. Par exemple la première activité de chorégraphie définit une interaction entre un patient et un centre de radiologie dans laquelle le message *Demande d'examen* est transmis.

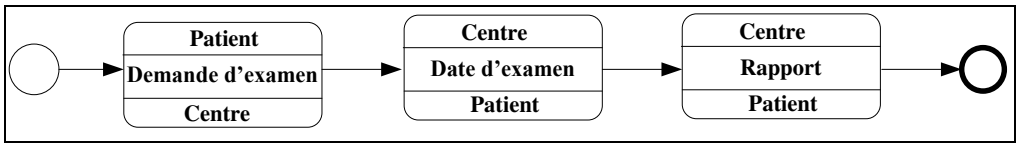


Figure I.14. Exemple de chorégraphie

Certes, les chorégraphies et les collaborations permettent la modélisation des PIO. Cependant, dans une chorégraphie un PIO est décrit comme un objet à part entière alors que dans une collaboration, un PIO n'est pas spécifié explicitement, mais il est modélisé à travers les processus qui le composent.

V. Conclusion

Dans ce chapitre, nous avons passé en revue les définitions des concepts de base relatifs aux processus métier intra- et inter-organisationnels, afin de fixer la terminologie que nous allons utiliser dans ce document. Nous avons également expliqué les perspectives à prendre en compte pour modéliser les processus. Nous avons aussi détaillé les types de coordination et les formes d'interopérabilité dans le cadre des processus inter-organisationnels. Finalement, nous avons détaillé la notation BPMN destinée à la modélisation graphique des processus.

Bien que BPMN 2.0 soit considéré comme le standard le plus répandu pour la modélisation des processus métier, cette notation souffre d'un certain nombre d'insuffisances qui sont :

- Un nombre important d'élément (plus de 100 éléments).
- Pas de distinction sémantique entre le modèle métier (non exécutable) et le modèle technique (exécutable). En effet, les efforts fournis pour avoir des diagrammes BPMN exécutables dégradent leur degré d'abstraction. Ainsi, ces diagrammes peuvent devenir complexes suite à l'adjonction de détails techniques facilitant leur interprétation par les moteurs d'exécution.
- Abandon de l'aspect opérationnel. BPMN explicite le travail à effectuer dans un processus sous forme de tâches (*i.e.,* activités atomiques) et de sous-processus (*i.e.,* activités composites). Ainsi, BPMN ne considère pas les opérations, *i.e.,* les actions élémentaires composant une tâche.

Après avoir présenté le contexte général de notre travail, nous détaillons dans le chapitre suivant le problème que nous adressons dans le cadre de cette thèse. En effet, plusieurs axes de recherche s'articulent autour du domaine de BPM. Nous nous intéressons plus particulièrement au problème de flexibilité des processus.

Chapitre II – Le versionnement pour la flexibilité des processus : Etat de l'art

Résumé. *La prise en compte des processus flexibles est un problème majeur du BPM. Plusieurs approches ont été proposées dans la littérature pour traiter ce problème. Ces approches différent en fonction du paradigme de modélisation utilisé pour représenter les processus (le paradigme orienté activité, le paradigme orienté données, le paradigme déclaratif, etc). Dans ce cadre, de nombreuses solutions traitant la flexibilité des processus ont été proposées, comme par exemple la technique des Worklet pour la flexibilité par incomplétude, l'approche Exlet pour la gestion des exceptions, l'approche préconisée dans Provop pour la configuration des variantes de processus, ou encore l'approche préconisée dans Adept, basée sur l'adaptation des schémas de processus et / ou de ses instances. Dans cette thèse, nous utilisons la technique de versionnement pour traiter le problème de modélisation des processus flexibles. Dans ce chapitre, nous défendons l'utilisation des versions pour la flexibilité des processus en nous basant sur les deux taxonomies les plus représentatives de l'état de l'art. Nous rendons également compte de l'état de l'art autour du versionnement dans les processus en présentant les contributions majeures. Nous présentons enfin les principaux travaux réalisés autour de la notion de contexte dans les processus. Cette notion de contexte est en effet fondamentale lorsqu'on aborde la problématique du visionnement des processus : il s'agit en effet d'être capable de préciser le contexte d'utilisation des versions modélisés afin de faciliter et rendre possible leur utilisation future.*

Sommaire du Chapitre II.

Durant les deux dernières décennies, il y a eu une évolution des systèmes d'information orientés données vers les systèmes d'information orientés processus. En effet, la place prise par les processus métier dans les organisations est devenue de plus en plus importante. Toutefois, les BPMS ne sont pas encore totalement adaptés aux besoins des organisations, notamment parce qu'ils ne supportent pas les processus flexibles. Ce problème de flexibilité des processus est crucial pour les organisations qui évoluent dans un environnement de plus en plus dynamique et ouvert à la concurrence : seules celles qui peuvent répondre de manière rapide et efficace aux exigences de leur environnement peuvent assurer leur survie. Les organisations sont donc amenées à réviser et modifier fréquemment leurs processus métiers. Par conséquent, le besoin d'un système de gestion de processus supportant les processus flexibles devient une nécessité.

Ce chapitre présente l'état de l'art des travaux abordant le problème de flexibilité en utilisant la technique de versionnement. Il se décompose en cinq sections.

La première section introduit et motive le problème de flexibilité des processus. Afin de caractériser le problème de la flexibilité des processus, elle présente ensuite les deux taxonomies les plus représentatives des travaux de l'état de l'art. Elle fait enfin un bilan de ces taxonomies, justifie le choix de la technique de versionnement pour aborder ce problème et donne un bref aperçu des principaux travaux effectués pour traiter le problème de la flexibilité des processus. La deuxième section du chapitre présente les principaux travaux autour du versionnement dans les processus. Elle introduit d'abord la notion de version. Elle détaille ensuite les travaux majeurs de l'état de l'art abordant le problème de la modélisation des processus flexibles en utilisant la technique de versionnement. L'insuffisance commune à ces travaux est qu'ils ne considèrent pas toutes les perspectives des processus (comportementale, organisationnelle, informationnelle et contextuelle) et qu'ils ne se basent pas sur un standard connu par les analystes, les concepteurs et les utilisateurs des processus métiers. Par ailleurs, la modélisation des versions de processus et leur future utilisation implique la spécification du contexte d'utilisation de ces versions. Ce contexte permet de définir la situation particulière dans laquelle une version sera utilisée. De ce fait, nous consacrons la troisième section du chapitre à la perspective contextuelle des processus. Plus précisément, cette section (i) permet de définir le contexte des processus métier, (ii) introduit la taxonomie de Rosemann [Rosemann *et al.,* 2008] pour la classification des informations contextuelles, et (iii) détaille les techniques de modélisation du contexte des processus.

La quatrième section présente la problématique traitée dans cette thèse. Elle énonce l'approche que nous préconisons pour traiter le problème de la flexibilité des processus : cette approche consiste à (i) l'adoption du paradigme procédural pour la représentation des processus sous forme d'activités coordonnées, (ii) l'adoption d'une approche basée sur les versions pour aborder le problème de la modélisation des processus flexibles, et (iii) l'adoption du standard BPMN pour une modélisation

normalisée des processus flexibles. Finalement, la cinquième section synthétise le chapitre et introduit le chapitre suivant.

I. Flexibilité des processus

I.1. Besoins, défis et motivation

Les entreprises sont sans cesse soumises à des nouvelles contraintes externes ou internes, en provenance des exigences économiques, des clients, ou des décisions stratégiques internes. Les entreprises les plus performantes sont donc celles qui s'adaptent le plus rapidement à ces contraintes externes ou internes en améliorant la qualité et la rapidité de leurs services ainsi que l'adéquation de leurs produits aux besoins des clients. Par conséquent, l'environnement dynamique dans lequel les entreprises évoluent leur impose d'être capables de rendre dynamiques (c'est-à-dire flexibles) leurs processus métiers. Un processus est qualifié de flexible s'il prend facilement en compte les changements fréquents imposés par divers facteurs externes et / ou internes. Ces changements peuvent remettre en cause la structure, la validité et l'adéquation d'un processus. Chaâbane [2012] a relevé principalement huit facteurs, non exclusifs, qui incitent à la modification et à l'adaptation des processus. Ces facteurs, ci-après expliqués, sont classifiés en deux types : (i) les facteurs environnementaux dus à l'apparition d'événements externes au processus, issus de son environnement d'exécution, et (ii) les facteurs internes au processus, relatifs à des difficultés rencontrées lors de leur définition ou de leur exécution. Les cinq premiers appartiennent au premier type, alors que les trois derniers sont du deuxième type.

1. Les évolutions technologiques : ces évolutions, tant au niveau matériel que logiciel, apportent des améliorations significatives et des solutions nouvelles, dont doivent tirer profit les entreprises, mais qui rendent obsolètes leurs équipements. Il faut alors adapter les processus existants et construire de nouvelles interfaces, utilisateurs et matériels.
2. Les événements externes et internes : certains événements, dont l'occurrence perturbe le bon fonctionnement des processus, nécessitent une réaction et une prise en charge rapide et efficace. L'origine de ces événements peut être interne et / ou externe à l'organisation. Les événements internes peuvent être dus à des problèmes techniques non prévus, nécessitant une modification du processus. Ces événements peuvent être une panne, l'absence d'un acteur, le cas d'un sinistre, etc. Quant aux événements externes, ils peuvent par exemple être la réception d'une commande spécifique nécessitant un traitement particulier différent du traitement habituel, le changement de législation, etc. Généralement, les adaptations qui sont dues à des événements internes et / ou externes n'ont d'effet que sur les cas futurs.
3. Les besoins des clients : dans un environnement qui devient de plus en plus ouvert à la concurrence, seules les entreprises qui veillent à satisfaire au mieux les besoins du marché, les exigences des clients en termes de qualité et de prix

4. Les changements de l'organisation : ils sont dus à l'application de nouveaux modes de gestion nécessitant une restructuration du travail. Généralement, les changements de l'organisation sont une conséquence directe du marché. Ces changements ont pour but d'optimiser les processus métiers et augmenter leur rentabilité.
5. Les changements stratégiques : ils s'expriment en termes d'orientations stratégiques, de clientèle visée, de partenariats, etc. Ces changements ont pour conséquence la reconfiguration de l'entreprise et l'optimisation des processus.
6. L'incertitude de l'information : étant donné la nature fluctuante et évolutive de l'environnement, les informations recueillies pour la définition des processus sont souvent incertaines et parfois contradictoires.
7. L'incomplétude de l'information : ce facteur est complémentaire au précédent. En effet, les informations recueillies peuvent être non seulement incertaines, mais dans certains cas sont insuffisantes. Les concepteurs des systèmes de gestion de processus, n'ayant pas le potentiel pour compléter les informations manquantes, doivent alors prévoir des solutions permettant aux utilisateurs du processus – les experts du domaine – de combler, lors de l'exécution, le manque qui existe dans la définition des processus.
8. Les conflits : ce facteur émerge dans le cas des processus inter-organisationnels. En effet, les conflits aux origines diverses (diversité des plateformes d'exécution, incompatibilité des informations, etc.) freinent le bon déroulement des processus. Il est par conséquent nécessaire de prévoir des solutions permettant de traiter les conflits lorsqu'ils surviennent entre les différents partenaires de ce processus.

I.2. Flexibilité des processus: Définitions

Plusieurs définitions du terme « flexibilité » ont été proposées dans la littérature. Malgré la diversité des domaines d'application de ce terme, toutes les définitions proposées mettent en avant le fait que la flexibilité décrit la capacité de modification d'une entité pour faire face aux contraintes évolutives tout en restant cohérente. Plus précisément, au sens littéraire du terme, le petit robert définit la flexibilité comme l'aptitude à changer facilement pour pouvoir s'adapter aux circonstances [Robert, 2002].

En économie, la flexibilité est ce qui caractérise la souplesse de l'organisation et des moyens de production d'une entreprise pour répondre aux fluctuations rapides de la demande et aux évolutions de l'environnement technico-économique. Elle est l'une des composantes de la réactivité industrielle.

En informatique, un système est dit flexible s'il peut s'adapter à plusieurs configurations (situations de fonctionnement) et contenir les solutions aux éventuels problèmes qui peuvent survenir.

Dans le domaine du BPM, il n'y a pas, à notre connaissance, de consensus ni sur ce qu'est un processus flexible, ni sur une solution pour la gestion de tels processus.

Nous énumérons dans ce qui suit les définitions de la littérature qui sont, à notre avis, les plus complètes à propos des processus flexibles.

Définition II.1 [Nurcan, 2008] :

> La flexibilité d'un processus se définit comme *la capacité de trouver un compromis entre, d'une part, satisfaire de manière rapide et facile les besoins de l'entreprise lorsqu'elle subit des changements organisationnels, fonctionnels et / ou opérationnels et, d'autre part, garder l'efficacité du processus ».*

Cette définition explicite les différentes dimensions pouvant être sujettes aux changements (les changements organisationnels, fonctionnels et opérationnels) nécessitant une adaptation du processus. Cependant, elle ne prend pas en compte la dimension informationnelle décrivant la structure des informations manipulées (produites et / ou utilisées) par le processus, qui peuvent elles aussi faire l'objet de changements.

Définition II.2 [Schonenberg *et al.,* 2008] :

> La flexibilité d'un processus se définit comme *la capacité à prendre en compte les changements prévisibles et non prévisibles au travers de la modification ou de l'adaptation de la seule partie de la définition du processus touchée par ces changements sans être obligé de redéfinir l'intégralité du processus.*

Cette définition met l'accent sur la capacité des systèmes à s'adapter aux éventuels changements. Plus précisément, les auteurs de [Schonenberg *et al.,* 2008] distinguent deux types de changements : les changements prévisibles et les changements non prévisibles. Les changements prévisibles correspondent à des situations ou à des configurations d'exécution connues à l'avance, au moment de la modélisation du processus. Quant aux changements non prévisibles, ils décrivent des modifications qui apparaissent lors de l'exécution des processus. Ces changements correspondent à des situations ou des configurations non connues préalablement. De plus, les changements non prévisibles peuvent être soit permanents, c'est-à- dire nécessitant des modifications de la définition du processus, soit occasionnels, c'est-à-dire spécifiques à un cas particulier et n'ayant aucune répercussion sur la définition du processus.

Chaâbane [2012] a récapitulé et fusionné les définitions II.1 et II.2 pour aboutir à une définition plus générale. En effet, selon ces auteurs la définition de [Schonenberg *et al.,* 2008] donne des prémices d'une taxonomie en fixant les différents types de flexibilité, alors que la définition de [Nurcan, 2008] indique les éléments pouvant être l'objet d'éventuelles modifications. C'est dans ce cadre qu'ils proposent une nouvelle définition déduite à partir des définitions précédentes.

Définition II.3 [Chaâbane 2012] :

> *La flexibilité d'un processus est sa capacité de prise en compte les changements prévisibles et non prévisibles qui se produisent suite à une modification de son schéma d'exécution, les acteurs qu'il invoque et / ou les ressources*

> *informationnelles qu'il manipule, tout en gardant l'efficacité et l'intégralité de la définition du processus.*

Dans nos travaux, nous avons choisi d'adopter la définition de Chaâbane [2012] puisqu'elle nous semble la plus complète. En effet, cette définition, d'une part, prend en compte les deux types de changements (prévisible et imprévisible) nécessitant la modification d'un processus et, d'autre part, considère la flexibilité des processus en prenant en compte les principales perspectives de modélisation de processus à savoir la perspective comportementale, la perspective organisationnelle et la perspective informationnelle.

I.3. Taxonomie des travaux existants

De nombreuses taxonomies caractérisant la notion de flexibilité des processus ont été proposées dans la littérature [Kumar *et al.,* 2006] [Narasipuram *et al.,* 2008] [Schonenberg *et al.,* 2008]. Ces taxonomies permettent, essentiellement, de mesurer la capacité des systèmes à supporter les processus flexibles. Nous détaillons dans ce qui suit les deux taxonomies les plus représentatives de l'état de l'art proposées dans les travaux de Nurcan [2008] et de Reichert et Weber [2012].

I.3.1. Taxonomie de Nurcan [Nurcan, 2008]

Nurcan [2008] distingue essentiellement deux types de flexibilité, à savoir la flexibilité *a priori* et la flexibilité *a postériori*. La distinction entre ces deux types de flexibilité repose sur les phases dans lesquelles les changements sont pris en compte (*Build time* ou *Run time*). Nous détaillons dans ce qui suit ces deux types de flexibilité en spécifiant les propriétés et les techniques de chacun de ces types.

La flexibilité a priori

La *flexibilité a priori* (dite aussi *flexibilité par sélection*) s'applique lors de la définition du processus, c'est-à-dire dans la phase de conception (*Build Time*). En effet, la définition du processus doit être spécifiée de façon à considérer les changements prévisibles de l'environnement opérationnel des processus. L'avantage majeur de ce type de flexibilité est que toutes les instances (les cas d'exécution) sont conformes à la définition du processus. La limite de ce type de flexibilité c'est qu'il ne peut être utilisé que lorsqu'on peut prévoir, lors de la définition, les éventuels changements.

Parmi les techniques permettant d'avoir une définition flexible d'un processus est de modéliser une définition incomplète de ce processus qui sera par la suite complétée en phase d'exécution. Nurcan [2008] distingue deux techniques de modélisation pour compléter les parties manquantes de la définition d'un processus dans la phase d'exécution (*i.e.,* les parties qui ne peuvent pas être définies au moment de la phase de conception) : la liaison tardive et la modélisation tardive.

- *Liaison tardive* : Cette technique consiste à modéliser une définition de processus incomplète comportant toutes les activités prévisibles ainsi que leur enchaînement et

à proposer plusieurs configurations possibles (définies sous forme de fragments d'activités) pour les parties imprévisibles. En phase d'exécution, le concepteur complète les parties manquantes de la définition du processus en sélectionnant le fragment d'activités approprié.

▪ *Modélisation tardive* : Cette technique consiste à proposer une définition incomplète d'un processus en phase de conception. La définition contient des parties qui sont laissées ouvertes à la création et à l'initiative du concepteur lors de l'exécution. Ainsi, au moment de l'exécution, le concepteur peut modéliser puis exécuter les parties manquantes. On peut avoir recours à cette technique (i) soit lorsqu'une partie de processus dépend du résultat de l'exécution des parties antérieures, (ii) soit lorsque des parties d'un processus ne peuvent être connues *a priori*, lors de la conception du processus.

La flexibilité a posteriori

La *flexibilité a posteriori* (dite aussi *flexibilité par adaptation*) permet de prendre en compte les changements imprévisibles se produisant dans l'environnement opérationnel du processus pendant son exécution. Dans ce cadre, Nurcan [2008] distingue deux principales propriétés de flexibilité : le type de changement et la nature de l'impact.

Type de changement : Nurcan [2008] distingue trois types de changements pouvant apparaitre au moment de l'exécution, à savoir les changements Ad-hoc, les changements correctifs et les changements évolutifs.

▪ *Changement Ad-hoc* : Un changement ad-hoc permet de considérer tout changement imprévu qui ne mérite pas d'être enregistré. Ce changement permet alors de traiter les exceptions qui se produisent lors de l'exécution. Un changement ad-hoc est appliqué sur les instances des processus, lorsque leurs schémas ne conviennent pas aux conditions d'exécution. Un tel changement a un impact local.

▪ *Changement Correctif* : Un changement correctif permet soit de corriger les erreurs de la définition des processus, soit de réagir aux exceptions qui peuvent se produire lors de l'exécution d'une instance de processus. Un changement correctif peut avoir un impact global (corriger les erreurs ou gérer les exceptions) ou local (gérer les changements ad-hoc).

▪ *Changement Evolutif* : Un changement évolutif permet de considérer les modifications à caractère permanent (c'est-à-dire qui doivent être gardées pour les exécutions futures), telles que les modifications qui se produisent pour prendre en compte une nouvelle réglementation, un nouvel outil technologique ou une nouvelle stratégie organisationnelle. Un changement évolutif a un impact global.

Nature de l'impact : Généralement, la flexibilité a posteriori nécessite une adaptation des instances en cours d'exécution. Cette adaptation peut ne porter que sur

l'instance en cours ou que sur les instances futures. Par conséquent, Nurcan [2008] différencie deux types d'impact : *impact local* et *impact global*.

- *Impact local :* un changement ayant un impact local ne peut toucher que l'instance de processus en cours d'exécution, sans être propagé jusqu'à la définition du processus. Ce type de changement correspond à une modification occasionnelle de l'instance.

- *Impact global :* un changement ayant un impact global touche le(s) instance(s) en cours d'exécution, mais se répercute aussi sur la définition du processus afin de pouvoir être pris en compte dans les exécutions à venir.

De plus, selon Nurcan [2008] la flexibilité ayant un impact global évoque le problème des instances en cours d'exécution dont la définition a été modifiée. Pour remédier à ce problème, Nurcan [2008] propose trois techniques de migration : *Migration par annulation, Migration avec propagation* et *Migration sans propagation*.

- *Migration par annulation* : cette technique consiste à annuler les instances en cours d'exécution (ces instances ont débuté l'exécution conformément à l'ancienne définition du processus, c'est-à-dire avant la modification) et à les relancer conformément à la nouvelle définition du processus.

- *Migration avec propagation* : Cette technique consiste à propager les modifications de la définition d'un processus aux instances qui sont en cours d'exécution. Il s'agit de modifier automatiquement les instances courantes, c'est-à-dire les instances dont l'exécution a débuté selon une ancienne définition de processus, en accord avec la nouvelle définition du processus sans perte d'informations. Cette technique est la plus difficile à mettre en œuvre. Elle est n'est pas toujours possible, notamment quand les changements touchent la partie déjà réalisée du processus.

- *Migration sans propagation* : Cette technique consiste à terminer les exécutions des instances en cours selon l'ancienne définition. Les modifications seront prises en compte lors des prochaines exécutions, c'est-à-dire que la nouvelle définition ne sera considérée que pour les nouvelles instances.

Finalement, la flexibilité a posteriori ayant un impact global peut être assurée en utilisant la technique de versionnement. Cette technique permet de définir plusieurs versions pour la même définition d'un processus. Elle offre les moyens pour permettre l'évolution des processus et anticiper sur les futurs changements.

La Figure II.1 résume la taxonomie de [Nurcan, 2008]. Elle présente les deux types de flexibilité, la flexibilité a priori et la flexibilité a postériori, ainsi que les propriétés et les techniques utilisées pour assurer ces deux types de flexibilité.

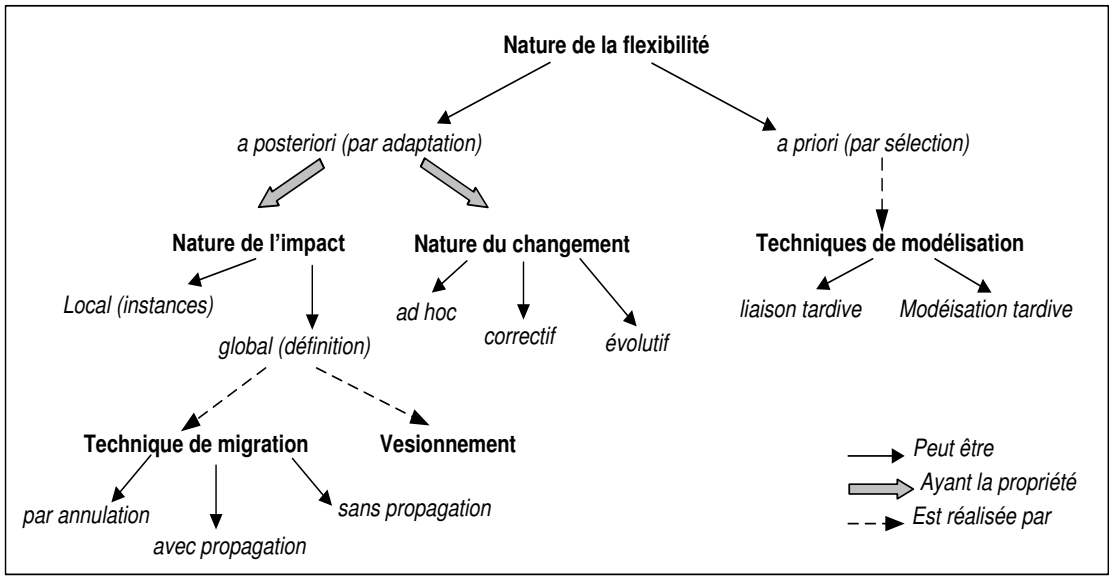


Figure II.1. Un extrait de la taxonomie de Nurcan [Nurcan, 2008]

I.3.2. Taxonomie de Reichert [Reichert et Weber, 2012]

Reichert et Weber [2012] distinguent les quatre besoins de flexibilité suivants : la variabilité (variability), l'incomplétude (looseness), l'adaptation (adaptation) et l'évolution (evolution). Chacun de ces besoins peut affecter les différentes perspectives de modélisation des processus. Dans ce qui suit, nous détaillons ces besoins.

La variabilité

La variabilité consiste à considérer différentes variantes d'un même processus. Ces variantes ont soit le même objectif, soit un objectif similaire mais diffèrent en fonction du contexte. En général, les variantes d'un même processus partagent la même définition de ce processus, dite processus de base (Basic Process). Cependant, cette définition fluctue d'une variante à une autre en fonction des circonstances rencontrées. Les fluctuations peuvent être dues aux facteurs suivants : différence entre règlements des pays et des régions, différence entre produits et services, différence entre groupes de clients ciblés et contraintes temporelles (tels que les changements de saisons et les périodes de vacances). Nous reviendrons en détail sur les facteurs incitant les variations des processus dans la section III de ce chapitre qui aborde le contexte des processus.

Exemple de variabilité : Dans un processus d'examen médical, plusieurs variantes peuvent être définies. Ces variantes diffèrent en fonction de la nature de l'examen (examen standard vs en urgence), la façon dont l'examen est géré (examen planifié par un rendez-vous vs examen immédiat) et le besoin d'activités spécifiques en fonction de l'environnement d'application donné.

Approches de variabilité : Plusieurs travaux de l'état de l'art ont traité des solutions pour la variabilité des processus [Gottschalk *et al.,* 2008], [Li, 2009], [La Rosa *et al.,* 2009], [Conforti *et al.,* 2015] et [Reichert *et al.,* 2015]. Ces travaux proposent des solutions permettant la configuration des variantes d'un processus selon

deux approches différentes : une approche basée sur le comportement des processus (behavior-based approach) et une approche basée sur une configuration structurelle des processus (Structural configuration approach).

L'approche basée sur le comportement des processus permet de prendre en compte toutes les variantes d'un même processus dans un seul modèle appelé modèle de référence d'un processus (reference process model). Ce modèle peut être représenté sous forme d'un système de transition étiqueté (Labled Transition System LTS) [Gottschalk *et al.,* 2008] ou bien avec un langage de modélisation de processus tel que C-EPC (Configurable Event-driven Process Chains) [Rosemann et Aalst, 2005]. De façon générale, dans cette approche, le modèle de référence d'un processus est configuré de sorte à ce qu'il comporte plusieurs alternatives (*i.e.,* chemins possibles depuis le début jusqu'à la fin du processus). Cependant, à un moment donné, un seul chemin est accessible. Cette configuration peut se faire soit par la dissimulation ou le blocage de certaines activités, soit par l'utilisation des nœuds de configuration (des activités et des éléments de routage) marquant les points de variation du modèle de référence.

L'approche basée sur une configuration structurelle définit aussi un processus de base (base process model) à partir duquel plusieurs variantes peuvent être déduites. En effet, ce processus de base comporte des points de variations auxquelles une série d'opérations peut être appliquée afin de déduire les variantes. Ces opérations permettent d'appliquer des changements structurels assurant l'insertion, la suppression et / ou le déplacement d'une activité ou bien d'un fragment (ensemble d'activités coordonnées) à partir du processus de base [Hallerbach *et al.,* 2010(a)] [Hallerbach *et al.,* 2010(b)] [Reichert *et al.,* 2015].

L'incomplétude

Dans ce type de flexibilité, seuls les objectifs des processus sont définis a priori. De ce fait, la définition du processus ne peut pas être entièrement connue à l'avance, c'est-à-dire en phase de conception des processus. Cette définition doit alors être complétée en phase d'exécution. Dans ce cas, les changements ne concernent qu'une partie de la définition du processus.

Exemple d'incomplétude : Le processus de traitement des patients hospitalisés dans une clinique ne peut pas être identique pour tous les patients. De ce fait, la définition de ce processus ne peut pas être totalement prévue à l'avance étant donné que le traitement donné à un patient dépend de plusieurs facteurs tels que l'état de santé du patient, les allergies et les intolérances chimiques, les résultats des examens et les décisions prises par le médecin.

Approches d'incomplétude : Dans la littérature, plusieurs modèles de décision ont été définis pour compléter la définition d'un processus au moment de l'exécution. Parmi ces modèles nous citons :

- La *liaison tardive*. La liaison tardive concerne les cas où le concepteur sait qu'une activité est à réaliser mais ne connaît pas la manière dont cette activité va se réaliser. Il la définit alors de manière abstraite, mais avec différentes concrétisations possibles. L'utilisateur du processus aura donc le choix, en cours d'exécution, entre ces concrétisations. Dans ce cadre, la littérature distingue essentiellement l'approche *Worklets* proposée dans [Adams *et al.,* 2006] [Adams, 2013] et l'approche de liaison tardive automatisée proposée dans [Klingemann, 2000]. L'approche *Worklets* préconise l'utilisation d'une technique basée sur des règles de sélection nommée *Ripple down rules* pour choisir la concrétisation appropriée au moment de l'exécution, alors que l'approche de liaison tardive automatisée utilise une technique basée sur les buts pour effectuer ce choix.

- La *modélisation tardive*. La modélisation tardive concerne les cas où le concepteur ne sait pas définir une activité lors de la conception du processus. Une telle activité est qualifiée d'abstraite ; sa définition est différée au moment de l'exécution du processus par le concepteur du processus. L'approche Pockets of flexibility proposée dans [Sadiq *et al.,* 2005] fournit une implémentation concrète de la modélisation tardive. Dans cette approche, les activités abstraites sont considérées comme des fragments composés par des activités non reliées et des contraintes définissant comment ces activités peuvent être reliées au moment de l'exécution.

- Raffinement itératif. Avec un raffinement itératif, la modélisation et l'exécution des processus se font de façon entrelacée. Le schéma du processus peut être itérativement modélisé et exécuté. Au moment de l'exécution, le concepteur peut alors redéfinir le schéma du processus en ajoutant de nouvelles activités ou en supprimant des activités existantes. Le système *Aleaska Simulator* est un exemple de système implémentant le raffinement itératif [Reichert et Weber, 2012].

- La composition Ad hoc. Avec une composition ad hoc, le schéma du processus est composé dynamiquement au moment de l'exécution. De ce fait, l'instance du processus évolue de façon ad hoc puisque les activités qui la forment ne sont pas prédéfinies. *Declare*, décrit dans [Pesic *et al.,* 2007(a)] et [Pesic, 2008] est un système supportant la composition ad hoc.

L'adaptation

Ce type de flexibilité permet l'adaptation de la définition d'un processus ou de ses instances pour faire face aux événements émergents. Ces événements mènent à des situations dans lesquelles un processus (*i.e.,* sa définition et / ou ses instances) ne reflète pas le mode opérationnel réel. Par conséquent, ce processus doit être adapté pour être en adéquation avec l'existant.

Plusieurs facteurs peuvent déclencher le besoin d'adaptation des processus parmi lesquels nous citons : l'apparition de situations spéciales (*i.e.,* non prévisibles) au moment de l'exécution, le traitement des cas exceptionnels du monde réel (*e.g.,* une réaction allergique d'un patient face à un traitement) et le traitement des erreurs (*e.g.,* activité échouée).

Exemple d'adaptation : L'absence d'une ressource responsable à l'exécution d'une ou de plusieurs activités d'un processus nécessite l'affectation de ces activités à une autre ressource ou bien l'annulation de ces activités si c'est possible.

Approches d'adaptation : De façon générale, les adaptations effectuées sur la définition du processus et / ou de ses instances peuvent être planifiées à l'avance. Dans ce cas, il est possible de planifier les déviations possibles par rapport à la définition du processus ; il s'agit alors du traitement des *Exceptions anticipées.* Ces exceptions peuvent émerger suite à un dysfonctionnement d'un système, suite à une indisponibilité d'une ressource ou bien suite à l'expiration de deadlines. Deux principales contributions illustrent cela. Tout d'abord, [Lerner *et al.,* 2010] propose trois catégories de patrons permettant le traitement des exceptions anticipées : la définition des alternatives comportant le traitement nécessaire si une exception est déclenchée, l'adjonction d'un comportement (tel que la fixation immédiate ou différée d'une activité) ou l'annulation d'un comportement (tel que le rejet d'une partie du processus). Ensuite, l'approche *Exlet,* proposée dans [Adams *et al.,* 2007], fournit un mécanisme permettant la gestion des exceptions anticipées. Dans cette approche, les auteurs associent à chaque activité d'un processus un ensemble de sous-processus pour la gestion des exceptions, appelés *Exlets*. Un Exlet définit le comportement à effectuer si une exception est détectée. Plus précisément, lorsqu'une exception se produit au moment de l'exécution, l'Exlet (*i.e.,* le sous processus traitant cette exception) est sélectionné dynamiquement en utilisant des règles de sélection hiérarchiques, appelées *Ripple down rules*.

Néanmoins, il n'est pas possible de prévoir toutes les exceptions qui peuvent se produire dans la définition d'un processus. Par conséquent, les instances d'un processus doivent être adaptées lors de l'exécution pour traiter les exceptions non anticipées. Ces exceptions sont traitées dans [Weber *et al.,* 2013] sous forme de changements Ad-hoc. Plus précisément, dans [Weber *et al.,* 2013] deux méthodes sont proposées pour assurer une adaptions structurelle d'un processus : (i) *les primitives de changement,* qui représentent des opérations de bas niveaux permettant d'adapter la définition d'un processus ou de ses instances en ajoutant un nœud ou un flux de séquence ou en supprimant un nœud ou un flux de séquence, et (ii) les *patrons d'adaptation,* qui représentent des opérations de haut niveau regroupant un ensemble de primitives de changement permettant l'adjonction, la suppression ou le déplacement d'un fragment de processus. Ces deux méthodes sont présentées en détail dans la section II.2.1.

L'évolution

Ce type de flexibilité considère les changements à caractère évolutif que peut avoir un processus. Ces changements peuvent être *permanents* ou *temporaires*. Les changements permanents sont valides à partir du moment où ils sont mis en œuvre alors que les changement temporaires sont valides pour une période de temps bien déterminée (*e.g.,* une période de promotion). De plus, l'évolution d'un processus peut

être une *évolution incrémentale* ou bien une *évolution révolutionnaire*. L'évolution incrémentale nécessite un simple ajustement par rapport au processus initial alors que l'évolution révolutionnaire nécessite un changement radical du processus (dans le cadre d'une innovation de ce processus par exemple).

Généralement, plusieurs facteurs peuvent causer un besoin d'évolution d'un processus. Ces facteurs peuvent être classés en deux types :

- **Facteurs externes** : Les facteurs externes peuvent être relatifs à (i) des changements des acteurs externes des processus (tel que le changement des besoins des consommateurs, des clients etc.,), (ii) des changements technologiques de l'environnement externe des organisations (telles que les innovations technologiques) ou bien (iii) des changements du contexte juridique dans lequel les organisations opèrent (telle que l'apparition d'une nouvelle réglementation).
- **Facteurs internes** : les facteurs internes peuvent être relatifs (i) aux erreurs de modélisation (tels que les deadlocks et les données manquantes) qui peuvent causer des problèmes lors de l'exécution des instances du processus, (ii) aux problèmes techniques (telle que la dégradation des performances à cause d'une augmentation de la quantité des données), ou bien (iii) aux stratégies internes d'une organisation (telle que l'adoption d'une nouvelle ligne directive).

Approches d'évolution : L'évolution d'un processus évoque le problème de traitement des instances de ce processus qui sont en cours d'exécution. Dans certains cas, il est suffisant d'appliquer les changements sur les nouvelles instances créées et de compléter l'exécution des instances en cours d'exécution conformément à l'ancienne définition de ce processus. Il s'agit alors d'une *évolution différée* qui peut être traitée avec la technique de versionnement. Dans d'autres cas, les changements doivent être appliqués sur les instances en cours d'exécution, d'où la nécessité de migration de ces instances. Dans ce cas, il s'agit d'une *évolution immédiate* utilisée surtout pour les processus nécessitant une longue durée d'exécution. Dans une évolution immédiate, la migration des instances en cours d'exécution peut être une *migration non contrôlée* ou bien une *migration contrôlée.* Dans une migration contrôlée, l'exécution de l'instance modifiée ne peut continuer que s'il n'y a aucun risque de blocage ou de données manquantes alors que dans une migration non contrôlée l'exécution poursuit sans aucun contrôle.

Plusieurs systèmes assurant l'évolution des définitions des processus et la migration de leurs instances ont été proposés dans littérature. Ces systèmes peuvent être classés en deux types d'approches : (i) les approches basées sur l'historique d'exécution des processus *History-Based Approaches,* tels que WIDE [Casati *et al.,* 1998] et ADEPT2 [Rinderle et Reichert, 2010] qui proposent des solutions de migration contrôlée en se basant sur les fichiers log résultant des anciennes exécutions, et (ii) les approches basées sur une représentation graphique des activités en cours d'exécution, appelées *Snapshot-based Approaches*. Cette approche a été adoptée par [Ellis *et al.,* 1995] en modélisant un processus ainsi que ses instances dans un seul

réseaux de Petri coloriés dans lequel des jetons de différentes couleurs représente les différentes instances de ce processus.

La Figure II.2 résume la taxonomie de [Reichert et Weber, 2012]. Elle présente les différents besoins de flexibilité ainsi que les techniques utilisées pour satisfaire ces différents besoins.

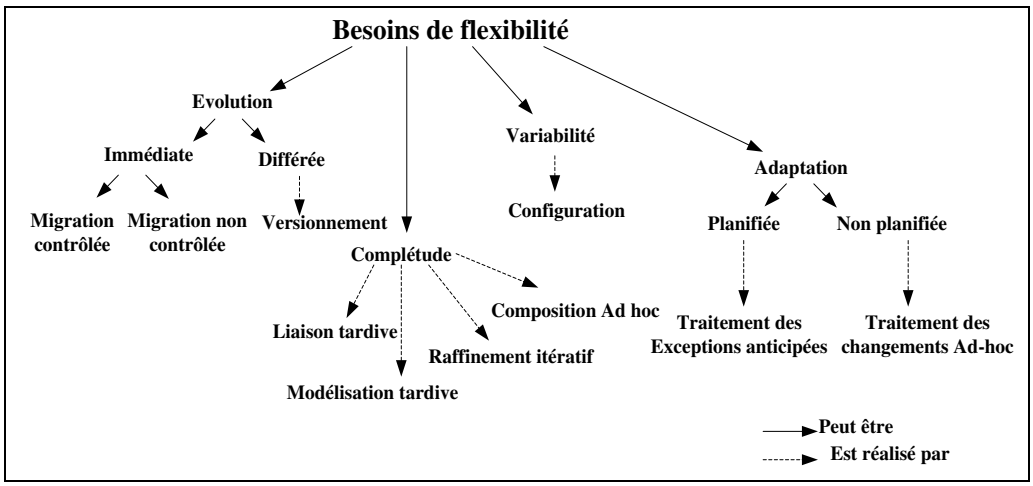


Figure II.2. Résumé de la taxonomie de Reichert

I.3.3. Synthèse des taxonomies

Les taxonomies présentées précédemment résument les types, les besoins et les techniques de flexibilité existant dans les travaux de l'état de l'art. Elles diffèrent l'une de l'autre au niveau de la classification du problème de flexibilité. Nurcan [2008] présente une taxonomie distinguant deux types de flexibilité et chacun de ces types est caractérisé par un ensemble de propriétés et techniques utilisées pour les assurer. Reichert présente une taxonomie plus consistante, au travers de la distinction de quatre besoins de flexibilité qui considèrent toutes les situations nécessitant une réaction face à un changement. De plus, cette taxonomie identifie les techniques utilisées pour assurer chacun de ces besoins.

D'après les deux classifications proposées par Nurcan [2008] et Reichert et Weber [2012], nous pouvons constater que la flexibilité des processus est nécessaire dans les deux phases de leur cycle de vie : la flexibilité durant la phase de conception et la flexibilité durant la phase d'exécution.

1) La flexibilité durant la phase de conception : Durant cette phase la flexibilité est relative au schéma (ou à la définition) de processus. Un schéma est dit flexible s'il contient des notions, des concepts et / ou des mécanismes permettant de considérer les changements prévisibles qui touchent les différents composants d'un processus (*e.g.,* activités et rôles). La flexibilité durant la phase de conception correspond au type de flexibilité a priori de Nurcan et aux besoins de la variabilité, de l'adaptation planifiée et de l'évolution de Reichert.

2) La flexibilité durant la phase d'exécution : Suite à des changements non prévisibles survenant au moment de l'exécution, le schéma d'une instance d'un processus peut ne pas être conforme au schéma de processus préalablement modélisé. De ce fait, cette instance doit être modifiée pour répondre à ces changements. Dans ce cas, les BPMS, et en particulier les moteurs d'exécution, doivent être capables de supporter la flexibilité des instances en cours d'exécution en offrant la possibilité de modifier le schéma de l'instance sans, toutefois, être amenés à modifier le schéma de ce processus. La flexibilité durant la phase d'exécution correspond au type de flexibilité a posteriori de Nurcan et aux besoins de l'incomplétude et de l'adaptation non planifiée de Reichert.

Plusieurs travaux de recherche ont proposé des solutions pour traiter la flexibilité durant la phase de conception. D'autres travaux se sont intéressés à la flexibilité dans la phase d'exécution. Les fruits de ces travaux sont des systèmes de gestion de processus capables de gérer les modifications qu'une instance peut avoir au cours de son exécution, à l'instar d'ADEPT1 [Reichert et Dadam, 1998], ADEPT2 [Reichert *et al.,* 2005] [Dadam et Reichert, 2009], YAWL [Aalst *et al.,* 2004], Declare [Pesic *et al.,* 2007(b)], SeaFlows [Ly *et al.,* 2011], ou AristaFlow [Lanz *et al.,* 2010]. Dans cette thèse, nous abordons la problématique de la modélisation des processus flexibles en considérant la modélisation de la flexibilité en phase de conception (a priori) et en phase d'exécution (a posteriori).

D'autre part, la difficulté du problème de la flexibilité des processus varie en fonction du paradigme de modélisation adoptée. En effet, dans un paradigme déclaratif, ou orienté décision, les solutions à ce problème sont plus naturelles (par exemple il suffit de pouvoir ajouter dynamiquement une règle à un processus pour le qualifier de flexible). Cependant, les solutions à ce problème ne sont pas intuitives et faciles si nous optons pour le paradigme procédural (orientée activité), qui décrit de manière explicite les activités du processus et leurs enchaînements.

Dans ce qui suit, nous présentons un bref aperçu des solutions de l'état de l'art traitant la flexibilité des processus, tout en les classant selon le paradigme de modélisation utilisé et en les positionnant par rapport à la taxonomie de Reichert. Dans la suite de ce document, nous nous basons sur la taxonomie de Reichert puisqu'elle est la plus récente et qu'elle nous semble la plus complète. En effet, cette taxonomie caractérise la flexibilité des processus en identifiant quatre besoins différents. Ainsi, un BPMS qui supporte les processus flexibles doit être capable de considérer ces besoins.

I.3.4. Survol des travaux de l'état de l'art traitant du problème de la flexibilité des processus

Les travaux classés par paradigme de modélisation

La littérature comporte de nombreux travaux apportant des solutions au problème de la flexibilité des processus. Nous présentons, dans cette partie, un bilan de ces travaux tout en les positionnant par rapport à la taxonomie de Reichert. Ces travaux

peuvent être classés selon le paradigme de modélisation adopté pour définir les processus. Ainsi, nous distinguons l'approche guidée par les données, l'approche guidée par les contraintes, l'approche guidée par les intentions, et l'approche guidée par les activités (dite aussi procédurale).

L'approche guidée par les données : Dans cette approche, la modélisation des processus est guidée par les données qu'ils manipulent. Si, les travaux relevant de ce paradigme considèrent à la fois les schémas de processus et leurs instances, ils mettent au premier plan les données qu'ils manipulent et leur cycle de vie pour en déduire les activités qui les composent. Plusieurs travaux ont adopté ce paradigme, [Cohn et Hull, 2009], [Damaggio *et al.,* 2011], [Estanol *et al.,* 2012], mais nous mettons plutôt COREPRO, [Müller *et al.,* 2007], [Müller *et al.,* 2008(a)], [Müller *et al.,* 2008(b)] en avant, qui, tout en respectant ce paradigme, ont abordé la problématique de l'adaptation des processus. Plus précisément, COREPRO est un environnement supportant la modélisation, l'exécution et l'adaptation de processus orientés-données. Il propose de prendre le cycle de vie des données d'un système comme point de départ de la modélisation de ses processus. L'adaptation des processus repose sur l'adaptation des données qu'ils manipulent. Cette adaptation consiste en un ensemble d'opérations de haut niveau permettant d'appliquer des modifications au niveau des données et non au niveau des processus eux-mêmes, ce qui garantit, d'après les auteurs, une plus grande facilité à les mettre en place. Le traitement de la flexibilité des processus en adoptant un paradigme orienté données satisfait, essentiellement, le besoin d'évolution de la taxonomie de Reichert.

L'approche guidée par les contraintes : Cette approche consiste à spécifier un ensemble d'activités et un ensemble de contraintes (exprimées par des règles) permettant de préciser la procédure d'exécution de ces activités. Deux catégories de travaux ont utilisé la notion de contraintes pour représenter les schémas de processus flexibles. Dans la première catégorie, les auteurs ont utilisé des règles ECA (Event-Condition-Action) pour définir les schémas des processus. Plus précisément, les règles sont utilisées pour décrire les contraintes d'exécution des activités des processus. Les travaux pionniers sont WIDE [Baresi *et al.,* 1999] et ceux menés dans VORTEX [Hull *et al.,* 1999]. Nous citons également les contributions à travers AgentWork [Müller *et al.,* 2004], AdaptFlow [Grenier *et al.,* 2004], BPTrigger [Park et Choi, 2004], ECAPE [Boukhbouze *et al.,* 2009] qui étend ECA par les notions de post-condition et post-événement. Dans la deuxième catégorie de travaux, les auteurs adoptent une démarche déclarative, en spécifiant les activités et les contraintes (règles) des processus. Dans ce cas, le schéma des processus n'est pas explicitement défini. Plusieurs travaux ont suivi cette approche. Nous pouvons citer BPAL [Lezoche *et al.,* 2008], mais c'est probablement DECLARE [Pesic et Aalst, 2006], [Pesic *et al.,* 2007(a)], [Pesic *et al.,* 2007(b)], [Pesic, 2008] qui constitue la proposition la plus complète. DECLARE est un prototype de système de gestion de processus qui utilise la logique temporelle pour modéliser et exécuter des processus métiers. Il précise l'ensemble des contraintes qui guident l'exécution du processus. Ces contraintes représentent les politiques qui ne

doivent pas être violées. Il est à noter que l'approche déclarative supporte la flexibilité de manière simple et intuitive. Il s'agit simplement d'ajouter ou de supprimer des contraintes pour s'adapter à une situation particulière [Pesic *et al.,* 2007(a)]. Ainsi, l'approche dirigée par les contraintes supporte les besoins de flexibilité suivants : (i) le besoin d'adaptation non anticipée nécessitant l'application des changements ad-hoc pour modifier l'instance en cours d'exécution, (ii) le besoin d'incomplétude et (iii) le besoin d'évolution qui consiste à modifier le schéma de processus pour les instances futurs. Cependant, les travaux adoptant cette approche sont complexes et difficiles à interpréter puisqu'ils ne permettent pas une représentation graphique du schéma de processus.

L'approche guidée par les buts : Les travaux adoptant cette approche préconisent l'utilisation des notions d'intention et de contexte pour la spécification et l'adaptation des processus [Nurcan, 2008], [Nurcan, 2011]. Parmi les travaux adoptant cette approche nous citons celui de [Saidani et Nurcan, 2006(a)], [Saidani et Nurcan, 2006(b)], où les auteurs pensent que les acteurs tiennent une place prépondérante dans une approche guidée par les buts car ils sont vus comme des individus ayant une capacité de décision. Ainsi, les responsabilités et les capacités d'un acteur peuvent varier selon les situations. Par exemple, dans une situation d'urgence, un acteur peut être autorisé à assumer une responsabilité, qui nécessiterait des qualifications supérieures dans une situation normale. Il est à noter que cette approche satisfait les besoins de l'incomplétude et de l'adaptation non anticipée. En effet, les utilisateurs impliqués dans l'exécution d'un processus peuvent, en fonction du contexte et des objectifs à atteindre, soit choisir les activités les plus appropriées pour compléter la définition de l'instance en cours d'exécution, soit adapter le schéma de l'instance en cours d'exécution pour être en accord avec le contexte de l'exécution. De ce fait, les acteurs n'ont pas à leur disposition de plan pré-établi, ils ont simplement des habitudes auxquelles ils peuvent déroger (dévier) suivant les cas.

L'approche guidée par les activités : Cette approche se base sur une représentation explicite des processus à travers leur schéma qui décrit les activités (tâches) réalisées et leur synchronisation, les ressources mobilisées pour la réalisation de ces activités ainsi que les données qu'elles consomment et produisent. Les travaux adoptant cette approche pour aborder le problème de flexibilité des processus se divisent en quatre catégories : les travaux autour de l'*évolution des processus*, les travaux autour de la disponibilité des partenaires, les travaux autour de la notion de *variante*, les travaux autour de la notion de *version*.

- Les travaux autour de l'évolution des processus : Les contributions des chercheurs ont considéré le problème de la flexibilité des processus en abordant l'évolution de leurs schémas et la migration de leurs instances. Ces travaux ont principalement porté sur l'évolution des schémas des processus intra-organisationnels. Ils ont, d'une part, spécifié des modèles décrivant les concepts relatifs aux schémas des processus et ont, d'autre part, défini des d'opérations de bas niveau pour l'évolution de ces schémas. Parmi ces travaux, nous pouvons citer ceux menés dans [Casati *et al.,*

1996]. Mais la principale contribution dans cette direction est probablement ADEPT [Dadam et Reichert, 2009]. D'autres, plutôt assez récents, ont investigué l'évolution des schémas des processus inter-organisationnels. Nous citons [Fdhila *et al.,* 2012] et [Fdhila *et al.,* 2015], qui proposent un ensemble d'algorithmes supportant la propagation des changements du processus d'un partenaire impliqué dans un processus inter-organisationnel sur les processus des autres partenaires de ce processus. Un autre travail intéressant est celui défini dans [Boukhedouma *et al.,* 2013(a)] et [Boukhedouma *et al.,* 2013(b)], qui proposent des patrons pour l'adaptation des processus (services) dans un contexte d'Architecture Orientée Services tout en considérant les formes d'interopérabilité : l'exécution chainée et le faiblement couplé. Les contributions autour de l'évolution des processus se focalisent essentiellement sur la perspective comportementale et supportent le besoin de l'évolution de la taxonomie de Reichert.

▪ Les travaux autour de la disponibilité des partenaires : [Chebbi *et al.,* 2006], [Andonoff *et al.,* 2009] et [Bouaziz et Andonoff, 2013] ont traité le problème de la disponibilité des partenaires, dans le cadre des processus inter-organisationnels lâches (*cf.* Chapitre I section IV). Les solutions proposées offrent des framework permettant la recherche de nouveaux partenaires et la négociation et l'élaboration de contrats entre ces partenaires. Ces contributions supportent une forme d'incomplétude de la taxonomie de Reichert, notamment l'incomplétude relative à la défaillance ou la disparition de partenaires dans le cadre de processus inter-organisationnels.

▪ Les travaux autour de la notion de variante : La notion de variante a été introduite pour traduire que plusieurs instances d'un processus, *i.e.,* une famille de processus, peuvent représenter les différentes configurations possibles du processus en question. Ainsi plusieurs variantes d'un même processus peuvent coexister ensemble, sachant que chaque variante a son propre schéma de processus. Généralement, un schéma générique de processus est défini. Ce schéma doit être, par la suite, configuré pour déduire les variantes [Rosemann et Aalst, 2005], [Hallerbach *et al.,* 2010(a)], [Döhring et Zimmermann, 2011]. Les contributions autour des variantes supportent le besoin de variabilité de la taxonomie de Reichert.

▪ Les travaux autour de la notion de version : Cette notion consiste à représenter les états significatifs d'un processus par différentes versions. Ces versions permettent de marquer les évolutions que peut subir un processus dans le temps. Elles peuvent également capturer les variations d'un processus par la définition de versions alternatives utilisées en fonction du contexte survenant. Les travaux de l'état de l'art abordant le problème de flexibilité en utilisant la technique de versionnement ont proposé de versionner les processus avec différents niveaux de granularité. En effet, certain travaux se sont focalisés seulement sur le versionnement des processus [Zhao *et al.,* 2007] [Ekanayake *et al.,* 2011]. D'autres travaux ont proposé, en plus du versionnement des processus, le versionnement de ses composants (*i.e.,* les activités, les rôles, etc.) [Kradofler et Geppert, 1999] [Dadam *et al.,* 2009], [Chaâbane, 2012].

Selon nous, l'utilisation de la technique de versionnement est très bien adaptée pour traiter le problème de la flexibilité des processus.

Dans ce travail, nous allons utiliser la technique de versionnement pour supporter les processus flexibles en se basant sur une approche guidée par les activités. En effet, le choix de cette approche est justifié par les raisons suivantes : (i) la possibilité de représenter graphiquement les schémas des processus ce qui facilite leur interprétation, (ii) la majorité des BPMS adoptent cette approche Yawl [Aalst *et al.,* 2004], Adept [Dadam et Reichert, 2009], FLOWER [Aalst *et al.,* 2005], [Aalst et Berens, 2001].

Le paragraphe qui suit, présente les arguments qui défendent le choix de la technique de versionnement pour traiter le problème de la flexibilité des processus en identifiant la capacité de cette technique à supporter les besoins de flexibilité proposés dans la taxonomie de Reichert.

Vers une solution basée sur les versions

Dans ce qui suit, nous apportons des réponses à la question suivante : dans quels cas les versions sont utiles pour considérer les processus flexibles ? La taxonomie proposée dans [Reichert et Weber, 2012] sert de support pour répondre à cette question.

D'après les besoins de flexibilité énoncés dans la taxonomie de Reichert, nous pensons que le versionnement peut être utilisé pour assurer le besoin de variabilité. En effet, des versions alternatives d'un même processus peuvent être définies dont chacune convient dans un contexte bien déterminé. En plus, le versionnement permet de considérer les exceptions prévisibles pouvant être anticipées au moment de la modélisation. Dans ce cas des versions alternatives peuvent aussi être modélisées pour considérer les comportements exceptionnels des processus. Finalement, nous voyons que le versionnement favorise également l'évolution des schémas de processus puisqu'il facilite la migration des instances de processus. En effet, le versionnement des schémas d'un processus permet à ses instances de s'exécuter conformément à des schémas différents, chacun étant représenté par une version du processus. La migration d'instance est donc facilitée puisqu'elle n'est plus obligatoire : une instance de processus en cours d'exécution n'a pas obligation de migrer pour se conformer au niveau schéma : elle peut continuer à s'exécuter en accord avec l'ancien schéma du processus.

La section suivante fait un état de l'art des contributions autour du versionnement dans les processus. Elle commence par introduire le concept de version. Puis, elle présente les principaux travaux de l'état de l'art traitant de la problématique de gestion des processus flexibles en utilisant les versions.

II. Le versionnement pour aborder le problème de la flexibilité des processus : Etat de l'art

Cette section introduit, dans un premier temps, la notion de version. Elle détaille, dans un second temps, les travaux majeurs de l'état de l'art traitant du problème des processus flexibles en adoptant la technique de versionnnement.

II.1. Notion de version

Une version correspond à un des états qu'une entité (*e.g.,* un processus) peut prendre durant son cycle de vie. De ce fait, il est possible de décrire les changements de cette entité à travers ces différentes versions. Ces versions sont rattachées par un lien de dérivation et forment une hiérarchie de dérivation comme illustré en Figure II.3. Chaque version de la hiérarchie, à l'exception de la première, est dérivée d'une version antérieure. Une version dérivée peut être soit une alternative, soit une évolution de la version origine de la dérivation. Une dérivation par évolution est utilisée pour considérer les changements qui peuvent se produire sur les entités tels que par exemple l'utilisation d'une nouvelle technologie ou bien l'alignement à une nouvelle législation. Une dérivation par alternative est utilisée pour considérer les variations ou les adaptations qu'une entité peut subir comme exemple le traitement d'un cas exceptionnel. Dans la Figure II.3, P.v1 est la première version du processus P, P.v2 est dérivée comme alternative de P.v1 alors que P.v3 est dérivée comme évolution de P.v1.

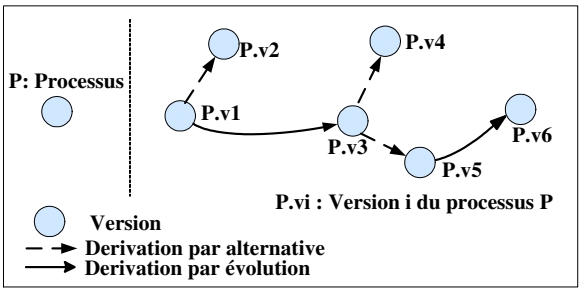


Figure II.3. Hiérarchie de dérivation des versions d'un processus

II.2. Versions et Flexibilité : Etat de l'art

Des travaux de l'état de l'art, tels que ceux présentés dans [Weber *et al.,* 2013], [Ekanayake *et al.,* 2011], [Zhao et Liu, 2013] et [Chaâbane, 2012], ont traité de la problématique de la gestion des processus flexibles en utilisant la technique de versionnement.

II.2.1. Les travaux de Weber *et al.*

Les travaux de [Weber et *al.,* 2008] [Weber *et al.,* 2013] ont défini des patrons récapitulant toutes les modifications qu'on peut apporter au schéma d'un processus. Les patrons proposés représentent des opérations de haut niveau utilisables pour traiter les besoins d'adaptation et d'incomplétude des processus flexibles. Ce travail est important bien qu'il ne traite pas le problème de la modélisation des versions de

processus, il sert de support dans nos contributions et dans les contributions traitant de l'adaptation des schémas des processus. Des exemples d'utilisation de ces patrons sont présentés dans la Figure II.4. Plus précisément, Weber et al. ont proposé d'utiliser les patrons d'adaptation pour les raisons suivantes :

- Faire évoluer le schéma d'un processus. La première partie de la Figure II.4 illustre un exemple d'évolution d'un schéma de processus S en un nouveau schéma S', et ceci en insérant deux nouvelles activités. La modification du schéma de processus a une répercussion sur les instances (de ce schéma) qui sont en cours d'exécution. Par conséquent, des solutions de migration ont été proposées pour modifier les instances en cours d'exécution conformément au nouveau schéma. Dans l'exemple présenté dans la Figure II.4, les instances I1 et I2 peuvent tenir compte des changements du schéma puisque l'activité B n'est pas accomplie alors que l'instance I3 ne peut pas supporter ces changements puisque l'exécution de l'activité B est achevée.

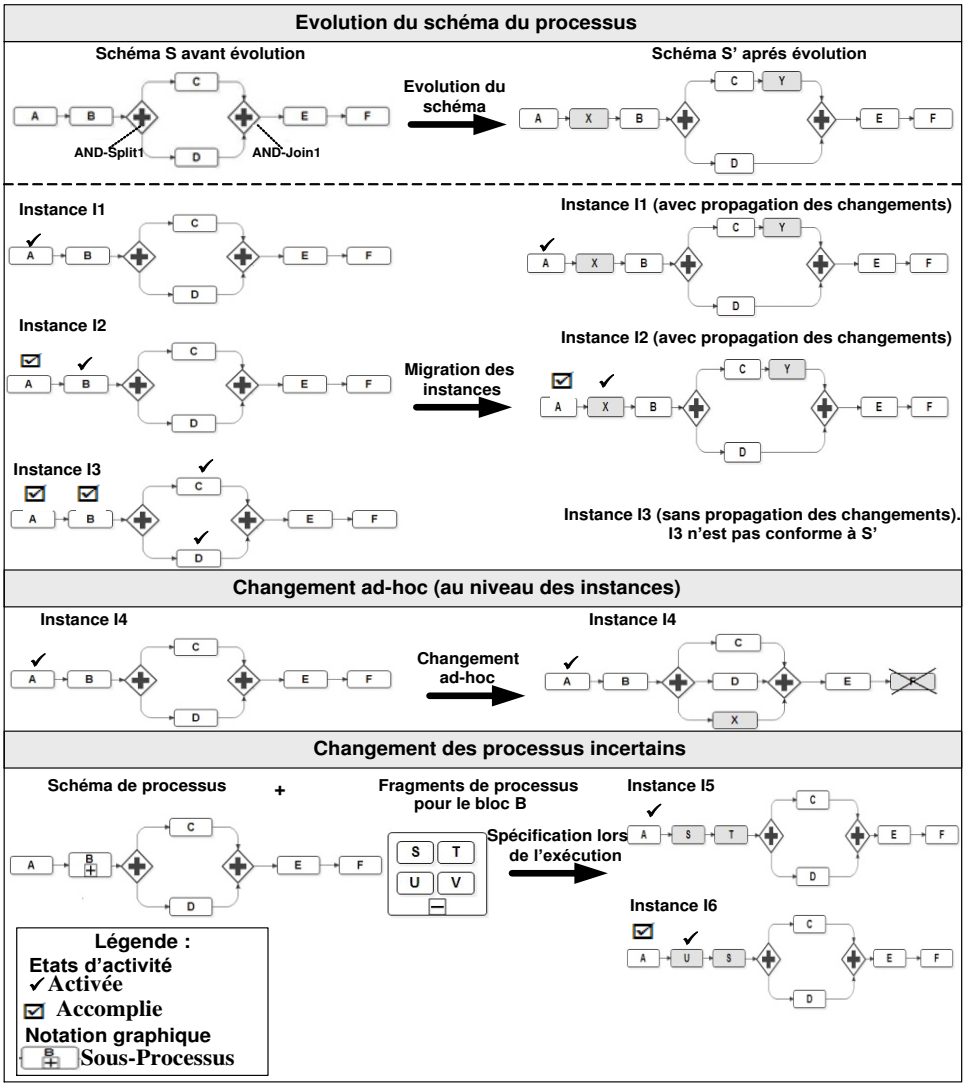


Figure II.4. Exemples d'utilisation des patrons d'adapation [Weber *et al.,* 2013]

- Gérer les changements ad-hoc. Les changements ad-hoc permettent de traiter les cas particuliers, *i.e.,* les exceptions non prévues qui apparaissent au moment de

l'exécution. Dans ce cas, il s'agit d'adapter l'instance en cours d'exécution sans toucher le schéma du processus. Dans la deuxième partie de la Figure II.4, le patron d'adaptation est appliqué pour modifier l'instance I4 en ajoutant la tâche X en parallèle entre C et D et en supprimant la tâche F. Ces modifications n'ont aucune répercussion sur le schéma du processus.

- Gérer des processus incertains dont le schéma ne peut être complété que lors de l'exécution. Dans la troisième partie de la Figure II.4, le schéma du processus est complété au moment de l'exécution de deux manières différentes (les instances I5 et I6).

Les patrons d'adaptation proposés dans les travaux de Weber sont classés selon deux catégories :

1- Les patrons d'adaptation permettant de transformer le schéma et / ou les instances de processus. Dans ce cas, Weber *et al.,* proposent quatorze patrons assurant des changements structurels des processus. Parmi ces patrons, nous présentons en Figure II.5 ceux qui permettent l'adjonction, la suppression, le déplacement, le replacement et l'échange d'un fragment de processus.
2- Les patrons de changement dans une région précise ne permettant pas de modifier la structure de la définition du processus. Ils offrent aux concepteurs, lors de l'exécution, un moyen pour terminer les parties incomplètes de la définition du processus.

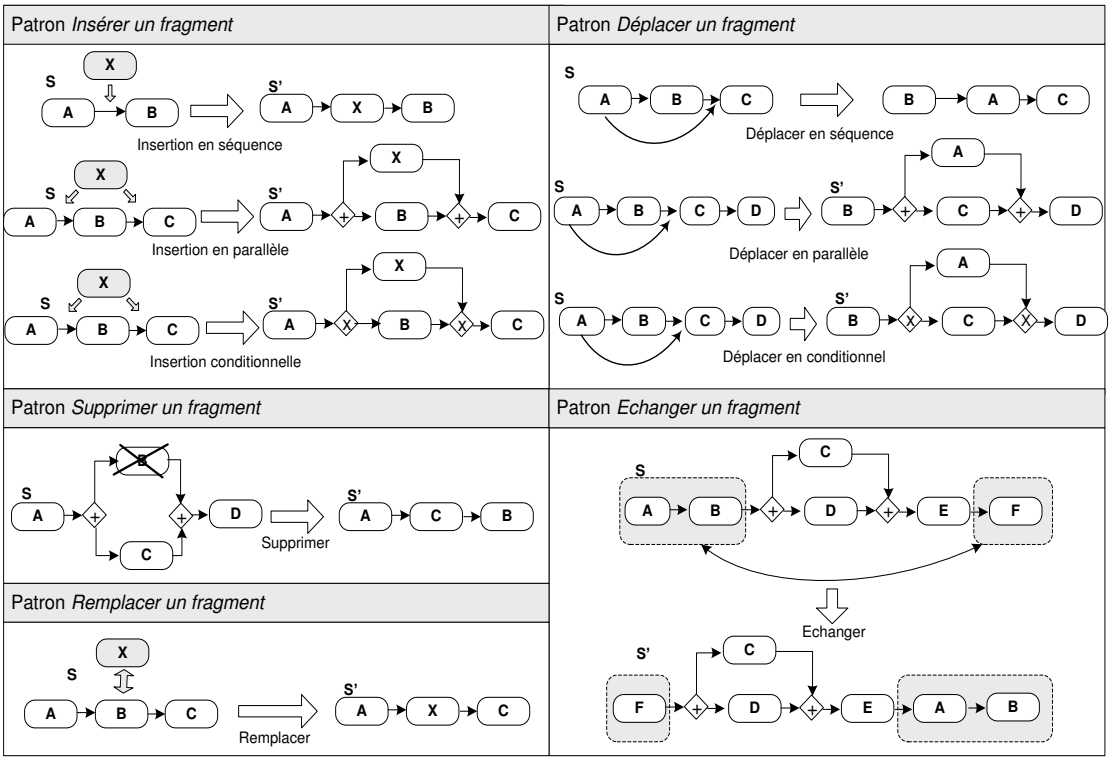


Figure II.5. Exprait des patrons d'adapation [Weber *et al.,* 2013]

Les patrons proposés dans [Weber *et al.,* 2013] couvrent la plupart des cas de changement de processus, mais ils ne traitent que la perspective comportementale et ne

considèrent pas les perspectives informationnelle, organisationnelle et contextuelle des processus.

II.2.2. Travaux de Ekanayake *et al.*

[Ekanayake *et al.,* 2011] proposent un modèle de versionnement et une structure de stockage des versions d'un processus inspirée des outils utilisés pour le versionnement dans le domaine du Génie Logiciel. Ce modèle, présenté en Figure II.6, est composé de plusieurs branches. Chaque branche représente un processus et comporte une ou plusieurs versions. Une version, à l'exception de la version initiale, est dérivée à partir d'une version antérieure appartenant à la même branche ou à une branche différente. La Figure II.6(a) illustre trois branches correspondant respectivement aux processus P1, P2 et P3. Le processus P1 comporte trois versions 1.0, 1.1 et 1.2 reliées par un lien de dérivation. La version 1.0 du processus P2 est dérivée à partir de la version 1.0 du processus P1, alors que la version 1.0 du processus P3 est dérivée à partir de la version 1.1 du processus P1. Les versions marquées par des cercles gris correspondent aux versions courantes de chaque branche.

L'idée de base du modèle de versionnement proposé est de décomposer chaque processus en fragments SESE (Single Entry Single Exit). Le processus décomposé est par la suite représenté sous forme d'une arborescence nommée RPST (Refined Process Structure Tree) composée par une hiérarchie de fragments SESE. Les auteurs de [Ekanayake *et al.,* 2011] traitent des changements d'un processus au travers du versionnement de ses fragments. Chaque fragment peut avoir une série de versions dont la dernière correspond à la version courante. Dans un RPST, chaque fragment (représenté dans un rectangle) a un numéro de version (représenté dans un cercle). La Figure II.6 (b) montre trois RPST correspondant aux versions 1.0 et 1.1 du processus P1 et à la version 1.0 du processus P2. Par exemple, la version 1.0 du processus P1 comporte la première version du fragment (F1) alors que la version 1.1 de ce processus comporte la deuxième version de ce fragment.

Les versions des processus peuvent avoir un ou plusieurs fragments en commun. Afin d'éviter les problèmes de redondance et de prolifération de versions, les auteurs de [Ekanayake *et al.,* 2011] proposent une solution de partage à un niveau vertical et à un niveau horizontal :

- Le niveau vertical : Les versions d'un même processus partagent un ou plusieurs fragments. Par exemple, dans la Figure II.6(b) les versions 1.0 et 1.1 du processus P1 partagent la première version du fragment F4 et les premières versions des fragments F5, F8, F9 et F10.
- Le niveau horizontal : Dans ce type de partage, deux versions appartenant à deux processus (*i.e.,* branches) différents partagent un certain nombre de fragments. Par exemple, dans la Figure II.6(b) la version 1.0 du processus P1 et la version 1.0 du processus P2 partagent la première version du fragment F3 (composée par les premières versions des fragments F6 et F7).

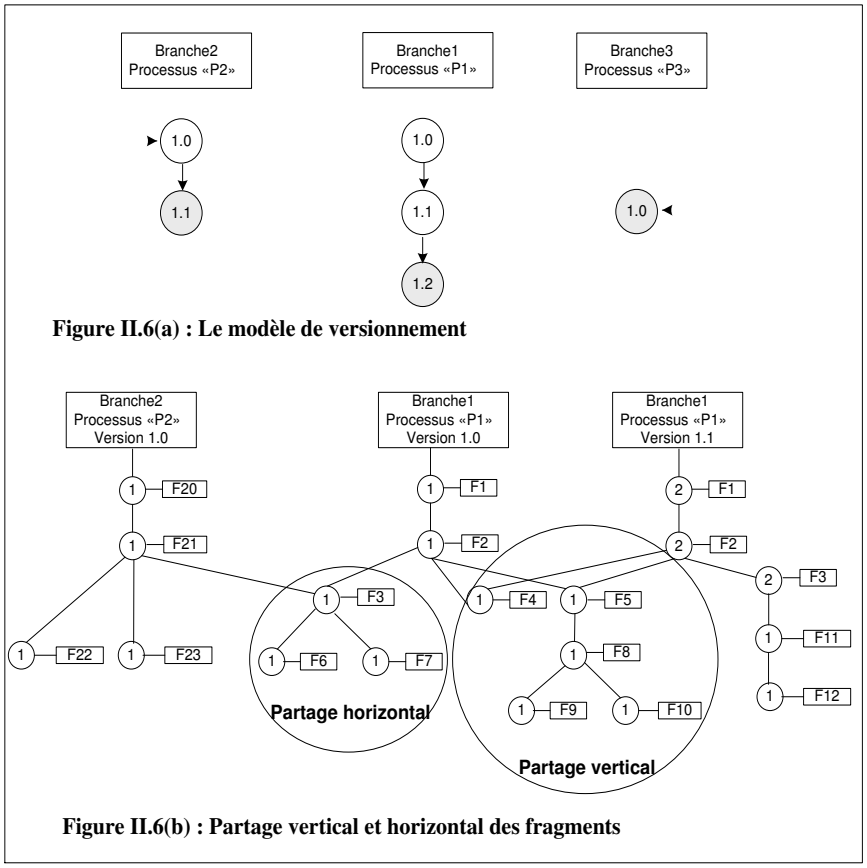


Figure II.6. Le modèle de versionnement de [Ekanayake *et al.,* 2011]

[Ekanayake *et al.,* 2011] proposent aussi une structure de stockage des versions de processus issues du modèle de versionnement. Cette structure permet la sauvegarde des processus (*i.e.,* branches), des fragments et de leurs différentes versions dans une base de données relationnelle. Les auteurs présentent aussi un ensemble d'algorithmes qui permettent la manipulation de la structure de stockage proposée. Plus particulièrement, ces algorithmes permettent l'insertion d'un nouveau processus, l'insertion d'un nouveau fragment, la recherche d'un fragment et la mise à jour d'un processus et de ses fragments.

La technique proposée possède les avantages suivants :

- elle garde trace des différents processus d'une organisation sous forme de fragments partagés horizontalement (*i.e.,* entre les différents processus) et verticalement (*i.e.,* entre les différentes versions de processus) ;
- elle favorise la propagation des changements. Par exemple si une erreur est détectée dans un fragment, la réparation concerne tous les processus contenant ce fragment ;
- elle favorise l'accès concurrent au schéma d'un processus par différents concepteurs. En effet, dans le cas où un concepteur a besoin de modifier le schéma d'un processus, il suffit de verrouiller uniquement les fragments concernés par la modification et non pas la totalité du processus.

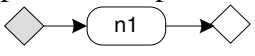
[Ekanayake *et al.,* 2011] adoptent le paradigme orienté activité pour la description des processus auxquels est appliquée la technique de versionnement. Le modèle proposé dans [Ekanayake *et al.,* 2011] prend en compte la perspective comportementale, mais ne traite pas la perspective informationnelle, la perspective organisationnelle et la perspective contextuelle. De plus, ce travail ne se base que sur des processus bien structurés (well-structured process) qui peuvent être composés en fragments SESE.

II.2.3. Travaux de Zhao et Liu

[Zhao et Liu, 2007] ont tout d'abord défini les exigences nécessaires pour qu'un système de gestion de processus puisse supporter la gestion des versions de processus. Ces exigences sont les suivantes :

- Représentation des versions : ce qui permet d'offrir des outils pour décrire la définition des versions de processus ainsi que leur évolution.
- Transformation des versions : ce qui permet d'assurer la migration des instances en cours sans être obligé d'arrêter leur exécution et de les reprendre dès le début.
- Exécution simultanée des versions : ce qui permet à plusieurs instances de processus issues de différentes versions d'être exécutées en même temps.
- Extraction des versions : ce qui permet de retrouver une version parmi plusieurs, c'est-à-dire que le système doit offrir des outils permettant de sélectionner une version. Cette exigence est importante car elle favorise la réutilisation des versions.

Cependant, Zhao et Liu n'abordent dans [Zhao et Liu, 2007] et [Zhao et Liu, 2013] que les problèmes de représentation et de transformation des versions. En effet, ils traitent du problème de l'évolution des schémas des processus en représentant les différentes versions dans un seul graphe. Plus précisément, les auteurs proposent de représenter le schéma d'une version de processus (*i.e.,* les activités et leur coordination) sous forme d'un graphe orienté, appelé VPG (Versionned Preserving directed Graph). Un VPG permet alors de capturer les évolutions du schéma d'un processus en modélisant dans un même graphe plusieurs versions pour éviter le problème de la prolifération des versions ; ainsi lors d'un changement du schéma de processus, on ne modifie qu'une partie du graphe, le reste du graphe (*i.e.,* la partie commune entre les deux versions) n'étant pas modifié.

Un VPG est composé d'un ensemble de nœuds, d'arcs et de relations d'exclusivité. Un nœud appartenant à un VPG peut représenter soit une activité du processus, soit un Gateway. Chaque nœud possède un point de pré-exécution et un point de post-exécution représentés sous forme de losanges comme suit : 

. Ces points décrivent les pré- et post-conditions relatives à l'exécution du nœud. Les arcs d'un VPG montrent le schéma de coordination des nœuds. Les arcs sont étiquetés par les identifiants des versions. Une relation

d'exclusivité définie entre deux arcs d'un VPG permet de distinguer le flux de deux versions de processus différentes.

La Figure II.7(b) illustre les trois versions de processus présentées en II.7(a). Après l'exécution de l'activité 1, représentée par le nœud n1, trois chemins alternatifs se présentent :

1. l'activité 2 (nœud n2) pour réaliser la version de processus V0,
2. l'activité 2 (nœud n2) en parallèle avec l'activité 5 (nœud n5) pour réaliser la version V1,
3. l'activité 5 (nœud n5) en parallèle avec l'activité 6 (nœud n6) pour réaliser la version V1.1. Dans cette version, l'activité 2 (nœud n2) est remplacée par l'activité 6 (nœud 6). Ce fait est représenté par une relation d'exclusivité modélisée dans la Figure II.7(b) par le symbole suivant :

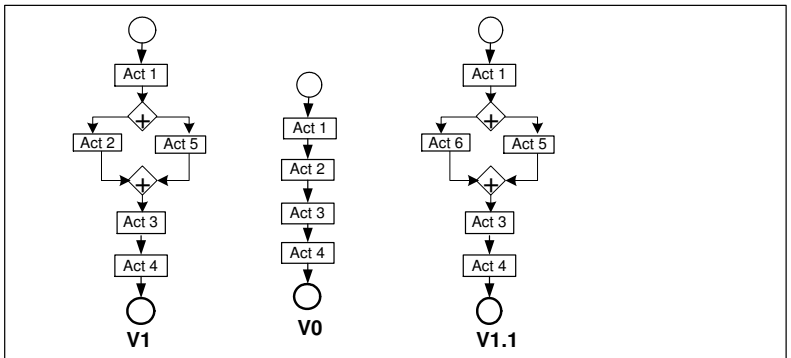


Figure II.7(a) : Description des schémas des versions du processus

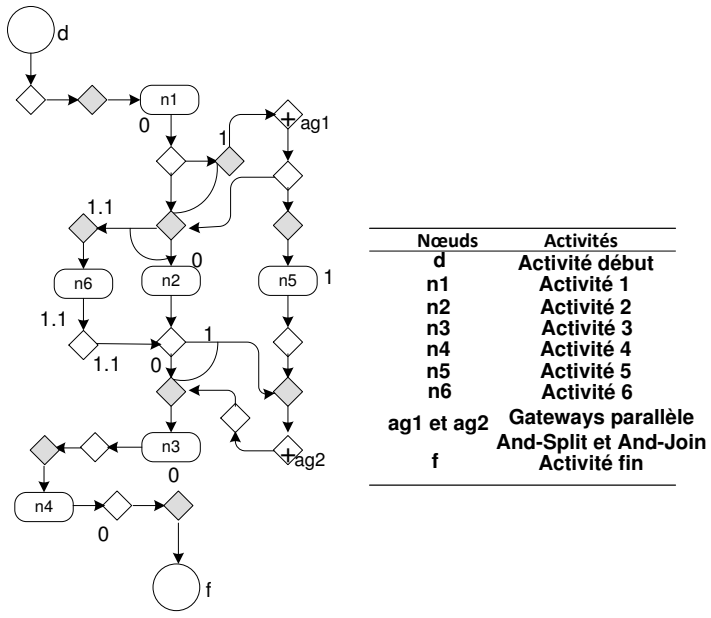


Figure II.7(b) : Illustration des versions à l'aide du VPG

Figure II.7. Représentation des schémas de version de processus [Zhao et Liu, 2013]

Quant au problème de transformation de versions, Zhao et Liu [2013] ont traité à la fois l'adaptation du VPG (*i.e.,* le schéma des versions du processus) et les instances

en cours d'exécution. Pour ce faire, ils se servent des patrons d'adaptation proposés dans [Weber *et al.,* 2013] pour adapter un VPG. En effet, un VPG peut être modifié en insérant, supprimant, remplaçant, déplaçant et échangeant des fragments. La Figure II.8 illustre des exemples de modification d'un VPG à l'aide des patrons d'adaptation.

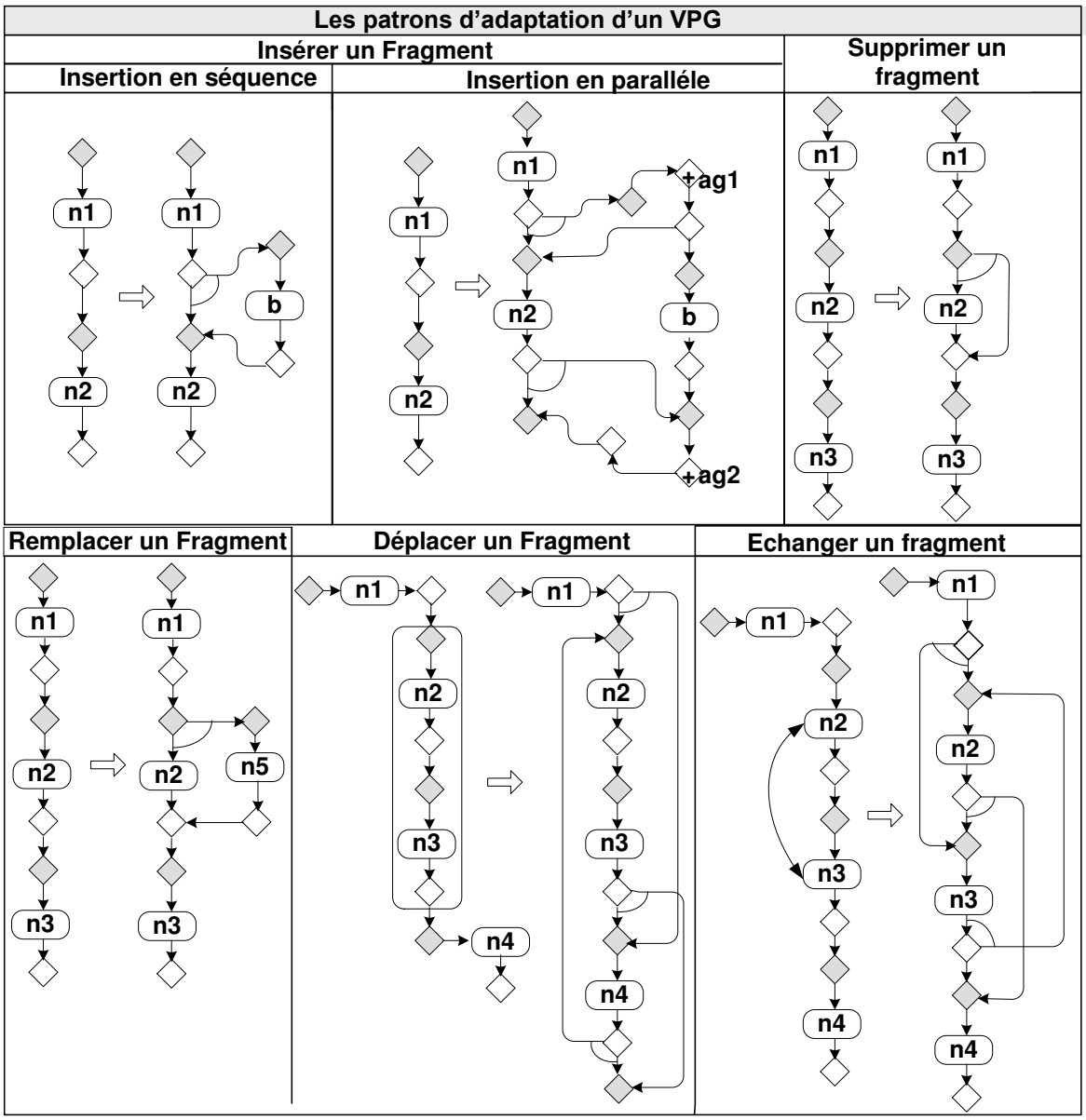


Figure II.8. Transformation d'un VPG à l'aide des patrons d'adaptation [Zhao et Liu, 2013]

Par exemple, pour insérer en séquence le nœud (b) dans le schéma décrit par la séquence d'activités n1→ n2, îı faut ajouter une relation d'exclusivité entre le point de post-exécution de n1 et le point de pré-exécution de n2. Ainsi on obtient le schéma suivant n1→ b→n2.

Zhao et Liu ont réussi à satisfaire deux exigences : la représentation des versions et la transformation des versions. En effet, le VPG proposé a permis de représenter dans un seul graphe toutes les versions d'un processus. De plus, les patrons proposés permettent d'assurer la modification du graphe et la migration d'une version à une

autre. Cependant, l'augmentation du nombre de versions d'un processus rend le graphe de plus en plus complexe (difficile à interpréter), ce qui rend l'extraction des versions difficile. En outre, Zhao et Liu n'abordent pas le problème de la compatibilité des versions. De plus, ils ne considèrent également qu'une seule perspective, qui est la perspective comportementale (décrivant les activités d'un processus et leur enchaînement) et n'accordent pas d'attention aux autres perspectives.

II.2.4. Les Travaux de Chaâbane

[Chaâbane, 2012] a défini un cadre conceptuel multi-dimensionnel et multi-versionnable pour assister les concepteurs à modéliser des processus intra-organisationnels flexibles. Plus précisément, dans ces travaux ces auteurs ont proposé un méta-modèle complet qui prend en compte les trois perspectives définies dans l'état de l'art du domaine, tout en adoptant une approche basée sur les versions. Le méta-modèle proposé, présenté en Figure II.9, est appelé VBP2M (Versions of Business Process Meta-Model). Ce méta-modèle comporte six classes versionnables permettant de garder trace des changements relatifs au travail réalisé dans un processus et du mode d'organisation relatif au travail en question. Ces classes sont les suivantes : Version de Processus, Version d'activité atomique, Version d'opération, Version de Rôle et Version d'unité organisationnelle. En effet, ces auteurs pensent que la gestion des versions des processus, des activités atomiques et des opérations, permet aux organisations de faire face aux modifications touchant les structures des processus afin de suivre les changements de leur environnement opérationnel ou dans la manière d'organiser le travail. De plus, ils pensent qu'il est intéressant de suivre l'évolution des informations manipulées par chaque version du processus, et d'historiser les changements que peut avoir un rôle ou une unité organisationnelle.

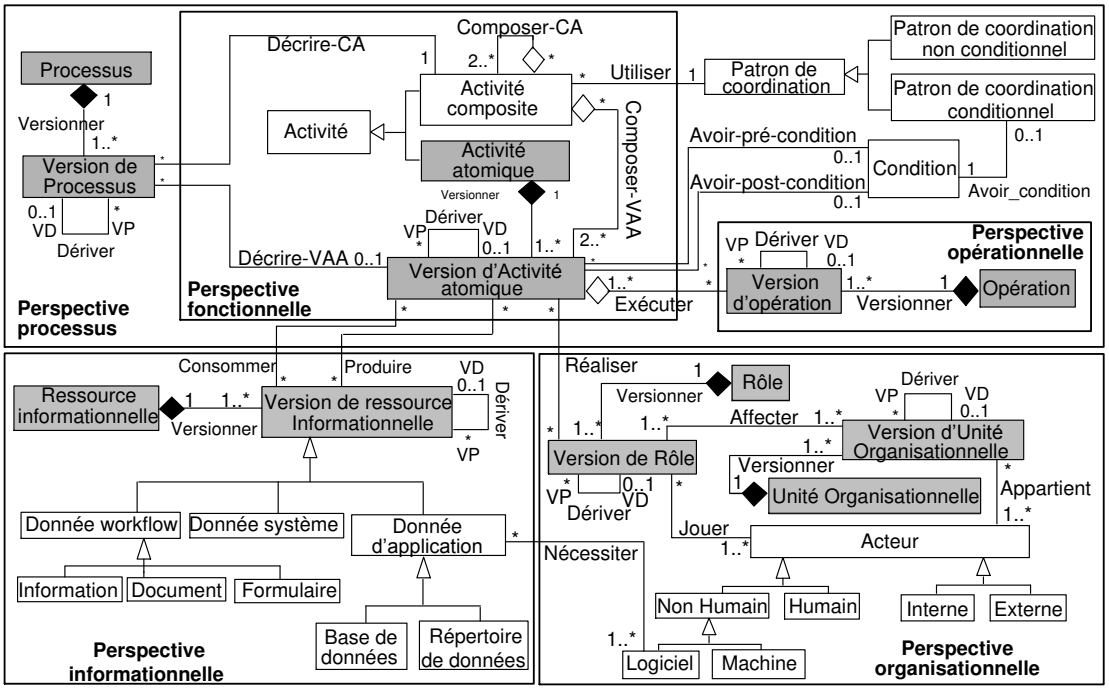


Figure II.9. Le méta-modèle VP2M [Chaâbane, 2012]

Outre le problème de la modélisation des processus flexibles, les travaux de ces auteurs ont traité les problèmes relatifs à la gestion des versions des processus définies conformément au méta-modèle VBP2M. Dans ce cadre, on note la proposition d'un langage VBPQL (*Versioned Business Process Query Language*) permettant la définition, la manipulation et l'interrogation des versions de processus.

II.3. Constat : Comparaison des travaux

Le tableau II.1 présente une étude comparative des travaux ci-dessus présentés. A cet effet, nous avons fixé des critères de comparaison afin d'évaluer les capacités de ces travaux à prendre en compte des processus intra- et inter-organisationnels flexibles. Ces critères sont les suivants :

- *Satisfaction des besoins de Flexibilité* : ce critère exige la prise en compte des besoins de flexibilité énoncés dans la taxonomie de Reichert. En effet, une solution assurant la flexibilité des processus doit traiter les changements prévisibles qui apparaissent suite à un besoin de variabilité, à un besoin d'évolution ou à un besoin d'adaptation (lors du traitement des exceptions anticipées). En outre, le traitement de la flexibilité doit aussi considérer les changements imprévisibles que peut subir un processus suite à un besoin d'adaptation non planifiée ou à une définition incomplète du processus.

- *Perspectives* : ce critère exige un modèle de processus permettant de prendre en compte les changements qui touchent les principales perspectives de modélisation des processus à savoir, *i.e.,* la perspective comportementale, la perspective informationnelle, la perspective organisationnelle et la perspective contextuelle.

- *Type de processus :* ce critère prend en compte à la fois les processus intra- et inter-organisationnels.

- *Traçabilité de l'évolution* : ce critère indique qu'il faut garder trace des changements que peut avoir un processus. En effet, il est important de considérer l'historique d'évolution des processus pour favoriser leurs réutilisations.

- *Utilisation d'un standard* : ce critère exige la modélisation des versions de processus conformément à une notation standard connue par la majorité des utilisateurs de processus (concepteurs, développeurs, etc.).

Dans le tableau II.1, nous avons utilisé les signes : (+) si le critère est satisfait, (+/-) si le critère est satisfait partiellement et (-) si le critère n'est pas satisfait.

Tableau II-1. Tableau comparatif des travaux de l'état de l'art utilisant la technique de versionnement

Critères		**Modèle de versionnement** [Ekanayake *et al.*, 2011]	**VPG** [Zhao et Liu, 2013]	**Patron d'adaptation** [Weber *et al.*, 2013]	**VBP2M** [Chaâbane, 2012]
Besoins de flexibilité	Variabilité	(-)	(-)	(-)	(+)
	Adaptation	(-)	(-)	(+)	(+/-)
	Incomplétude	(-)	(-)	(+/-)	(-)
	Evolution	(+)	(+)	(+)	(+)
Perspectives	Comportementale	(+)	(+)	(+)	(+)
	Informationnelle	(-)	(-)	(-)	(+)
	Organisationnelle	(-)	(-)	(-)	(+)
	Contextuelle	(-)	(-)	(-)	(+/-)
Type de Processus	Processus intra-organisationnels	(+)	(+)	(+)	(+)
	Processus inter-organisationnels	(-)	(-)	(-)	(-)
Traçabilité de l'évolution		(+)	(+)	(-)	(+)
Utilisation d'un standard		(+/-)	(-)	(+)	(-)

D'après le tableau II-1, nous pouvons constater que les travaux étudiés considèrent partiellement les besoins de flexibilité de la taxonomie de Reichert. Si, tous les travaux prennent en considération le besoin d'évolution, les besoins de variabilité et d'adaptation sont partiellement pris en compte dans les travaux de Weber *et al.* et de Chaâbane et le besoin de l'incomplétude n'est supporté que dans le travail de Weber *et al.* Quant à la prise en compte des perspectives, tous les travaux étudiés sont capables de prendre en compte les changements de la perspective comportementale. Mais les trois perspectives informationnelle, organisationnelle et contextuelle ne sont supportées que par les travaux de Chaâbane. Pour les types de processus, tous les travaux concernent évidemment les processus intra-organisationnels, mais aucun n'a traité des processus inter-organisationnels. En ce qui concerne le critère de traçabilité de l'évolution, permettant de favoriser la réutilisation des définitions, il est satisfait dans les travaux de Ekanayake *et al.*, de Zhao et Liu et de Chaâbane. Finalement, seuls les travaux de Ekanayake *et al.* et de Weber *et al.* se sont basés sur des standards.

A la lumière de cette étude comparative, nous pouvons constater que les travaux de Chaâbane. répondent mieux aux critères que nous avons fixés. Cependant, ces travaux proposent leur propre méta-modèle sans pouvoir profiter des standards de modélisation des processus, ce qui réduit leur acceptabilité auprès des utilisateurs BPM. C'est pour cette raison que nous avons choisi de nous inspirer de ces travaux pour étendre BPMN, qui est reconnu comme une notation standard adaptée aux

concepteurs de processus, afin de supporter la flexibilité des processus intra- et inter-organisationnels.

III. Le contexte des processus métier

Après cette revue des travaux de l'état de l'art traitant du problème de la modélisation des processus flexibles en adoptant une approche basée sur les versions, cette section détaille la notion de contexte qui permet de spécifier le pourquoi des (versions de) processus. En fait, au travers de la spécification du contexte, nous pouvons répondre aux questions suivantes : pourquoi une version (de processus, d'activité, de rôle, ...) a-t-elle été définie ? dans quelle situation une version peut être utilisée ?

Cette section présente tout d'abord des définitions de la notion de contexte. Elle détaille ensuite la taxonomie la plus complète de l'état de l'art pour la classification des informations contextuelles. Finalement, elle explique et discute des techniques de modélisation du contexte.

III.1. Définitions

La notion de contexte est apparue dans de nombreuses disciplines, tels que l'intelligence artificielle, le traitement des langues naturelles et la recherche intelligente d'informations. De nombreux travaux issus de ces communautés de recherche tentent de donner une définition de ce qu'est un contexte. Toutefois, il n'existe pas une définition à la fois générique et pragmatique de cette notion. En effet, la majorité des propositions de définition sont soit très abstraites, soit très spécifiques à un domaine particulier. Nous donnons dans ce qui suit quelques définitions issues de la littérature.

Définition II.5 [Dey, 2001] :

> *Le contexte se définit comme toute information qui caractérise la situation d'une entité ; une entité étant une personne, un lieu ou un objet considéré comme pertinent relativement à une interaction entre un utilisateur et une application, incluant l'utilisateur et l'application eux-mêmes.*

Cette définition couvre le contexte des différents acteurs d'un système d'information. Cependant, elle est très générique puisque elle trace un espace infini et illimité de ce qui fait partie du contexte.

Définition II.6 [Rosemann *et al.*, 2008] :

> *Le minimum d'informations contenant tous les renseignements pertinents qui influent sur la modélisation et l'exécution d'un processus.*

Cette définition est spécifique au domaine des processus métier. Elle prend en compte le contexte durant les phases du cycle de vie du processus (*i.e.,* la phase de conception et la phase d'exécution). Cependant, elle n'indique pas quels types d'information doivent être décrits dans un contexte. Cette définition a été enrichie dans les travaux de [Chaâbane, 2012].

Définition II.7 [Chaâbane, 2012] :

> *L'ensemble des informations quantitatives et qualitatives qui caractérisent un processus et influent sur sa modélisation et son exécution.*

Cette définition nous semble plus complète puisqu'elle spécifie les types d'informations contextuelles et elle couvre les phases du cycle de vie d'un processus.

III.2. Taxonomie des informations contextuelles

Plusieurs taxonomies ont été proposées pour catégoriser les informations contextuelles des processus. Parmi celles examinées dans l'état de l'art, nous mettons en avant celle de Rosemann [Rosemann *et al.,* 2008], qui est la plus complète. Cette taxonomie classifie les informations de contexte en quatre types, à savoir le contexte immédiat, le contexte interne, le contexte externe et le contexte environnemental. La Figure II.10 illustre cette taxonomie.

- Le contexte immédiat touche directement le côté opérant du processus. Les éléments du contexte immédiat sont reliés essentiellement à l'aspect fonctionnel du processus, à savoir les activités, les opérations, les ressources informationnelles et les ressources organisationnelles.
- Le contexte interne couvre les stratégies et les objectifs à atteindre dans un processus.
- Le contexte externe capture les éléments qui sont au-delà de la sphère de l'organisation. Bien que ce contexte ne soit pas à proximité immédiate de l'organisation, il a un impact sur ses processus. Ce contexte contient des informations sur les clients, les collaborations, les messages et les profils des partenaires.
- Le contexte environnemental est la couche la plus externe. Il capture l'environnement global dans lequel l'organisation opère, tels que les facteurs économiques (taux de change, prix pétrole) et les facteurs climatiques (Séisme).

III.3. Techniques de modélisation du contexte des processus métiers

Dans cette section, nous détaillons les trois principales techniques pour modéliser le contexte : paires/triplets, orientée modèle et orientée ontologie.

III.3.1. La technique paires/triplets

La technique paires/triplets consiste à décrire le contexte sous forme de paires (attribut, valeur), où l'attribut désigne une information contextuelle et la valeur représente la valeur courante affectée à cette information. Les travaux de Dey [Dey, 2000] présentent le contexte sous forme d'un ensemble de paires. Pour la mise en œuvre de cette technique, [Dey, 2000] utilise un serveur d'environnement dynamique capable d'observer l'environnement d'utilisation d'une application. Plus précisément, ce serveur capte les changements des valeurs des informations décrivant le contexte.

Quant aux auteurs de [Schmidt *et al.,* 1999], ils ont introduit la dimension du floue lors de la description d'un contexte. En effet, ils proposent d'ajouter une pondération pour chaque paire. D'où, le contexte est défini selon [Schmidt *et al.,* 1999] par un triplet (attribut, valeur, degré de certitude). Par exemple, le triplet (Distance, proche, 95) qualifie une distance comme proche avec un degré de certitude égal à 95%.

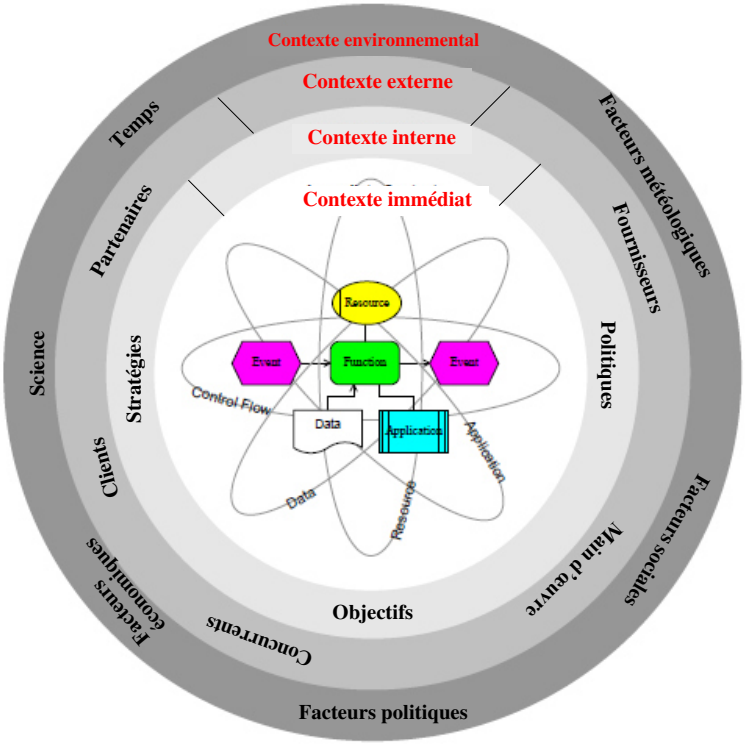


Figure II.10. Taxonomie de Rosemann [Rosemann *et al.,* 2008]

III.3.2. La technique orientée ontologie

La technique orientée ontologie est une technique formelle qui tire profit des caractéristiques des ontologies pour modéliser le contexte. Les modèles d'ontologie sont utilisés dans différents domaines. Ils ont été utilisés pour détecter des problèmes dans la sémantique des processus [Fellmann *et al.,* 2011] ou de similarités entre les modèles de processus [Makni *et al.,* 2012]. Ils ont aussi été utilisés pour la représentation et l'adaptation du contexte dans le domaine des services web [Boukadi, 2009] et de l'informatique pervasive et mobile [Saleemi *et al.,* 2011].

Les travaux de [Boukadi, 2009] se situent dans le domaine des services web. Ils proposent une ontologie de catégorisation statique, comparable à un diagramme de classes. Cette ontologie offre un dictionnaire détaillé des informations contextuelles pouvant influencer le comportement d'un processus. Ces informations sont structurées à l'aide des relations de type "est-un". L'auteur a défini trois catégories pour le contexte : le contexte environnemental (tels que le temps et la localisation), le contexte fonctionnel (telle que l'expérience d'un acteur) et le contexte non fonctionnel (telles que la sécurité et la qualité). Ces catégories de contexte permettent de percevoir la situation d'un processus métier dans son environnement.

Les travaux de [Saidani *et al.,* 2015] ont proposé une approche pour la modélisation du contexte des processus dans le but de l'adaptation du processus selon le contexte. En effet, les auteurs ont proposé un méta-modèle qui spécifie les concepts utilisés par les informations contextuelles ainsi que les relations entre ces concepts. Quant au niveau modèle, ils ont proposé un modèle basé sur les ontologies, qui permet de faciliter le partage des connaissances et d'effectuer des raisonnements sur les informations contextuelles afin de déduire de nouvelles informations. Ce modèle distingue deux niveaux : un niveau générique et un niveau spécifique. Le niveau générique ne dépend pas d'un domaine particulier. Il est représenté par une ontologie de haut niveau qui décrit les caractéristiques générales du contexte d'une entité. Le niveau spécifique est relatif à un domaine particulier. Il est exprimé au travers d'une ontologie de domaine.

III.3.3. La technique orientée modèle

La technique orientée modèle utilise des modèles formels regroupant les concepts des processus afin de spécifier les informations de contexte. Elle offre aussi des méta-modèles de description qui peuvent être réutilisés par plusieurs applications. Deux types de contexte découlent de cette technique : le contexte implicite, qui est donné par les relations entre les concepts du modèle, et le contexte explicite, qui consiste à décrire le contexte via des métriques (dites aussi paramètres).

Les travaux de [Korherr *et al.,* 2007] proposent un méta-modèle générique basé sur les perspectives suivantes : perspective comportementale, perspective informationnelle, perspective organisationnelle et perspective contextuelle. La perspective contextuelle proposée considère les mesures Qualité, Coût et Temps pour modéliser les objectifs des processus. Les auteurs évaluent par la suite les langages de modélisation existants, tels que le diagramme d'activité (DA) d'UML, EPC, BPMN, Petri Net, en les comparant avec le méta-modèle proposé. Le résultat de l'évaluation montre que les différents langages étudiés ne supportent pas la perspective contextuelle. Les auteurs proposent alors l'intégration de cette perspective dans les méta-modèles des langages étudiés.

Les travaux de [Saidani et Nurcan, 2009] ont utilisé une approche basée sur les buts afin d'assurer la flexibilité, tout en se focalisant sur les rôles. Ils ont proposé un modèle appelé RBPM (Role driven approach for modeling flexible BPs). Cette approche consiste à utiliser des connaissances contextuelles afin d'améliorer la cohérence des activités. Le contexte a été spécifié d'une manière implicite. Les auteurs décrivent le contexte par des relations qui existent entre des entités, telle que la relation "Can play" entre les deux entités "Actor" et "Role". Par exemple, si l'acteur "Ali" joue le rôle d'un "assistant" et il possède une bonne expérience, il peut être affecté au rôle "directeur" si jamais les acteurs jouant le rôle "directeur" sont indisponibles. En fait, selon Saidani et Nurcan, cette affectation est reliée à un contexte spécifique.

Les travaux de Chaâbane [Chaâbane, 2012] ont traité de la modélisation du contexte des versions de processus en définissant une perspective contextuelle associée

aux autres perspectives du méta-modèle VBP2M. Cette perspective est présentée dans la Figure II.11. Dans cette perspective, chaque version de processus est associée à un contexte décrivant, via les paramètres fonctionnels et quantitatifs, l'utilisation de cette version de processus. Les paramètres fonctionnels portent sur les versions des activités atomiques et / ou sur les versions des rôles. En effet, selon Chaâbane le contexte d'une version de processus change lorsqu'il y a une modification fonctionnelle (c'est-à-dire les versions d'activités formant la version du processus) ou une modification organisationnelle (c'est-à-dire la version du rôle invoqué pour réaliser une version d'activité).

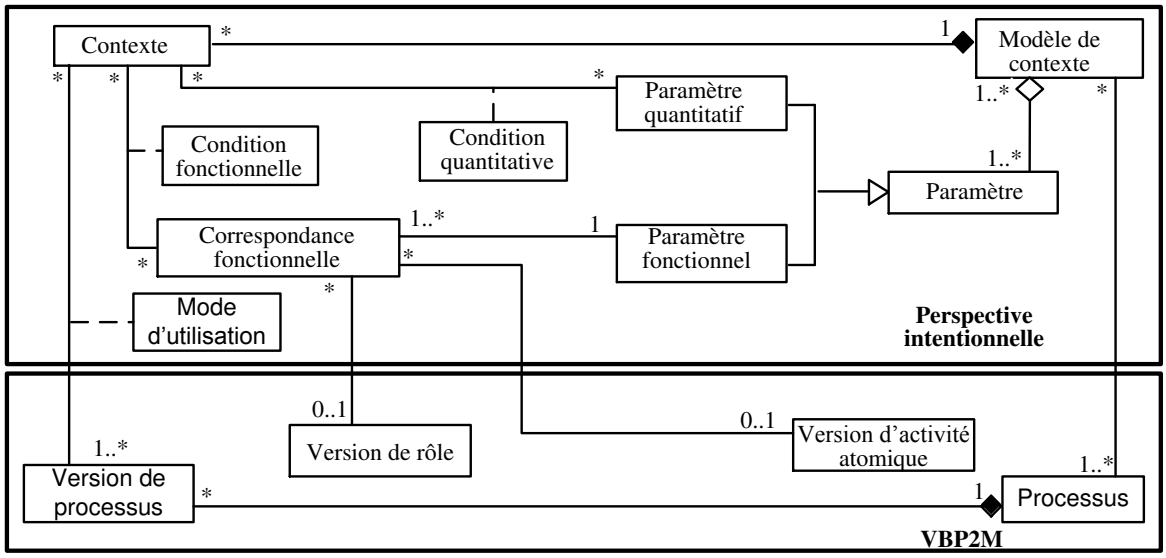


Figure II.11. Modélisation du contexte avec VP2M [Chaâbane, 2012]

III.3.4. Comparaison des techniques de modélisation du contexte

Nous présentons dans ce qui suit un tableau comparatif des techniques de modélisation des contextes des processus, détaillées précédemment. Cette comparaison est basée sur les critères suivants : (i) la simplicité et la facilité de l'implémentation de la technique, (ii) la spécification du type de l'information contextuelle, notamment en se basant sur les taxonomies existantes de l'état de l'art, telle que la taxonomie de Rosemann, et (iii) la spécification des relations entre les informations contextuelles afin de prendre en compte les dépendances entre ces informations.

Dans le tableau II.2, nous avons utilisé les signes : (+) si le critère est satisfait, (+/-) si le critère est satisfait partiellement et (-) si le critère n'est pas satisfait.

Tableau II-2. Tableau comparatif des techniques de modélisation du contexte des processus

Critères / Techniques	**Simplicité et facilité de mise en œuvre**	**Spécification du type de l'information contextuelle**	**Spécification de la relation entre les informations contextuelle**
Paires/Triplets	+	-	-
Orientée ontologies	-	+	+
Orientée modèle	+/-	+	+

D'après ce tableau, nous remarquons que la technique paires/triplets est la technique la plus simple à implémenter. Cependant cette technique est caractérisée par un faible niveau d'expression vu qu'elle n'offre pas une description complète du contexte. Cette technique permet ni l'expression des relations qui peuvent exister entre les informations de contexte, ni la spécification de leur type. Par exemple, dans un processus de fabrication, le contexte de l'activité « Production » est décrit comme suit : un mode de fonctionnement automatique est soit une qualité de production de niveau supérieur, soit une quantité de production supérieure à 100 unités. Ce contexte peut être exprimé par les trois paires suivantes (Mode de fonctionnement, Automatique), (Qualité demandée, supérieur) et (Quantité demandée, >100). De ce fait, en utilisant cette technique il n'est pas possible de définir pour chaque attribut le type de l'information contextuelle qu'il détient, l'opérateur et les relations sémantiques entre ces attributs.

Quant à la technique orientée ontologie, elle permet de décrire le contexte de manière plus riche que ce qui est permis par la technique paires/triplets. En effet, cette technique utilise des concepts ontologiques et des relations sémantiques pour spécifier le type des informations contextuelles et les relations entre ces informations. Cependant, cette technique est difficile à utiliser et à implémenter surtout qu'il n'y a pas un consensus pour la construction des ontologies et que chaque processus nécessite une ontologie décrivant le domaine.

Bien que la technique orientée ontologie semble utile pour résoudre les problèmes sémantiques dans un cadre inter-organisationnel, nous avons opté dans nos travaux pour l'utilisation de la technique orienté modèle afin de modéliser le contexte des processus intra-organisationnels. En effet, cette technique offre des modèles formels assez simples à instancier, permettant de décrire le contexte des processus. Ces modèles permettent la spécification des types des informations contextuelles et des

relations sémantiques entre ces informations. De plus, les modèles issus de la technique orientée modèle sont assez faciles à mettre en œuvre puisqu'ils n'exigent pas l'utilisation d'outils de développement compliqués. En effet, les instances des modèles peuvent être stockées dans une base de données relationnelle ou bien dans une base de données XML, ce qui permet une manipulation et une interrogation simples et rapides de ces instances.

IV. Problématique traitée

Cette thèse adresse le problème de la flexibilité des processus. Ce problème peut être vu sous deux angles complémentaires : (i) la flexibilité traitée durant la phase de conception, qui consiste à avoir des définitions de processus capables d'être modifiées pour suivre les changements de leur environnement d'exécution et (ii) la flexibilité traitée durant la phase d'exécution, qui consiste à proposer des solutions pour prendre en compte les changements non prévisibles que peuvent subir les instances en cours d'exécution, sans toutefois interrompre leur exécution. Dans le cadre de cette thèse, nous apportons des solutions au problème de la modélisation des changements que peut avoir un processus aussi bien en phase de conception qu'en phase d'exécution.

Les auteurs de [Reichert et Weber., 2012] ont défini une taxonomie récapitulant les besoins de flexibilité et les techniques assurant ces besoins. A la lumière de cette taxonomie, nous projetons d'utiliser une approche guidée par les activités en utilisant la technique de versionnement pour traiter le problème de la flexibilité. En effet, nous défendons l'utilisation des versions pour : (i) traiter la variabilité, et ceci en définissant plusieurs versions utilisées alternativement selon le contexte, (ii) traiter les adaptations planifiées en créant des versions décrivant les exceptions qui peuvent être prévues à l'avance et (iii) garder trace des évolutions du schéma d'un processus.

La majorité des travaux de l'état de l'art qui ont abordé le problème de la flexibilité de processus en adoptant un paradigme orienté activité et en utilisant la technique de versionnement se sont concentrés sur les activités et leur coordination (*i.e.,* la perspective comportementale). Mais ils ont accordé très peu ou pas d'attention aux trois autres perspectives (organisationnelle, informationnelle et contextuelle). En outre, peu de ces travaux se sont basés sur des notations standards dans les solutions proposées.

A travers cette thèse, nous souhaitons trouver des réponses aux questions relatives aux problèmes suivants :

1) Modélisation des versions des processus intra-organisationnels avec BPMN ; voir comment rendre le standard BPMN capable de supporter les versions des processus privés.

2) Modélisation du contexte des versions des processus intra-organisationnels. Il s'agit de proposer des solutions permettant d'aider au choix d'une version de processus dans une situation donnée.

3) Modélisation des versions des processus inter-organisationnels spécifiées soit comme des collaborations, soit comme des chorégraphies en BPMN ; chercher comment pouvoir modéliser des versions de collaborations et de chorégraphies.

4) Gestion des versions des processus intra- et inter-organisationnels ; voir comment pouvoir manipuler les versions des processus modélisées.

5) Implanter un outil permettant la modélisation et la gestion des versions des processus intra- et inter-organisationnels.

V. Conclusion

Ce chapitre a mis l'accent sur un problème primordial du BPM, celui relatif à la modélisation des versions des processus flexibles. Ce problème nous a induit un autre problème celui, de la spécification du contexte des processus pour décrire les situations incitant à la définition et à l'utilisation de ces versions.

Plus précisément, nous y avons tout d'abord présenté le problème général relatif au traitement des processus flexibles en donnant les définitions et les taxonomies les plus représentatives de la flexibilité dans l'état de l'art, ce qui nous a permis de mieux cerner ce problème. A partir de ces taxonomies, nous avons pu argumenter l'utilisation de la technique de versionnement pour traiter le problème de la flexibilité des processus. Nous avons par la suite détaillé les principaux travaux de l'état de l'art traitant du problème de la flexibilité des processus en utilisant les versions afin de dégager les avantages et les limites de ces travaux. La limitation de la plupart de ces travaux consiste au fait qu'ils ne considèrent que la perspective comportementale et qu'ils ne se basent pas sur des standards dans les solutions qu'ils proposent. De ce fait, nous visons, dans le cadre de cette thèse, d'utiliser la technique de versionnement pour aborder le problème de la flexibilité et de considérer les différentes perspectives de modélisation de processus (perspective comportementale, perspective organisationnelle, perspective informationnelle et perspective intentionnelle), en se basant sur le standard BPMN.

Dans ce chapitre, nous avons également évoqué le problème de modélisation du contexte de processus, qui consiste à trouver une solution pour la spécification du pourquoi une version est créée et dans quelles situations elle peut être utilisée. Pour ce faire, nous avons détaillé la taxonomie qui nous semble la plus complète pour classifier les informations contextuelles en différents types. Nous avons ensuite présenté une étude comparative des techniques de modélisation de contexte. Suite à cette étude, nous avons choisi d'utiliser la technique orientée modèle pour prendre en compte le contexte des versions des processus.

Dans le chapitre suivant, nous allons détailler notre première contribution qui consiste à proposer deux extensions à BPMN. La première extension a pour objectif de considérer les versions des processus intra-organisationnels dans BPMN, alors que la deuxième extension enrichit BPMN pour permettre la prise en compte du contexte des versions des processus.

Chapitre III – BPMN4V-PP pour la modélisation des versions des processus intra-organisationnels

Résumé: *Comme nous l'avons montré dans le chapitre précédant, la technique de versionnement est reconnue comme la technique la plus appropriée pour traiter le problème de la flexibilité des processus intra- et inter-organisationnels. D'un autre côté, BPMN est considéré comme le standard le plus répandu pour la modélisation des processus. Cependant, cette notation ne considère ni les processus flexibles, notamment à travers la modélisation des versions de processus, ni la spécification du contexte de ces versions de processus. Ce chapitre apporte des solutions pour étendre BPMN afin qu'il puisse supporter la modélisation des versions des processus intra-organisationnels. D'où la définition d'une nouvelle notation dédiée à la modélisation des versions des processus privés, nommée BPMN4V-PP (BPMN for versions of Private Process). Ce chapitre détaille, aussi, une deuxième extension de BPMN qui permet de considérer le contexte des versions de processus. Cette extension est nommée BPMN4V-Context.*

Sommaire du Chapitre III.

BPMN 2.0 est le standard recommandé par l'OMG pour la modélisation des processus métier intra- et inter- organisationnels. Il est largement utilisé dans les entreprises pour modéliser des processus comme une synchronisation d'activités et d'événements. Néanmoins, comme expliqué dans le chapitre 2, cette notation ne se prête pas à la modélisation des processus flexibles.

Ce chapitre apporte des solutions à la modélisation des processus intra-organisationnels flexibles avec BPMN 2.0. Pour cela nous proposons le méta-modèle BPMN4V-PP (BPMN for Versions of Private Processes), une extension de BPMN 2.0 pour la modélisation de processus intra-organisationnels sous la forme de versions de *private processes*. BPMN4V-PP prend en compte les principales perspectives de modélisation des processus, à savoir la perspective comportementale, la perspective informationnelle et la perspective organisationnelle. Les aspects statiques et dynamiques des versions de processus sont étudiés et un exemple est introduit pour illustrer la modélisation des versions des processus intra-organisationnels. Nous présentons par la suite une extension de BPMN4V-PP, appelée BPMN4V-Context, qui permet la prise en compte du contexte d'utilisation des versions des processus créées conformément à BPMN4V-PP. Cette nouvelle extension prend en compte la perspective contextuelle des processus, perspective qui n'est pas supportée par BPMN 2.0.

La suite de ce chapitre est organisée comme suit. La section 1 présente le méta-modèle BPMN4V-PP pour la prise en compte des processus intra-organisationnels flexibles en se concentrant sur les aspects statiques du méta-modèle et en illustrant la définition de plusieurs versions d'un processus d'*Examen Radiologique*. La section 2 présente une étude de l'aspect dynamique des versions de processus qui sont des instances du méta-modèle BPMN4V-PP. La section 3 détaille les extensions apportées à BPMN pour la modélisation du contexte des versions de processus. Finalement, la section 4 fait une synthèse du chapitre et introduit le chapitre suivant.

I. BPMN4V-PP : un méta-modèle pour la modélisation des versions de processus intra-organisationnels

Cette section présente les extensions apportées à BPMN 2.0 pour modéliser des versions des processus intra-organisationnels. Dans un premier temps, elle présente le méta-modèle BPMN 2.0 tel qu'il a été proposé dans son tutorial [OMG, 2011]. Dans un deuxième temps, elle détaille le modèle de versionnement que nous proposons pour prendre en compte les changements portant sur la définition des processus. Dans un troisième temps cette section présente BPMN4V-PP qui résulte de la fusion du méta-modèle BPMN 2.0 et du modèle de versionnement afin de modéliser des versions de processus. Enfin, cette section montre, à l'aide d'un exemple, comment modéliser des versions d'un processus par instanciation du méta-modèle BPMN4V-PP.

I.1. Méta-modèle de BPMN 2.0

La Figure III.1 présentée ci-dessous donne un extrait du méta-modèle de BPMN 2.0 [OMG, 2011] centré sur les processus privés de BPMN, *i.e.,* les processus intra-organisationnels. Ce méta-modèle intègre des concepts supportant la modélisation des principales perspectives des processus, à savoir la perspective comportementale, la perspective organisationnelle et la perspective informationnelle.

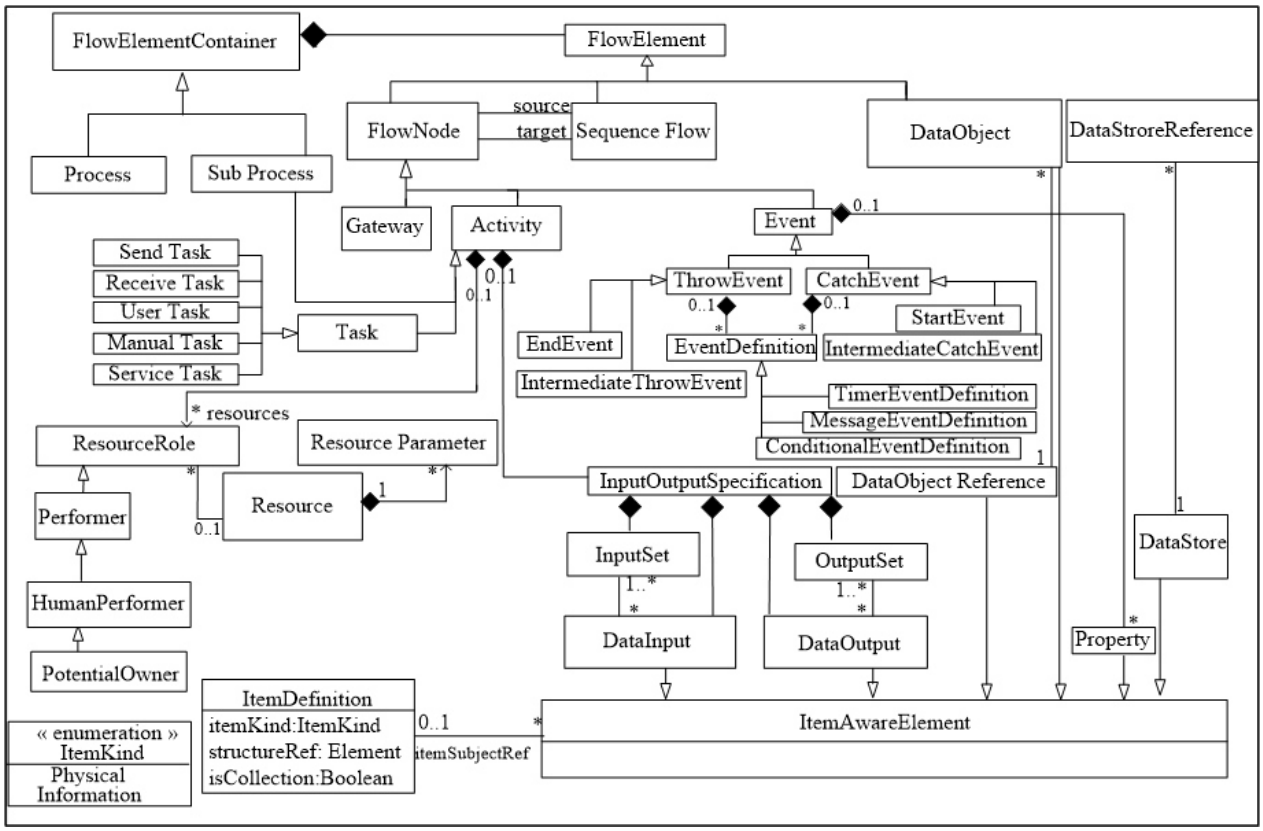


Figure III.1. Extrait du méta-modèle de BPMN 2.0 [OMG, 2011]

La perspective comportementale est représentée essentiellement par la classe *FlowElementContainer*. Cette dernière est une classe abstraite qui représente un regroupement de *FlowElements*. Un *FlowElementContainer* peut être un *Process* ou un *Sub-process*. Un *FlowElement* peut être un *FlowNode*, une *SequenceFlow* ou bien un *DataObject*. Une *SequenceFlow* permet de définir l'ordre des *FlowNodes* dans un *Process*. Un *FlowNode* est un élément du processus qui peut être source ou cible d'une *SequenceFlow*. Il peut être un Branchement (*Gateway*), une activité (*Activity*) ou bien un événement (*Event*). Un *Gateway* permet de contrôler l'interaction des *SequenceFlows* dans un *Process*. Il peut être utilisé pour réunir ou dissocier des *SequenceFlows*. Une activité représente un travail effectué dans un Process ; elle peut être atomique (*Task*) ou composite (*Sub-Process*). BPMN 2.0 définit plusieurs types de *Tasks* parmi lesquels nous citons : les tâches manuelles (*Manual Task*), les tâches semi-automatiques (*User Task*), les tâches automatiques (*Service Task*), les tâches d'envoi de message (*Send Task*) et les tâches de réception de message (*Receive Task*). Un événement doit être utilisé pour débuter un processus (*Start Event*), finir un *processus*

(*End Event*) et peut éventuellement représenter un résultat intermédiaire produit et utilisé par les activités du processus (*Intermediate Event*). Les événements sont de deux types : les événements émetteurs (*ThrowEvent*) utilisés pour déclencher une activité (ou un *Gateway*) dans un *processus* et les événements récepteurs (*CatchEvent*) utilisés pour capturer un résultat qui se produit dans le *Process*. Un événement peut être composé par un ou plusieurs *EventDefinition*. Un *EventDefinition* fait référence aux déclencheurs d'un *CatchEvent* ou bien aux résultats d'un *ThrowEvent*. Les *EventDefinitions* sont de plusieurs types comme par exemple *MessageEventDefinition*, *TimerEventDefinition*, *ConditionalEventDefinition*.

Concernant la perspective organisationnelle, un *ResourceRole* représente l'acteur responsable à l'exécution d'une activité. Il peut faire référence à une *Resource* représentant la ressource organisationnelle concernée par cet acteur. Pour chaque resource, BPMN définit un certain nombre de paramètres *ResourceParameter* pour représenter les informations caractérisant cette *Resource*. Par exemple la *Resource* « Directeur Régional » peut avoir comme paramètres « nom » et « région ».

Concernant la perspective informationnelle, un *ItemAwareElement* représente une information créée, manipulée et / ou utilisée durant l'exécution d'un *Process*. Un *ItemAwareElement* peut avoir un *ItemDefinition* permettant de définir le type d'information portée par cet élément *itemKind* (Physical ou Information) ainsi que sa structure *structureRef*. Un *ItemAwareElement* peut être un *DataObject*, un *DataObjectReference*, un *DataStore*, *DataStoreReference*, une *Property*, un *DataInput* ou un *DataOutput*. Un *DataStore* est utilisé pour modéliser les données stockées dans un processus. On peut réutiliser un même *DataStore* plusieurs fois dans un processus via le concept *DataStoreReference*. Un *DataObject* et une *Property* sont utilisés pour modéliser une donnée consommée et / ou produite dans un processus. Un même *DataObject* peut être réutilisé plusieurs fois dans un même processus via le concept *DataObjectReference*. Un *DataInput* permet de modéliser une information consommée par une activité. L'ensemble des *DataInput* nécessaires pour déclencher une activité forment un *InputSet*. Un *DataOutput* permet de modéliser une information produite par une activité. L'ensemble des *DataOutputs* d'une activité forment un *OutputSet*. Une *InputOutputSpecification* est un regroupement de *DataInputs*, *DataOutputs*, *InputSets* et *OutputSets*.

I.2. Le modèle de versionnement

Comme nous l'avons mentionné dans la chapitre II, la technique de versionnement permet de décrire les changements d'une entité à travers la définition de ses différentes versions. Ces versions sont rattachées entre elles par un lien de dérivation, ce qui forme une hiérarchie de dérivation. Chaque version de la hiérarchie, à l'exception de la première, est dérivée d'une version antérieure.

Dans le but de prendre en compte la hiérarchie de dérivation entre les versions d'une entité, nous adaptons le patron de versionnement proposé dans [Chaâbane, 2012] par l'intégration du type de dérivation. Ce patron est présenté en Figure III.2. L'idée de

ce patron est d'indiquer comment on doit procéder si l'on souhaite gérer des versions. Ce patron sera par la suite appliqué aux classes du méta-modèle précédemment introduit, pour lesquelles nous voulons modéliser des versions. De telles classes ont été appelées classes versionnables dans [Sciore, 1994].

Ce patron de versionnement est simple ; les concepts versionnables du méta-modèle sont décrits par deux classes : la première, appelée « Versionable », permet de représenter les entités pour lesquelles on veut gérer des versions, tandis que la seconde, appelée « Version-of-Versionable », permet de représenter les versions des entités de la classe « Versionable ». Ces deux classes sont liées par deux relations : la relation « Is-version-of » et la relation « Derived-from ». La relation « Is-version-of » est une relation de composition liant la classe composée « versionable » avec la classe composante « Version-of-Versionable » contenant les versions : elle représente le lien d'appartenance d'une version à une entité. La relation « Derived-from » est une relation réflexive sur la classe « Version-of-Versionable ». Elle traduit le fait qu'une version (à l'exception de la première) est dérivée d'une (autre) version antérieure,– constituant ainsi une hiérarchie de dérivation (*cf.* Figure III.2). Les rôles de cette relation sont : (i) PV (Previous Version) au sens de version-prédécesseur assurant le lien entre une version dérivée et l'éventuelle version qui la précède dans la hiérarchie de dérivation (c'est-à-dire la version qui a servi de version de base à la dérivation), et (ii) VS (Version Successor) au sens de version-successeur, qui fait le lien entre une version et les éventuelles versions qui la succèdent dans la hiérarchie de dérivation. Cette relation comporte l'attribut de lien « Type » utilisé pour spécifier le type de la dérivation : par évolution ou par alternative.

La classe Versionable comporte les attributs id et nom du concept versionnable. Quant aux attributs relatifs aux versions, ils sont modélisés dans la classe Version-of-Versionable, et correspondent aux attributs classiques d'une version, à savoir, son numéro, son créateur, sa date de création et son état [Sciore, 1994].

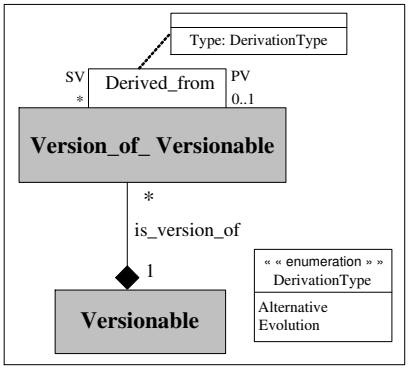


Figure III.2. Patron de versionnement

I.3. BPMN4V-PP

Après avoir détaillé le méta-modèle de BPMN 2.0 et le patron de versionnement, nous présentons dans cette section le méta-modèle BPMN4V-PP (BPMN for Versions of private processes) pour la modélisation des versions de processus intra-

organisationnels. BPMN4V-PP est obtenu par application du patron de versionnement sur certaines classes du méta-modèle BPMN 2.0. La Figure III.3 illustre le méta-modèle BPMN4V-PP résultat de l'extension de BPMN 2.0 [Ben Said *et al.,* 2014(a)].

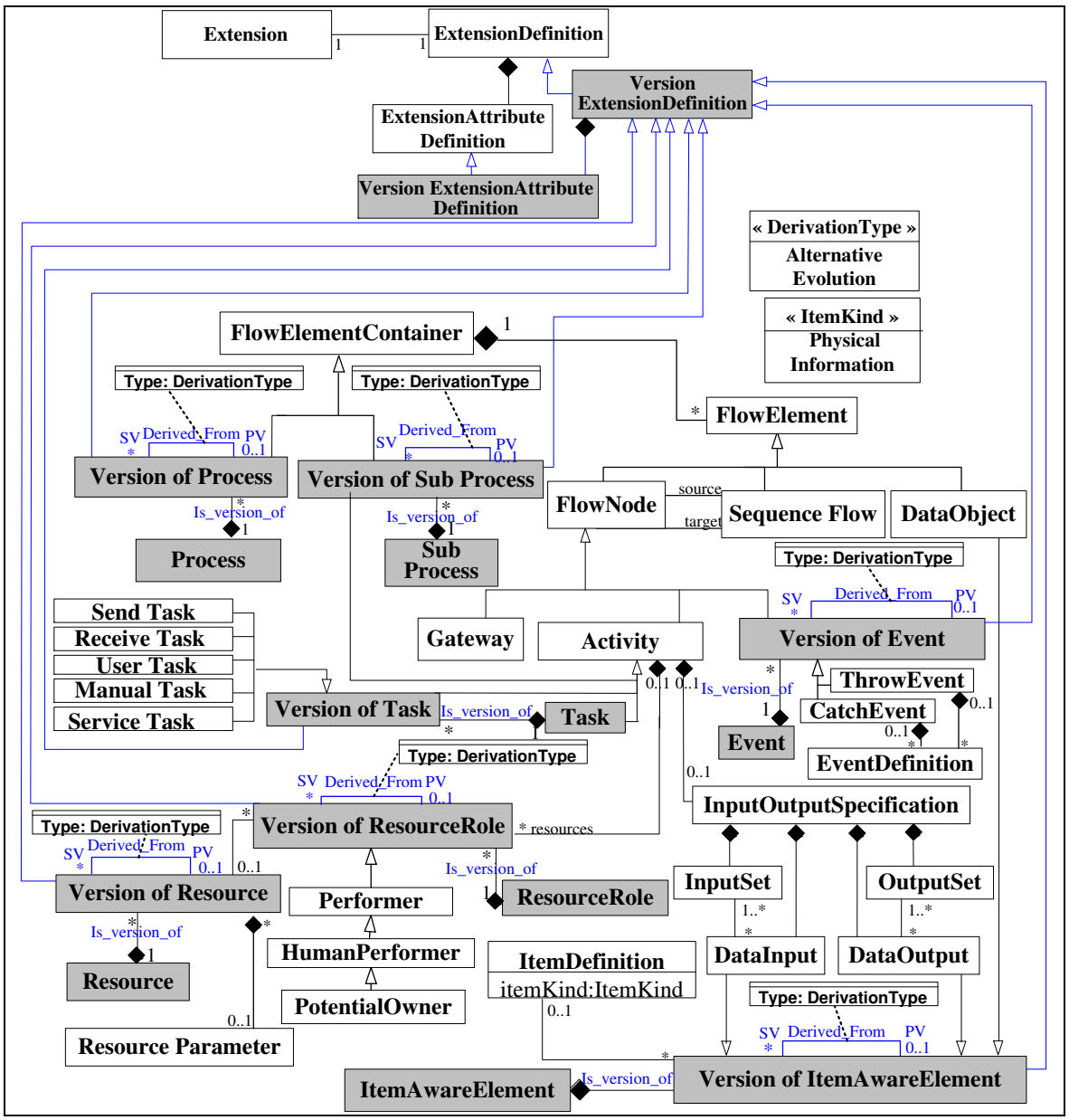


Figure III.3. Méta-modèle BPMN4V-PP

Nous avons décidé de versionner sept classes issues des différentes perspectives du méta-modèle de BPMN 2.0 pour pouvoir garder trace des changements touchant le travail réalisé dans un processus, ainsi que le mode d'organisation et les informations manipulées lors de l'accomplissement de ce travail. Ces classes sont : *Process*, *Sub-Process*, *Task*, *Event*, *Resource*, *ResourceRole* et *ItemAwareElement*.

Une nouvelle version d'un concept versionnable (*e.g., Process)* est définie en fonction des changements qui peuvent se produire. En général, ces changements peuvent correspondre à l'adjonction de nouvelles informations (modélisées par des

propriétés ou des relations) ou bien à la modification ou la suppression d'informations existantes. Ces changements touchent toutes les perspectives de modélisation d'un processus.

Concernant la perspective organisationnelle, nous proposons de créer des versions de *Resource* et *ResourceRole* afin de suivre l'évolution des acteurs et des rôles responsables à l'accomplissement des activités d'un *Process*. Plus précisément, nous créons une nouvelle version d'une *Resource* lorsqu'il y a un changement dans ces paramètres (*i.e., ResourceParameter*). Par exemple, la ressource « Directeur régional » est caractérisé par les deux paramètres suivants : « nom » et « région ». Une nouvelle version de cette ressource doit être définie si, dans certains cas, il est préférable de considérer un troisième paramètre qui est par exemple « expérience ». Par ailleurs, nous créons une nouvelle version d'un *ResourceRole* s'il y a changement des privilèges données à l'acteur concerné. Par exemple, supposons que la *ResoureRole* « le directeur régional Ali » a été chargé de l'accomplissement de quatre activités différentes. Mais suite à une automatisation de certaines activités, ce directeur n'est chargé dorénavant que de deux activités. Une nouvelle version de cette *ResourceRole* doit alors être définie.

Concernant la perspective informationnelle, il est intéressant de garder trace des changements en gérant des versions pour la classe *ItemAwareElement* afin de considérer les modifications des informations manipulées par chaque version de processus. Nous créons une nouvelle version d'un *ItemAwareElement* lorsqu'il y a un changement dans son type itemKind et / ou dans sa structure. Par exemple, « rapport d'un examen médical » est un *ItemAwareElement* représentant un document papier qui doit être rédigé et signé manuellement par un médecin (donc la valeur de la propriété *itemKind* est *Physical*). Suite à un changement technologique, ce rapport devient un document électronique (son *itemKind* devient *Information*). Ceci nécessite la définition d'une nouvelle version de ce rapport.

Concernant la perspective comportementale, nous pensons que gérer des versions pour les concepts *Task, Event, Sub-Process* et *Process* est suffisant pour que les organisations puissent faire face aux modifications relatives à la structure d'un processus (déterminant l'aspect comportemental du processus). Le versionnement de ces quatre classes nous permet de suivre les changements de l'environnement opérationnel des processus ou les changements apportés dans la manière d'organiser le travail. Plus précisément, une nouvelle version de tâche (*Task*) peut être créée s'il y a un changement de son type. Par exemple, supposons que la tâche « Rédaction du rapport d'examen » est jusque-là une tâche manuelle (*Manual Task*). Suite à un changement technologique, cette tâche devient une tâche semi-automatique (*User Task*), d'où la nécessité de définir une nouvelle version de cette tâche. De plus, une nouvelle version de tâche est créée s'il y a un changement des ressources (*i.e., ResourceRole* et / ou *Resource*) nécessaires à son accomplissement et / ou des informations qu'elle manipule (*i.e.,* ses *ItemAwareElement*). En ce qui concerne les événements, nous créons une nouvelle version d'événement s'il y a un changement

dans son *EventDefinition*. Par exemple, considérons l'événement « Panne du matériel d'examen » ayant un *SignalEventDefinition* qui correspond à un signal marquant qu'une machine utilisée dans la réalisation d'un examen radiologique est en panne. Suite à un changement technologique, cet événement change de définition pour devenir un *MessageEventDefinition* correspondant à la réception d'un message détaillant la panne de la machine. Cela entraîne la création d'une nouvelle version de cet événement. Pour les concepts *Process* et *Sub-Process*, nous créons une nouvelle version lorsqu'il y a un changement des activités ou des événements impliqués dans ce *Process* (ou *Sub-Process*) ou bien lorsqu'il y a changement de la coordination de ces activités et événements.

Par ailleurs, BPMN 2.0 offre un mécanisme d'extension permettant de considérer des éléments personnalisés dans le standard. Ce mécanisme est composé par des éléments d'extension qui permettent de joindre de nouveaux attributs et / ou éléments à la notation. Les principaux éléments d'extension de BPMN sont ExtensionAttributeDefinition et ExtensionDefinition. Une ExtensionAttributeDefinition définit la liste des attributs qui peuvent être liés à un élément BPMN, alors qu'une ExtensionDefinition spécifie un nouveau type d'élément à ajouter à la notation.

Dans le méta-modèle BPMN4V-PP, nous avons ajouté la classe VersionExtensionDefinition qui est une ExtensionDefinition représentant un nouveau type d'élément se rapportant aux versions. En effet, cette classe représente une classe générique de tous les concepts versionnables de notre méta-modèle. Notons qu'une VersionExtensionDefinition est composée par un ensemble d'attributs décrivant le détail des versions (*e.g.,* le numéro de version, le nom du créateur, l'état de la version, le nom du créateur, etc.). Ces attributs sont considérés dans la classe VersionExtensionAttributeDefinition.

I.4. Exemple : le processus d'*Examen Radiologique*

Cette section a pour objectif d'illustrer comment on peut modéliser des versions de processus par instanciation du méta-modèle BPMN4V-PP. Pour cela, nous avons repris et étendu dans la Figure III.4 l'exemple introduit dans [Reichert et Weber, 2012] pour assurer la flexibilité d'un processus d'examen des patients d'un centre de radiologie médicale.

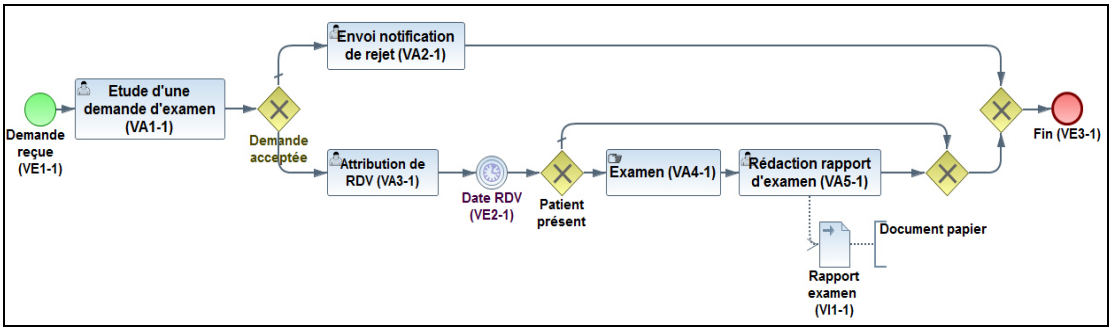


Figure III.4. Première version du processus d'*Examen Radiologique*

Ce processus est formé initialement de cinq tâches : *Etude d'une demande d'examen* (A1), *Envoi d'une notification de rejet* (A2), *Attribution de RDV* (A3), *Examen* (A4) et *Rédaction rapport d'examen* (A5). Nous décrivons, dans ce qui suit, trois versions de ce processus et plusieurs versions de tâches, de ressources et d'informations impliquées dans la définition de ces versions.

La Figure III.4 présente la première version du processus d'*Examen Radiologique*. Cette version a la sémantique suivante : Le processus débute à la réception d'une demande d'examen déposée par un patient (la version VE1-1 de l'événement *Demande Reçue*). L'arrivée de cette demande permet de déclencher la première version de la tâche *Etude d'une demande d'examen* identifiée par VA1-1. Après cette étude et dans le cas où la demande n'est pas acceptée, une notification de rejet peut être envoyée au patient pour expliquer les raisons du refus de sa demande. D'où la définition de la première version de la tâche *Envoi de notification de rejet* identifiée par VA2-1. Si la demande est acceptée, la première version de la tâche *Attribution de RDV*, identifiée par VA3-1, permet de fixer un rendez-vous pour le patient. Les versions des tâches *Etude d'une demande d'examen*, *Envoi de notification de rejet* et *Attribution de RDV* sont à exécuter par la secrétaire moyennant la première version du *ResourceRole* « App-G-CIM » (Application de Gestion du centre d'imagerie médicale). Cette version de *RessourceRole*, identifiée par VRR1-1, est une application web mise en place par le centre de radiologie pour automatiser les tâches relatives à la gestion des patients, la gestion des rendez-vous et la gestion des examens planifiés. Le jour du rendez-vous et en présence du patient, un médecin radiologue effectue l'examen du patient (la première version de la tâche *Examen* identifiée par VA4-1) et rédige le rapport d'examen correspondant (la première version de la tâche *Rédaction rapport d'examen* identifiée par VA5-1). Ce rapport correspond à la première version de *l'itemAwareElement Rapport Examen* identifiée par VI1-1. Dans cette version, le rapport d'examen correspond à un document papier remis au patient à la fin de l'examen.

Cependant, à la réception d'un patient en situation d'urgence (*e.g.,* patient accidenté), la secrétaire vérifie que l'examen radiologique demandé peut être réalisé dans le centre. Si la demande est acceptée, le patient n'attend pas un rendez-vous pour être examiné. Il s'agit alors d'un cas prioritaire pour lequel l'examen radiologique se fait immédiatement. De plus, d'autres examens cliniques peuvent être nécessaires. Suite à ces examens le patient peut subir d'autres examens radiologiques. La Figure III.5 montre une nouvelle version alternative du processus d'*Examen Radiologique* qui prend en charge les patients en cas d'urgence. Cette version diffère de la première version par la suppression de la tâche *Attribution de RDV* et par l'adjonction de la première version de la tâche *Examen complémentaire,* identifiée par VA6-1.

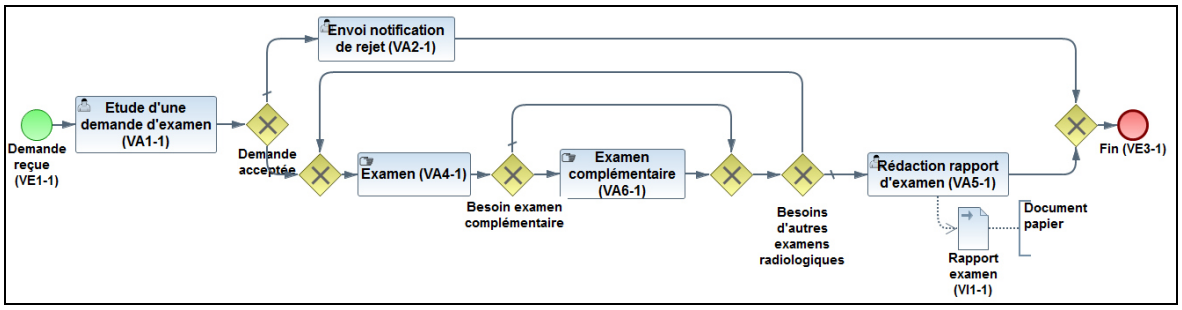


Figure III.5. Deuxième version du processus d'*Examen Radiologique*

D'autre part, les membres de la direction du centre radio ont décidé d'augmenter la capacité d'accueil du centre. Pour ce faire, ils ont opté pour la mise en place de nouvelles salles d'examen et pour l'acquisition de nouveaux matériels. De plus, pour être compétitif et s'adapter aux nouvelles exigences technologiques, la direction a décidé de moderniser l'application web « App-G-CIM ». Elle est devenue une application Mobile qui offre les services suivants : (i) Envoi de sms pour notifier les patients des résultats de leurs demandes d'examen, (ii) Rappel aux patients de leur rendez-vous par envoi d'une notification deux jours avant la date fixée, (iii) Envoi électronique du rapport d'examen au médecin demandeur. En effet, le rapport d'examen devient un document électronique envoyé au médecin demandeur à travers l'application « App-G-CIM ». Par ailleurs, une nouvelle législation impose qu'un médecin radiologue sénior valide le rapport réalisé à la fin de l'examen. Le rapport final change alors de structure et doit être signé par les deux médecins. Ces changements sont représentés dans la nouvelle version du processus, donnée en Figure III.6, par : (i) l'adjonction des premières versions des deux nouvelles tâches *Envoi d'une notification de rappel* (identifiée par VA7-1) et *Validation du rapport* (identifiée par VA8-1), (ii) la définition de nouvelles versions des tâches *Etude d'une demande d'examen* identifiée par VA1-2, *Envoi d'une notification de rejet* identifiée par VA2-2, *Attribution de RDV* identifiée par VA3-2 et *Rédaction rapport d'examen* identifiée par VA5-2 suite aux changements de leurs types, et (iii) la définition de nouvelles versions du *RessourceRole* « App-G-CIM » identifiée par VRR1-2) et *l'ItemAwareElement Rapport Examen* (identifiée par VI1-2).

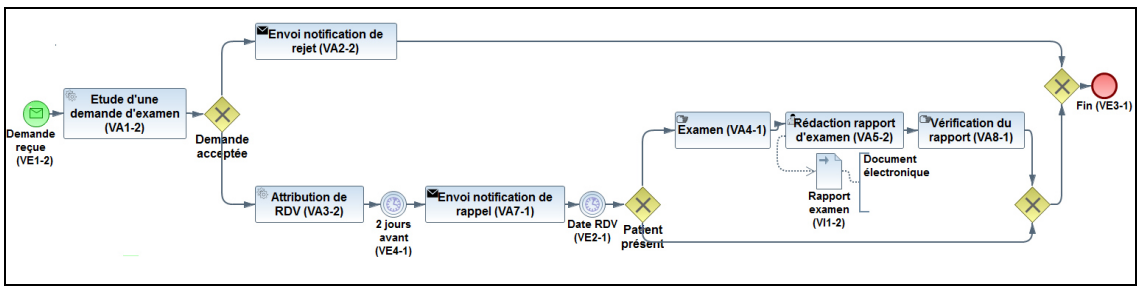


Figure III.6. Troisième version du processus d'*Examen Radiologique*

La mémorisation des produits de la modélisation de ces différentes versions de processus se fait par instanciation du méta-modèle BPMN4V-PP. Les Figures III.7,

III.8 et III.10 récapitulent les différentes versions du processus d'*Examen Radiologique* ainsi que les versions de leurs tâches, d'événements, ressources et informations. Quant aux Figures III.9 et III.11, elles présentent des extraits de deux diagrammes d'objets UML obtenus par instanciation du méta-modèle BPMN4V-PP conformément à l'exemple du processus d'*Examen Radiologique*. Pour des raisons de lisibilité, la Figure III.9 se limite à l'extrait correspondant à l'instanciation des classes issues de la perspective comportementale pour les trois versions du processus, alors que la Figure III.11 se limite à l'extrait correspondant à l'instanciation des classes issues des perspectives organisationnelle et informationnelle relatives aux versions des activités et des événements de ce processus.

Les Figures III.7 et III.9 montrent le processus d'*Examen Radiologique*, identifié par P1, ainsi que les trois versions VP1-1, VP1-2 et VP1-3 liées à P1 par la relation de composition « Is_version_of ». La version VP1-2 est dérivée par alternative à partir de la version VP1-1, alors que la version VP1-3 est drivée par évolution de la même version VP1-1, d'où les relations « Derived_From » entre ces versions.

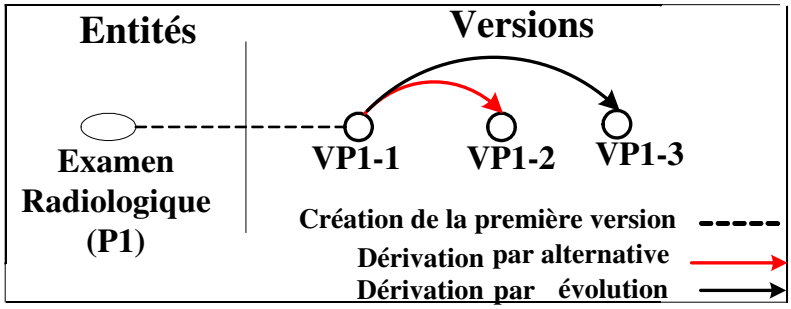


Figure III.7. Hiérarchie de dérivation de l'entité Processus

En outre, la Figures III.8, avec la Figure III.9, montrent les versions des tâches et des événements appartenant à la définition des versions du processus d'*Examen Radiologique*.

Plus précisément, l'événement *Demande Reçue* (identifiée par E1) présente deux versions VE1-1 et VE1-2. La version VE1-1 correspond à un *None Start Event* indiquant que le processus débute lorsque le patient dépose une demande d'examen. La version VE1-2 est dérivée par évolution à partir de la version VE1-1 et correspond à un événement ayant un *MessageEventDefinition* permettant de marquer que le processus débute à la réception d'un message comportant la demande d'examen.

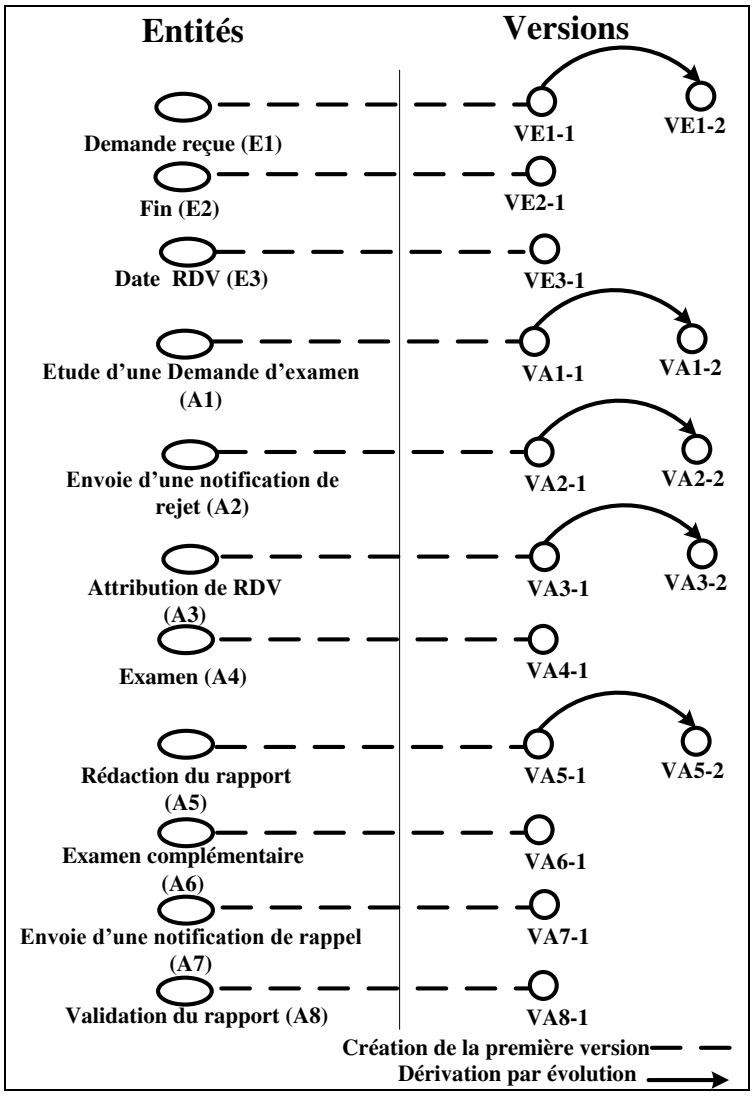


Figure III.8. Hiérarchie de dérivation des entités Tâches et Evénements

Les tâches *Etude d'une demande d'examen* (identifiée par A1), *Envoi d'une notification de rejet* (identifié par A2) et *Attribution de RDV* (identifiée par A3) présente chacune deux versions différentes. Les versions VA1-1, VA2-1 et VA3-1, correspondant respectivement aux premières versions de ces tâches, spécifient des exécutions semi-automatiques. Les versions VA1-2, VA2-2 et VA3-2 sont dérivées par évolution à partir de respectivement VA1-1, VA2-1 et VA3-1. Les versions VA1-2 et VA3-2 spécifient des exécutions automatiques (*ServiceTask*) effectués par l'application Mobile « App-G-CIM ». La version VA2-2 spécifie un envoi (*SendTask*) automatique d'une notification à travers l'application « App-G-CIM ».

Les événements « Fin » (identifié par E2) et « Date RDV » (identifié par E3) et les tâches « Examen » (identifiée par A4), « Examen complémentaire » (identifiée par A6), « Envoi d'une notification de rappel » (identifiée par A7) et « Validation du rapport » (Identifiée par A8) n'ont qu'une seule version chacune.

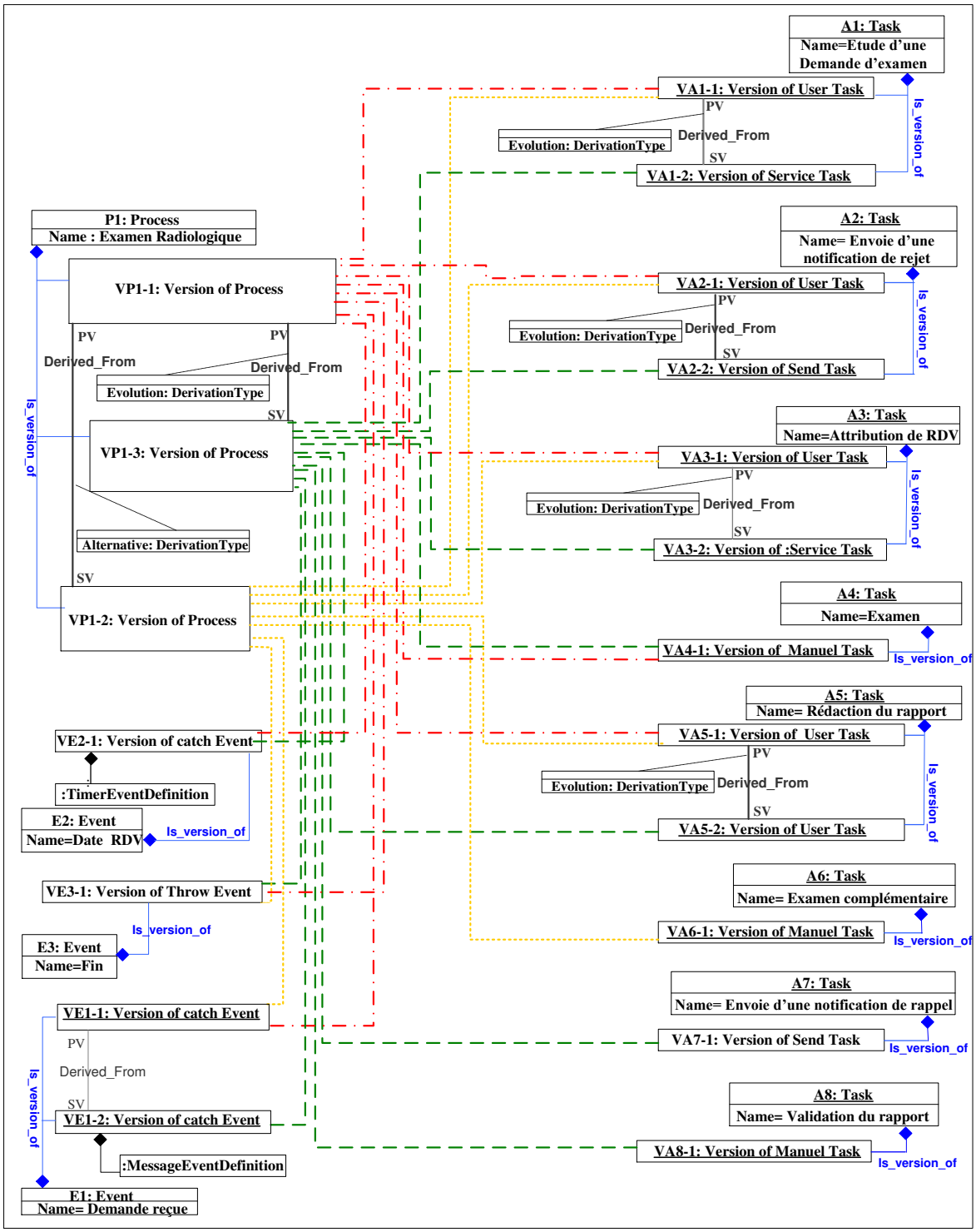


Figure III.9. Instantiation des classes issues de la perspective comportementale du méta-modèle BPMN4V-PP

Les Figures III.10 et III.11 présentent les versions des entités *Resource*, *ResourceRole* et *ItemAwareElement* relatives aux trois versions du processus d'*Examen Radiologique*. En effet, la Figure III.10 montre la hiérarchie de dérivation de la *Resource* « Application » (identifiée par R1), le *ResourceRole* « App-G-CIM » (identifié par RR1) et *l'ItemAwareElement* « Rapport d'examen » (identifié par I1).

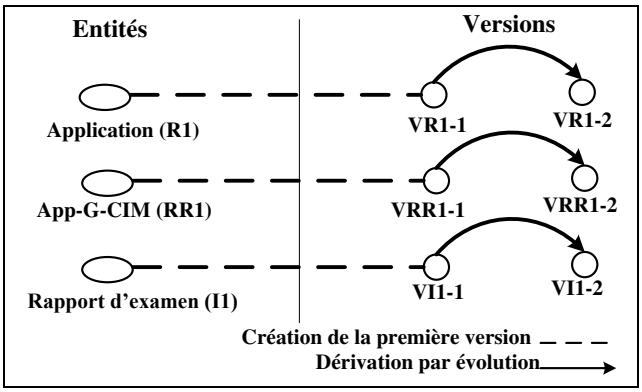


Figure III.10. Hiérarchie de dérivation des entités *Resource*, *ResourceRole* et *ItemAwareElement*

La Figure III.11 montre une instanciation des classes du méta-modèle BPMN4V-PP issues des perspectives organisationnelle et informationnelle. Plus précisément, cette Figure montre que les versions VR1-1 et VR1-2 sont deux versions de la ressource « Application ». La version VR1-1 est caractérisée par le *ResourceParameter* « Technologie ». La version VR1-2 est dérivée par évolution de la version VR1-1 et spécifie, en plus du paramètre « Technologie », le paramètre « Date de mise à jour de l'application ». A ces deux versions de ressource correspondent les deux versions de la *ResourceRole* « App-G-CIM » VRR1-1 et VRR1-2. La version VRR1-1 est invoquée dans la réalisation de la première version de la tâche *Attribution de RDV*. Alors que la deuxième version VRR1-2 permet la réalisation de la deuxième version de cette même tâche.

La Figure III.11 montre aussi que les versions VI1-1 et VI1-2 sont deux versions de *l'ItemAwareElement* « Rapport d'examen ». La version VI1-2 (qui correspond à un rapport sous forme d'un document électronique) est une version dérivée par évolution à partir de la version VI1-1 (qui correspond à un rapport sous forme d'un document papier). Par conséquent, la tâche « Rédaction de rapport » (identifiée par A5) présente aussi deux versions VA5-1 et VA5-2. La version VA5-1, qui correspond à une rédaction manuelle du rapport, produit la version *d'ItemAwareElement* VI1-1 (c.à.d. la version papier du rapport). La version VA5-2, qui correspond à rédaction du rapport via l'application, est dérivée par évolution de la version VA5-1 suite au changement de la version d'information produite par cette version de tâche. VA5-2 produit alors la deuxième version de *l'ItemAwareElement* « Rapport d'examen » VI1-2, c.à.d. la version électronique du rapport.

II. Etude de la dynamique des versions de processus intra-organisationnels

Cette section aborde les aspects dynamiques de BPMN4V-PP. Plus précisément, elle présente dans un premier temps un diagramme d'états-transitions UML décrivant les états possibles d'une version ainsi que les différentes opérations réalisables sur ces versions en fonction de leur état. Elle détaille dans un second temps chacune de ces opérations.

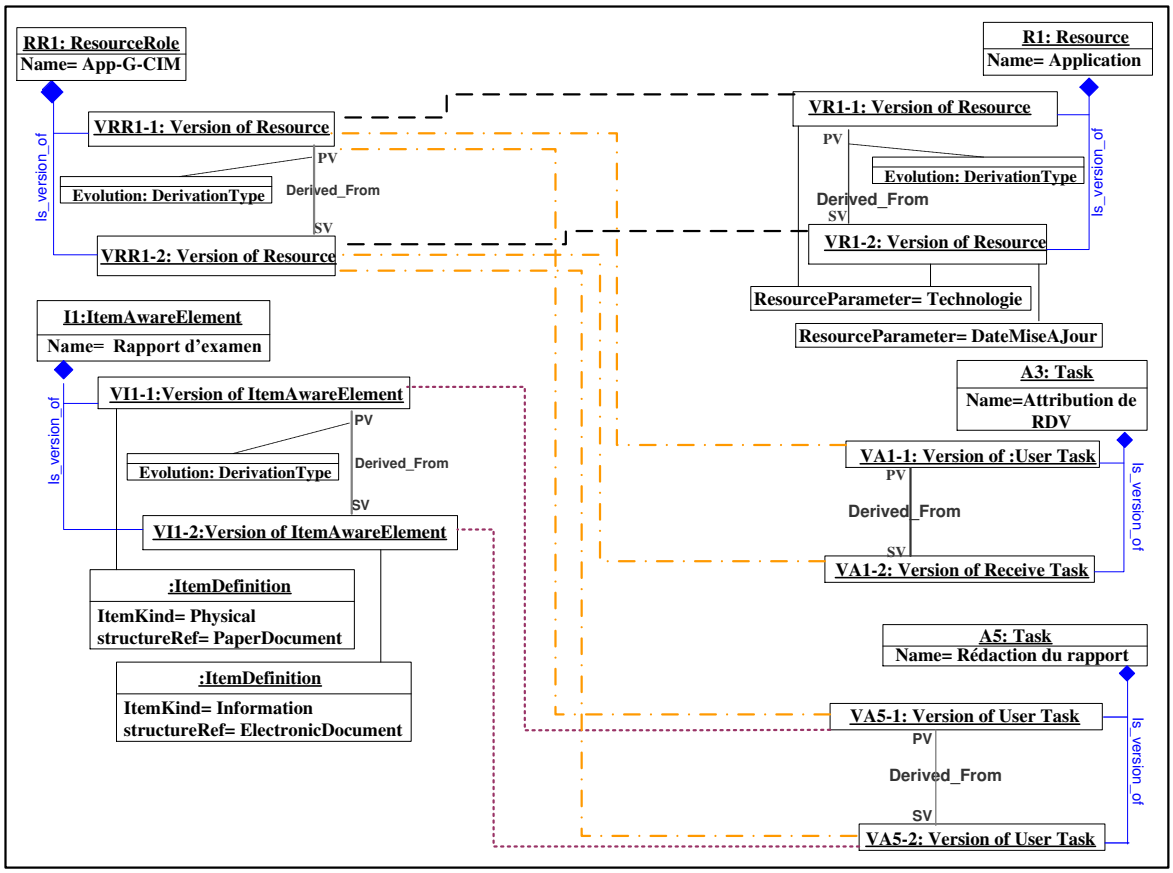


Figure III.11. Instantiation des classes issues des perspectives informationelle et organisationnelle du méta-modèle BPMN4V-PP

II.1. Diagramme d'états-transitions d'une version

La Figure III.12 donne le diagramme d'états-transitions UML introduisant les différents états que peut avoir une version durant son cycle de vie et les différentes transitions reliant ces états. Ce diagramme indique aussi les opérations possibles à partir d'un état donné. Les transitions de ce diagramme sont décrites par un couple Evénement/Action qui a la sémantique suivante : si un Evénement e est détecté alors l'Action a est déclenchée [Ben Said *et al.,* 2015(a)].

Lors de l'apparition de l'événement de création d'une nouvelle entité (*e.g.,* processus), l'opération de création est déclenchée pour créer la nouvelle entité ainsi que sa première version. L'état de la version créée est « *En travail (T)* ». Dans cet état, la version est en cours de définition ; elle n'est pas dans son état final. Elle peut être modifiée à l'aide de l'opération de modification. Les opérations de création et de modification peuvent être spécifiées à l'aide d'un ensemble de primitives. Bien que ces primitives varient d'un concept (*i.e.,* élément BPMN) à un autre, elles partagent la même sémantique : elles définissent les valeurs des propriétés et des relations de ces concepts. Elles sont présentées en détail dans la section II.2.

Une version dans l'état « *En travail* » peut être supprimée avec l'opération de suppression ou bien validée avec l'opération de validation. Lorsque la validation est

effectuée, la version concernée devient « *Stable (S)* ». Cet état indique alors que la version est dans un état final : elle ne peut plus subir de modifications.

Finalement, une version à l'état « *Stable* » peut servir de base pour la création d'une nouvelle version en utilisant l'opération de dérivation. La version obtenue par dérivation est dans l'état « *En travail* ». Avant d'être modifiée, cette version a la même définition que la version origine de la dérivation.

Le diagramme d'états-transitions présenté dans la Figure III.12 donne une idée sur les opérations possibles pour une version. Dans ce qui suit, nous détaillons chacune de ces opérations.

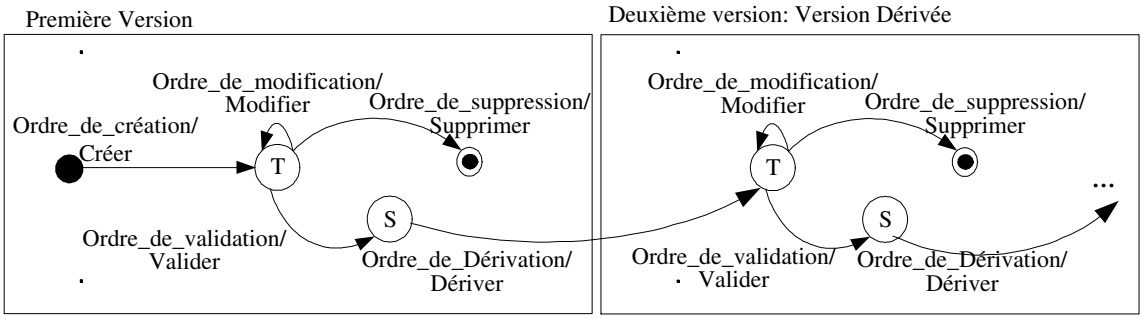


Figure III.12. Digramme d'états-transitions d'une version

II.2. L'opération de création d'une version

Comme nous l'avons mentionné précédemment, l'opération de création permet la création d'une entité (versionnable) ainsi que sa première version. Elle est définie au travers d'un ensemble de primitives qui varient selon l'élément BPMN concerné par la création. Le tableau III-1 illustre la définition de cette opération en spécifiant les primitives correspondantes à chaque élément. Bien que ces primitives soient différentes d'un élément à un autre, elles partagent le même objectif consistant à donner des valeurs aux propriétés et / ou aux relations de ces éléments.

La création d'un *processus* (ou *sub-process*) et de sa première version inclut les primitives permettant l'adjonction (+) des activités (*Tasks* ou *Sub-Process*), événements, *SequenceFlow* et *Gateways* qui composent ce processus (ou *Sub-process*). Concernant les tâches, la création se fait via les primitives permettant l'adjonction des informations manipulées (*ItemAwareElement*) et des ressources impliquées (*ResourceRole).* La création d'un événement est assurée par la définition de son *EventDefinition* (*e.g., MessageEventDefinition* ou *ConditionalEventDefinition*) ou bien par la définition des informations supportées par cet événement. Quant à la création d'une information (*i.e., ItemAwareElement*), elle est réalisée par une primitive assurant la définition/valorisation de son *ItemDefinition* comportant le type de l'information (*i.e., itemKind*) et sa structure (*i.e., StructureRef*). La création d'une *Resource* consiste en l'adjonction des paramètres (*i.e., ResourceParameter*) décrivant cette ressource. Finalement, les primitives relatives à la création d'un *ResourceRole* permettent la spécification (i) de la *Resource* référencée par cet *ResourceRole* et (ii) de ses

autorisations attribuées (ceci par l'adjonction des activités censées être accomplies par cet *ResourceRole).*

Tableau III-1. Primitives de l'opération de création (cas des processus privés)

Légende + : adjonction

Elément BPMN4V-PP	Primitives
Process/Sub-Process	+Activity +Event +Sequence Flow + Gateway
Task	+ItemAwareElement +ResourceRole
Event	+EventDefinition +ItemAwareElement
ItemAwareElement	+ItemDefinition (itemKind et itemStructure)
Resource	+Parameter
ResourceRole	+Resource +Activity

II.3. L'opération de modification d'une version

De même que l'opération de création, l'opération de modification est définie au travers d'un ensemble de primitives assurant la modification d'une version. Le tableau III-2 illustre les primitives correspondant à chaque élément versionnable de BPMN4V-PP concerné par une modification.

La modification d'une version d'un *Process* (ou *Sub-Process)* inclut les primitives permettant l'adjonction (+) ou la suppression (-) des versions de *Task,* des versions de *Sub-Process*, des versions *d'Event*, des *SequenceFlow* et des *Gateways* qui composent cette version de *Process* (ou *Sub-Process*). Concernant les tâches, la modification d'une version se fait à travers des primitives permettant l'adjonction ou la suppression des informations manipulées (*ItemAwareElement*), des ressources impliquées (*ResourceRole)* ou bien la modification du type (suppression d'un type existant et adjonction d'un autre) de cette version de tâche (*e.g., UserTask*, *SendTask* ou *ReceiveTask*). La modification d'une version d'événement est assurée par l'adjonction ou la suppression d'un *EventDefinition* ou bien des informations supportées par cet événement. Quant à la modification d'une version d'information (*i.e., ItemAwareElement*), elle est réalisée par une primitive assurant la modification de son *ItemDefinition,* en particulier son type (*i.e., itemKind*) et / ou sa structure (*i.e., structureRef*). La modification d'une version de *Resource* consiste en l'adjonction ou la suppression des paramètres (*i.e., ResourceParameter*) décrivant cette ressource. Finalement, les primitives relatives à la modification d'une version de *ResourceRole*

permettent la modification de la *Resource* référencée par cet *ResourceRole* ou bien la modification de ses autorisations en ajoutant ou en supprimant des activités censées être accomplies par cet *ResourceRole*.

Tableau III-2. Primities de l'opérations de modification d'une version (cas des processus privés)

Légende + : adjonction ; - suppression

Eléments BPMN4V-PP	Primitives
Version_of_Process/ Version_of_Sub-Process	+/-Activity +/- Event +/- Sequence Flow +/- Gateway
Version_of_Task	+/- ItemAwareElement +/- ResourceRole Modifier le Type (supprimer le type existant et le remplacer à un nouveau type)
Version_of_Event	+/-EventDefinition +/-ItemAwareElement
Version_of_ItemAwareElement	Modifier ItemDefinition (supprimer les valeurs des propriétés itemKind et itemStructure et les remplacer par autres valeurs)
Version_of_Resource	+/- Parameter
Version_of_ResourceRole	Modifier la Resource (supprimer la ressource et la remplacer par une autre) +/-Activity

II.4. Opération de validation

L'opération de validation s'applique sur une version dans l'état « *En travail* ». Elle permet de faire passer la version à l'état « *Stable* », qui est l'état final d'une version. Par conséquent, la version ne pourra plus subir de modifications. De plus, la validation d'une version peut déclencher la validation d'autres versions reliées à cette version. En effet, lorsque le concepteur souhaite valider une version de processus, c'est qu'en réalité il souhaite valider le processus avec toutes les versions qui le composent. La Figure III.13 illustre la propagation de la validation. Dans cette Figure les flèches noires correspondent aux validations initiales d'une version d'un concept versionnable (*i.e.,* validation par l'application directe de l'opération de validation), alors que les flèches rouges correspondent aux validations par propagation.

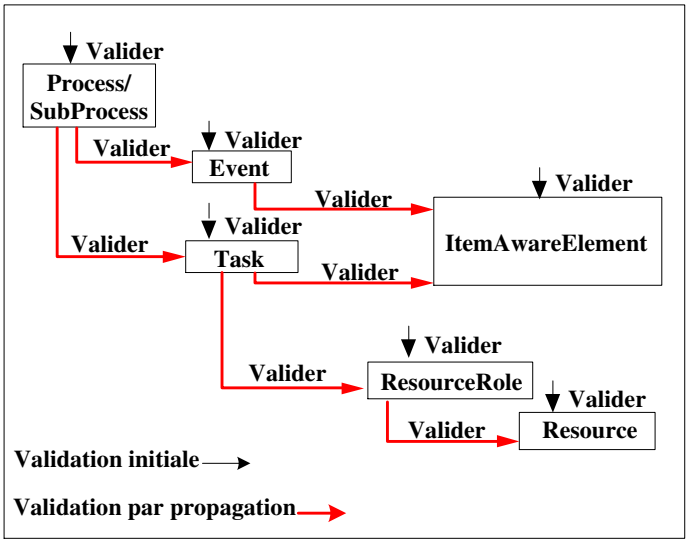


Figure III.13. Propagation de la validation

La validation d'une version d'un *process* (ou *sub-process)* entraine la validation des tâches et des événements qui le composent. De plus, la validation d'une version d'une tâche entraine la validation de tous ses composants (*e.g.,* versions des *ItemAwareElement* et des *ResourceRole* impliqués dans cette version). La validation d'une version d'événement entraîne la validation de l'information (*i.e., ItemAwareElement*) supportée par cette version. De même, la validation d'une version de *ResourceRole* entraîne la validation de la version de *Resource* correspondante.

II.5. Opération de dérivation

L'opération de dérivation permet la création d'une nouvelle version à partir d'une version dans l'état « *Stable* ». Avant d'être modifiée, la version créée possède la même définition que la version source de dérivation. De plus, la dérivation d'une version peut déclencher la dérivation d'autres versions. La Figure III.14 illustre la propagation de la dérivation. Cette propagation est due aux relations de composition entre les classes du méta-modèle BPMN4V-PP : *Process* (*FlowElementContainer*), *Activity*, *Event* (*FlowElement*), *ResourceRole, Resource* et *ItemAwareElement*. Par conséquent, la dérivation d'une version de *Resource* entraîne la dérivation de la version *ResourceRole* correspondante. De plus, la dérivation d'une version de *ResourceRole* ou d'une version d'un *ItemAwareElement* entraîne la dérivation de la version de tâche concernée. De même, la dérivation d'une version *d'ItemAwareElement* entraîne la dérivation de la version d'événement qui supporte cette version d'information. Finalement, la dérivation d'une version d'une tâche ou d'une version d'un événement entraîne la dérivation de la version de *Process* (ou *Sub-process*) correspondante.

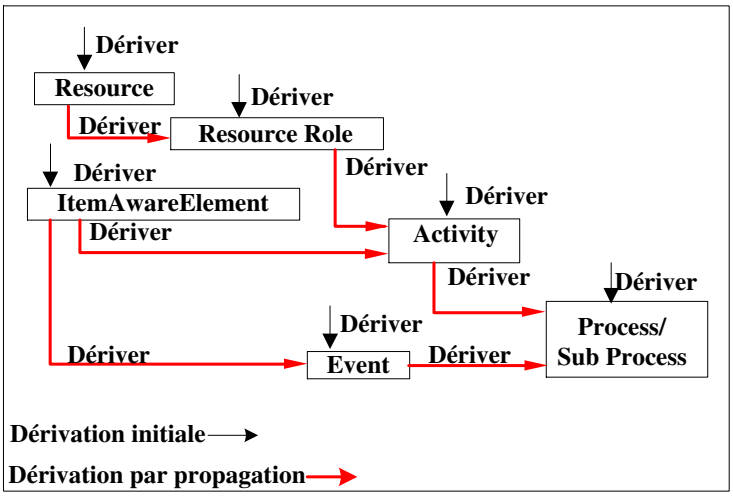


Figure III.14. Propagation de la dérivation

III. Modélisation des contextes des versions

L'objectif de cette section est d'aider au choix d'une version dans une situation donnée. La solution préconisée ajoute une nouvelle dimension à BPMN4V-PP pour décrire le contexte d'utilisation de chaque version modélisée. Il s'agit donc d'étendre BPMN4V-PP afin de considérer le contexte des versions des processus. Tout d'abord, cette section présente le méta-modèle de contexte utilisé pour modéliser les informations contextuelles des processus. Par la suite, elle décrit le méta-modèle BPMN4V-Context qui combine le modèle de contexte et le méta-modèle BPMN4V-PP en utilisant un ensemble de classes et de liens. Finalement, elle illustre la modélisation de contexte par un exemple relatif à des versions du processus d'*Examen Radiologique*, à travers une instanciation du méta-modèle BPMN4V-Context.

III.1. Méta-modèle de contexte

Le méta-modèle de contexte, présenté dans la Figure III.15, permet la définition de tout modèle de contexte *ContextModel* comme un ensemble de paramètres de contexte *ContextParameter*. Un paramètre de contexte correspond à une information qui caractérise une situation particulière pour laquelle une condition doit être définie. Par exemple, le paramètre de contexte « Disponibilité d'une salle d'examen » est utilisé pour décrire l'état d'une ressource (disponible ou non disponible). Un paramètre de contexte possède une certaine nature *ContextNature.* En se référant à la taxonomie de Rosemann [Rosemann *et al.,* 2008], détaillée dans le chapitre II, le contexte peut avoir quatre natures différentes : contexte environnemental, contexte externe, contexte interne et contexte de processus. En plus, un paramètre de contexte est caractérisé par un type qui fait référence à la perspective à laquelle ce paramètre appartient. De ce fait, nous considérons les paramètres liés au comportement d'un processus *BehavioralParameter* regroupant les informations contextuelles relatives à la perspective comportementale, comme par exemple le temps nécessaire à l'accomplissement du processus ou bien le mode de fonctionnement d'une tâche. Nous

considérons, aussi, les paramètres des ressources *ResourceParameter* et les paramètres des informations *DataParameter* faisant référence aux informations contextuelles relatives respectivement à la perspective organisationnelle et à la perspective informationnelle. Finalement, nous considérons les paramètres de but *GoalParameter* regroupant les informations contextuelles relatives aux objectifs à atteindre, comme par exemple la qualité d'un service. Ces paramètres forment la perspective contextuelle des processus [Ben Said *et al.,* 2014(b)].

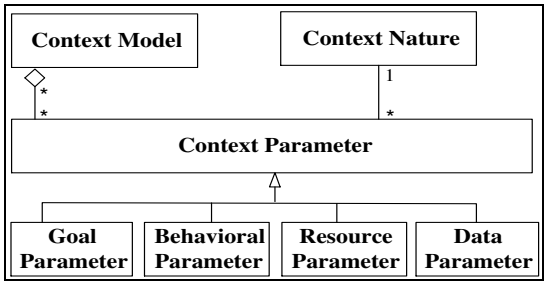


Figure III.15. Méta-modèle de contexte

III.2. Méta-modèle BPMN4V-Context

Le méta-modèle BPMN4V-Context, présenté dans la Figure III.16, est le résultat de l'intégration du méta-modèle de contexte dans le méta-modèle de BPMN4V-PP [Ben said *et al.,* 2014(b)]. Cette intégration est réalisée en utilisant des relations et des classes d'association reliant les classes versionnables du Méta-Modèle BPMN4V-PP avec les classes du méta-modèle de contexte. Le méta-modèle BPMN4V-Context résultant décrit le contexte des versions, et en particulier des versions de processus, en précisant ses objectifs (*i.e.,* pourquoi cette version est créée) ainsi que son contexte d'utilisation (dans quel cas une version est utilisée). Ce dernier peut être spécifié en définissant le contexte des composants versionnables de la version considérée. Si cette version est une version de processus, il s'agit du contexte des versions de tâches, des versions de sous-processus, des versions d'événements, des versions de ressources et des versions d'informations auxquelles elle est rattachée.

Plus précisément, le contexte d'une version d'un processus peut être exprimé en fonction des paramètres de contexte. Ces paramètres forment ainsi le modèle de contexte du processus concerné. Ils sont associés aux concepts du BPMN4V-PP issus des différentes perspectives, *i.e.,* la perspective comportementale, la perspective organisationnelle, la perspective informationnelle et la perspective contextuelle. Par exemple, le modèle de contexte du processus d'*Examen Radiologique* comporte les paramètres suivants : le nombre de patients examinés par jour (issu de la perspective contextuelle), l'urgence (issu de la perspective comportementale), la disponibilité d'une salle d'examen (issu de la perspective organisationnelle) et le type d'un document (issu de la perspective informationnelle).

Il est à noter que le concept modèle de contexte du méta-modèle BPMN4V-Context est le regroupement logique des paramètres qui décrivent le contexte d'un processus. Les versions de ce processus partagent le même modèle mais se distinguent

par les valeurs attribuées à ces paramètres. Ce concept est différent de la notion de modèle de contexte, qui est une instanciation du méta-modèle BPMN4V-Context.

Nous expliquons dans ce qui suit comment modéliser le contexte d'une version de processus à l'aide des paramètres de contexte en exprimant, d'une part, les objectifs de cette version et, d'autre part, son contexte d'utilisation.

III.2.1. Modélisation des objectifs d'une version de processus

L'objectif de création d'une version de processus, ainsi que de ses composants versionnables, peut être exprimé en fonction des paramètres de but *GoalParameter*. Ces paramètres utilisent des conditions de but *Goalconditions* pour spécifier pourquoi une version a été créée.

Par exemple, NombrePatientsExaminés/jour est un *GoalParameter* faisant partie du modèle de contexte du processus d'*Examen Radiologique*. La première version de ce processus est définie pour le cas où le nombre des patients examinés par jour est inférieur à 100, alors la deuxième version de ce processus convient pour le cas où ce nombre est supérieur à 100 patients. Dans ce cas, nous définissons deux conditions de but (*Goalconditions).* La première condition est « <100 patients » pour la spécification des raisons de définition de la première version du processus d'*Examen Radiologique*. La deuxième condition « >100 patients » pour la définition des raisons de définition de la deuxième version de ce processus.

Par ailleurs, un *Goalparameter* utilisé pour définir le but d'une version doit faire partie des paramètres formant le modèle de contexte du processus concerné. Dans le méta-modèle proposé et visualisé en Figure III.17 nous utilisons la contrainte UML *CI1* pour la définition de cette restriction.

III.2.2. Modélisation du contexte d'utilisation d'une version de processus

Le contexte d'utilisation d'une version de processus permet de spécifier dans quel cas cette version est utilisée. Ce contexte peut être exprimé en fonction des paramètres de contexte (notamment les *BehavioralParameters*, les *DataParameters* et les *ResourceParameters*) et des conditions d'affectation appelées *Assignment conditions*. Ces conditions spécifient pourquoi une version fait partie de la définition d'une version de processus. Elles sont alors des expressions booléennes permettant de décrire pour une version de processus pourquoi les versions des tâches, des sous-processus, des événements, des ressources et des informations sont utilisées dans la définition de cette version.

Plus précisément, le méta-modèle BPMN4V-Context est le méta-modèle BPMN4V-PP enrichi par l'adjonction des trois classes d'association suivantes : *Context of FlowNode Assignment, Context of Resource Assignment* et *Context of Data Assignment*. Ces classes utilisent les *Assignment conditions* pour définir les conditions d'affectation des composants versionnables d'une version de processus. En effet, la classe d'association *Context of FlowNode Assignment* permet de définir dans quels cas

une version d'une activité (*i.e., Task* ou *Sub-Process*), ou bien une version d'un événement, est utilisée dans une version de processus. En plus, la classe d'association *Context of Resource Assignment* est utilisée pour définir le contexte d'affectation d'une version de *ResourceRole* à une version d'activité, *i.e.,* dans quel cas une version de *ResourceRole* est invoquée pour la réalisation d'une version d'activité. De même, la classe d'association *Context of Data Assignment* permet de spécifier les conditions d'affectation d'une version d'information (*i.e., ItemAwareElement*) à une version d'activité. En particulier, cette classe d'association est utilisée pour définir pourquoi une version d'activité produit ou consomme une version d'information. Prenons l'exemple d'un *itemAwareElement* représentant une demande d'examen. Cette information peut avoir deux versions différentes dont chacune a une structure particulière. La structure de première version correspond à une demande d'examen d'un patient dans un état normal alors que la structure de la deuxième version convient à une demande d'examen d'un patient dans un état grave. De ce fait, la première version de l'*itemAwareElement Demande d'examen* est assignée à la première version de l'activité *Réception d'une demande d'examen* dans le cas où le patient est dans un état normal, alors que la deuxième version de cette information est affectée à la deuxième version de l'activité *Réception d'une demande d'examen* si le patient est dans un état urgent.

Par ailleurs, un paramètre de contexte utilisé dans la définition d'une condition d'affectation doit faire partie de l'ensemble des paramètres de contexte qui forment le modèle de contexte du processus concerné. Dans le méta-modèle BPMN4V-Context, cette restriction est prise en compte dans les contraintes d'intégrités *CI2*, *CI3* et *CI4*.

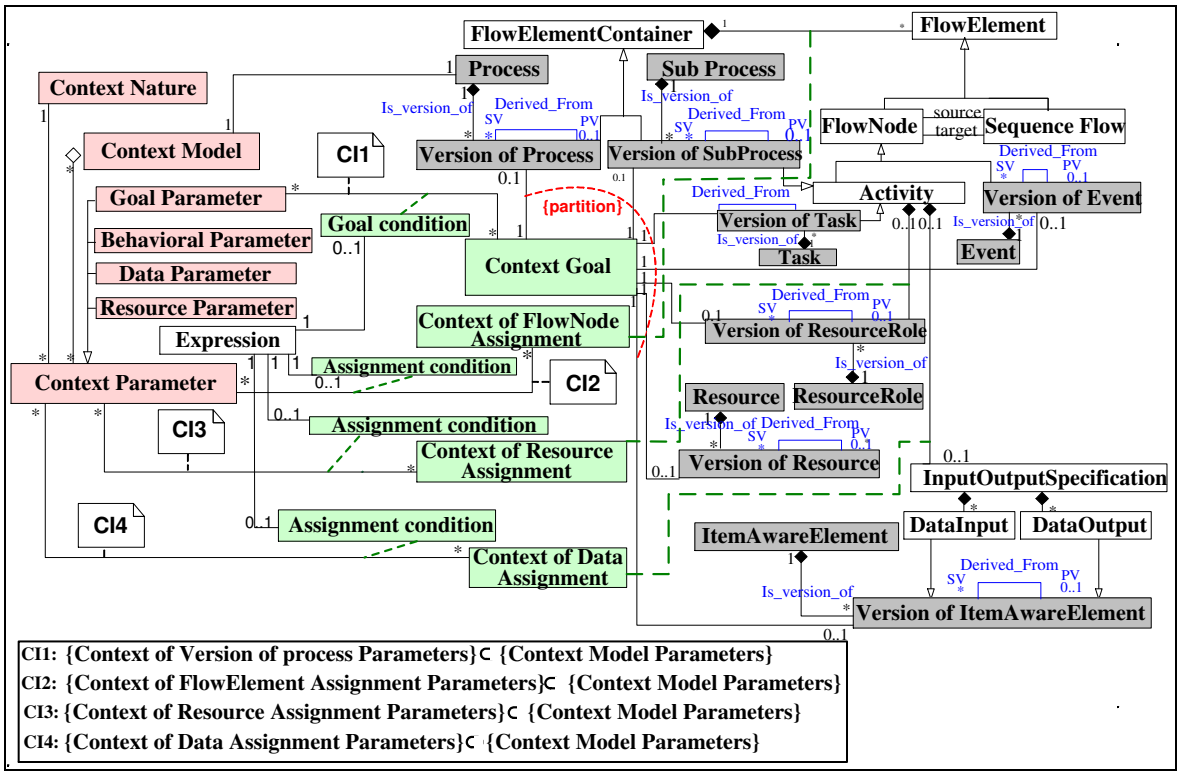


Figure III.16. Méta-modèle de BPMN4V-Context

III.3. Définition du contexte des versions du processus d'*Examen Radiologique*

Afin de mieux illustrer les extensions apportées au méta-modèle BPMN4V-PP pour la prise en compte des contextes des versions de processus, nous reprenons l'exemple du processus d'*Examen Radiologique* présenté dans la section I.4. Dans la suite nous explicitons le contexte de chaque version de ce processus à travers un ensemble de paramètres. Ces paramètres permettent de définir soit le but de création d'une version (*i.e.,* pourquoi une version a été créée), soit le contexte de son utilisation (*i.e.,* les situations dans lesquelles une version peut être utilisée).

Les paramètres de contexte qui caractérisent le contexte des trois versions du processus d'*Examen Radiologique* sont les suivants : le nombre des patients examinés/jour, l'urgence du cas à traiter, le type de l'application et le type du document. Le tableau III-3, présenté ci-dessous, résume les paramètres de contexte utilisés dans la définition des versions du processus d'examen radiologique ainsi que les conditions de chaque paramètre. D'autre part, la Figure III.17 donne un extrait de l'instanciation de la première et la troisième version du processus d'*Examen Radiologique* conformément au méta-modèle BPMN4V-Context.

Tableau III-3. Les paramètres de contexte et les conditions des versions de processus d'examen radilogique

Nature du contexte	Paramètre de contexte (Type)	Contexte 1ère version	Contexte 2ème version	Contexte 3ème version
Contexte Interne	NombrePatientsExaminés/ jour (Goal Parameter)	<100	<100	>100
Contexte Immédiat	Urgence (ProcessParameter)	Non	Oui	Non
	TypeApplication (ResourceParameter)	Est_une ApplicationWeb	Est_une ApplicationWeb	Est_une ApplicationMobile
	TypeDocument (DataParameter)	Est_un DocumentPapier	Est_un DocumentPapier	Est_un Document Electronique

La première et la deuxième version du processus d'*Examen Radiologique* (VP1-1 et VP1-2 sont deux alternatives définies au moment de la création du centre. Etant donné que l'investissement initial est limité, le nombre des patients examinés par jour n'est pas important. La troisième version du processus d'*Examen Radiologique* (VP1-3) est définie suite à une décision stratégique prise par la direction du centre qui consiste à faire augmenter la capacité d'accueil, et par conséquent le nombre de patients examinés par jour. Ces informations contextuelles sont prises en compte à travers le paramètre de contexte « NombrePatientsExaminés/jour ». Ce paramètre est un paramètre de but qui provient du contexte interne (de la taxonomie de Rosemann). Les deux conditions de but (*Goal condition*) <100 et >100 (*cf. Gc1* et *Gc2* de la FigureIII.17) correspondent aux valeurs possibles de ce paramètre. La première condition convient à la première et à la deuxième version du processus d'*Examen*

Radiologique alors que la deuxième condition correspond à la troisième version de ce processus.

Les versions du processus d'*Examen Radiologique* différent au niveau de l'organisation du travail. En effet, la première et la troisième version correspondent au mode de travail ordinaire du centre, alors que la deuxième version concerne la prise en charge des urgences. De ce fait, le paramètre de processus « Urgence » permet de spécifier si le processus est urgent ou pas. Ce paramètre prend la valeur *Oui* dans le cas où le patient doit être examiné en priorité, et par conséquent le processus est qualifié d'urgent (cas de la deuxième version VP1-2), et il prend la valeur *Non* dans le cas contraire (cas de la première et la troisième versions VP1-1 et VP1-3).

De plus, dans la première et la deuxième version du processus d'*Examen Radiologique*, la secrétaire se sert de l'application web « App-G-CIM » pour effectuer les tâches relatives à la gestion des demandes d'examen (*Etude des demandes*, *Notification des patients* et *Attribution des RDV*) et le médecin rédige le rapport d'examen qui est un document papier donné au patient à la fin de l'examen. Dans la troisième version de ce processus, cette application est devenue une application Mobile. D'où l'automatisation des tâches *Etude des demandes*, *Notification des patients* et *Attribution des RDV*. Dans cette version, le document produit par la tâche *Rédaction Rapport d'examen* devient un document électronique.

Le contexte des tâches, des ressources et des informations de ces trois versions est décrit à travers les paramètres de contexte suivants : « ModeFonctionnement » (*BehavioralParameter)*, « TypeApplication » (*ResourceParameter*) et « TypeDocument » (D*ataParameter*). Ces paramètres proviennent du contexte immédiat (*cf.* taxonomie de Rosemann détaillée dans le chapitre II). De plus, les valeurs prises par ces paramètres décrivent le contexte d'utilisation de chaque version de processus. Ces valeurs correspondent aux conditions d'affectation des différentes versions de tâches, de ressources et d'informations contenues dans les définitions des versions de processus d'*Examen Radiologique*.

Par exemple, le contexte d'utilisation du *ResourceRole* « App-G-CIM » pour la réalisation de la tâche *Attribution de RDV* peut être défini à travers les instances *CRA1* et *CRA2* de la classe *Context of Resource Assignment.* En effet, deux versions de la tâche *Attribution de RDV* (*VA3-1* et *VA3-2*) et deux versions du *ResourceRole App-G-CIM* (*VRR1-1* et *VRR1-2*) ont été définies. La condition d'affectation *Est_une Application Web* (*Ac4*) correspond à la valeur du paramètre de contexte « TypeApplication ». Cette condition permet de spécifier que la version *VA3-1* est réalisée par la version du *RessourceRole VR1-1* lorsque *VR1-1* est une application web. Alors que la condition d'affectation *Ac3* précise que la version *VR1-2* est invoquée dans la réalisation de la version de tâche *VA3-2* lorsque *VR1-2* est une application mobile (*i.e.,* la valeur du paramètre « TypeApplication » est *Est_une ApplicationMobile*).

Concernant la tâche « Rédaction de rapport d'examen », deux versions ont été créées *VA5-1* et *VA5-2*. *VA5-1* appartient à la première version du processus d'*Examen Radiologique VP1-1,* alors que *VA5-2* est contenue dans la définition de la troisième version de ce processus. Le contexte des versions de cette tâche peut être défini avec les conditions d'affectation *Ac5* et *Ac6,* relatives au paramètre de contexte *TypeDocument,* définissent le contexte d'utilisation des versions *d'ItemAwareElement VI1-1* et *VI1-2* par les versions de tâches *VA5-1* et *VA5-2*. En effet, *Ac5* permet de spécifier que la version de la tâche *VA5-1* produit la version *VI1-1* lorsque *VI1-1* est un document papier. La condition *Ac6* précise que la version *VI1-2* est produite par la tâche *VA5-2* dans le cas où cette version est un document électronique.

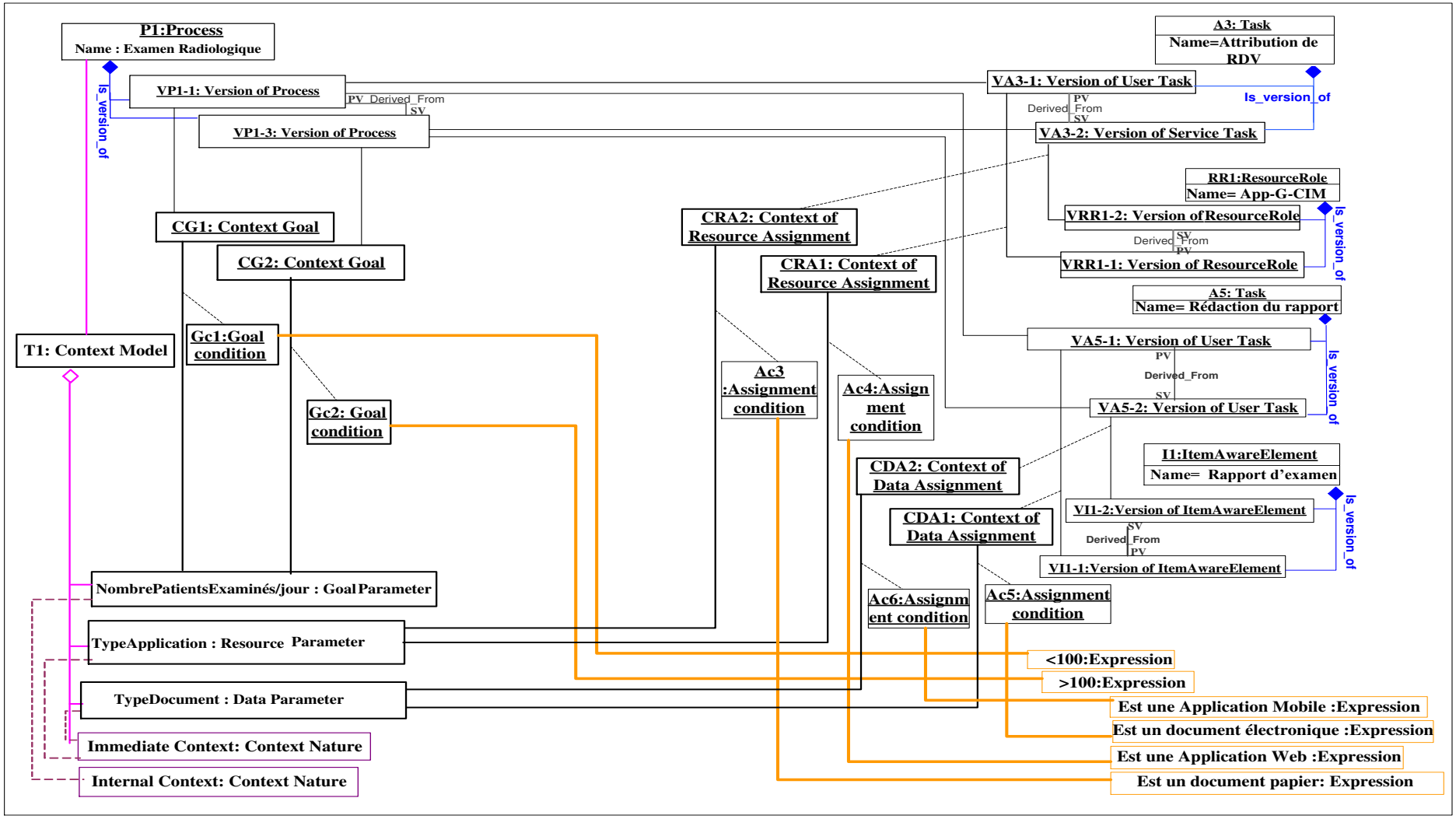


Figure III.17. Extrait de l'instantiation des contextes des versions du processus d'*Examen Radiologique*

IV. Conclusion

Les contributions de ce chapitre apportent des solutions au problème de la flexibilité des processus intra-organisationnels. En particulier, ces contributions offrent une solution conceptuelle pour la modélisation des processus privés flexibles. Cette solution est basée sur le standard BPMN 2.0 et sur les versions.

La première contribution de ce chapitre consiste à proposer le méta-modèle BPMN4V-PP (BPMN for Versions of private processes) qui représente une extension du méta-modèle de BPMN 2.0 en intégrant la technique de versionnement. Les instances du méta-modèle BPMN4V-PP sont des versions de processus privés composées par des versions d'activités, des versions d'événements, des versions de ressources et des versions d'informations. L'avantage de BPMN4V-PP est donc qu'il est capable de modéliser les processus sous forme de versions, ce qui a permis de garder trace de l'historique de l'évolution de ces processus sous forme d'une hiérarchie appelée hiérarchie de dérivation.

Cependant, l'ensemble des versions modélisées conformément à BPMN4V-PP nécessite une solution permettant la gestion de ces versions. D'où notre deuxième contribution qui consiste à proposer un diagramme d'états-transitions d'UML décrivant les états possibles d'une version ainsi que les différentes transitions reliant ces états. Ces transitions sont réalisables au travers d'un ensemble d'opérations permettent la création, la modification, la validation et la dérivation d'une version. L'avantage de cette contribution est qu'elle simplifie la gestion des versions de processus instances de BPMN4V-PP à travers la définition des opérations assurant la manipulation de ces versions.

Le méta-modèle BPMN4V-PP prend en compte les trois perspectives de modélisation des processus, à savoir la perspective comportementale, la perspective organisationnelle et la perspective informationnelle. Cependant, il ne considère pas la perspective contextuelle permettant de spécifier le pourquoi des processus. Nous avons donc proposé une deuxième extension de BPMN afin de prendre en compte cette autre dimension. D'où la troisième contribution de ce chapitre, le méta-modèle BPMN4V-Context. L'avantage de BPMN4V-Context est qu'il permet d'aider le concepteur à choisir, pour une situation donnée, la version adéquate d'un processus.

Dans le chapitre suivant, nous passons à l'étude de la modélisation des processus inter-organisationnels flexibles à l'aide de la technique de versionnement.

Chapitre IV – BPMN4V-CC : pour la modélisation de versions de processus inter-organisationnels

Résumé. *Après avoir traité la modélisation des versions des private process conformément à BPMN4V-PP dans le chapitre précédant, nous proposons dans ce chapitre une deuxième extension de BPMN, nommée BPMN4V-CC (BPMN for versions of collaborations and choreographies), qui permet de considérer des versions de collaborations et de chorégraphies. Cette extension a pour objectif (i) la modélisation et la gestion des versions de collaborations et (ii) la déduction des versions de chorégraphies à partir des versions de collaborations.*

Sommaire du Chapitre IV.

En plus des processus intra-organisationnels, BPMN 2.0 supporte la modélisation des processus qui traversent les frontières des organisations, appelés processus inter-organisationnels. Dans de tels processus, l'accent est mis sur la modélisation des interactions entre les organisations participantes. Plus précisément, BPMN offre deux types de diagrammes permettant la modélisation de ces interactions : 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 n'est apparu qu'avec la version 2.0 de BPMN.

Dans ce chapitre, nous étudions comment versionner des diagrammes de collaboration et de chorégraphie de BPMN 2.0 pour traiter le problème de flexibilité des processus inter-organisationnels. Plus précisément, les contributions de ce chapitre sont les suivantes : (i) le méta-modèle BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies) qui est une extension du méta-modèle de BPMN 2.0 pour modéliser les changements des collaborations, (ii) un ensemble d'opérations (de création, de modification, de validation et de dérivation) permettant la manipulation des versions de collaborations modélisées comme des instances du méta-modèle BPMN4V-CC, (iii) un ensemble de six patrons d'adaptation facilitant les opérations de création et de modification des versions de collaboration, et enfin (iv) un ensemble d'algorithmes pour la génération d'une version de chorégraphie à partir d'une version de collaboration étant donné qu'une chorégraphie n'est qu'une autre représentation d'une collaboration en se focalisant sur l'acheminement des messages.

La suite de ce chapitre est organisée comme suit. La section 1 détaille le méta-modèle BPMN4V-CC pour la modélisation des versions de collaboration. La section 2 présente les opérations applicables sur les versions de collaborations à savoir les opérations de création, de modification, de suppression, de validation et de dérivation de versions de collaborations. La section 3 propose six patrons d'adaptation pour faciliter la mise en œuvre des opérations de création et de modification des versions de collaborations. La section 4 explique comment nous pouvons déduire automatiquement une version de chorégraphie à partir d'une version de collaboration. Finalement, la section 5 fait une synthèse du chapitre et introduit le chapitre suivant.

I. Modélisation des versions de collaboration : le méta-modèle BPMN4V-CC

Cette section présente les extensions apportées à BPMN 2.0 pour supporter les processus inter-organisationnels flexibles. Dans un premier temps, elle présente un extrait du méta-modèle de BPMN 2.0 considérant les diagrammes de collaboration et de chorégraphie [OMG, 2011]. Dans un deuxième temps, elle présente le méta-modèle BPMN4V-CC qui est le résultat de la fusion du méta-modèle de BPMN 2.0 et le modèle de versionnement présenté dans la sous-section III.I.2). Enfin, cette section montre à l'aide de l'exemple du processus d'*Examen Radiologique*, revisité pour la circonstance, comment modéliser des versions de collaboration par instanciation du méta-modèle BPMN4V-CC.

I.1. Modélisation des collaborations et des chorégraphies avec BPMN 2.0

La Figure IV.1, présentée ci-dessous, montre un extrait du méta-modèle de BPMN 2.0 [OMG, 2011] illustrant les concepts qui permettent la modélisation des collaborations et des chorégraphies avec BPMN 2.0. En effet, une collaboration décrit les interactions entre deux ou plusieurs processus appartenant à des organisations différentes. Chaque organisation représente un participant qui peut être une entité *PartnerEntity* (*e.g.,* organisation) ou bien un rôle *PartnerRole* (*e.g.,* client). Un *Participant* est souvent responsable de l'exécution d'un processus *Process*. Un processus impliqué dans une collaboration est un *FlowElementContainer* qui peut contenir des *SequenceFlows* et des *FlowNodes* (*Gateways*, *Events*, *Tasks*). Plus précisément, ce processus représente une synchronisation d'un ensemble de tâches et d'événements en utilisant des *SequenceFlows* et des *Gateways*. En plus, les participants d'une collaboration sont amenés à envoyer et à recevoir des messages à travers des *MessageFlows.* Un *MessageFlow* représente un flux de message entre deux nœuds d'interaction *InteractionNodes*. Un nœud d'interaction représente un élément qui peut être la source (dans une relation *Send*) ou bien la cible (dans une relation *Receive*) d'un *MessageFlow.* Un nœud d'interaction peut être une tâche, un événement ou bien un participant. Un message porté par un *MessageFlow* possède un *ItemDefinition* qui décrit le type de l'information portée par ce message (*itemKind*) et sa structure (*structureRef*).

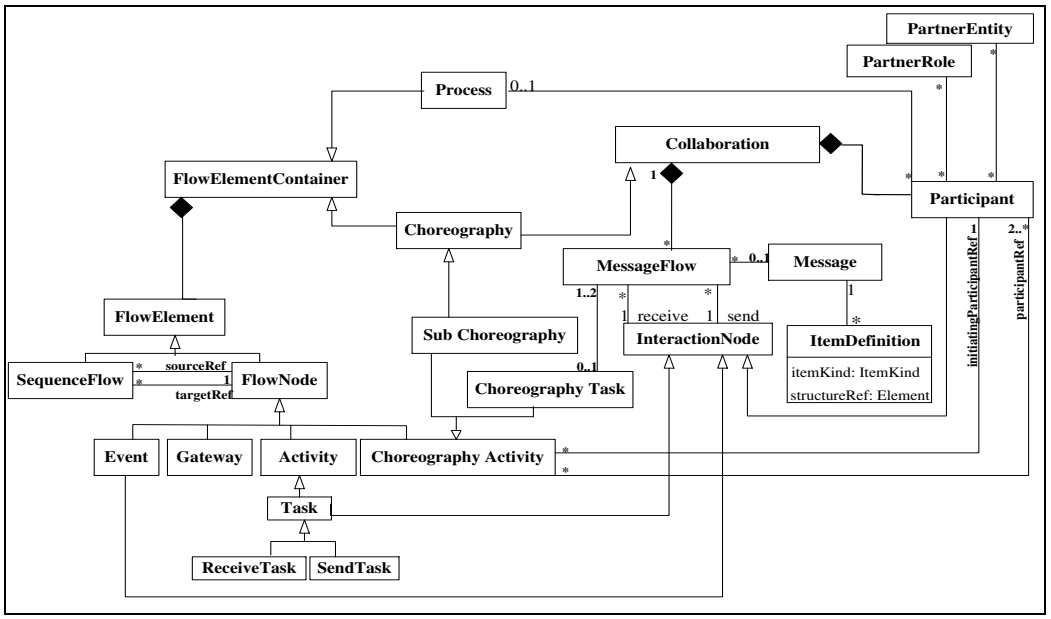


Figure IV.1. Extrait du méta-modèle de BPMN 2.0 pour les collaborations [OMG, 2011]

Il est à noter que les tâches et les événements des processus d'une collaboration forment les nœuds d'interaction. Ils définissent alors la partie publique de ces processus. Par conséquent, toutes les tâches et tous les événements de la partie privée ne sont pas visualisés.

Une chorégraphie est un *FlowElementContainer* qui peut contenir des *Sequenceflows* et des *FlowNodes* (*Event*, *Gateway* ou *ChoreographyActivity*). Une activité de chorégraphie *ChoreographyActivity* est un point du flux de la chorégraphie dans lequel une interaction se produit entre deux ou plusieurs participants. Elle peut être une *ChoreographyTask* ou bien une *SubChoreography*. Une *ChoreographyTask* est une activité atomique dans une chorégraphie qui représente une interaction dans laquelle au plus deux messages sont échangés (un message et éventuellement une réponse) entre deux participants. Une *SubChoreography* est une activité composite dans une chorégraphie qui contient un flux d'un ensemble d'activités de chorégraphie *ChoreographyActivities.*

I.2. BPMN4V-CC pour la prise en compte des versions de collaboration

Nous utilisons la technique de versionnement pour traiter la flexibilité des processus inter-organisationnels. Nous proposons ainsi d'utiliser le patron de versionnement, présenté dans la section I.2 du chapitre précédant, pour versionner certaines classes du méta-modèle de BPMN 2.0. La Figure IV.2 présente le méta-modèle BPMN4V-CC (BPMN for Versions of Collaborations and choreographies) résultat de la fusion de ce patron de versionnement avec le méta-modèle BPMN 2.0 pour les collaborations.

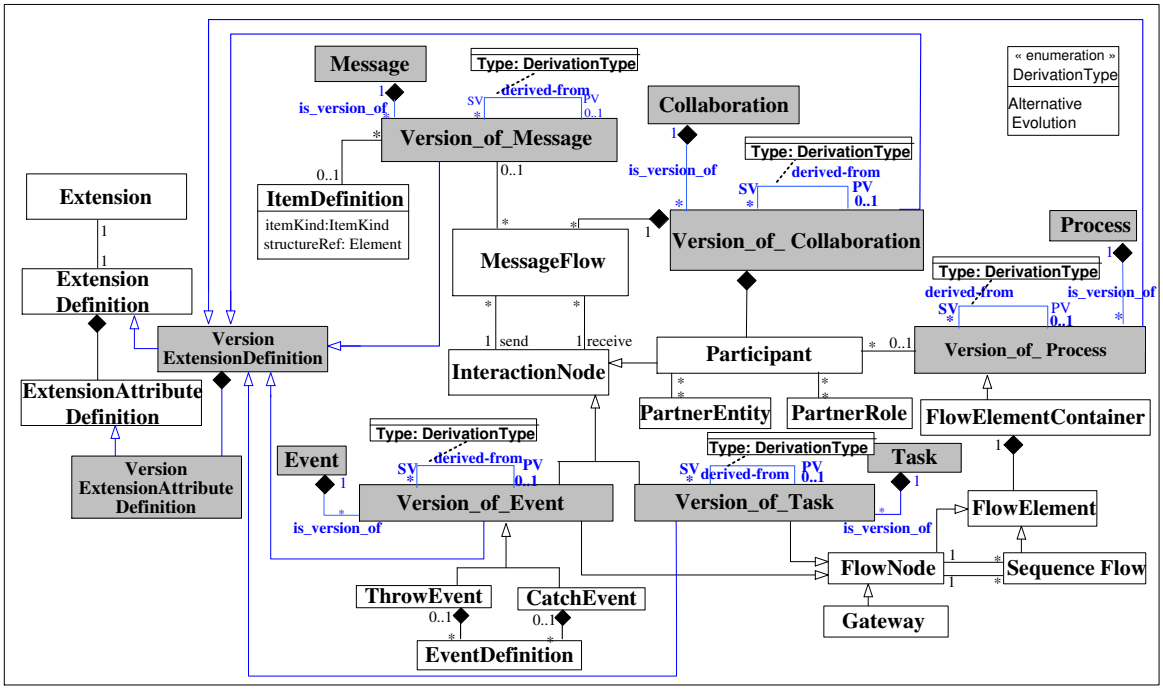


Figure IV.2. Méta-modèle BPMN4V-CC

Afin de prendre en compte les changements que subit un processus inter-organisationnel, nous avons décidé de versionner cinq classes issues du méta-modèle de BPMN 2.0, à savoir les classes *Collaboration*, *Message*, *Process*, *Task* et *Event*. Chacune de ces classes représente un concept clé d'une collaboration et joue un rôle important dans la définition d'un processus inter-organisationnel.

De manière générale, une nouvelle version d'un concept versionnable (*e.g., Collaboration)* est définie en fonction des changements qui peuvent se produire. Ces changements peuvent correspondre à l'adjonction de nouvelles informations (*i.e.,* propriété ou relation) ou bien à la modification ou la suppression d'informations existantes.

Plus précisément, concernant les messages, nous considérons que la modification des propriétés *itemKind* et / ou *itemStructure* entraine la création d'une nouvelle version de message. Par exemple, nous considérons, dans la Figure IV.3, la première version du message *M* (identifiée par VM1-1), représentant un document papier (la valeur de la propriété *itemKind* est *Physical*), envoyé entre la première version de la tâche T1 (identifiée par VA1-1) et la première version de la tâche T2 (identifiée par VA2-1). Suite à un changement technologique, ce message devient un document électronique envoyé par e-mail (*i.e.,* la valeur de la propriété *itemKind* est *Information).* Par conséquent, de nouvelles versions du message M et des tâches T1 et T2 doivent être créées.

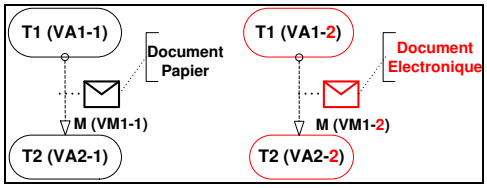


Figure IV.3. Exemple de changement entrainant la dérivation d'une version de message selon BPMN

Concernant les processus, nous créons une nouvelle version lorsqu'il y a des changements dans les tâches et / ou les événements impliqués dans ce processus, ou bien dans la manière dont ils sont coordonnés. De même, nous créons une nouvelle version d'une tâche ou d'un événement lorsqu'il y a des changements des messages qu'ils échangent. Par exemple, nous représentons, dans la Figure IV.4, la première version de la tâche d'envoi (*SendTask*) T1 (identifiée par VA1-1) qui envoie la première version du message M1 (identifiée par VM1-1) à la première version de la tâche de réception (*ReceiveTask)* T2 (identifiée par VA2-1). Suite à un changement organisationnel, T1 reçoit le message M2 de T3 (identifié par VA3-1) puis elle envoie le message à T2. Dans ce cas, une nouvelle version de la tâche T1 doit être créée (VA1-2) puisque elle est devenue une tâche générique (*Task*) qui reçoit un message et envoie un autre.

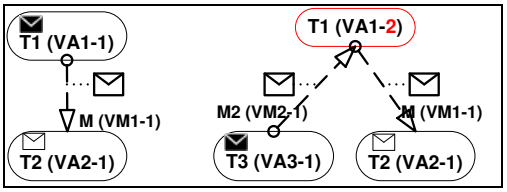


Figure IV.4. Exemple de changement entrainant la dérivation d'une version de Tâche

Quant aux collaborations, tous les changements qui touchent les processus partenaires entraînent la création d'une nouvelle version de collaboration. En effet,

l'adjonction ou la suppression d'un participant à la collaboration nécessite une adaptation des processus déjà existants. Il devient important d'intégrer le nouveau participant ou de pallier l'absence du participant supprimé. D'autre part, une nouvelle version de collaboration peut être créée suite aux changements des processus ou bien des messages échangés dans cette collaboration. Etant donné que ces messages échangés jouent un rôle important dans le flux d'une collaboration, tout changement dans un message envoyé ou reçu affecte les tâches et / ou les événements impliqués dans ce message, et par conséquent le processus concerné. L'adjonction (respectivement la suppression) d'un message dans une collaboration entraîne éventuellement l'adjonction (respectivement la suppression) de son nœud d'envoi et de son nœud de réception, ce qui entraîne la modification des schémas des processus partenaires concernés par ce changement. Par exemple, considérons dans la Figure IV.5 la première version de la collaboration C (VC1-1) composée des premières versions des processus A (VP1-1) et B (VP2-1). L'adjonction de la première version du message M3 dans la collaboration C nécessite l'insertion de deux nouvelles Tâches A3 (VA5-1) et B3 (VA6-1), d'où la modification des schémas des processus A et B. Par conséquent nous définissions la deuxième version de la collaboration C (VC1-2) dérivée à partir de VC1-1, ainsi que, les deuxièmes versions des processus A (VP1-2) et B (VP2-2) dérivées respectivement de VP1-1 et VP2-1.

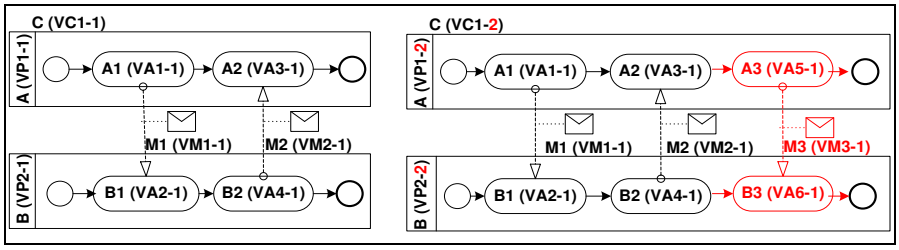


Figure IV.5. Exemple de changement entraînant la dérivation d'une collaboration

De même que le méta-modèle BPMN4V-PP, toutes les classes versionnables du méta-modèle BPMN4V-CC héritent de la classe VersionExtensionDefinition composée par un ensemble d'attributs décrivant le détail des versions dans la classe VersionExtensionAttributeDefinition.

I.3. Exemple : le processus d'*Examen Radiologique*

Nous illustrons dans ce qui suit comment nous pouvons modéliser des versions de collaboration par instanciation du méta-modèle BPMN4V-CC. Pour ce faire, nous reprenons l'exemple du processus d'examen du centre de radiologie présenté dans le chapitre précédant et nous considérons que le centre radiologique et la clinique sont deux organisations différentes travaillant en collaboration. A travers cette collaboration, nous décrivons comment un patient hospitalisé dans une clinique peut bénéficier des services offerts par le centre de radiologie. Nous introduisons également différents scénarii de changements qui peuvent mener à la modification de cette collaboration (*i.e.,* créer de nouvelles versions). La première version de la collaboration *Examen Radiologique* (identifiée par VC1-1) est donnée dans la Figure IV.6. Cette

version implique la première version du processus de la clinique (identifiée par VP1-1) et la première version du processus du centre de radiologie (identifiée par VP2-1). VC1-1 débute lorsqu'un patient hospitalisé dans la clinique a besoin d'un *Examen Radiologique*. Tout d'abord, la clinique envoie au centre de radiologie une demande d'examen, d'où la définition de la première version de la tâche *Envoi demande d'examen* (identifiée par VA1-1) et la première version de la tâche *Réception demande d'examen* (identifiée par VA2-1). Après vérification, la demande peut être rejetée. Dans ce cas, le centre renvoie à la clinque une notification qui explique les causes du refus en utilisant la première version de la tâche *Envoi notification de rejet* (identifiée par VA3-1) et la première version de la tâche *Réception notification de rejet* (identifiée par VA5-1). Dans le cas où la demande est acceptée, le centre de radiologie notifie la clinique de la date d'examen, d'où la définition de la première version de la tâche *Envoi date de RDV* (identifiée par VA4-1) et *Réception date de RDV* (identifiée par VA6-1). A la date fixée, le patient est transporté (la première version de la tâche *Transport Patient* identifiée par VA7-1) au centre pour être examiné. Durant l'examen, le médecin radiologue peut interviewer le patient, demander et consulter son historique pathologique avant de réaliser les clichés souhaités. Après l'examen, le médecin rédige et envoie le rapport d'examen à la clinique en utilisant la première version de la tâche *Envoi rapport d'examen* (identifiée par VA9-1) et la première version de la tâche *Réception demande d'examen* (identifiée par VA10-1). Il est à noter que toutes les tâches de cette version de collaboration sont des tâches manuelles et que tous les documents échangés sont des documents papiers. Par exemple, le rapport envoyé suite à l'*Examen Radiologique* est un document papier représentant la première version du message *Rapport d'examen* (identifiée par VM5-1).

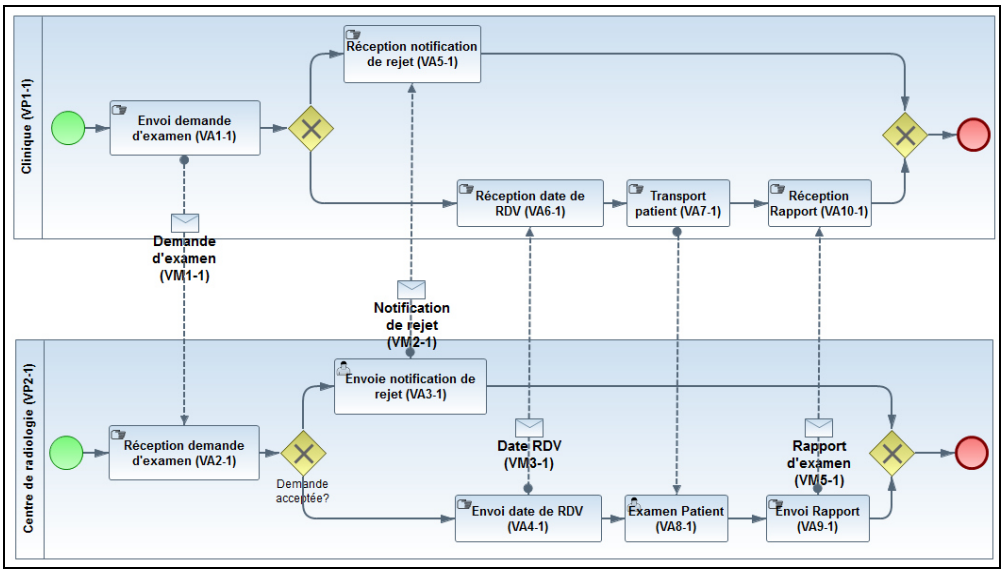


Figure IV.6. Première version de la collaboration *Examen Radiologique*

La Figure IV.7 montre la deuxième version de cette collaboration, identifiée par VC1-2. Cette version, dérivée à partir de la première version (VC1-1), est utilisée lorsque le patient ne peut pas être transporté au centre pour être examiné. Dans ce cas,

un technicien radiologue transporte le matériel du centre vers la clinique pour effectuer les clichés radiologiques dans la chambre du patient. Il est à noter que dans cette version le technicien ne réalise pas un examen complet du patient. Il va juste réaliser les clichés demandés par le médecin. De ce fait, VC1-2 implique la deuxième version du processus de la clinique (identifiée par VP1-2 et dérivée à partir de VP1-1) et la deuxième version du processus du centre de radiologie (identifiée par VP2-2 et dérivée à partir de VP2-1). La dérivation de ces deux versions de processus est effectuée suite au changement de leurs définitions, et ceci (i) en remplaçant la première version de la tâche *Transport Patient* par la première version de la tâche *Transport matériel* (identifiée par VA11-1) dans la version du processus VP2-2, et (ii) en insérant la première version de la tâche *Réalisation des clichés* (identifié par VA12-1) dans la version de processus VP1-2.

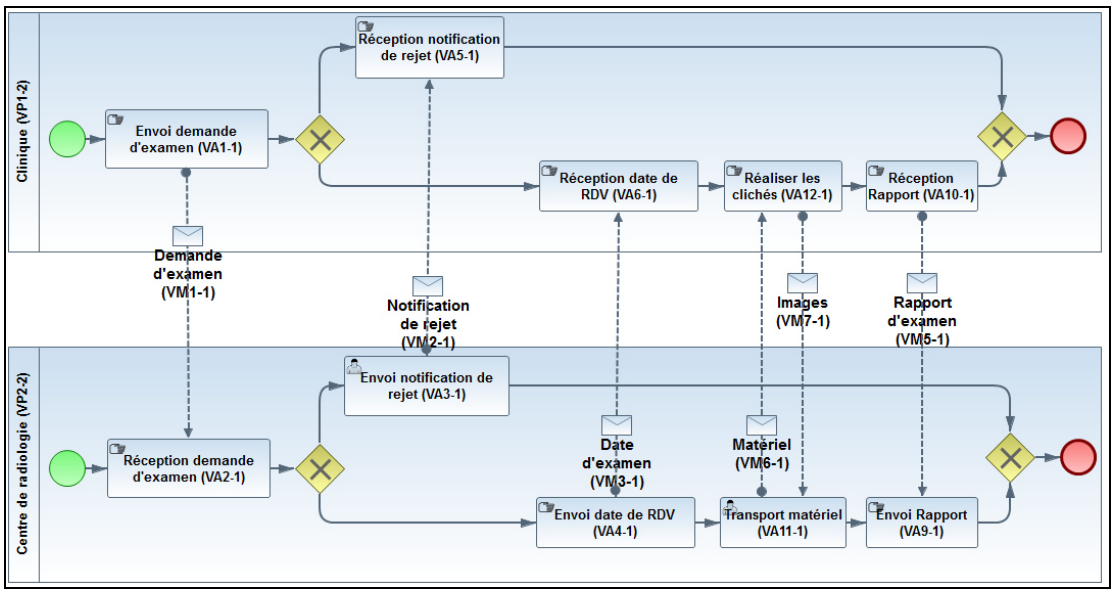


Figure IV.7. Deuxième version de la collaboration *Examen Radiologique*

Dans le but d'améliorer la qualité de leurs services, le centre et la clinique mettent en œuvre une application web qui supporte l'automatisation de leurs communications. Ainsi, une nouvelle version de la collaboration, identifiée par VC1-3 et dérivée à partir de VC1-1, est définie, comme montré dans la Figure IV.8. Dans cette version, les messages sont transmis d'une façon électronique à travers l'application Web suite à l'automatisation des tâches d'envoi et de réception de cette version de collaboration. Plus précisément, la version VC1-3 est composée de la troisième version du processus de la clinique (identifiée par VP1-3 et dérivée à partir de VP1-1) et la troisième version du processus du centre (identifiée par VP2-3 et dérivée à partir de VP2-1). Ces deux versions de processus ont été dérivées suite au changement de leurs schémas. En effet, VP1-3 est composée de la deuxième version de la tâche *Envoi demande d'examen* (identifiée par VA1-2 et dérivée à partir de VA1-1), la deuxième version de la tâche *Réception notification de Rejet* (identifiée par VA5-2 et dérivée à partir de VA5-1), la deuxième version de la tâche *Réception date de RDV* (identifiée par VA6-2 et dérivée à partir de (VA6-1), la première version de la tâche *Transport*

patient (identifiée par VA7-1), la première version de la tâche *Fournir informations complémentaires* (identifiée par VA14-1) et la deuxième version de la tâche *Réception Rapport* (identifiée par VA10-2 et dérivée à partir de VA10-1). Quant à VP2-3, elle est composée de la deuxième version de la tâche *Réception demande d'examen* (identifiée par VA2-2 et dérivée à partir de VA2-1), la deuxième version de la tâche *Envoi notification de Rejet* (identifiée par VA3-2 et dérivée à partir de VA3-1), la deuxième version de la tâche *Envoi date RDV* (identifiée par VA4-2 et dérivée à partir de VA4-1), la première version de la tâche *Examen* (identifiée par VA8-1), la première version de la tâche *Demande informations complémentaires* (identifiée par VA13-1) et la deuxième version de la tâche *Envoi Rapport* (identifiée par VA9-2 et dérivée à partir de VA9-1). Par ailleurs, la version de collaboration VC1-3 comporte les deuxièmes versions des messages *Demande d'examen* (identifiée VM1-2 dérivée à partir de VM1-1), *Notification de rejet* (identifiée par VM2-2 dérivée à partir de VM2-1), *Date de RDV* (identifiée par VM3-2 dérivée de VM3-1) et *Rapport d'examen* (identifiée par VM5-2 et dérivée à partir de la version VM5-1). La dérivation de ces versions de message est due suite à la modification de leurs structures, puisque l'envoi et la réception se font d'une façon automatique à travers l'application web mise en œuvre.

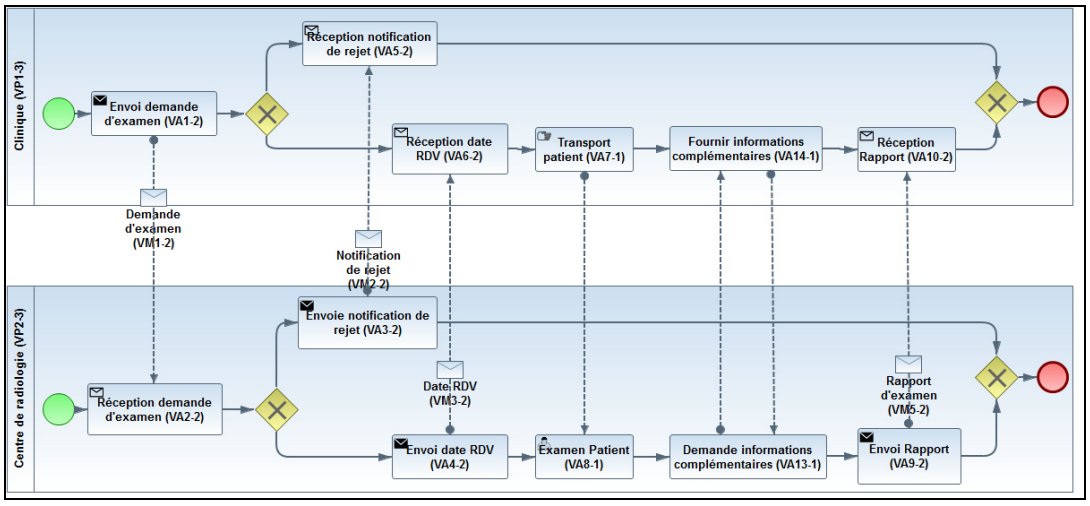


Figure IV.8. Troisième version de la collaboration *Examen Radiologique*

En se référant à la taxonomie de Reichert [Reichert et Weber, 2012], nous considérons que la deuxième version de la collaboration *Examen Radiologique* VC1-2 (présentée dans la Figure IV.7) représente une variante par rapport au déroulement normal de cette collaboration définie dans la première version (présentée dans la Figure IV.6). Il s'agit alors du traitement d'un besoin de variabilité se rapportant aux patients dont l'état de santé ne permet pas leur transport au centre. La troisième version VC1-3 (présentée en Figure IV.8) est définie suite à un besoin d'évolution de la première version afin de tenir compte des nouvelles technologies mises en place par les deux partenaires. La Figure IV.9 donne les hiérarchies de dérivation de la collaboration *Examen Radiologique*, des processus du centre de radiologie et de la clinique, impliqués dans cette collaboration, des tâches *envoi et réception rapport d'examen* et du message *Rapport d'examen*.

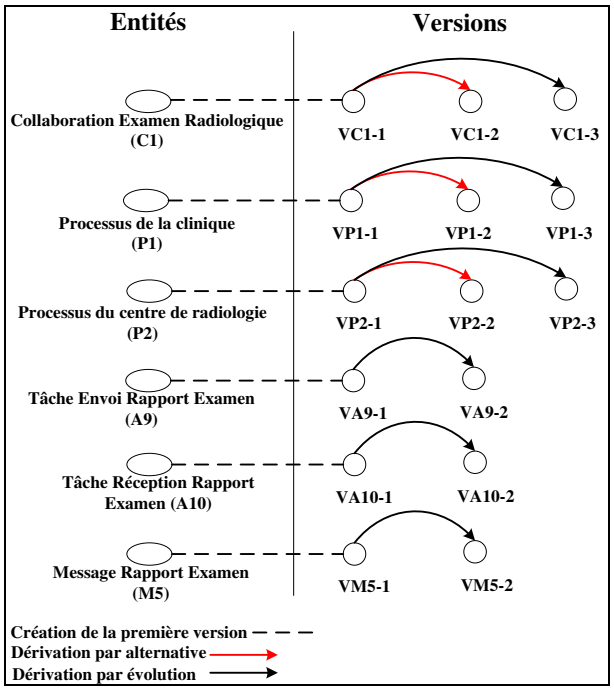


Figure IV.9. Hiérarchie de dérivation de la collaboration *Examen Radiologique*

Les résultats de la modélisation des trois versions de la collaboration *Examen Radiologique* sont mémorisés par instanciation du méta-modèle BPMN4V-CC. La Figures IV.10, présentée ci-dessous, est un diagramme d'objet UML illustrant l'instanciation de ce méta-modèle. Pour des raisons de lisibilité, la Figure IV.10 se limite à présenter une instanciation de la première et la troisième version de la collaboration *Examen Radiologique*. Une instanciation complète est donnée en Annexe A1.

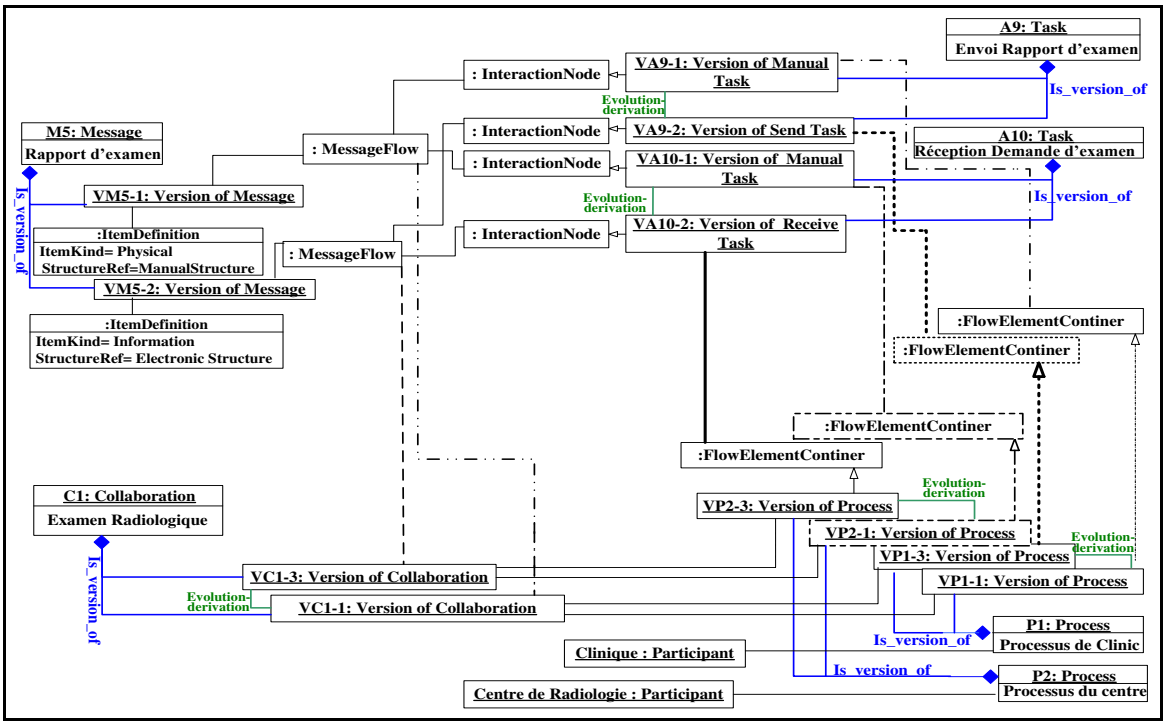


Figure IV.10. Extrait de l'instanciation de la première et de la troisième versions de la collaboration examen radilogique.

II. Dynamique des versions de collaboration

Cette section décrit les aspects dynamiques de BPMN4V-CC. Ce problème a été abordé dans le chapitre précédent pour le cas des processus intra-organisationnels, en proposant un Diagramme d'Etats-Transitions (DET) UML décrivant les différents états possibles que peut prendre une version et les différentes opérations permettant la transition d'un état à un autre (*cf.* Chapitre III § II.1). Dans ce chapitre, nous adoptons le même principe pour gérer les versions des processus inter-organisationnels : nous nous basons sur le même DET pour définir le cycle de vie des versions des concepts versionnables d'une collaboration. Plus précisément, la première version est créée en utilisant l'opération de création. La version créée, avec un état « *En travail (T)* », peut être modifiée en utilisant l'opération de modification. Par la suite, cette version peut être soit supprimée, soit validée et passée à l'état « *Stable* ». L'état « *Stable* » est un état final pour une version : elle ne peut plus être modifiée, mais évidemment peut être instanciée (*i.e.,* servir de support à l'exécution de cas). Cette version peut également servir de base pour la création d'une nouvelle version en utilisant l'opération de dérivation. La version obtenue, ayant l'état « *En travail* », possède la même définition que la version à partir de laquelle elle a été dérivée.

Les opérations de gestion des versions de collaboration permettent alors la création, la modification, la suppression, la validation et la dérivation d'une version. Dans ce qui suit, nous détaillons chacune de ces opérations [Ben Said *et al.,* 2015(b)].

II.1. L'opération de création d'une version

L'opération de création permet la création d'une entité versionnable ainsi que sa première version. Cette opération est supportée par un ensemble de primitives correspondant à des actions de bas niveau. Ces primitives, détaillées dans le tableau IV-1, diffèrent en fonction du concept BPMN sur lequel elles portent.

Tableau IV-1. Primitives de l'opération de création (cas des collaborations)

Légende + : adjonction

Elément BPMN4V-CC	**Primitives**
Collaboration	+Participant/process + Message/MessageFlow
Participant/process	+Task + Event + SequenceFlow + Gateway
Event	+ EventDefinition
Message	+ItemDefinition

Une collaboration (ainsi que sa première version) est créée via des primitives permettant l'adjonction (+) des processus et des participants (*i.e., Process* et *Participant*) et des interactions entre ces participants (*i.e., Message* et *MessageFlow*). De la même manière, la création d'un processus participant à une version de collaboration est assurée à travers des primitives permettant l'adjonction (i) des tâches (*i.e., Task*) et (ii) des événements (*i.e., Event*) qui le composent ainsi que (iii) le schéma de coordination de ces tâches et événements. Le schéma de coordination est défini par des *SequenceFlows* et des *Gateways*. Concernant la création des événements et des messages, elle inclut des primitives permettant respectivement la définition de la propriété *EventDefinition* de l'événement et de la propriété *ItemDefinition* du message.

II.2. L'opération de modification d'une version

L'opération de modification permet de modifier une version ayant l'état « *En travail* ». De même que pour l'opération de création, cette opération est spécifiée en utilisant des primitives. Le tableau IV-2 présente les différentes primitives de modification de chaque concept versionnable de BPMN4V-CC.

En effet, une version de collaboration peut être modifiée via des primitives permettant l'adjonction (+) ou la suppression (-) des versions de processus participants et / ou des interactions entre ces participants (*i.e., Versions de Message* et *MessageFlow*). De la même manière, la modification d'une version de processus participant à une version de collaboration est assurée à travers des primitives permettant l'adjonction et / ou la suppression (i) des versions de tâches, (ii) des versions d'événements et (iii) du schéma de coordination de ces versions de tâches et d'événements. Le schéma de coordination est défini par des *SequenceFlows* et des *Gateways*. Concernant la modification des tâches, des événements et des messages, elle repose sur des primitives permettant respectivement la définition du type de la tâche, la définition de la propriété *EventDefinition* de l'événement et la définition de la propriété *itemDefinition* du message.

Tableau IV-2. Primitives de l'opération de modification (cas des collaborations)

Légende + : adjonction ; - suppression

Elément BPMN4V-CC	Primitives
Version_of_Collaboration	+/- Participant/process +/- Message/MessageFlow
Version_of_process	+/- Task +/- Event +/- SequenceFlow +/- Gateway
Verion_of_Event	Modifier EventDefinition (supprimer l'EventDefinition existant et le remplacer par un autre EventDefinition)

Version_of_Task	Modifier Type (supprimer le type existant et le remplacer par un nouveau type)
Version_of_Message	Modifier ItemDefinition (supprimer les valeurs des propriétés itemKind et itemStructure et les remplacer par d'autres valeurs)

II.3. L'opération de validation

L'opération de validation permet de faire passer une version de l'état « *En travail* » à l'état « *Stable* ». Nous rappelons qu'une version « *Stable* » ne peut pas subir de modifications. De plus, la validation d'une version peut déclencher la validation d'autres versions reliées à cette version lorsqu'elles ne sont pas déjà validées. En effet, lorsque le concepteur souhaite valider une version de collaboration, c'est qu'en réalité il souhaite valider la collaboration avec toutes les versions qui la composent (*i.e.,* ses processus, ses messages, ses tâches et ses événements). La Figure IV.11 illustre la propagation de la validation. Dans cette figure, les flèches noires correspondent aux validations initiales (*i.e.,* la validation par l'application directe de l'opération de validation) alors que les flèches rouges correspondent aux validations par propagation.

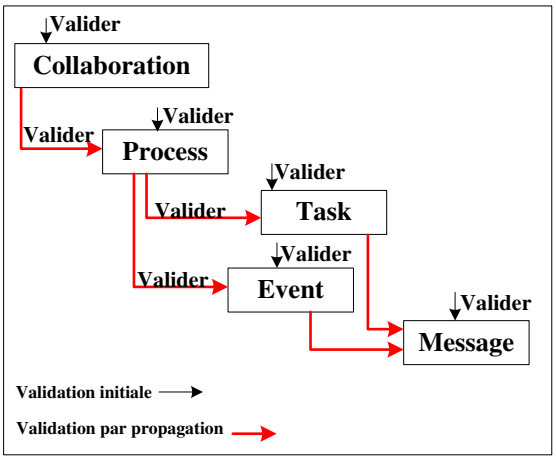


Figure IV.11. Propagation de la validation

En effet, la validation d'une version de collaboration peut entraîner la validation des versions des processus qui la composent. De même, la validation d'une version d'un processus peut entraîner la validation de ses composants versionnables, *i.e.,* ses versions de tâches et ses versions d'événements. De la même manière, la validation d'une version de tâche (ou d'événement) peut entrainer la validation de la version de message envoyée par cette version.

II.4. L'opération de dérivation

Cette opération permet la création d'une nouvelle version à partir d'une version existante « *Stable* ». La version créée est dans l'état « *En travail* ». Avant d'être modifiée, elle possède la même définition que la version à partir de laquelle elle est dérivée. De plus, la dérivation d'une version peut déclencher la dérivation d'autres

versions. La Figure IV.12 illustre la propagation de la dérivation. Cette propagation est due aux relations de composition qui existent entre les classes du méta-modèle BPMN4V-CC : *Collaboration*, *Message*(*MessageFlow*), *Process* (*Participant*), *Task* et *Event*. Par conséquent, la dérivation d'une version de message entraine la dérivation des tâches ou des événements d'envoi et de réception de ce message. En plus, la dérivation d'une version de tâche ou d'événement entraîne la dérivation de la version du processus correspondante. De même, la dérivation d'une version de processus entraine la dérivation de la version de collaboration correspondante. Dans cette figure, nous avons adopté la même convention que dans la figure précédente : les flèches noires correspondent aux dérivations initiales alors que les flèches rouges correspondent aux dérivations par propagation.

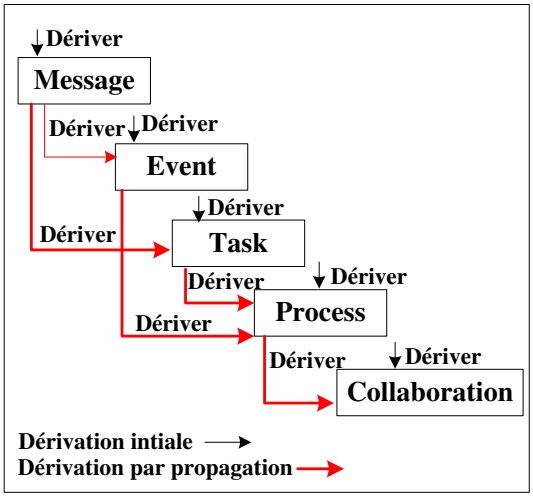


Figure IV.12. Propagation de la dérivation

III. Patrons d'adaptation des versions de collaboration

Afin de faciliter l'utilisation des opérations de création et de modification d'une version de collaboration, nous proposons six patrons d'adaptation. L'élément central considéré dans ces patrons est l'interaction d'une collaboration. Une interaction (*cf.* la Figure IV.13) est un triplé composé par un *Messageflow* portant un message, un nœud d'envoi et un nœud de réception de ce message. Les nœuds d'envoi et de réception sont appelés des nœuds d'interaction (*InteractionNode*) qui peuvent être une tâche ou bien un événement [Ben Said *et al.,* 2015(b)].

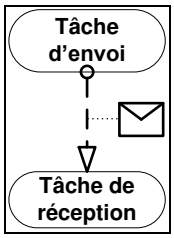


Figure IV.13. Interaction dans une collaboration

La Figure IV.14 présente les patrons d'adaptation que nous proposons pour la modification des schémas des versions de collaboration. Ces patrons sont inspirés des patrons d'adaptation proposés dans [Weber *et al.,* 2013] pour modifier le schéma d'un processus intra-organisationnel. Les patrons que nous proposons sont les suivants : ajouter une interaction, supprimer une interaction, déplacer une interaction, remplacer une interaction, échanger une interaction et modifier une interaction.

CPAP1: ajouter une interaction	**CPAP4**: remplacer une interaction
CPAP2: supprimer une interaction	**CPAP5**: échanger des interactions
CPAP3: déplacer une interaction	**CPAP6**: modifier une interaction

Figure IV.14. Vue d'ensemble des patrons d'adaptation d'une collaboration

Dans la suite de la section, nous expliquons, tout d'abord, le principe général d'utilisation des patrons d'adaptation. Puis, nous détaillons chacun de ces patrons en présentant les différents cas qui se déclinent de chaque patron ainsi que les algorithmes implémentant ces cas.

III.1. Principe général d'utilisation des patrons d'adaptation

Un patron d'adaptation peut être utilisé soit pour créer une première version de collaboration, soit pour modifier une version existante (obtenue par dérivation et étant dans l'état « *En travail* »). Dans ces deux cas, l'utilisation des patrons permet de passer une version de collaboration d'un schéma initial à un nouveau schéma. Ce changement est le résultat de : (i) l'adjonction, (ii) la suppression, (iii) le remplacement, (iv) le déplacement, (v) l'échange et / ou (vi) la modification d'une ou de plusieurs interactions de cette version de collaboration.

Un patron d'adaptation d'une collaboration « *En travail* » peut déclencher des opérations de dérivation des éléments qui la composent (*i.e.,* les versions de processus, les versions de tâches, les versions de messages et / ou les versions des événements) s'ils sont à l'état « *Stable* ».

Afin de mieux expliquer le principe d'utilisation des patrons d'adaptation, considérons l'exemple de la Figure IV.15. Dans cet exemple, la première version de la collaboration C est identifiée par VC1-1. Cette version est une version « *Stable* » qui invoque les premières versions des processus A (identifiée par VP1-1) et B (identifiée par VP2-1). La dérivation de VC1-1 conduit à la création d'une nouvelle version de collaboration identifiée par VC1-2, ayant le même schéma que VC1-1 et ayant l'état « *En travail* ». Cette nouvelle version est à modifier, par la suite, via les patrons d'adaptation. On a utilisé dans cet exemple, un patron pour l'adjonction d'une nouvelle interaction entre la tâche A2 et une nouvelle tâche B3. Par conséquent, l'utilisation de ce patron entraine (i) la définition de VA2-2, la deuxième version de la tâche A2, dérivée à partir de la première version identifiée par VA2-1, (ii) l'adjonction d'une nouvelle tâche B3 et sa première version identifiée par VA5-1, (iii) l'adjonction d'une nouvelle version de message entre VA2-2 et VA5-1, et (iv) la définition des nouvelles

versions des processus A et B (VP1-2 dérivées de VP1-1 et VP2-2 dérivées de VP2-1). La séquence des flux entre ces éléments est également modifiée. Les modifications effectuées à l'aide des patrons d'adaptation sont indiquées en rouge.

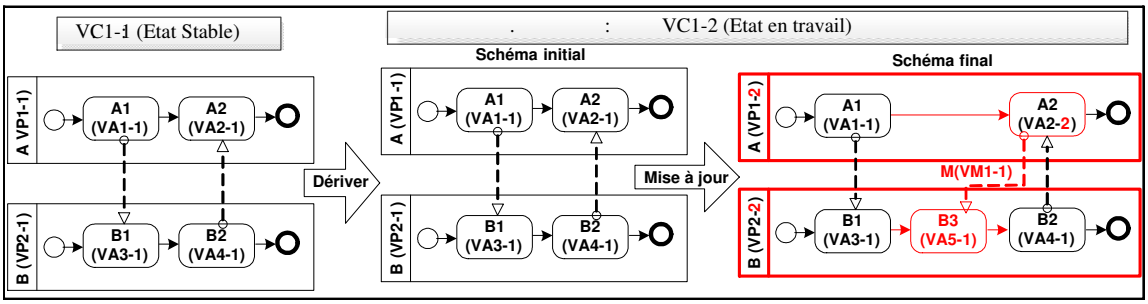


Figure IV.15. Principe d'utilisation d'un patron d'adaptation

Comme nous l'avons détaillé dans les tableaux IV-1 et IV-2, les opérations de création et de modification d'une version de collaboration peuvent être réalisées en utilisant un ensemble d'actions élémentaires dites aussi primitives de bas niveaux. Les patrons d'adaptation viennent pour simplifier ces deux opérations par la définition de primitives de haut niveau qui regroupent un ensemble de primitives de bas niveau. Ces primitives correspondent aux méthodes des classes versionnables du méta-modèle BPMN4V-CC. Ces méthodes sont détaillées dans le tableau IV-3.

Tableau IV-3. Méthodes des classes versionnables du méta-modèle BPMN4V-CC

Classes	Méthodes
Version_of_Collaboration	estStable() : indique si la version de collaboration est en état Stable ou non.
	ajouterMessage(M) : ajoute le message M dans la version de collaboration.
	ajouterMessageFlow(M,E,R) : ajoute un *MessageFlow* qui porte le message M et qui relie le nœud d'envoi E et le nœud de réception R.
	supprimerMessage(M) : supprime le message M de la version de collaboration.
	getMessageFlow(M,E,R) : renvoie le *MessageFlow* qui relie les nœuds E et R et qui porte le message M.
	supprimerMessageFlow(M,E,R) : supprime un *MessageFlow* qui porte le message M et qui relie le nœud d'envoi E et le nœud de réception R.
	estStable() : indique si la version de processus est en état Stable ou non.
	supprimerSequenceFlow(E,R) : permet la suppression du *SequenceFlux* entre les *FlowNodes* E et R d'une version de processus.

Version_of_Process	ajouterSequenceFlow(E,R) :permet d'ajouter le *SequenceFlow* entre les *FlowNodes* E et R d'une version de processus.
	derive() : permet la dérivation d'une version du processus.
	insérerNoeud(N) : permet l'insertion du nœud N dans une version de processus.
	insérerGatewayParallel () : permet l'insertion d'un Gateway parallèle dans une version de processus.
	insérerGatewayConditional() : permet l'insertion d'un Gateway conditionnel dans une version de processus.
	ajouterNoeud (T,S,FNp,FNs) : permet l'insertion du nœud d'interaction T entre les deux FlowNode FNp et FNs (FNp le nœud précédant de T et FNs le nœud suivant de T). La manière dont le nœud inséré est coordonnée avec les autres nœuds est indiquée par le paramètre S qui peut prendre les valeurs suivantes : séquence, parallèle ou conditionnelle.
	supprimerNoeud(T) : permet de supprimer le nœud T de la version de processus.
InteractionNode (Version_of_Task ou version_of_event)	estStable() : indique si la version de tâche (ou la version d'événement) est dans l'état Stable ou non.
	derive() : permet de dériver la version de tâche (ou la version d'événement).
	calculerNbMsg() : permet de calculer le nombre de messages échangés par la version de tâche (ou la version d'événement).
	getPrecedent() : permet de renvoyer le nœud qui précède la version de tâche (ou la version d'événement).
	getSuivant() : permet de renvoyer le nœud qui suit la version de tâche (ou la version d'événement).
Message	estStable() : indique si la version de message est dans l'état Stable ou non.
	derive() : permet de dériver la version message
	setItemDefinition(id) : affecte l'ItemDefinition id à la version de message.

III.2. Patron d'adaptation « ajouter une interaction » (Add Interaction)

Le patron « ajouter une interaction » permet la modification d'une version de collaboration par l'adjonction d'une nouvelle interaction. Nous distinguons quatre cas possibles pour réaliser ce patron :

- Cas#1 ajouter un message ainsi que ses nœuds d'envoi et de réception.
- Cas#2 ajouter un message et un nœud d'envoi et utiliser un nœud de réception existant.
- Cas#3 ajouter un message et un nœud de réception et utiliser un nœud d'envoi existant.
- Cas#4 ajouter un message entre des nœuds d'envoi et de réception existants.

Nous nous limitons dans cette section à présenter le premier cas (cas#1). Les trois autres cas sont présentés dans l'annexe A2.

Cas#1 : ajouter un message, son nœud d'envoi et son nœud de réception

Le cas#1 du patron « Ajouter une interaction » permet d'ajouter un message, le nœud d'envoi et le nœud de réception de ce message. Ainsi, nous devons, tout d'abord, ajouter les deux nouveaux nœuds dans les processus participants correspondants. Puis nous devons définir un nouveau message et le *MessageFlow* reliant les nouveaux nœuds, qui le porte.

En fonction du type de routage utilisé pour coordonner les nouveaux nœuds insérés dans les processus de la collaboration, nous distinguons les trois options suivantes : insertion en séquence, insertion parallèle et insertion conditionnelle. Nous présentons dans ce qui suit une description de ces options ainsi que les algorithmes qui les implémentent.

✓ Description des options du cas#1

La Figure IV.16 illustre le cas#1 du patron « ajouter une interaction » sur un exemple abstrait. Dans cet exemple, nous présentons les différentes options d'insertion des nouveaux nœuds dans les processus de la collaboration. Chacune de ces options permet de : (i) dériver les processus participants (s'ils sont à l'état « *Stable* ») (ii) insérer la première version de la tâche d'envoi A3 (identifiée par VA5-1) et la première version de la tâche de réception B3 (identifiée par VA6-1) dans les deuxièmes versions des processus A (identifiée par VP1-2) et B (identifiée par VP2-2), (iii) définir la première version du message M (identifiée par VM1-1), et (iv) définir un *MessageFlow* portant le message M et reliant les versions de tâches VA5-1 et VA6-1.

La première option correspond à une insertion en séquence des versions de tâches VA5-1 (première version de A3) et VA6-1 (première version de B3) respectivement dans les versions des processus VP1-2 (deuxième version de A) et VP2-2 (deuxième version de B), tandis que la deuxième correspond à une insertion parallèle et la troisième option correspond à une insertion conditionnelle. Dans la Figure IV.16, les modifications effectuées suite à l'utilisation du cas#1 du patron « ajouter une interaction » sont présentées en rouge.

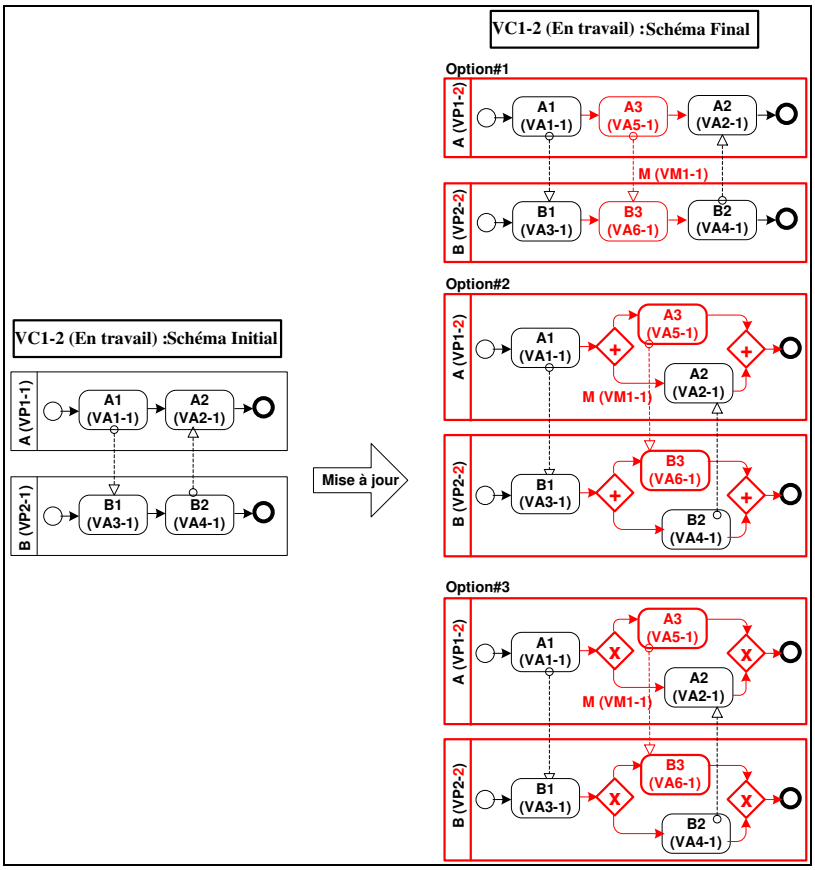


Figure IV.16. Cas#1 du patron « ajouter une interaction »

✓ Algorithmes du cas#1

Afin de mieux expliciter le cas#1 du patron « ajouter une interaction », nous détaillons dans ce qui suit l'algorithme de la méthode ajouterInteractionCas#1 implémentant ce cas. Cet algorithme fait appel aux méthodes des classes versionnables du méta-modèle BPMN4V-CC décrites dans le tableau IV-3. Plus précisément, cet algorithme utilise la méthode « ajouterNoeud(T,S,FNp,FNs) » de la classe *Version of process* qui permet d'ajouter un nœud d'interaction dans un processus participant. Les paramètres de cette méthode sont : T nœud d'interaction à insérer entre les deux *FlowNodes* FNp (le nœud qui précède T) et FNs (FNs le nœud qui suit T) et S indique le type de routage utilisé lors de l'insertion. S peut avoir les valeurs suivantes : séquence, parallèle ou conditionnelle. L'algorithme de la méthode « ajouterNoeud » est le suivant.

```
Algorithme ajouterNoeud(T:interactionNode, S:String, FNp:FlowNode,
FNs: FlowNode)
Variables
PGsplit, PGjoin : ParallelGateway
CGsplit, CGjoin : ConditionalGateway
Début
    this.supprimerSequenceFlow(Fnp,Fns)
    this.insererNoeud(T)
Selon S
S= 'Sequence':
    this.ajouterSequenceFlow(FNp,T)
    this.ajouterSequenceFlow(T,FNs)
```

```
S= 'Parallel':
    PGsplit= this.insererGatewayParallel()
    this.ajouterSequenceFlow(FNp,PGsplit)
    this.ajouterSequenceFlow(PGsplit,FNs)
    this.ajouterSequenceFlow(PGsplit,T)
    PGjoin= this.insererGatewayParallel()
    this.ajouterSequenceFlow(FNs,PGjoin)
    this.ajouterSequenceFlow(T,PGjoin)
    this.ajouterSequenceFlow(PGjoin,(FNs.getSuivant()))
S= 'Conditional':
    CGsplit= this.insererGatewayConditional()
    this.ajouterSequenceFlow(FNp,CGsplit)
    this.ajouterSequenceFlow(CGsplit,FNs)
    this.ajouterSequenceFlow(CGsplit,T)
    PGjoin= this.insererGatewayConditional()
    this.ajouterSequenceFlow(FNs,CGjoin)
    this.ajouterSequenceFlow(T,CGjoin)
    this.ajouterSequenceFlow(CGjoin,(FNs.getSuivant()))
Fin Selon
Fin
```

L'algorithme de la méthode « ajouterInteractionCas#1 » qui implémente le cas#1 du patron « ajouter une interaction » possède les paramètres suivants : M la version de message à insérer, E le nœud d'interaction à insérer dans la version du processus Pe entre les deux *FlowNodes* Fnpe (le nœud qui précède E) et Fnse (le nœud qui suit E), s1 le type de routage utilisé lors de l'insertion de E, R le nœud d'interaction à insérer dans la version du processus Pr entre les deux *FlowNodes* Fnpr (le nœud qui précède R) et Fnsr (le nœud qui suit R) et s2 le type de routage utilisé lors de l'insertion de R. D'abord, cet algorithme permet l'insertion des nouveaux nœuds « E » (nœud d'envoi) et « R » (nœud de réception) respectivement dans les versions de processus Pe et Pr. Cette insertion peut donner lieu à la dérivation de ces deux versions de processus si'elles sont dans un état « *Stable* » ou si elles ne sont pas déjà dérivées. Par la suite, cet algorithme permet la définition de la version de message « M » dans la collaboration. Finalement, il permet l'adjonction d'un nouveau *MessageFlow* qui relie les nœuds ajoutés E et R et qui porte la version de message M.

```
Algorithme ajouterInteractionCas#1(M:version_of_message, E:
InteractionNode, Pe:version_of_process, s1:String, Fnpe:FlowNode,
Fnse:FlowNode, R:InteractionNode, Pr:version_of_process, s2:String,
Fnpr: FlowNode, Fnsr:FlowNode)
Variables
Ped, Prd : version of process
Début
//Dériver Pe si elle est Stable et insérer le nœud d'interaction E
Si Pe.estStable() alors
    Ped = Pe.derive()
    Ped.ajouterNoeud(E,s1,Fnpe,Fnse)
Sinon
    Pe.ajouterNoeud(E,s1,Fnpe,Fnse)
Finsi
//Dériver Pr si elle est Stable et insérer le nœud d'interaction R
Si Pr.estStable() alors
    Prd = Pr.derive()
    Prd.ajouterNoeud(R,s2,Fnpr,Fnsr)
Sinon
    Pr.ajouterNoeud(R,s2,Fnpr,Fnsr)
```

```
Finsi
//Ajouter la version du message M dans la collaboration
    this.ajouterMessage(M)
//Ajouter un MessageFlow qui relie les nœuds E et R et qui porte la
version du message M
    this.ajouterMessageFlow(M,E,R)
Fin
```

III.3. Patron d'adaptation « supprimer une interaction » (Delete Interaction)

Le patron « supprimer une interaction » permet la modification d'une version de collaboration par la suppression d'une interaction de cette version. Quatre cas possibles, ci-après présentés, peuvent se décliner suite à l'utilisation de ce patron.

- Cas#1 : supprimer un message ainsi que ses nœuds d'envoi et de réception.
- Cas#2 : supprimer un message et redéfinir ses nœuds d'envoi et de réception s'ils sont utilisés dans d'autres interactions.
- Cas#3 : supprimer un message ainsi que son nœud d'envoi et redéfinir son nœud de réception.
- Cas#4 : supprimer un message ainsi que son nœud de réception et redéfinir son nœud d'envoi.

Dans le premier, le troisième et le quatrième cas la suppression des nœuds (*i.e.,* versions de tâches ou versions d'événements) entraine leur disparition de la version de collaboration à modifier mais pas la destruction définitive de ces versions puisqu'ils peuvent être réutilisés dans la définition d'autres versions de processus.

Dans cette section, nous nous limitons à détailler le premier cas (cas#1). Les trois autres cas et l'algorithme qui implémente les quatre cas sont présentés dans l'annexe A2.

Cas#1 : supprimer un message, le nœud d'envoi et le nœud de réception

Le cas#1 du patron « supprimer une interaction » est utilisé quand les nœuds de l'interaction à supprimer ne sont pas impliqués dans d'autres messages de la collaboration. De ce fait, ce patron permet la suppression du message de l'interaction ainsi que ses nœuds d'envoi et de réception.

La Figure IV.17 montre un exemple d'utilisation du cas#1 de ce patron. D'après cet exemple, nous illustrons la suppression du message M entre les tâches A2 et B2 de la collaboration C. Cette suppression entraîne (i) la définition des deuxièmes versions des processus A (VP1-2) et B (VP2-2) dérivées respectivement à partir de VP1-1 (la première version de A) et VP2-2 (la première version de B) si'elles sont à l'état « *Stable* » ou elles ne sont pas déjà dérivées (ii) la suppression de *MessageFlow* portant la première version du message M (VM1-1) et reliant la première version de la tâche A2 (identifiée par VA2-1) et la première version de la tâche B2 (identifiée VA5-1), (iii) la suppression des deux versions de tâches VA2-1 et VA5-1 car elles ne participent pas

à d'autres *MessageFlow* et (iv) la suppression de la version de message VM1-1. Les versions de processus VP1-1 et VP2-1 ont été dérivées car leurs schémas ont été modifiés.

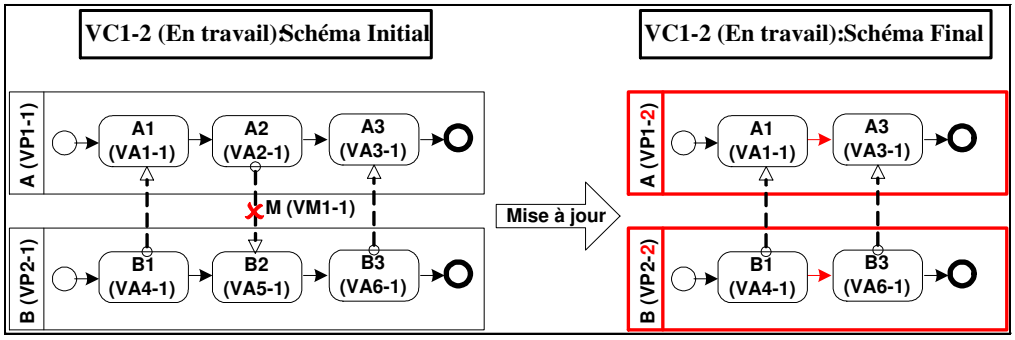


Figure IV.17. Cas#1 du patron « supprimer une interaction »

III.4. Patron d'adaptation « déplacer une interaction » (Move Interaction)

Le patron « déplacer une interaction » permet la modification d'une version de collaboration par le déplacement d'une interaction d'une position initiale vers une position finale. Ce patron peut être défini soit en utilisant un ensemble de primitives de bas niveaux qui représentent les méthodes des classes du méta-modèle BPMN4V-CC concernées par la modification (*e.g.,* ajouterNœud et supprimerNœud), soit en utilisant les patrons « ajouter une interaction » et « supprimer une interaction ».

Nous distinguons trois options possibles pour déplacer une interaction d'une position à une autre. Ces options diffèrent en fonction du type de routage utilisé pour réinsérer les nœuds de l'interaction à déplacer dans les processus correspondants (*i.e.,* insertion en séquence, insertion parallèle ou insertion conditionnelle). Nous présentons dans ce qui suit une description de ces options ainsi que l'algorithme qui les implémente.

✓ Description des options du patron « déplacer une interaction »

La Figure IV.18, illustre, à travers un exemple d'utilisation du patron « déplacer une interaction », les différentes options de déplacement. Cet exemple traite de l'interaction entre les tâches A1 et B1, qui doit être déplacée de sa position initiale (en début des processus A et B) vers une position finale (entre les tâches A2 et A3 du processus A et les tâches B2 et B3 du processus B). Chacune de ces options permet : (i) l'utilisation du cas#1 du patron « supprimer une interaction » pour supprimer l'interaction entre les premières versions des tâches A1 et B1 (identifiées par VA1-1 et VA4-1) à partir des deuxièmes versions des processus A et B (identifiées par VP1-2 et VP2-2), et (ii) l'application du cas#1 du patron « ajouter une interaction » pour réinsérer l'interaction supprimée dans sa nouvelle position. Les versions des processus VP1-2 et VP2-2 sont définies par dérivation à partir de VP1-1 et VP2-1 suite à la modification de leurs schémas.

Les différentes options de ce patron différent en fonction du type de routage utilisé pour réinsérer l'interaction supprimée. La première option correspond à une insertion en séquence des versions de tâches VA1-1 (première version de A1) et VA4-1 (première version de B1) dans les versions de processus VP1-2 et VP2-2, tandis que la deuxième correspond à une insertion parallèle et la troisième option correspond à une insertion conditionnelle. Dans la Figure IV.18, les modifications effectuées à l'aide du patron d'adaptation « déplacer une interaction » sont présentées en rouge.

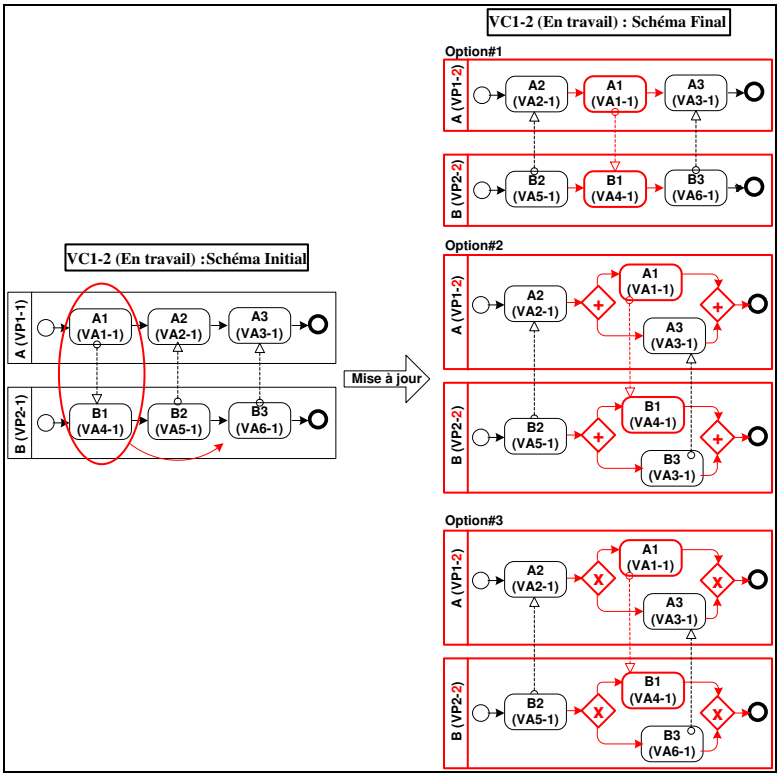


Figure IV.18. Exemple d'utilisation du patron « déplacer une interaction »

✓ Algorithme du patron « déplacer une interaction »

Pour bien expliquer le patron « déplacer une interaction », nous proposons, dans ce qui suit, l'algorithme de la méthode « déplacerInteraction » qui implémente ce patron. Les paramètres de cette méthode sont les suivants : la version du message à déplacer M, le nœud d'interaction E de la version du processus Pe à déplacer entre les deux *FlowNodes* Fnpe (le nouveau nœud qui précède E) et Fnse (le nouveau nœud qui suit E), le type de routage s1 utilisé lors de la réinsertion de E, le nœud d'interaction R de la version du processus Pr à déplacer entre les deux *FlowNodes* Fnpr (le nouveau nœud qui précède R) et Fnsr (le nouveau nœud qui suit R), et le type de routage s2 utilisé lors de l'insertion de R.

L'algorithme de la méthode « déplacerInteraction » fait appel aux méthodes des patrons « ajouter une interaction » et « supprimer une interaction ». Plus précisément, il utilise la méthode « supprimerInteraction(M,E,Pe,R,Pr) » pour supprimer le message M, le nœud d'envoi E du processus Pe, et le nœud de réception R du processus Pr. Dans un second temps, cet algorithme appelle la méthode « ajouterInteractionCas#1(M,

E,Pe,s1,Fnpe,Fnse,R,Pr,s2,Fnpr,Fnsr) » qui permet d'ajouter le message M, le nœud d'envoi E dans le processus Pe entre les *FlowNodes* Fnpe et Fnse et le nœud de réception R dans le processus Pr entre les *FlowNodes* Fnpr et Fnsr. L'algorithme de la méthode « déplacerInteraction » est le suivant.

```
Algorithme déplacerInteraction(M:version_of_message,
E:InteractionNode, Pe:version_of_process, s1:String, Fnpe:FlowNode,
Fnse:FlowNode, R: InteractionNode, Pr:version_of_process, s2:String,
Fnpr:FlowNode, Fnsr:FlowNode)
Début
    this.supprimerInteraction(M,E,Pe,R,Pr)
    this.ajouterInteractionCas#1(M,E,Pe,s1,Fnpe,Fnse,R,Pr,s2,Fnpr,
Fnsr)
Fin
```

III.5. Patron d'adaptation « remplacer une interaction » (Replace Interaction)

Le patron « remplacer une interaction » permet la modification d'une version de collaboration par la substitution d'une interaction par une autre. Comme pour le patron « déplacer une interaction », ce patron peut être défini par la combinaison des patrons : « ajouter une interaction » et « supprimer une interaction ». Dans ce qui suit, nous présentons une description de ce patron ainsi que l'algorithme qui l'implémente.

✓ Description du patron « remplacer une interaction »

La Figure IV.19 montre un exemple d'utilisation du patron « remplacer une interaction ». Dans cet exemple, nous remplaçons l'interaction composée par la première version du message M1 (identifiée par VM1-1) et les premières versions des tâches A2 et B2 (identifiées par VA2-1 et VA5-1) par une nouvelle interaction composée par la première version du message M2 (identifiée par VM2-1) et les premières versions des tâches X et Y (identifiées par VA7-1 et VA8-1). Pour ce faire, nous devons supprimer, dans un premier temps, l'interaction à remplacer en utilisant le patron « supprimer une interaction » et insérer, dans un second temps, à sa place la nouvelle interaction en utilisant le patron « ajouter une interaction ». Dans la Figure IV.19, les modifications effectuées à l'aide du patron d'adaptation « remplacer une interaction » sont présentées en rouge.

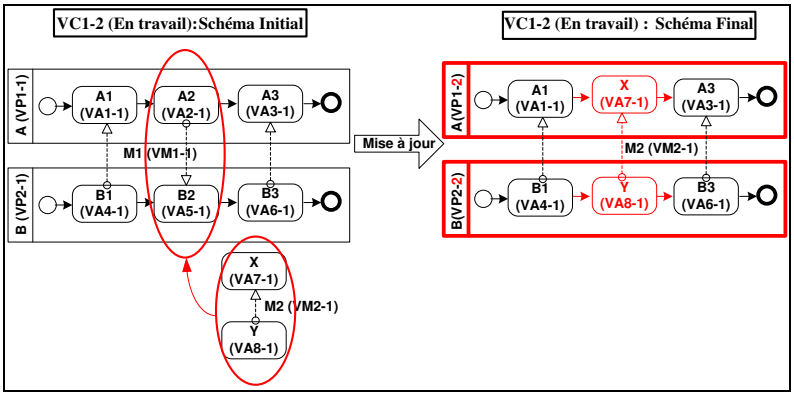


Figure IV.19. Exemple d'utilisation du patron d'adaptation « Remplacer une interaction »

✓ Algorithme du patron « Remplacer une interaction »

Cette section détaille l'algorithme de la méthode « remplacerInteraction » qui implémente le patron « remplacer une interaction ». Cette méthode utilise les paramètres suivants : la version du message à remplacer M, le nœud d'envoi E de l'interaction à remplacer appartenant à la version du processus Pe, le nœud de réception R de d'interaction à déplacer appartenant à la version du processus Pr, la nouvelle version du message M1, le nouveau nœud d'envoi En et le nouveau nœud de réception Rn.

L'algorithme de la méthode « remplacerInteraction » permet alors de remplacer l'interaction composée par le message M et les nœuds E et R par l'interaction formée par le message M1 et les tâches En et Rn. Plus précisément, cet algorithme permet, tout d'abord, de récupérer les positions des nœuds de l'interaction à remplacer. En effet, dans les variables Fnpe et Fnpr, nous récupérons les *FlowNodes* qui précèdent respectivement les nœuds E et R. Alors que, dans les variables Fnse et Fnsr nous affectons les *FlowNodes* qui suivent les nœuds E et R. Par la suite, cet algorithme fait appel à la méthode « supprimerInteraction(M,E,Pe,R,Pr) » pour supprimer l'interaction entre les nœuds E et R. Finalement, il appel la méthode « ajouterInteractionCas#1 (M1, En,Pe,'Sequence',Fnpe,Fnse,Rn,Pr,'Sequence',Fnpr,Fnsr) » pour ajouter la nouvelle interaction entre les nœuds En et Rn dans les positions récupérées. L'algorithme de la méthode « remplacerInteraction » est le suivant.

```
Algorithme remplacerInteraction(M:version_of_message, E:
InteractionNode, Pe: version_of_process, R: InteractionNode, Pr:
version_of_process, M1 :version_of_message, En: InteractionNode, Rn :
InteractionNode)
 Variables
 Fnpe,Fnse,Fnpr,Fnsr :FlowNode
 Début
 // Récupération du FlowNode qui précède le nœud E
     Fnpe=E.getPrecedent()
 // Récupération du FlowNode qui suit le nœud E
     Fnse=E.getSuivant()
 // Récupération du FlowNode qui précède le nœud R
     Fnpr=R.getPrecedent()
 // Récupération du FlowNode qui suit le nœud R
     Fnsr=R.getSuivant()
 // Suppression de l'interaction entre les nœuds E et R
     this.supprimerInteraction(M,E,Ped,R,Prd)
 // Insertion de la nouvelle interaction
     this.ajouterInteractionCas#1(M1,En,Pe,'Sequence',Fnpe,Fnse,Rn,
Pr,'Sequence',Fnpr,Fnsr)
 Fin
```

III.6. Patron d'adaptation « échanger une interaction » (Swap Interaction)

Le patron « échanger une interaction » permet la modification d'une version de collaboration à travers la permutation de deux interactions. Ce patron peut être réalisé par l'utilisation combinée des patrons : « ajouter une interaction » et « supprimer une interaction », ou bien par l'utilisation directe des méthodes introduites dans le tableau

IV-3. Dans ce qui suit nous présentons une description de ce patron ainsi que l'algorithme qui l'implémente.

✓ Description du patron « échanger une interaction »

La Figure IV.20 montre un exemple d'utilisation du patron « échanger une interaction ». A travers cet exemple, nous illustrons comment nous pouvons échanger l'interaction composée par la première version du message M1 (identifiée par VM1-1), la première version de la tâche B1 (VA4-1) et la première version de la tâche A1 (VA1-1) par l'interaction composée par la première version du message M2 (VM2-1), la première version de tâche la B3 (VA6-1) et la première version de la tâche A3 (VA3-1). Pour ce faire, nous proposons de supprimer ces deux interactions puis les réinsérer dans les nouvelles positions. Dans la Figure IV.20, les modifications effectuées à l'aide du patron d'adaptation « échanger une interaction » sont présentées en rouge.

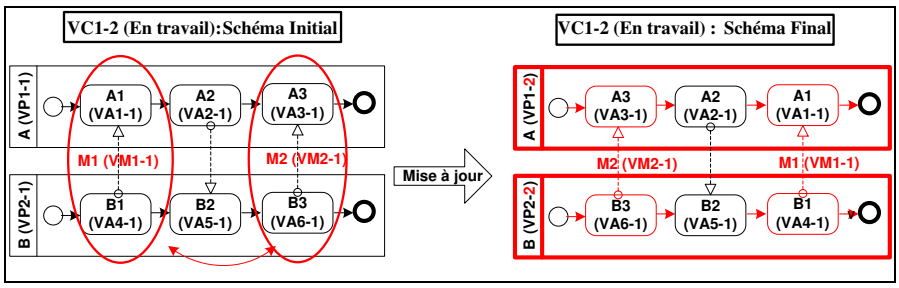


Figure IV.20. Le patron « échanger une inetraction »

✓ Algorithme du patron « échanger une interaction »

Pour mieux expliquer le patron « Echanger une interaction », nous présentons, dans ce qui suit, l'algorithme de la méthode « échangerInteraction » qui implémente ce patron. Cette méthode utilise les paramètres suivants : M1 et M2 les deux versions de message à échanger, E1 et E2 les deux nœuds à échanger, appartenant respectivement aux versions des processus Pe1 et Pe2, et R1 et R2 les deux nœuds à échanger appartenant respectivement aux versions des processus Pr1 et Pr2. Plus précisément, cet algorithme, permet d'échanger l'interaction formée par le message M1 et les nœuds E1 et R1 par l'interaction formée par le message M2 et les nœuds E2 et R2. Pour ce faire, cet algorithme permet : (i) la récupération des positions des nœuds E1, E2, R1 et R2 par la détermination du *FlowNode* qui suit et du *FlowNode* qui précède chacun de ces nœuds, (ii) la suppression des deux interactions à échanger et (iii) la réinsertion de ces deux interactions dans les positions récupérées. L'algorithme de la méthode « échangerInteraction » est le suivant.

```
Algorithme échangerInteraction(M1:version_of_message, M2:
version_of_message, E1:InteractionNode, E2:InteractionNode, Pe1:
version_of_process, Pe2:version_of_process, R1:InteractionNode, R2:
InteractionNode, Pr1:version_of_process, Pr2:version_of_process)
 Variables
 Fnpe1,Fnse1,Fnpr1,Fnsr1,Fnpe2,Fnse2,Fnpr2,Fnsr2: FlowNode
 Début
 //Récupération de la position du nœud d'envoi E1
```

```
    Fnpe1=E1.getPrecedent()
    Fnse1=E1.getSuivant()
//Récupération de la position du nœud réception R1
    Fnpr1=R1.getPrecedent()
    Fnsr1=R1.getSuivant()

//Récupération de la position du nœud d'envoi E2
    Fnpe2=E2.getPrecedent()
    Fnse2=E2.getSuivant()
//Récupération de la position du nœud réception R2
    Fnpr2=R2.getPrecedent()
    Fnsr2=R2.getSuivant()
//Suppression des interactions à échanger
    This.supprimerInteraction(M1, E1, Pe1, R1, Pr1)
    This.supprimerInteraction(M2, E2, Pe2, R2, Pr2)
//Réinsertion des interactions à échanger
    This.ajouterInteractionCas#1(M2,E2,Pe1,'Sequence',Fnpe1,Fnse1,R2,
Pr1,'Sequence',Fnpr1,Fnsr1)
    This.ajouterInteractionCas#1(M1,E1,Pe2,'Sequence',Fnpe2,Fnse2,R1,
Pr2,'Sequence',Fnpr2,Fnsr2)

Fin
```

III.7. Patron d'adaptation « modifier une interaction » (Modify Interaction)

Le patron « modifier une interaction » permet la modification d'une interaction appartenant à une version de collaboration. Cette modification peut toucher le message, le nœud d'envoi et / ou le nœud de réception de l'interaction à modifier. La Figure IV.21 illustre quatre cas possibles du patron « modifier une interaction », qui se déclinent en dix sous-cas. Ces cas varient en fonction du composant de l'interaction concernée par la modification. Plus précisément, le cas#1 correspond à la modification du nœud d'envoi, le cas#2 correspond à la modification du nœud de réception, le cas#3 correspond à la modification à la fois des nœuds d'envoi et de réception et le cas#4 correspond à la modification du message de l'interaction. Dans les cas#1, cas#2 et cas#3, les nœuds à modifier peuvent être remplacés par des nœuds existants dans la collaboration ou bien par de nouveaux nœuds. Nous nous limitons dans cette section à présenter le premier cas (cas#1). Les autres cas sont présentés dans l'annexe A2.

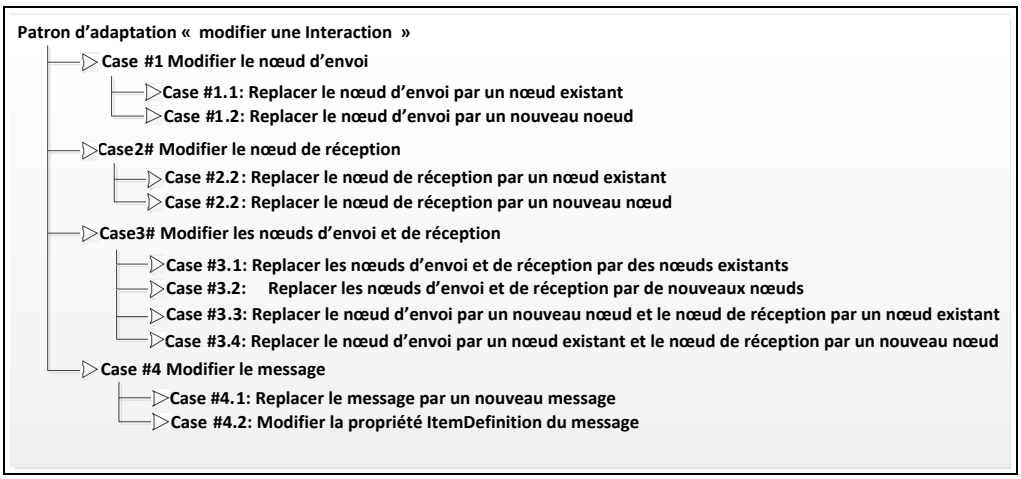


Figure IV.21. Cas possibles du patron « modifier une interaction »

Cas#1 : modifier le nœud d'envoi

Le cas#1 du patron « modifier une interaction » permet de modifier une interaction en changeant son nœud d'envoi de message. Cette modification peut être selon deux sous-cas. Le premier sous-cas consiste à remplacer le nœud d'envoi par un nœud existant dans la collaboration, tandis que dans le deuxième sous-cas le nœud d'envoi est remplacé par un nouveau nœud. Dans ce qui suit nous présentons une description de ces sous-cas ainsi que l'algorithme qui les implémente.

✓ Description des sous-cas du cas#1

- **Cas#1.1 : remplacer le nœud d'envoi par un nœud existant**

La Figure IV.22 illustre par un exemple l'application du cas#1.1 du patron « modifier une interaction ». Dans cet exemple, il s'agit de modifier la tâche d'envoi de l'interaction portant la première version du message M (VM1-1) et reliant les premières versions des tâches A2 et B2 (identifiées respectivement par VA2-1 et VA5-1). Plus précisément, le nœud VA2-1 doit être remplacé par la première version de la tâche A1 (identifiée par VA1-1) déjà existante dans la première version du processus A (VP1-1). L'utilisation de ce cas de patron permet : (i) la définition de la deuxième version de la tâche A1 (VA1-2) dérivée à partir de VA2-1 suite au changement de son type ainsi que la deuxième version du processus A (VP1-2), (ii) la spécification de la version VA1-2 comme le nœud source du *MessageFlow* portant la version de message VM1-1, et (iii) la suppression de la première version de la tâche A2 (VA2-1) si elle ne participe pas dans d'autres *MessageFlow* (option1). Il est à noter que la version de tâche VA2-1 doit être maintenue si elle est invoquée dans d'autres *MessageFlow* (option2). Dans la Figure IV.22, les modifications à l'aide du cas#1.1 du patron d'adaptation « modifier une interaction » sont présentées en rouge.

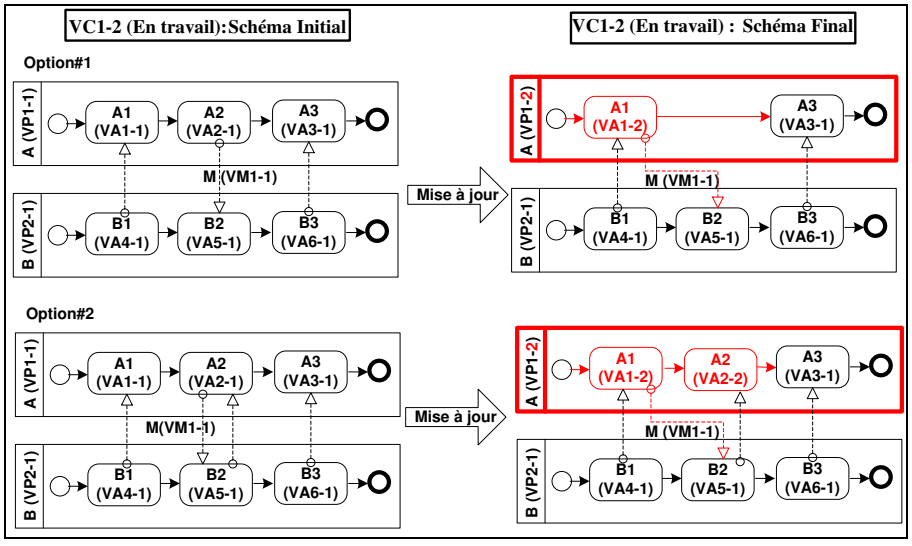


Figure IV.22. Cas#1.1 du patron « modifier une interaction »

- **Cas#1.2 : Remplacer le nœud d'envoi par un nouveau nœud**

La Figure IV.23 illustre par un exemple l'application du cas#1.2 du patron « modifier une interaction ». Dans cet exemple, le nœud d'envoi de l'interaction formée

par la première version du message M (identifiée par VM1-1) et des premières versions des tâches A2 et B2 (identifiées par VA2-1 et VA5-1) est remplacé par le nouveau nœud A4 inséré dans le processus A. Plus précisément, l'utilisation de ce cas de patron permet (i) la définition de la deuxième version du processus A (identifiée par VP1-2) dérivée à partir de sa première version (VP1-1) suite au changement de son schéma, (ii) l'insertion de la première version de la tâche A4 (identifiée par VA7-1) dans VP1-2 et (iii) la définition de tâche A4 comme le nœud source du *MessageFlow* portant le message M. En fonction du type de routage utilisé pour insérer le nouveau nœud d'envoi A4, nous distinguons trois options possibles : insertion en séquence (option1), insertion parallèle (option2) et insertion conditionnelle (option3). Dans la figure IV.23, les modifications effectuées à l'aide de cas#1.2 du patron d'adaptation « modifier une interaction » sont présentées en rouge.

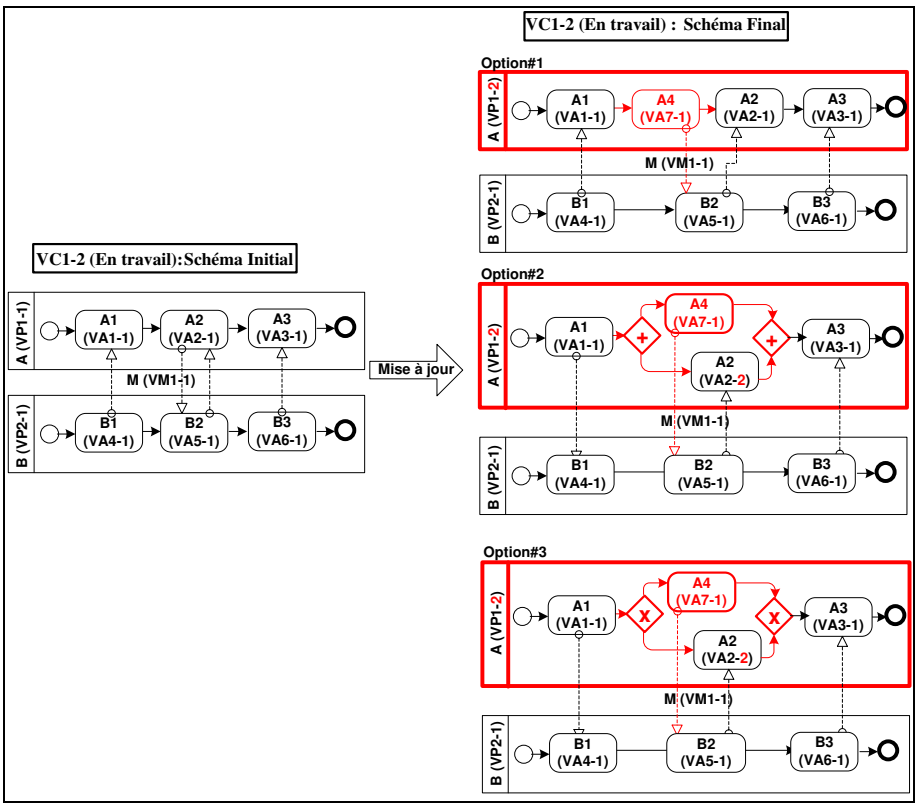


Figure IV.23. Cas#1.2 du patron « modifier une interaction »

✓ Algorithmes du cas#1

Nous détaillons dans cette section l'algorithme de la méthode « modifierInteractionCas#1 » permettant la modification du nœud source d'une interaction. Cette méthode utilise les paramètres suivants : la version de message M, le nœud d'interaction E à modifier appartenant à la version de processus Pe, le nouveau nœud d'interaction NE à insérer entre les *FlowNodes* Fnp (le nœud qui précède NE) et Fns (le nœud qui suit NE), le type de routage s utilisé lors de l'insertion de NE ; s peut avoir les valeurs suivantes : séquence, parallèle ou conditionnelle.

L'algorithme permet de modifier le nœud d'envoi de l'interaction composée par la version de message M et les nœuds d'interaction E et R, tout en considérant les sous-cas du cas#1. Plus précisément, cet algorithme permet (i) la récupération du *MessageFlux* de l'interaction à modifier, (ii) l'insertion du nœud NE s'il n'existe pas dans la version du processus Pe, d'où la dérivation de cette version de processus, (iii) la modification de la source du *MessageFlow* récupéré en remplaçant le nœud d'interaction E par le nœud NE, et (iv) la suppression du nœud E s'il n'est pas impliqué dans d'autres *MessageFlux* ou bien la dérivation de ce nœud s'il est maintenu.

```
Algorithme modifierInteractionCas#1(M:version_of_message,
   E:InteractionNode, NE:InteractionNode, Pe:version_of_process, S:
   String, Fnp: FlowNode, Fns:FlowNode, R:InteractionNode)
Variables
   Ped: version of process
   Ed, NEd : InteractionNode
   Mf: MessageFlow
Début
   //Récupération du MessageFlow à modifier
   Mf= This.getMessageFlow(M,E,R)

   Si Pe.estStable() alors
        Ped = Pe.derive ()
   Sinon
        Ped=Pe
   Fin Si

   Si Ped.exist(NE) alors
        Si NE.estStable()
                NEd=NE.derive()
        Sinon
                NEd=NE
        Fin si
         //Modification de la source du MessageFlow
        Mf.modifierNoeudEnvoi(NEd)
   Sinon
        //Adjonction du nouveau nœud NE
        Ped.ajouterNoeud(NE,S,Fnp,Fns)
        Mf.modifierNoeudEnvoi(NE)
   Fin Si
        //Suppression du nœud E s'il n'est pas impliqué dans d'autre
   messageFlow
   Si E.calculerNbMsg()=1 alors
        Ped.supprimerNoeud(E)
   Sinon
        si E.estStable()
                Ed=E.derive()

        Fin Si
   Fin Si

Fin
```

IV. Génération d'une version de chorégraphie à partir d'une version de collaboration

Outre la modélisation et la gestion des versions de collaborations, BPMN4V-CC permet aussi la génération automatique des versions de chorégraphies à partir des versions de collaborations (qui sont des instances du méta-modèle BPMN4V-CC). Cette génération est due, du fait qu'une chorégraphie n'est qu'un type particulier d'une collaboration. En effet, en se référant au méta-modèle de BPMN 2.0 présenté précédemment dans la Figure IV.1, nous pouvons constater qu'une chorégraphie est une spécialisation d'une collaboration qui met en évidence les messages échangés entre les différents partenaires. De ce fait, nous proposons une solution permettant la transformation d'une version de collaboration en une version de chorégraphie, au lieu de modéliser directement des versions de chorégraphies.

Cette section présente la démarche que nous proposons pour la génération d'une version de chorégraphie à partir d'une version de collaboration. Elle donne, dans un premier temps, le principe général de la génération en présentant les différents étapes de la démarche proposée et détaille, dans un second temps, les algorithmes implémentant ces étapes.

IV.1. Principe de la génération

La démarche de génération d'une version de chorégraphie à partir d'une version de collaboration, présentée en Figure IV.24, comporte les quatre étapes suivantes :

- Etape1 construit une arborescence, nommée Arbre-VP (Arbre de Version de Processus), pour chaque version de processus impliquée dans la version de collaboration. La construction d'un Arbre-VP nécessite une décomposition de la version du processus en fragments.
- Etape2 relie les Arbre-VP construits dans l'étape précédente pour la construction d'une forêt nommée Arbres-VP-Liés. Plus précisément, cette forêt est composée par (i) des Arbres-VP qui correspondent aux versions de processus de la version de collaboration et (ii) des liens qui correspondent aux messages échangés.
- Etape3 déduit un Arbre-VC (Arbre de version de chorégraphie) qui est une représentation de la version de chorégraphie sous forme d'une arborescence.
- Etape4 déduit le diagramme de chorégraphie correspondant à l'Arbre-VC de l'étape précédente.

Dans le reste de cette section, nous détaillons chacune de ces étapes en explicitant les algorithmes qui les implémentent [Ben Said *et al.,* 2016].

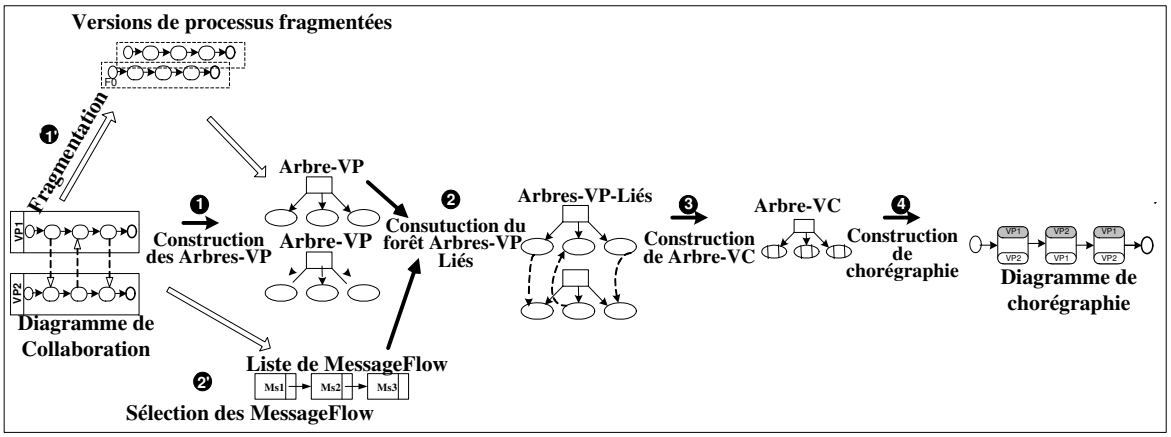


Figure IV.24. Démarche de génération d'une version de chorégraphie à partir d'une version de collaboration

IV.2. Construction des Arbres-VP à partir d'une version de collaboration

La première étape de la démarche de génération consiste à construire un Arbre-VP (Arbre de Version de processus) pour chaque version de processus impliquée dans la version de la collaboration à transformer. Pour ce faire, nous décomposons, dans un premier temps, chaque version de processus en fragments et nous déduisons, dans un second temps, les Arbres-VP correspondants.

IV.2.1. Fragmentation de processus

La fragmentation de processus consiste en la décomposition de chaque version de processus impliquée dans la version de collaboration en fragments canoniques ayant une seule entrée et une seule sortie, appelés Fragments SESE (Single Entry Single Exit). Pour ce faire, nous proposons d'utiliser l'algorithme proposé par [Polyvyanyy et *al.*, 2012] pour décomposer un processus en fragments SESE. La Figure IV.25 illustre le résultat de la fragmentation de la première version du processus de centre de radiologie impliquée dans la première version de la collaboration *Examen Radiologique* (présentée dans la Figure IV.6) selon l'algorithme proposé par [Polyvyanyy et *al.*, 2012]. Cette décomposition produit les fragments SESE suivants : F0, F1, F2. Comme son nom l'indique, chaque fragment SESE a une seule flèche d'entrée et une seule flèche de sortie. Par exemple, le fragment F2 est une séquence des tâches *Envoi date de RDV*, *Examen* et *Envoi Rapport*, alors que, le fragment F1 est un branchement de la tâche *Envoi Notification de rejet* et du fragment F2. De plus, un fragment SESE peut intégrer d'autres fragments SESE. Par exemple, F0 est un fragment SESE qui regroupe l'événement initial, la tâche *Réception demande d'examen*, le fragment F1 et l'événement final.

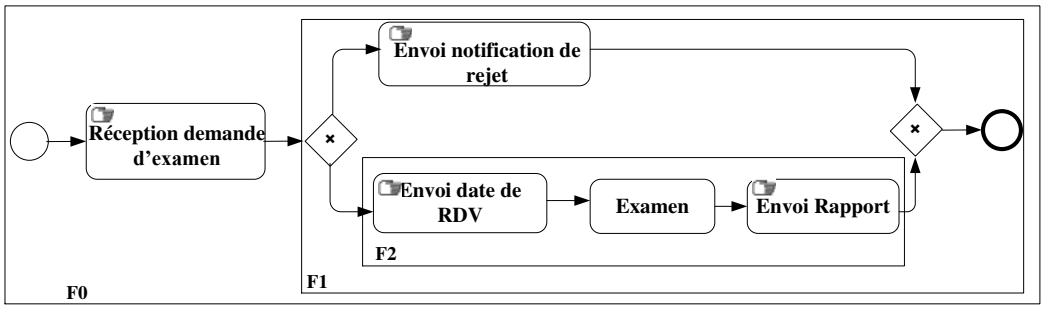


Figure IV.25. Décomposition de la permière version du processus du centre de radiologie en fragments SESE

IV.2.2. Déduction des Arbre-VP à partir des Fragments

Les Arbres-VP représentant les versions de processus d'une collaboration sont déduites à partir des fragments SESE. En général, un Arbre-VP est une arborescence ayant la structure présentée ci-dessous. Cette structure est représentée avec le syntaxe du langage ML [Milner et *al.,* 1997].

```
Arbre-VP::= Nœud-VP
Nœud-VP::=NœudTerminal-VP | NœudNonTerminal-VP
NœudNonTerminal-VP::=SEQ({Nœud-VP})|CHO({Nœud-VP})|PAR({Nœud-VP})
|REP(Nœud-VP)
NœudTerminal-VP::=Tâche
```

Un Arbre-VP est défini comme étant un nœud appelé Nœud-VP. Un Nœud-VP peut être de deux types : un nœud terminal NœudTerminal-VP ou bien un nœud non terminal NœudNonTerminal-VP. Un NœudNonTerminal-VP a une certaine nature. Il peut être une séquence (SEQ), un choix (CHO), un parallélisme (PAR) ou bien une répétition (REP) d'un ensemble de Nœud-VP. Un NœudTerminal correspond à une version de tâche d'un fragment SESE qui permet l'envoi ou la réception d'un message.

Nous proposons, dans ce qui suit, l'algorithme *Construire-Arbre-VP* qui implémente la transformation d'une version de processus fragmentée (*i.e.,* décomposée en fragments SESE) en Arbre-VP. Cet algorithme utilise les règles de transformation présentées dans le tableau IV-4.

Tableau IV-4. Les règles de transformation d'un fragment en Arbre-VP

Fragment	Arbre-VP
Version_of_Task	NœudTerminal-VP/Tâche
Fragment	NœudNonTerminal-VP
Type de Routage utilisé	Nature du NœudNonTerminal-VP (SEQ, CHO, PAR ou REP)

En outre, l'algorithme *Construire-Arbre-VP* utilise un ensemble de fonctions permettant la manipulation des fragments et des arbres. Ces fonctions sont décrites dans le tableau IV-5.

Tableau IV-5. Les fonctions permettant la construction d'un Arbre-VP

Concepts	Fonctions	Description
Fragment	estTâche()	renvoie Vrai si un fragment est une version de tâche et Faux sinon.
	estEvénement()	renvoie Vrai si un fragment est une version d'événement et Faux sinon.
	getComposant()	renvoie l'ensemble des versions de tâches et/ou des fragments qui composent un fragment.
	getTypeRoutage ()	renvoie le type de routage utilisé pour coordonner les versions de tâches d'un fragment.
	définirNœudTerminal-VP()	définit le NœudTerminal-VP qui correspond à une version de tâche.
	définirNœudNonTerminal-VP (r)	définit le NœudNonTerminal-VP qui correspond à un fragment dont les composants sont synchronisés en utilisant le type de routage r.
Arbre-VP	getNoeudCourant()	renvoie le nœud courant de l'Arbre-VP.
	ajouterNœud-VP(n, p)	ajoute le Nœud-VP *n* à un Arbre-VP comme un fils du nœud *p*.

L'algorithme *Construire-Arbre-VP* assurant la transformation d'un fragment en Arbre-VP est le suivant.

```
Algorithme Construire-Arbre-VP(f:Fragment):VP-Tree
Local n,nc:Noeud-VP
      vpt: Arbre-VP
      c:Fragment
Début
   nc=vpt.getNoeudCourant()
   Si f.estTâche() alors
```

```
        n=f.définirNœudTerminal-VP()
        vpt.ajouterNœud-VP(n,nc)
        retourner vpt
   Sinon
        si non estEvénement(f) alors
                /* f est un fragment */
        n=f.définirNœudNonTerminal-VP(f.getTypeRoutage())
        vpt.ajouterNœud-VP(n, nc)
        Pour chaque c de f.getComposant()
                retourner Construire-Arbre-VP(c)
        Fin pour
   Fin si
Fin
```

La Figure IV.26 montre le résultat de la première étape de génération conformément à la première version de la collaboration *Examen Radiologique*. L'algorithme *Construire-Arbre-VP,* présenté précédemment, doit être exécuté pour chaque version de processus (préalablement fragmentée) impliquée dans cette version de collaboration. Chaque Arbre-VP obtenu est défini comme une séquence de NœudTerminal-VP, qui correspondent aux versions de tâches, et de NœudNonTerminal-VP, qui correspondent aux fragments. Il est à noter que les événements de début et de fin de chaque version de processus ne sont pas pris en compte dans un Arbre-VP. Ils seront ajoutés dans l'étape4 lors de la déduction du diagramme de chorégraphie.

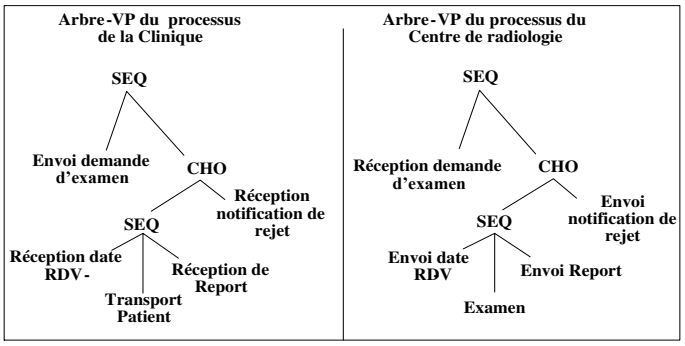


Figure IV.26. Arbres-VP de la première version de la collaboration examen radilogique

IV.3. Etape2 : Construction de la forêt Arbres-VP-Liés à partir des Arbres-VP

L'étape 1 de la démarche de génération que nous proposons permet de construire un ensemble d'Arbres-VP qui représentent les versions des processus impliquées dans la version de collaboration à transformer. L'étape suivante de cette démarche a pour objectif de lier ces Arbres-VP pour prendre en compte les messages échangés dans cette version de collaboration. Le résultat de cette étape est la construction d'une forêt formée par un ensemble d'Arbres-VP liés, appelée Arbres-VP-Liés, définie selon la structure suivante :

```
Arbres-VP-Liés ::= ({Arbre-VP},{Lien})
Lien ::= (NœudSource, NœudCible,Message)
NœudSource ::= Tâche
NœudCible ::= Tâche
```

Une forêt Arbres-VP-Liés est composée d'un ensemble d'Arbres-VP et de liens. Un lien est défini comme un triplet composé d'un NoeudSource, d'un NœudCible et d'un Message. Les nœuds source et cible d'un lien représentent des nœuds terminaux NœudTerminal-VP des Arbres-VP (*i.e.,* les versions des tâches), alors que le message est l'information transmise entre ces nœuds.

Comme dans la première étape, nous présentons dans ce qui suit l'algorithme *Construire-Arbres-VP-Liés* permettant la construction des liens entre les nœuds des Arbres-VP. Cet algorithme utilise les règles de transformation décrites dans le tableau IV-6.

Tableau IV-6. Règles de transformation d'un MessageFlow en Arbres-VP-Liés

MessageFlow	Arbres-VP-Liés
MessageFlow	Lien
sourceRef d'un MessageFlow	NoeudSource d'un lien
targetRef d'un MessageFlow	NœudCible d'un lien
messageRef d'un MessageFlow	Message d'un lien

En plus, l'algorithme *Construire-Arbres-VP-Liés* utilise un ensemble de fonctions qui permettent la manipulation des *MessageFlow* de la version de collaboration et de la forêt Arbres-VP-Liés à construire. Ces fonctions sont décrites dans le tableau IV-7.

Tableau IV-7. Fonctions utilisées par l'algorithme Construire-Arbres-VP-Liés

Concepts	Fonctions	Description
MessageFlow	getMessage()	Renvoie le message porté par un *MessageFlow*
	getNoeudSource()	Renvoie le NoeudTerminal-VP qui correspond à la tâche représentant la source du *MessageFlow*
	getNoeudCible()	Renvoie le NoeudTerminal-VP qui correspond à la tâche représentant la cible du *MessageFlow*
Arbres-VP-Liés	ajouterArbre-VP(t)	Ajoute l'arbre-VP *t* à la forêt Arbres-VP-Liés

	ajouterLien(l)	Ajoute le lien *l* à la forêt Arbres-VP-Liés
	définirLien(n1, n2, m)	Définit un lien reliant le NœudTerminal-VP *n1* et le NœudTerminal-VP *n2* et portant le message *m*

L'algorithme *Construire-Arbres-VP-Liés* assurant la transformant des Arbres-VP en Arbres-VP-Liés est présenté ci-dessous. Cet algorithme considère les deux paramètres correspondant à l'ensemble des Arbres-VP à lier et à l'ensemble des *MessageFlow* échangés entre les versions des processus de la collaboration.

```
Algorithme Construire-Arbres-VP-Liés(ensemblede-VPT:{Arbre-
   VP},ensemblede-MF: {MessageFlow}): Arbres-VP-Liés
Variable n1, n2: Nœud-VP, l: Lien
   mf: MessageFlow, vpt: Arbre-VP
   l-vpt: Arbres-VP-Liés
Début
   Pour chaque vpt de ensemblede-VPT
        l-vpt.ajouterArbre-VP(vpt)
   Fin Pour
   Pour chaque mf de ensemblede-MF
        n1=mf.getNoeudSource()
        n2=mf.getNoeudCible()
        l=définirLien(n1,n2,mf.getMessage())
        l-vpt.ajouterLien(l)
   Fin Pour
   Retourner l-vpt
Fin
```

La Figure IV.27 donne le résultat de cette étape suite à son application sur la première version de la collaboration *Examen Radiologique* et fait référence aux Arbres-VP présentés dans la Figure IV.26. Les liens de la forêt Arbres-VP-Liés sont présentés par des flèches rouges.

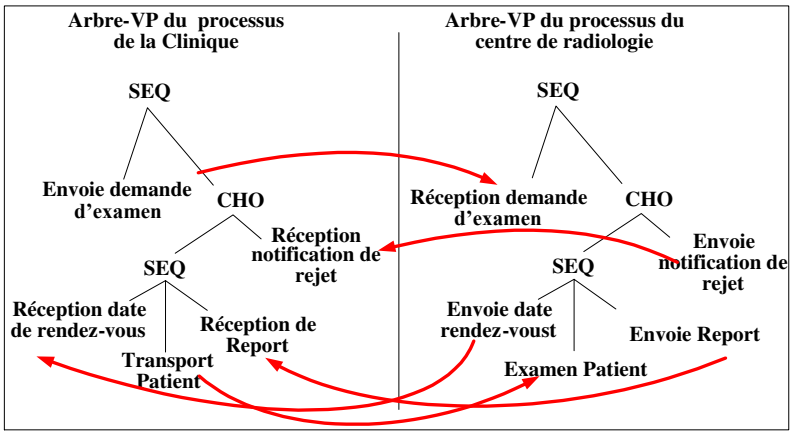


Figure IV.27. Arbres-VP-Liés de la première version de la collaboration « examen radiologique »

IV.4. Etape3 : Transformation de la forêt Arbres-VP-Liés en Arbre-VC

La troisième étape de notre démarche de génération d'une version de chorégraphie à partir d'une version de collaboration consiste à déduire un Arbre-VC (Arbre de Version de chorégraphie) à partir de la forêt Arbres-VP-Liés résultant de l'étape précédente. Un Arbre-VC a la structure suivante :

```
Arbre-VC ::= Nœud-VC
Nœud-VC ::= NœudTerminal-VC | NœudNonTerminal-VC
NœudNonTerminal-VC ::= SEQ({Nœud-VC})| CHO({Nœud-VC}) | PAR({Nœud-VC})
|REP(Nœud-VC)
NœudNonTerminal-VC ::= TâcheChorégraphie
TâcheChorégraphie ::= (ParticipantEmetteur,Message, ParticipantRécepteur)
```

Un Arbre-VC est représenté par un Nœud-VC. Un Nœud-VC peut être un nœud non terminal (NœudNonTerminal-VC) ou bien un nœud terminal (NœudTerminal-VC) qui représente une tâche de chorégraphie (TâcheChorégraphie). Un NœudNonTerminal-VC peut être une séquence (SEQ), un choix (CHO), un parallélisme (PAR) ou bien une répétition (REP) d'un ensemble de Nœud-VC. Une *TâcheChorégraphie* est définie comme un triplet composé par un *ParticipantEmetteur*, un *Message* et un *ParticipantRécepteur*. Ainsi, elle indique qu'un message est envoyé d'un participant à un autre.

Nous détaillons, dans ce qui suit, l'algorithme *Construire-Arbre-VC* assurant la déduction d'un Arbre-VC à partir d'une forêt Arbres-VP-Liés. Cet algorithme utilise les règles de transformation présentées dans le tableau IV-8.

Tableau IV-8. Règles de transformation d'une forêt Arbres-VP-Liés en Arbres-VC

Arbres-VP-Liés	Arbre-VC
NœudNonTerminal-VP	NœudNonTerminal-VC
Nature des NoeudNonTerminal-VP (SEQ, CHO, PAR, REP)	Nature des NoeudNonTerminal-VC (SEQ, CHO, PAR, REP)
Tâche et Lien	TâcheChorégraphie
Tâche source d'un lien	ParticipantEmetteur d'une TâcheChorégraphie
Tâche cible d'un lien	ParticipantRécepteur d'une TâcheChorégraphie
Message d'un lien	Message d'une TâcheChorégraphie

L'algorithme *Construire-Arbre-VC* utilise également un ensemble de fonctions qui permettent la manipulation des Arbres-VP-Liées et de l'Arbre-VC à construire. Ces fonctions sont décrites dans le tableau IV-9.

Tableau IV-9. Les fonctions permettant la construction d'un Arbre-VC

Concepts	Fonctions	Description
Nœud-VP d'une forêt Arbres-VP-liées	estTâche()	renvoie Vrai si un Nœud-VP est une tâche, et Faux sinon.
	getLien()	Renvoie le lien auquel le Nœud-VP est impliqué
	Frère()	Renvoie le Nœud-VP qui représente le frère d'un Nœud-VP
	FilsSuivant()	Renvoie le fils suivant d'un NœudNonTerminal-VP
	getParticipant()	Renvoie le participant auquel un Nœud-VP est impliqué
	Parcourir()	Renvoie Vrai s'il est nécessaire de parcourir le sous-arbre dans lequel le Nœud-VP est impliqué et faux sinon.
	listeFils()	Renvoie la liste des fils du Nœud-VP
	getTypeRoutage()	Renvoie le type de routage utilisé dans un NœudNonTerminal-VP
Lien d'une forêt Arbres-VP-liées	getSource()	Renvoie le NoeudTerminal-VP qui représente la source d'un lien
	getCible()	Renvoie le NoeudTerminal-VP qui représente la cible d'un lien

	getMessage()	Renvoie le message porté par un lien
Arbre-VC	getCourant()	Renvoie le nœud courant d'un Arbre-VC
	définirTâcheChorégraphy(pi,m,p)	Définit la TâcheChorégraphie composée par le message m, le participant émetteur du message Pi et le participant récepteur P.
	DéfinirNoeudNonTerminal-VC (n, r)	Définit le NœudNonTerminal-VC qui correspond au NœudNonTerminal-VP n et qui utilise le type de routage r
	ajouterNoeud-VC(n,p)	Ajoute le Nœud-VC n à un Arbre-VC comme un fils du NœudNonTerminal-VC p

L'algorithme de la fonction *Construire-Arbre-VC* inclus un seul paramètre correspondant à un Nœud-VP de la forêt Arbres-VP-Liés à transformer. La première valeur de ce paramètre est *Racine(Initial(l-vpt))* qui correspond à la racine de l'Arbre-VP qui débute la forêt Arbres-VP-Liés *l-vpt* à transformer (*i.e.,* l'arbre qui envoie le premier message). Dans l'exemple de la collaboration *Examen Radiologique*, l'Arbre-VP correspondant au processus de la clinique est celle qui débute la forêt Arbres-VP-Liés.

```
Variable Globale vct: Arbre-VC

Algorithme Construire-Arbre-VC(n: Noeud-VP)
Variable Locale n2: Noeud-VP, l:Lien, m: Message
               ctn: NoeudNonTerminal-VC
Begin
   Si n.estTâche() Alors
        l=n.getLien() m=l.getMessage()
        Si n=l.getSource() Alors
               n2=l.getCible()
        Sinon
               n=l.getSource ()
               n2=l.getCible ()
```

```
        Finsi
        /* Définition d'une TâcheChorégraphie ct */

        ct=définirTâcheChorégraphie(n.getParticipant(),m,n2.getParticipan
   t())
        /* Ajouter le noeud ct à l'Arbre-VC */
        vct.addVC-Node(ct,vct.getCourant())
                Si n2.parcourir() alors
                /* Parcourir l'Arbre-VP */
                Construire-Arbre-VC(frère(n2))
                Finsi
   Sinon
        /* n est un Noeud NonTerminal-VP, il faut définir le Nœud
   NonTerminal-VC qui lui correspond*/
        ctn=définirNoeudNonTerminal-VC(n,n.getPattern(n))
        /* Ajouter ce noeud à l'Arbre-VC */
        Vct.addVC-Node(ctn,vct.getCourant())

           Selon n.getPatronCoordination()
                SEQ, RPT:
                        Construire-Arbre-VC(n.filsSuivant())
                        Break
                CHO, PAR:
                        Pour chaque c in n.listFils()
                                Construire-Arbre-VC(c)
                        Fin pour
             Fin selon
   Finsi
Fin
```

La Figure IV.28 montre l'Arbre-VC, le résultat de cette étape qui correspond à la première version de la collaboration *Examen Radiologique*.

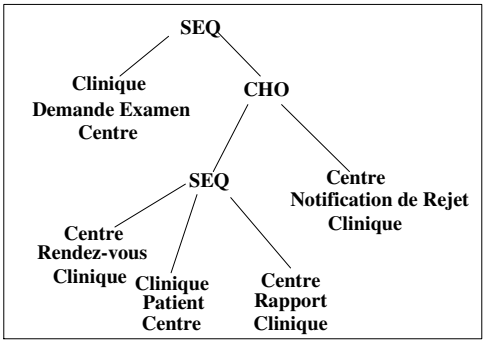


Figure IV.28. Arbre-VC relatif à la première version de la collaboration *Examen Radiologique*

IV.5. Etape4 : Déduction des Chorégraphies

La dernière étape de la démarche de génération que nous proposons consiste à déduire le diagramme de chorégraphie BPMN à partir de l'Arbres-VC généré dans l'étape précédente. Pour ce faire, nous proposons l'algorithme *Construire-Chorégraphie* assurant cette transformation. Cet algorithme utilise les règles de transformation présentées dans le tableau IV.10. Plus précisément, ce tableau illustre qu'un nœud terminal d'un Arbre-VC (NœudTerminal-VC) correspond à une tâche de chorégraphie de BPMN (*i.e., ChoreographyTask*), alors qu'un nœud non terminal (en particulier la nature de ces nœuds, *i.e.,* SEQ, CHO, PAR, REP) est utilisé pour

identifier le type de routage utilisé pour la synchronisation de ces ChoreographyTask (*i.e.,* les SequenceFlow et les Gateways). Il est à noter que nous devons, aussi, spécifier les événements marquant le début et la fin du diagramme de chorégraphie obtenu.

Tableau IV-10. Règles de transformation d'un Arbres-VC en un diagramme de Chorégraphie de BPMN

Arbre-VC	Chorégraphie de BPMN
NœudNonTerminal-VC	*SequenceFlow* and/or *Gateway*
TâcheChorégraphie	*ChoreographyTask*
Participant émetteur d'une TâcheChorégraphie	Initiating participant d'un *ChoreographyTask*
Participant récepteur d'une TâcheChorégraphie	Participant d'un *ChoreographyTask*
Message d'une TâcheChorégraphie	Message d'un *ChoreographyTask*

L'algorithme *Construire-Chorégraphie* utilise un ensemble de fonctions, détaillées dans le tableau IV.11, permettant la manipulation d'un Arbre-VC et d'un diagramme de chorégraphie BPMN.

Tableau IV-11. Fonctions permettant la construction du diagramme de chorégraphie

Concepts	Fonctions	Description
Arbre-VC	estNœudTerminal-VC()	Renvoie Vrai si un nœud est un nœud terminal, et Faux sinon.
	getListeFils()	Renvoie la liste des nœuds fils d'un NœudNonTerminal-VC
Chorégraphie de BPMN	définirChoreographyTask(n) n : NœudTerminal-VC	Définit un ChoreographyTask de BPMN qui correspond au NœudTerminal-VC *n*
	définirTypeRoutage(n) n : NœudNonTerminal-VC	Définit le type de routage (SequenceFlow et Gateways) utilisé pour la synchronisation des ChoreographyTask à partir de la nature du NœudNonTerminal-VC n

	ajouterChoreographyTask(t) t : ChoreographyTask	Ajoute la ChoreographyTask t à un diagramme de Chorégraphie
	ajouterTypeRoutage(p) r : Gateway et/ou SequenceFlows	Ajouter le type de routage r à un diagramme de Chorégraphie

L'algorithme *Construire-Chorégraphie* prend en paramètre un Arbre-VC et renvoie le diagramme de Chorégraphie BPMN correspondant.

```
Algorithme Construire-Chorégraphie(vct: Arbre-VC): Chorégraphie
Local t: BPMN-ChoreographyTask
   ch: Chorégraphie
   c: Arbre-VC
Début
   Si vct.estNœudTerminal-VC()alors
        t = définirChoreographyTask(vct)
        ch.ajouterChoreographyTask(t)
        retourner ch
   Sinon
        // vct est un NoeudNonTerminal-VC
        ch.ajouterTypeRoutage(définirTypeRoutage(vct))
        Pour chaque c de getListeFils(vct)
               Retourner Construire-Chorégraphie(c)
        Fin Pour
   Fin si
Fin
```

La Figure IV.29 montre le résultat de cette étape conformément à la première version de la collaboration *Examen Radiologique* tout en considérant l'Arbre-VC présenté dans la Figure IV.28.

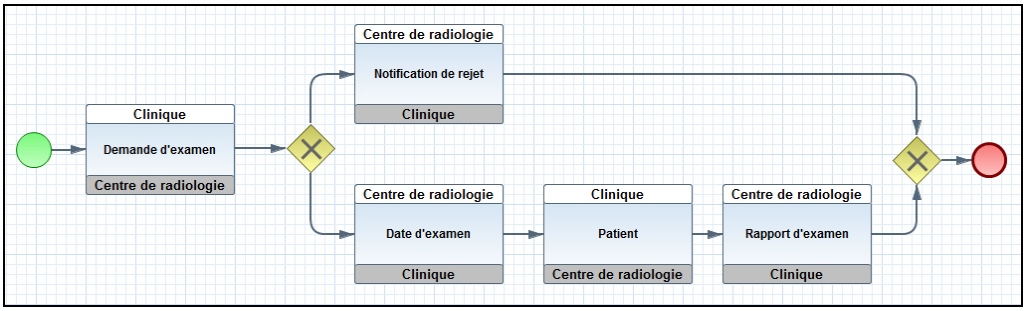


Figure IV.29. Diagramme de chorégraphie relatif à la première version de la collaboration *Examen Radiologique*

V. Conclusion

Dans ce chapitre, nous nous sommes intéressés à la problématique de la flexibilité des processus inter-organisationnels, modélisés sous forme de collaborations et de chorégraphies en BPMN 2.0, tout en adoptant une approche basée sur les versions et sur les patrons. Ainsi, nous avons donné des solutions permettant de résoudre les problèmes suivants :

- Problème de modélisation des versions de collaborations. Pour le résoudre, nous avons proposé des extensions au méta-modèle de BPMN 2.0 pour qu'il puisse supporter les versions des collaborations. Le méta-modèle résultant de cette extension est nommé BPMN4-VCC (BPMN for Versions of Collaborations and Choreographies). L'avantage de ce méta-modèle est qu'il permet de prendre en compte les changements des processus inter-organisationnels à travers la modélisation des versions de collaborations.

- Problème de manipulation des versions de collaborations. Les opérations de création, de modification, de dérivation et de validation des versions de collaborations que nous avons définies permettent au concepteur de gérer les versions de collaborations modélisées conformément à BPMN4V-CC. Afin de simplifier les opérations de création et de modification, nous avons proposé six patrons d'adaptation. Ces patrons permettent de faire passer une version de collaboration d'un schéma à un autre en ajoutant, supprimant, déplaçant, remplaçant, échangeant ou modifiant une interaction.

- Problème de modélisation des versions de chorégraphies. Nous avons proposé une démarche formée de quatre étapes pour la transformation d'une version de collaboration en une version de chorégraphie. L'avantage de cette démarche c'est qu'elle permet la déduction automatique d'une version de chorégraphie à partir d'une version de collaboration, et d'éviter ainsi de modéliser cette version de chorégraphie à partir de zéro. En effet, nous avons proposé des algorithmes appropriés pour assurer cette déduction. Nous avons opté pour cette solution étant donné que les collaborations subsument les chorégraphies. En effet, une chorégraphie n'est qu'une autre représentation d'une collaboration dans laquelle on focalise sur l'orchestration des messages transmis entre les différents partenaires.

Cependant, les contributions détaillées dans ce chapitre présentent un certain nombre d'insuffisances qui seront traitées dans nos futurs travaux. La première est que certains concepteurs peuvent préférer de modéliser directement des chorégraphies sans passer par les collaborations. De ce fait, nous devons étendre BPMN pour permettre la modélisation directe des versions de chorégraphies. La deuxième est relative à la cohérence des schémas des versions de collaborations obtenus suite à l'utilisation des patrons d'adaptation. En effet, nous n'avons considéré que les cas des patrons d'adaptation qui modifient le schéma d'une version de collaboration sans toucher à sa cohérence. Le schéma obtenu doit alors respecter les propriétés de compatibilité et de cohérence énoncées dans [Decker et Weske, 2007]. Il ne comporte pas, par exemple, des cas de blocages (dead-lock). Une étape de vérification des schémas de collaboration produits par l'application des patrons d'adaptation est alors nécessaire. La troisième est relative aux algorithmes de transformation d'une version de collaboration en une version de chorégraphie. Ces algorithmes sont basés sur les deux hypothèses suivantes. La première considère que le schéma de la version de collaboration à transformer est bien structuré et cohérent, ce qui permet la décomposition des versions de processus en fragments SESE. La deuxième suppose

que cette version ne comporte pas (i) des événements intermédiaires, (ii) des événements qui participent à des interactions (*i.e.,* des événements qui sont source ou cible d'un MessageFlow), et (iii) des traitements itératifs.

Le chapitre suivant aborde l'implémentation des extensions apportées à BPMN 2.0 pour la modélisation des versions de processus intra- et inter-organisationnels. Il présente l'outil BPMN4V-Modeler assurant la modélisation et la gestion des versions de processus et de collaboration modélisées conformément aux méta-modèles BPMN4V-PP et BPMN4V-CC.

Chapitre V – BPMN4V-Modeler : un outil d'aide à la modélisation et la gestion des versions des processus intra- et inter-organisationnels

Résumé. *Ce chapitre présente l'outil BPMN4V-Modeler permettant la validation par expérimentation de nos contributions. Il détaille les différentes fonctionnalités de cet outil qui permet de modéliser et de manipuler les versions de processus intra-organisationnels (i.e., les processus privés) et inter-organisationnels (i.e., les collaborations* et *les chorégraphies) conformément à BPMN4V-PP et BPMN4V-CC. Ce chapitre présente également les détails relatifs à l'implémentation de ces fonctionnalités.*

Sommaire du Chapitre V.

BPMN4V-Modeler est un outil dédié à la modélisation et la gestion des versions des processus intra- et inter-organisationnels modélisées conformément aux méta-modèles BPMN4V-PP et BPMN4V-CC. Cet outil étend les fonctionnalités du plug-in existant Eclipse BPMN2 Modeler par :

- l'intégration de la notion de version : BPMN4V-Modeler permet la modélisation (i) des versions de processus privés comportant des versions d'activités et des versions d'événements et faisant appel à des versions de ressources et des versions d'informations, et (ii) des versions de collaborations formées par des versions de processus et des versions de messages ;
- la faculté de manipuler les versions des processus privés et des versions des collaborations : BPMN4V-Modeler assure la gestion des versions en termes de création, modification, validation et dérivation des versions.

Dans ce chapitre, nous détaillons les différentes fonctionnalités de l'éditeur BPMN4V-Modeler en indiquant comment nous les avons implémentées et en illustrant son utilisation à travers des exemples. Plus précisément, la première section détaille les outils que nous avons utilisés dans le développement de l'éditeur. La deuxième section donne un aperçu général de BPMN4V-Modeler en mentionnant les principales extensions ajoutées par rapport au plug-in Eclipse BPMN2 Modeler. Les troisième et quatrième sections détaillent l'utilisation de l'éditeur pour la modélisation des versions du processus privé *Examen radiologique* et les versions de la collaboration *Examen radiologique* présentées respectivement dans les chapitres 3 et 4. Quant à la cinquième section, elle donne les résultats de l'évaluation de nos contributions. Finalement, la sixième section fait une synthèse du chapitre.

I. Les outils de développement utilisés

Pour développer l'outil BPMN4V-Modeler, nous nous sommes basés sur les outils de développement suivants : l'environnement de développement Eclipse et le plug-in Eclipse BPMN2 Modeler. Nous donnons dans ce qui suit une brève description de ces outils.

I.1. Environnement de développement Eclipse

Eclipse est une plateforme de développement écrite en Java, fruit du travail d'un consortium de grandes entreprises (IBM, Borland, Rational Rose, HP, etc.). C'est un Environnement de Développement Intégré EDI (Integrated Developement Environnement IDE en anglais) performant et libre ; qui a su trouver sa place comme l'un des environnements de développement Java les plus populaires. La grande force de cet IDE réside dans l'ouverture de son noyau qui permet l'adjonction de très nombreux plug-ins. Il est possible, par exemple, d'intégrer des éditeurs XML, HTML, BPMN, etc. ou encore de déployer ses applications vers la quasi-totalité des serveurs du marché.

Plus précisément, le framework de la plateforme Eclipse est formé d'un ensemble de plug-ins extensibles permettant la création de nouveaux plug-ins qui étendent les

fonctionnalités des précédents à travers l'outil PDE (Plug-in Developement Environement). Le principe d'extensibilité d'Eclipse est basé sur les notions de points d'extension et d'extensions. En effet, tout plug-in Eclipse est extensible via des points d'extension dans son fichier plugin.xml. Un point d'extension peut être décrit par un fichier XSD (XML Schema Definition) ou un fichier DTD (Document Type Definition) définissant l'extension possible à partir de ce point. Tout plug-in qui étend les fonctionnalités d'un plug-in extensible déclare des extensions conformes aux grammaires (décrites dans les fichiers XSD ou DTD) des points d'extension correspondants. La Figure V.1 présente le principe d'extensibilité d'Eclipse.

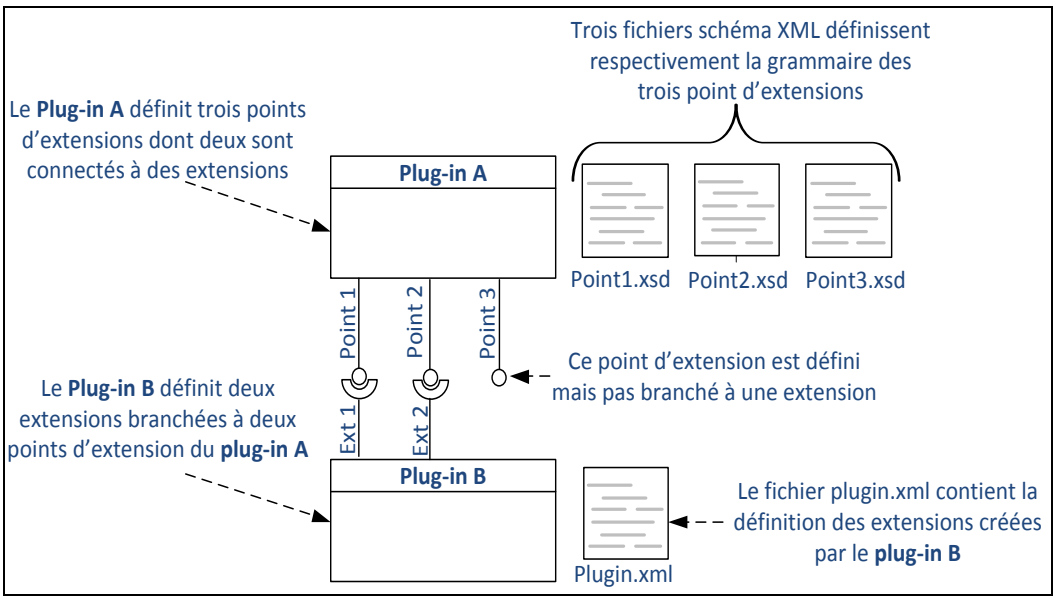


Figure V.1. Principe d'extensibilité d'Eclipse

Cette Figure montre que le plug-in A est extensible du fait qu'il définit les trois points d'extensions : Point 1, Point 2 et Point 3. Les grammaires associées à ces trois points d'extensions sont définies respectivement dans les fichiers point1.xsd, point2.xsd et point3.xsd. Le plug-in B étend le plug-in A par la définition de deux extensions qui correspondent aux points d'extension Point 1 et Point 2 du plug-in A. Ces extensions sont définies dans le fichier plugin.xml sous forme de code XML représentant deux instances des schémas point1.xsd et point2.xsd .

I.2. Plug-in Eclipse BPMN2 Modeler

Eclipse BPMN2 Modeler est un plug-in Eclipse dédié à la modélisation des diagrammes BPMN. Il offre un environnement graphique permettant d'assister les concepteurs dans la création et l'édition de processus privés, de collaborations et de chorégraphies. Cet outil permet de :

- prendre en compte de la majorité des éléments graphiques et des attributs de la notation BPMN 2.0,
- personnaliser des éléments de la notation par l'adjonction de nouveaux attributs,

- ajouter de nouveaux éléments graphiques à la notation dans la palette, et
- sélectionner de l'ensemble des éléments et attributs à utiliser lors de la modélisation des diagrammes.

L'éditeur Eclipse BPMN2 modeler est soutenu par RedHat qui est un leader des fournisseurs de solutions libres dans le monde. Le développement de cet outil est basé sur les plug-ins Eclipse Graphiti et EMF (Eclipse Modeling Framework). Nous présentons, dans ce qui suit, l'interface graphique offerte par cet éditeur ainsi les plug-ins utilisés pour son développement.

I.3. Interface

La Figure V.2 présente la fenêtre principale du plug-in BPMN2 Modeler. Cette fenêtre est composée de cinq types de sous-fenêtres : une vue de navigation, une fenêtre d'édition, une palette, une vue d'arborescence et un vue des propriétés.

- **La vue de navigation ❶** : permet de parcourir et d'afficher les diagrammes modélisés.
- **La fenêtre d'édition ❷ :** représente l'espace de travail où se fait l'édition des diagrammes. Chaque diagramme modélisé est présenté dans un onglet. Pour ajouter un élément graphique, il suffit que l'utilisateur de l'éditeur le sélectionne à partir de la palette et faire un *glisser-déposer* dans la fenêtre d'édition.

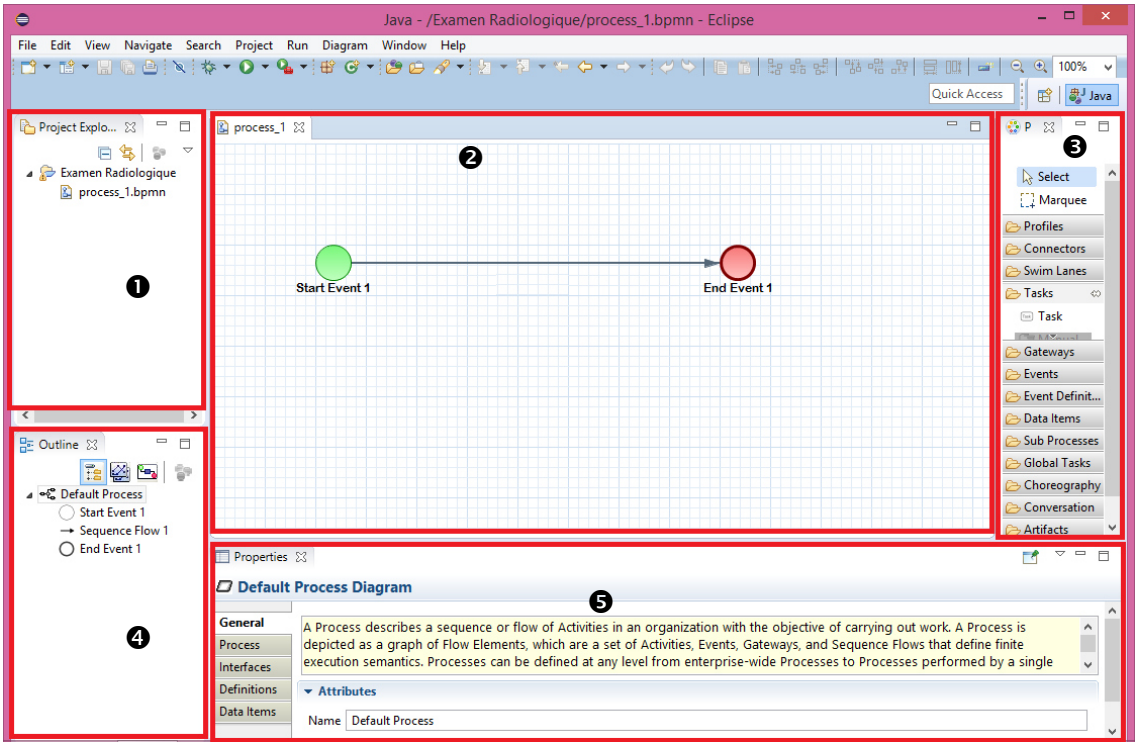


Figure V.2. Interface graphique du plug-in Eclipse BPMN2 Modeler

- **La palette ❸** : comporte tous les éléments graphiques de la notation BPMN 2.0. Ces éléments sont organisés en catégories : les connecteurs, les tâches, les Gateways, les Events, etc.

▪ **La vue d'arborescence** ❹ : offre un aperçu sur les éléments du diagramme existant dans la fenêtre d'édition. Les éléments de cette vue sont présentés sous forme d'une arborescence et sont synchronisés avec les éléments de la fenêtre d'édition. En effet, si un élément du diagramme de la fenêtre d'édition est sélectionné, il sera souligné dans la vue d'arborescence. Inversement, lorsqu'un élément est souligné dans la vue d'arborescence, il sera également mis en relief dans la fenêtre d'édition.

▪ **La vue des propriétés** ❺ : permet d'afficher les propriétés de l'élément sélectionné dans la fenêtre d'édition afin de pouvoir les modifier. Cette fenêtre est composée d'un ensemble d'onglets dont le nombre et le contenu différent d'un élément à un autre. Cependant, le premier onglet « Description » est commun à tous les éléments. Il comporte le nom et une brève description de l'élément sélectionné.

I.4. Plug-in EMF (Eclipse Modeling Framework)

Le projet Eclipse Modeling Framework (EMF) est un framework de modélisation et de génération de code utilisé pour le développement d'applications basées sur un modèle de données structurées. Il permet de créer deux types de modèles : les modèles *ecore* et les modèles de domaine. Un modèle *ecore* représente le méta-modèle définissant les concepts de l'application à modéliser ainsi que leurs relations. Les modèles de domaine correspondent à des instanciations différentes du modèle *ecore*. Par exemple, le plug-in Eclipse BPMN2 Modeler implémente un modèle *ecore* définissant les différents éléments et attributs, et les relations entre les éléments de la notation BPMN 2.0. Les modèles de domaine de ce modeleur correspondent aux diagrammes BPMN qui sont des instances du modèle *ecore*. La Figure V.3 montre un extrait du modèle *ecore* du plug-in BPMN2.

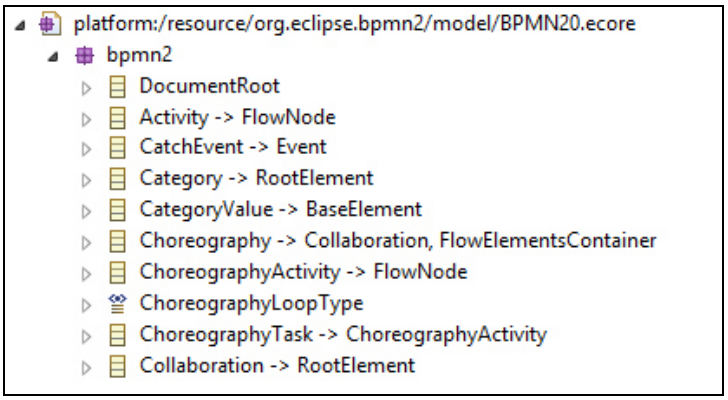


Figure V.3. Extrait du modèle ecore du plug-in Eclipse BPMN2 Modeler

I.5. Plug-in Graphiti

Graphiti est un plug-in Eclipse dédié à l'implémentation des éditeurs graphiques. Il fournit une API Java bien structurée qui simplifie la création des modeleurs en offrant une fenêtre d'édition, une palette et un ensemble d'interfaces assurant la

création et l'édition des diagrammes. Graphiti utilise des modèles de domaine définissant des concepts de l'éditeur à implémenter, basés sur les projets EMF, et manipule des objets Java supportant la représentation graphique de ces concepts.

L'architecture du plug-in Graphiti se base principalement sur les éléments suivants : Domain Model, Pictogram Model, Link Model et Diagram Type Agent.

- Domain Model : ce modèle comporte l'ensemble des éléments supportés par l'éditeur. Généralement, ce modèle doit être conforme à un modèle *ecore* d'EMF contenant les concepts ainsi que leurs relations. Dans le cas du plug-in Eclipse BPMN2 Modeler, Domain Model est conforme au méta-modèle de BPMN tel qu'il est présenté par l'OMG.
- Pictogram Model : ce modèle comporte les informations graphiques des concepts supportés par l'éditeur à implémenter, tels que la forme d'un élément et le style de trait.
- Link Model : ce modèle assure les liens entre les concepts de Domain Model et leurs informations graphiques définies dans Pictogram Model.
- Diagram Type Agent : c'est le composant qui supporte le traitement des interactions entre l'utilisateur et l'éditeur, tels que le redimensionnement de la taille d'un élément, la suppression d'un élément et l'insertion d'un élément à partir de la palette. Il permet de synchroniser les trois modèles présentés précédemment. Par exemple, si l'utilisateur de l'éditeur fait un *glisser-déposer* pour ajouter une tâche à partir de la palette, Diagram Type Agent permet de (i) créer une instance de l'élément tâche dans Domain Model, (ii) enregistrer les informations graphiques de cette tâche (position, taille, etc.) dans Pictogram Model, et (iii) faire le lien entre l'instance de Domain Model et les informations de Pictogram Model (Link Model).

II. Le plug-in BPMN4V-Modeler

II.1. Vue d'ensemble de BPMN4V-Modeler

La Figure V.4 montre que le plug-in BPMN4V-Modeler étend le plug-in Eclipse BPMN2 Modeler à travers la définition d'un ensemble d'extensions associées aux points d'extension définis dans ce plug-in conformément au principe d'extensibilité d'Eclipse évoqué dans la sous-section I.1. Par exemple, afin d'étendre la palette du plug-in Eclipse BPMN2 Modeler, nous avons défini un ensemble d'extensions conforment au point d'extension org.eclipse.bpmn2.runtime.

De plus, le plug-in BPMN4V-Modeler est en interaction avec une base de données XML nommée BPMN4V-Database. Le choix d'une base de données XML est justifié du fait que les données sont structurées sous la forme d'un arbre ou d'un graphe ce qui garantit la rapidité et la simplicité d'accès à ces données. BPMN4V-Database contient toutes les versions de processus modélisées avec l'éditeur BPMN4V-Modeler. En effet, lorsqu'une version est créée ou modifiée à partir de l'éditeur, des requêtes Xquery sont exécutées pour prendre en compte les nouvelles spécifications de cette

version dans la base de données. De plus, les versions sauvegardées dans la base peuvent être sélectionnées et utilisées dans la définition d'autres versions.

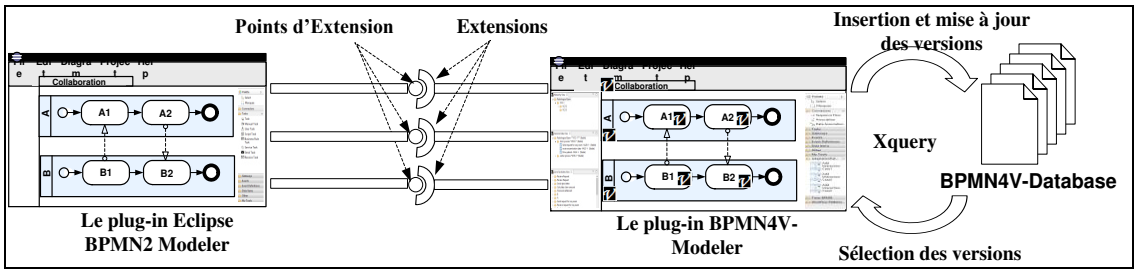


Figure V.4. Vue d'ensemble du plug-in Eclipse BPMN4V-Modeler

II.2. Composants introduits par BPMN4V-Modeler

Nous présentons en Figure V.5 l'interface du plug-in BPMN4V-Modeler comportant les principales nouveautés de ce plug-in par rapport au plug-in Eclipse BPMN2 Modeler. Ces nouveautés peuvent se résumer comme suit :

- ❶ Décoration des éléments concernés, dans la fenêtre d'édition, avec l'icône suivante afin de souligner le fait que BPMN4V-Modeler supporte la notion de version. Cette icone est utilisée pour marquer les concepts versionnables (processus, activités et événements) d'un diagramme.

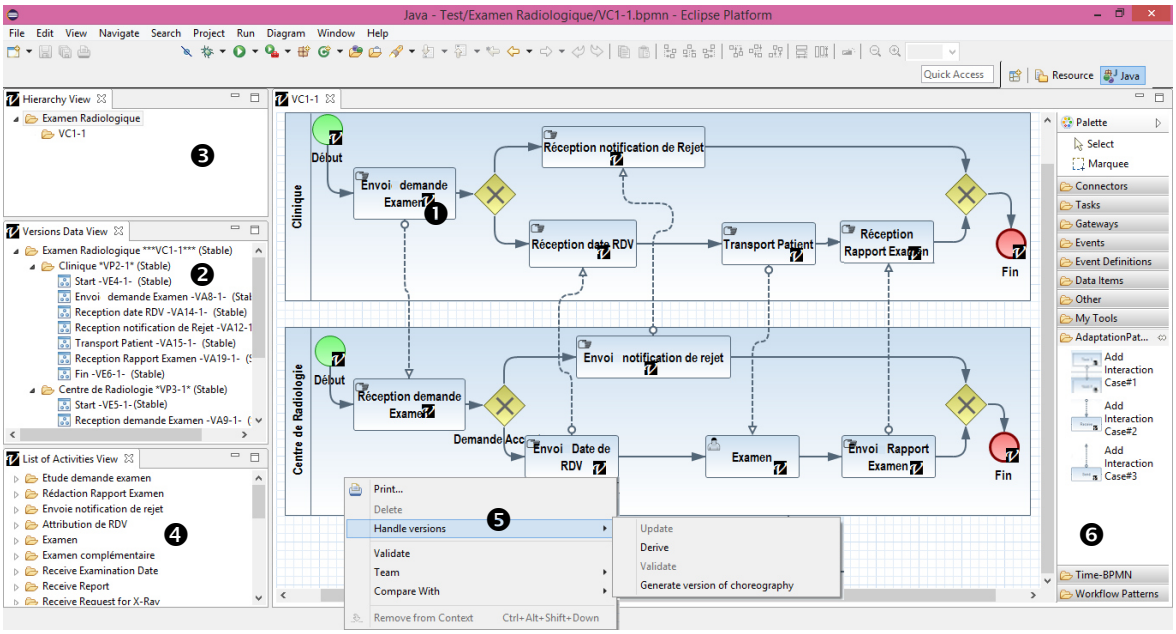


Figure V.5. Interface graphique du plug-in BPMN4V-Modeler

- ❷,❸ et ❹ nouvelles vues Eclipse qui illustrent le détail des versions.
- ❺ Nouveau menu contextuel supportant la gestion des versions.
- ❻ Nouvelle catégorie d'éléments dans la palette pour supporter les patrons d'adaptation.

Nous présentons, dans ce qui suit, une description des nouveautés que comporte le plug-in BPMN4V-Modeler, tout en expliquant le détail de leur mise en œuvre.

II.2.1. Nouvelles Vues Eclipse

Description des vues

BPMN4V-Modeler permet la spécification des détails des versions affichées dans la fenêtre d'édition à travers la définition de trois nouvelles vues Eclipse (Eclipse View) qui sont : *Versions Data View*, *Hierarchy View* et *List of Activities View*.

- La vue Versions Data View, présentée dans la partie ❷ de la Figure V.5, montre le détail des différentes versions qui composent le diagramme actif dans la fenêtre d'édition. Plus précisément, cette vue donne le nom de chaque composant, l'identifiant de la version et l'état de la version (« *En travail* » ou « *Stable* »). Par exemple, la vue présentée en Figure V.5 indique que le diagramme actif dans la fenêtre d'édition correspond à la première version de la collaboration *Examen Radiologique* identifiée par VC1-1 et ayant l'état « *Stable* ». Cette vue indique aussi que cette version de collaboration est composée par la première version du processus de la *Clinique* (identifiée par VP2-1) et la première version du processus du *Centre de Radiologie* (identifiée par VP3-1). De plus, cette vue montre le détail des versions des événements et des activités que comportent les versions des processus VP2-1 et VP3-1. Par exemple, VP2-1 comporte les premières versions de l'événement de début (VE4-1), de la tâche *Envoi demande Examen* (VA8-1), de la tâche *Réception notification de rejet* (VA12-1), de la tâche *Réception date RDV* (VA14-1), de la tâche *Transport Patient* (VA15-1), de la tâche *Réception Rapport Examen* (VA19-1) et de l'événement de fin (VE6-1). La vue indique aussi que toutes ces versions sont à l'état « *Stable* ».

- La vue Hierarchy View présente la hiérarchie de dérivation du concept versionnable (élément) sélectionné dans la fenêtre d'édition. Par exemple, dans la partie ❸ de la Figure V.5, la vue montre l'arborescence de dérivation de la collaboration *Examen Radiologique* puisque le concept versionnable sélectionné est cette collaboration.

- La vue List of Activities View, présentée dans la partie ❹ de la Figure V.5 comporte toutes les activités modélisées préalablement et leurs versions. Cette vue est utilisée dans le cas où le concepteur désire utiliser une version d'activité modélisée dans un autre processus. Ainsi, il peut sélectionner la version à partir de la vue et la faire glisser dans la fenêtre d'édition pour l'ajouter à la définition de la version de processus ou de collaboration en cours de création.

Pour édifier les vues du plug-in BPMN4V-Modeler, nous avons eu recours au point d'extension org.eclipse.ui.views. Nous avons créé trois extensions, instances de ce point, qui correspondent aux trois nouvelles vues. Etant donné qu'il y a une forte similarité entre ces vues, nous présentons, dans ce qui suit, uniquement le détail d'implémentation de la vue « Versions Data View ».

Implémentation de la vue « Versions Data View »

Nous décrivons dans ce qui suit comment nous avons utilisé le point d'extension org.eclipse.ui.views pour définir la vue « Versions Data View ».

La partie gauche de la Figure V.6 montre une définition de ce point d'extension, présentée sons forme d'une description DTD. Cette description indique qu'une extension instance de ce point d'extension peut comporter plusieurs vues (View). Chacune de ces vues est caractérisée par un identifiant (id), un nom (name), le nom de la classe qui l'implémente (class) et éventuellement une icône et une description.

Point extension

```
<!ELEMENT extension (view)*>
<!ATTLIST extension point CDATA #REQUIRED>
<!ELEMENT view (description?)>
<!ATTLIST view
id CDATA #REQUIRED
name CDATA #REQUIRED
class CDATA #REQUIRED
icon CDATA #IMPLIED>
<!ELEMENT description (#PCDATA)>
```

Definition

```
<extension point="org.eclipse.ui.views">
<view id="org.eclipse.bpmn2.modeler.ui.views.VersionsDataView"
name="Versions Data View"
class="org.eclipse.bpmn2.modeler.ui.views.VersionsDataView"
icon="icons/SMALL/version.png"/>
</extension>
```

Figure V.6. Description du point d'extension et de l'extension de la vue « Versions Data View »

La partie droite de la Figure V.6 présente une instance du point d'extension org.eclipse.ui.views qui permet la prise en compte de la vue « Versions Data View ». L'attribut class, qui a pour valeur « org.eclipse.bpmn2.modeler.ui.views.VersionsDataView », spécifie le nom de la classe permettant (i) la création d'un nouvel objet graphique (widget) sous forme d'arborescence (un TreeViewer) et (ii) l'instanciation de la classe VersionsDataContentProvider pour considérer le contenu de ce TreeViewer. Cette dernière classe sélectionne le détail des versions des concepts versionnables contenus dans le diagramme actif de la fenêtre d'édition pour remplir le TreeViewer. La Figure V.7 présente un extrait des classes VersionsDataView et VersionsDataContentProvider.

```
public class VersionsDataView extends ViewPart {

public static TreeViewer viewer;
@Override
public void createPartControl(Composite parent)
{viewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);

 final VersionsDataContentProvider contentProvider = new VersionsDataContentProvider();
 viewer.setContentProvider(contentProvider);}}

public class VersionsDataContentProvider implements IStructuredContentProvider, ITreeContentProvider {

 private TreeParent invisibleRoot;

 public void updateModel(ModelHandler mh){

 invisibleRoot = new TreeParent("");
 invisibleRoot.removeChildren();
 Definitions definitions = mh.getDefinitions();
 List<RootElement> rootElements = definitions.getRootElements();

  for (RootElement element : rootElements)
       {if (element instanceof Collaboration)

          {VersionofCollaboration vc = (VersionofCollaboration) element;
          TreeParent col = new TreeParent(SelectCollaborationVersionDetail(vc.getId()));
          invisibleRoot.addChild(col);
          List<Participant> p= vc.getParticipants();
          Iterator<Participant> i=p.iterator();
           while (i.hasNext())
              {Participant par=i.next();
               VersionofProcess vprocess= par.getProcessRef();
               TreeParent treeObject = new TreeParent(
                SelectProcessVersionDetail(vprocess.getId());
               col.addChild(treeObject);
               createFlowElementTree(treeObject,vprocess.getFlowElements());
              }
          }
       }}}
```

Création d'un TreeViewer

Instanciation de la classe qui supporte le contenu du TreeViewer

Sélection du détail des versions des concepts versionnables du diagramme pour remplir le TreeViewer

Figure V.7. Extrait du code des classes VersionsDataView et VersionsDataContentProvider

II.2.2. Nouveau Menu contextuel « Handle versions »

Le menu contextuel « Handle versions » permet l'implémentation de la dynamique des versions. Ce menu convient pour tous les concepts versionnables appartenant au diagramme actif de la fenêtre d'édition (tels que les Tasks, les events et les Process). Par exemple, le menu présenté dans la partie ❺ de la Figure V.5 correspond à la version de la collaboration VC1-1. Ce menu supporte non seulement les opérations du diagramme d'états-transitions UML présenté dans le Chapitre III, §II.1 (Update, Derive et Validate), mais aussi la génération automatique des versions de chorégraphies à partir des versions de collaborations.

```
public class BPMNToolBehaviorProvider extends DefaultToolBehaviorProvider
implements IFeatureCheckerHolder {
{
@Override
public IContextMenuEntry[] getContextMenu(ICustomContext context) {

  IContextMenuEntry ret[] = null;
  IFeatureProvider featureProvider = getFeatureProvider();
  PictogramElement[] pes = context.getPictogramElements();
  Object bo = featureProvider.getBusinessObjectForPictogramElement(pes[0]);

  if (bo instanceof VersionofCollaboration){

       ContextMenuEntry subMenu = new ContextMenuEntry(null, context);
                        subMenu.setText("Handle versions");
                        subMenu.setSubmenu(true);


      final IFeatureProvider fp = getFeatureProvider();
      UpdateVersionCollaboration up=new UpdateVersionCollaboration(fp);
      ContextMenuEntry subMenuUpdate = new ContextMenuEntry(up, context);
                      subMenuUpdate.setText("Update");
                       subMenu.add(subMenu);

      DeriveVersionCollaboration dp=new DeriveVersionCollaboration(fp);
       ContextMenuEntry subMenuDerive = new ContextMenuEntry(dp, context);
                        subMenuDerive.setText("Derive");
                        subMenu.add(subMenuDerive);

     ValidateVersionCollaboration vp =new ValidateVersionCollaboration(fp);
     ContextMenuEntry subMenuValidate = new ContextMenuEntry(vp, context);
                     subMenuValidate.setText("Validate");
                     subMenu.add(subMenuValidate);

     GenerateChoreography ch =new GenerateChoreography(fp);
     ContextMenuEntry subMenuch = new ContextMenuEntry(ch, context);
                       subMenuch.setText("Generate version of choreography");
                       subMenu.add(subMenuch);

                      ret = new IContextMenuEntry[] { subMenu0 };
                           }
}

               return ret;}
```

Identifié l'élément sélectionné dans le diagramme

Création du sous-menu Handle versions

Création des commandes du sous-menu Handle versions et définition des leurs actions

Figure V.8. Extrait de la classe BPMNToolBehaviorProvider

La définition du nouveau menu contextuel dans BPMN4V-Modeler nécessite la modification du code de la classe BPMNToolBehaviorProvider du plug-in Eclipse BPMN2 Modeler. En effet, cette classe fournit un ensemble d'outils, tels que les info-bulles (ToolTips), les boutons de contexte (ContextButtons) et les menus contextuels (ContextMenu). Ces outils permettent la définition du comportement des diagrammes BPMN ainsi que des éléments qui les composent. Plus précisément, nous avons implémenté la méthode getContextMenu donnée en Figure V.8 ci-dessus présentée. Cette méthode identifie l'élément actif dans le diagramme et construit le menu contextuel correspond. Par exemple, si l'élément actif est une version de collaboration, le menu Handle versions est construit avec ses différentes commandes (Update, Derive,

Validate and Generate version of choreography). De plus, les commandes du menu contextuel sont en concordance avec l'état de l'élément sélectionné (« *Stable* » ou « *En travail* »). Par exemple, les commandes Update et Validate des versions de collaborations ne sont disponibles que si l'état de la version est « *En travail* », alors que les commandes Derive et Generate version of choreography ne sont activées que si l'état est « *Stable* ».

En outre, nous avons implémenté les classes décrivant le comportement des commandes du menu Handle versions. Par exemple, nous présentons en Figure V.9 un extrait de la classe GenerateChoreography qui spécifie le comportement à exécuter lorsque la commande Generate version of choreography est sélectionnée. L'activation de cette commande ne se fait que si la version de collaboration est à l'état « *Stable* ».

```
public class GenerateChoreography extends AbstractCustomFeature {

public GenerateChoreography (IFeatureProvider fp) {
              super(fp);}

@Override
public void execute(ICustomContext context) {
              generateChoreography(getDiagram()); }

@Override
public boolean canExecute(ICustomContext context) {

boolean ret = false;
PictogramElement[] pes = context.getPictogramElements();
Object bo = getBusinessObjectForPictogramElement(pes[0]);

if (bo instanceof VersionofCollaboration && bo.state.compareTo("Stable")==0 )
ret = true;
return ret;}
}
```

Définition du comportement à exécuter si the commande Generate version of choreography est sélectionnée

La commande Generate version of choreography peut être exécutée lorsque l'état de la version de collaboration est Stable

Figure V.9. Extrait de la classe GenerateChoreography

II.2.3. Mise en œuvre du patron « ajouter une interaction » par l'adjonction d'une nouvelle catégorie d'éléments dans la palette

Le plug-in BPMN4V-Modeler permet la mise en œuvre des patrons d'adaptation permettant la modification des schémas des versions de collaboration (*cf.* Chapitre IV). En effet, afin de considérer ces patrons, nous avons ajouté une nouvelle catégorie d'éléments dans la palette comme le montre la partie ❻ de la Figure V.5. Pour ce faire, nous avons eu recours au mécanisme d'extension d'Eclipse à travers l'utilisation du point d'extension « org.eclipse.bpmn2.modeler.runtime », et en particulier l'élément customTask associé à ce point.

Dans ce qui suit, nous donnons une description de la grammaire de l'élément customTask. Puis nous illustrons les instanciations de cette grammaire permettant l'adjonction d'une nouvelle catégorie d'éléments dans la palette.

L'élément customTask

L'élément customTask du point d'extension org.eclipse.bpmn2.modeler.runtime décrit comment nous pouvons considérer un élément personnalisé dans la palette du plug-in BPMN4V-Modeler. Nous présentons dans la Figure V.10 un XSD définissant la grammaire de cet élément dont les attributs sont détaillés par cette grammaire. Certains de ces attributs sont obligatoires alors que d'autres sont optionnels. Parmi les attributs de l'élément customTask nous citons :

- id : qui est l'identifiant unique de la customTask.
- name : qui indique le nom de la customTask.
- feautureContainer : qui indique la classe qui contient le code permettant l'insertion de la customTask dans un diagramme.
- description : qui spécifie la description de la customTask.
- type : qui indique le concept du modèle *ecore* que cette customTask représente.
- category : qui indique la catégorie des éléments de la palette dans laquelle la customTask apparait.
- icon : qui permet de spécifier l'image de la customTask qui apparait dans la palette.

```
<element name="customTask">
<attribute name="id" type="string" use="required"/>
<attribute name="name" type="string" use="required"/>
<attribute name="description" type="string"/>
<attribute name="type" type="string"/>
<attribute name="featureContainer" type="string" use="required"/>
<attribute name="category" type="string"/>
<attribute name="icon" type="string"/>
</element>
```

Figure V.10. Extrait du fichier comportant la grammaire de l'élément customTask

Les extensions associées à l'élément customTask

Les extensions associées à l'élément customTask permettent l'adjonction d'une nouvelle catégorie d'éléments, nommée « Adaptation Pattern », dans la palette du plug-in BPMN4V-Modeler. Cette catégorie permet de considérer le patron d'adaptation « ajouter une interaction », évoqué dans chapitre IV, §III.2, qui permet l'adjonction d'une nouvelle interaction à une version de collaboration. Elle comporte trois éléments correspondant aux trois cas de ce patron. Ces éléments représentent :

- Le cas#1 de ce patron permet d'ajouter une interaction composée d'une tâche d'envoi, d'une tâche de réception et d'un message.
- Le cas#2 permet d'ajouter une interaction entre une tâche d'envoi existante et une nouvelle tâche de réception.
- Le cas#3 permet d'ajouter une interaction entre une nouvelle tâche d'envoi et une tâche de réception existante.

Quant au cas#4 de ce patron qui permet d'ajouter un message entre des tâches d'envoi et de réception existants, il est implicitement considéré par l'éditeur par la connexion de deux tâches existantes à l'aide d'un *MessageFlow* et en associant un message à ce *MessageFlow*.

Les extensions associées au point d'extension customTask

```
<extension
     <customTask
       category="AdaptationPattern"
       name="Add Interaction Case#1"
       description="Create the source task and target task"
       featureContainer="org.eclipse.bpmn2.modeler.examples.customtask.
       MyAdaptationPatternFeatureContainerCase1"
       icon="CPAI1.png">
     </customTask>

     <customTask
       category="AdaptationPattern"
       description="Select the source task and create the target task"
       featureContainer="org.eclipse.bpmn2.modeler.examples.customtask.
       MyAdaptationPatternCase2"
       icon="CPAI2.png"
       name="Add Interaction Case#2">
      </customTask>

      <customTask
        category="AdaptationPattern"
        description="Create the source task and select the target task"
        featureContainer="org.eclipse.bpmn2.modeler.examples.customtask.
        MyAdaptationPatternCase3"
        icon="CPAI3.png"
        name="Add Interaction Case#3">
      </customTask>

</extension>
```

Le résultat obtenu

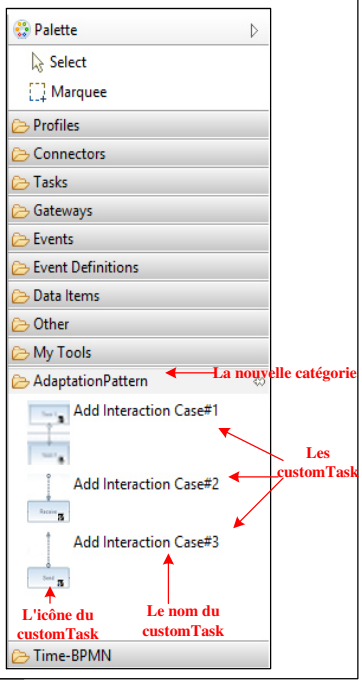


Figure V.11. Les extensions de l'élément customTask

La partie gauche de la Figure V.11 montre une extension du point d'extension org.eclipse.bpmn2.modeler.runtime. Cette extension comporte trois instanciations différentes de la grammaire de l'élément customTask représentant les trois premier cas du patron « ajouter une interaction ». Chaque customTask définit, essentiellement, la catégorie à laquelle il appartient (category), son nom (name), une description qui détaille son comportement (description), la classe java qui implémente l'insertion de ce customTask dans un diagramme (featureContainer) et l'icône associée à ce customTask dans la palette. La partie droite de la Figure V.11 montre le résultat des extensions effectuées sur la palette.

Outre la définition de la nouvelle catégorie « Adaptation Pattern » dans la palette, nous avons implémenté les classes déclarées dans les attributs featureContainer de l'élément customTask. Ces classes décrivent le comportement à effectuer lorsque un utilisateur de l'éditeur veut utiliser un élément de cette catégorie pour modifier le schéma de la version de collaboration activée dans la fenêtre d'édition.

Par exemple, la classe MyAdapatationPatternFeatureContainerCase1, présentée en Figure V.12, spécifie le traitement nécessaire si l'utilisateur de l'éditeur fait un *glisser-déposer* de l'élément « Add Intercation Cas#1 » à partir de la palette. Plus précisément, cette classe permet la vérification de l'état de la version de collaboration à modifier. Si la version est à l'état « *Stable* », un message d'erreur interdisant la modification de cette version est affiché à l'écran. Sinon, la version de la collaboration est modifiée par l'insertion d'une nouvelle tâche d'envoi, une nouvelle tâche de réception et un *MessageFlow* reliant ces deux tâches.

```
public class MyAdapatationPatternFeatureContainer extends
CustomShapeFeatureContainer {

private final static String TYPE_VALUE = "AddInteractioncase#1";
private final static String CUSTOM_TASK_ID =
"org.eclipse.bpmn2.modeler.examples.customtask.MyPattern1" ;
public MyAdapatationPatternFeatureContainer() {
                    // TODO Auto-generated constructor stub
          }
@Override
public Object[] create(ICreateContext context) {

String stateC = select_version_collaboration_state(BPMN2Editor.currentInput.getName());
  if (stateC.compareTo("Stable")==0)
{
JOptionPane.showMessageDialog(null, "The version of collaboration  is Stable. Derive stable
version before use pattern", "Derive stable version before use pattern",
JOptionPane.ERROR_MESSAGE);
return null;
}

  else {

ModelHandler handler 0= ModelHandler.getInstance(getDiagram());
 Process    SourceProcess  = (Process)
handler0.findElement(SelectSourceParticipant(context));
ModelHandler handler1 = ModelHandler.getInstance(getDiagram());
 Process    TargetProcess = (Process)handler1.findElement(SelectTargetParticipant(context));

Task SourceTask = null;
SourceTask = (Task) createBusinessObject(context);
SourceTask.setName("Source Task");
ModelHandler handler2 = ModelHandler.getInstance(getDiagram());
handler.addFlowElement(handler2.findElement(SourceProcess,SourceTask);
PictogramElement Source = addGraphicalRepresentation(context, SourceTask);

Task TargetTask = null;
TargetTask = (Task) createBusinessObject(context);
TargetTask.setName("Target Task");
ModelHandler handler3 = ModelHandler.getInstance(getDiagram());
handler.addFlowElement(handler3.findElement(TargetProcess), TargetTask);
PictogramElement Target = addGraphicalRepresentation(context, TargetTask);

MessageFlow message = null;
ModelHandler handler4 = ModelHandler.getInstance(getDiagram());
m = handler.createMessageFlow(source, target);
Anchor s = GraphitiUi.getPeService().createChopboxAnchor((ContainerShape) Source);
Anchor    t = GraphitiUi.getPeService().createChopboxAnchor((ContainerShape) Target);
CreateConnectionContext c = new CreateConnectionContext();
c.setSourceAnchor(s);
c.setTargetAnchor(t);
AddConnectionContext addContext = new AddConnectionContext(c.getSourceAnchor(),
c.getTargetAnchor());
addContext.setNewObject(message);
getFeatureProvider().addIfPossible(addContext);
putBusinessObject(context, message);

return new Object[] { SourceTask, TargetTask, message, Source, Target };
    }
}
}
```

Vérifier l'état de la version de collaboration et autoriser la modification si l'état est en travail

Insertion de la tâche d'envoi

Insertion de la tâche de réception

Insertion du MessageFlow

Figure V.12. Extrait de la classe MyAdapatationPatternFeatureContainerCase1

II.2.4. Mise en œuvre du patron « supprimer une interaction »

Afin d'implémenter les cas du patron « supprimer une interaction », nous avons revisité le code de la classe MessageFlowFeatureContainer responsable de la gestion (création, modification et suppression) des *MessageFlows* d'un diagramme de collaboration. Cette classe comporte les méthodes *canDelete* et *delete* assurant la suppression d'un *MessageFlow*. Nous présentons dans la Figure V.13 un extrait du code de cette classe.

```
public class MessageFlowFeatureContainer extends BaseElementConnectionFeatureContainer {

public Task src = null;
public Task dest = null;
public MessageFlow messageFlow = null;
public Process Ps = null;
public Process Pd = null;


@Override
public boolean canDelete(IDeleteContext context) {


public void delete(IDeleteContext context) {

}
```

Figure V.13. Extrait du code de la classe MessageFlowFeatureContainer

La méthode *canDelete* effectue les contrôles nécessaires avant l'application du patron de suppression. Elle permet de vérifier l'état de la version de collaboration à modifier. Si l'état est « *Stable* », un message d'erreur interdisant l'application du patron s'affiche à l'écran. Sinon, cette méthode permet d'identifier le *MessageFlow* objet de la suppression ainsi que les tâches d'envoi et de réception qui lui sont associées. Elle détermine aussi les versions de processus auxquelles ces tâches appartiennent. Dans le cas où une de ces deux versions de processus est à l'état « *Stable* », un message s'affiche à l'écran pour demander la confirmation de la dérivation de la version de processus. Un extrait du code de la méthode *canDelete* est donné dans la Figure V.14.

```
@Override
public boolean canDelete(IDeleteContext context) {

String state=
select_version_collaboration_state(getDiagram().getName());
if (state.compareTo("Stable")==0)   {
   JOptionPane.showMessageDialog(null,"This is a Stable version. It have
   to be derived before updating. ","Derive stable version before use
   pattern", JOptionPane.ERROR_MESSAGE);
   return  false;}
else{
   PictogramElement pe = context.getPictogramElement();
   Connection messageFlowConnection = (Connection) pe;
   MessageFlow messageFlow = (MessageFlow)
   BusinessObjectUtil.getFirstBaseElement(messageFlowConnection);



   src = (Task) messageFlow.getSourceRef();
   dest = (Task) messageFlow.getTargetRef();
   Ps = getProcess(src, getDiagram());
   Pd = getProcess(dest, getDiagram());
   String statePs = select_version_process_state(Ps);
   String statePd = select_version_process_state(Pd);
if (statePs.compareTo("Stable")==0 || statePd.compareTo("Stable")==0
{
   int reply =JOptionPane.showConfirmDialog(null,"The Versions of
   processes"+ Ps.getName()+"and/or"+ Pd.getName()+"have to be derived
   since they are stable. Would you like to continue?","Derive processes",
   JOptionPane.YES_NO_CANCEL_OPTION);
      if (reply == JOptionPane.YES_OPTION)
            return true;
      else
            return false;}
}
}
```

Vérifier l'état de la version de collaboration et autoriser la modification si l'état est « En travail »

Identification du MessageFlow à supprimer

Identification des versions des tâches source et cible du MessageFlow à supprimer ainsi que de leurs versions de processus

Vérifier l'état des versions de processus et demander la confirmation de la dérivation

Figure V.14. Extrait du code de la méthode canDelete

Quant à la méthode *delete,* elle calcule, tout d'abord, le nombre de *MessageFlow*s de la tâche d'envoi *nbs* et le nombre de *MessageFlow*s de la tâche de réception *nbt* de l'interaction à supprimer. Ces deux nombres (*i.e.*, nbs et nbt) nous permettent de déterminer si les tâches d'envoi et de réception doivent être supprimées

ou non. En effet, la tâche est supprimée lorsque le nombre de *MessageFlow*s calculé est égal à 1. Par la suite, cette méthode supprime le *MessageFlow* identifié par la méthode *canDelete*. Finalement, elle supprime la tâche d'envoi et / ou la tâche de réception qui ne sont pas invoquées dans d'autres *MessageFlow* (*i.e.*, les tâches dont le nombre de *MessageFlows* calculé est égal à 1) et dérive les versions de processus contenant ces deux tâches. La Figure V.15 donne un extrait du code de la méthode *delete*.

```
public void delete(IDeleteContext context) {

Boolean deleteSource= false;
Boolean deleteTarget= false;

      if (canDelete(context)==  true)

         {int nbs = CalculateTaskNumMsgFlow(src,getDiagram().getName());
                  if (nbs==1)
                  deleteSource=true

          int nbt = CalculateTaskNumMsgFlow(dest,getDiagram().getName());
                  if (nbt==1)
                  deleteTarget=true

         ConnectionDecorator decorator = findMessageDecorator(messageFlowConnection);

                  if (decorator != null)
                  ContainerShape messageShape = BusinessObjectUtil.getFirstElementOfType
                   decorator,ContainerShape.class);

                  if (messageShape != null)
                  ContainerShape labelShape = BusinessObjectUtil.getFirstElementOfType
                  (messageShape,ContainerShape.class);

                  if (labelShape != null)
                  peService.deletePictogramElement(labelShape);
                  peService.deletePictogramElement(messageShape);
                  peService.deletePictogramElement(decorator);
                  super.delete(context);


                  if (deleteSource==true && deleteTarget==true)
                        {deleteBusinessObject(src);
                         derive(Ps);
                         deleteBusinessObject(dest);
                         derive(Pd);
                         deleteSource=false;
                         deleteTarget=false;}

                  else

                  if (deleteSource==true && deleteTarget==false)
                        {deleteBusinesObject(src);
                         derive(Ps);
                         deleteSource=false;}

                  else

                  if (deleteSource==false && deleteTarget==true)
                      {deleteBusinessObject(dest);
                       derive(Pd);
                       deleteTarget=false;}
      }

}
```

Calcul du nombre de MessageFlow reliés à la tâche source

Calcul du nombre de MessageFlow reliés à la tâche cible

Suppression du MessageFlow

Cas#1 suppression des tâches d'envoie et de réception

Cas#3 suppression de la tâche d'envoie

Cas#4 suppression de la tâche de réception

Figure V.15. Extrait du code de la méthode delete

III. Modélisation des versions du processus privé *Examen Radiologique*

Nous montrons dans cette section comment un concepteur peut utiliser l'éditeur BPMN4V-Modeler pour modéliser les versions du processus *Examen Radiologique* présenté précédemment dans le chapitre III (*cf.* §I.4). Par ailleurs, une

démonstration en ligne est disponible dans la vidéo[5] illustrant les fonctionnalités de l'éditeur.

III.1. Création de la première version du processus *Examen Radiologique*

Tout d'abord le concepteur utilise l'interface de création, présentée en Figure V.16, pour créer le processus *Examen Radiologique* ainsi que sa première version (identifiée par VP1-1). Dans cette interface, il doit sélectionner le bouton radio « Create » puis saisir le nom du processus dans la boite de dialogue qui s'ouvre. Une fois le concepteur clique sur le bouton « Finish », le processus se crée. La fenêtre d'édition contient la première version de ce processus VP1-1 qui est à l'état « *En travail* ».

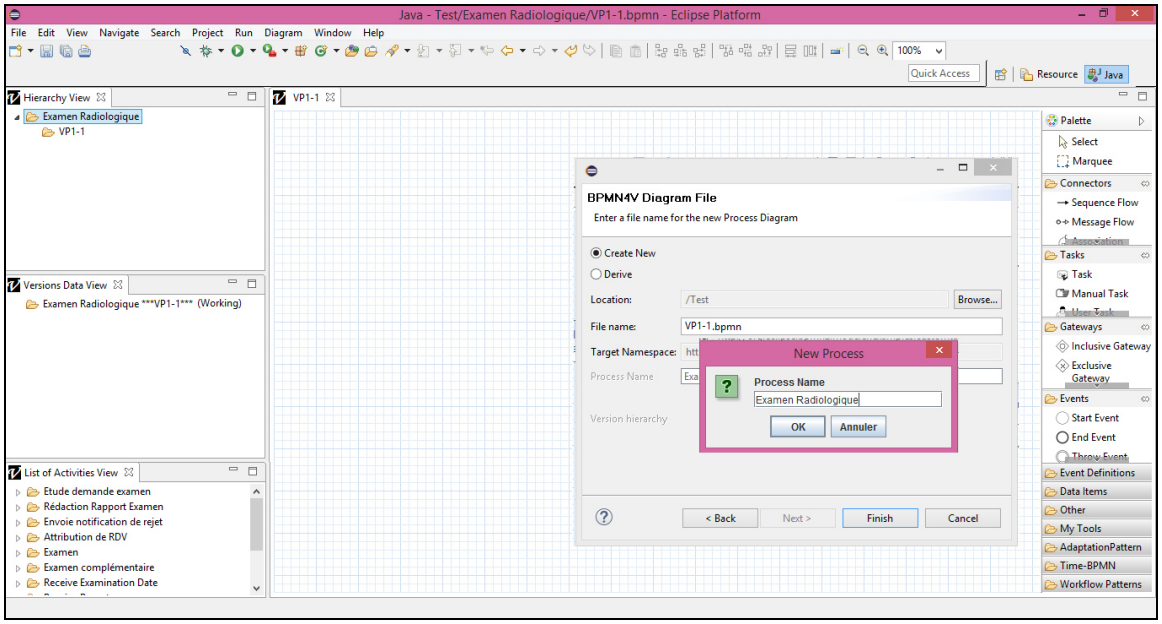


Figure V.16. Interface de création du processus *Examen Radiologique* et de sa première version

Le concepteur définit ensuite les versions d'activités et d'événements qui composent VP1-1. Pour ce faire, il peut soit insérer les composants de cette version à partir de la palette, soit utiliser la vue « List of Activities View » pour ajouter à la fenêtre d'édition une version d'activité modélisée préalablement dans une autre version de processus. Une fois les versions d'activités et d'événements définies, le concepteur ajoute à partir de la palette le type de routage (*i.e.,* les Gateways et les sequenceFlows) reliant les versions ajoutées. La Figure V.17 montre le résultat de création de la première version VP1-1 du processus *Examen Radiologique*.

5 https://www.youtube.com/watch?v=OHX7cciMXO8

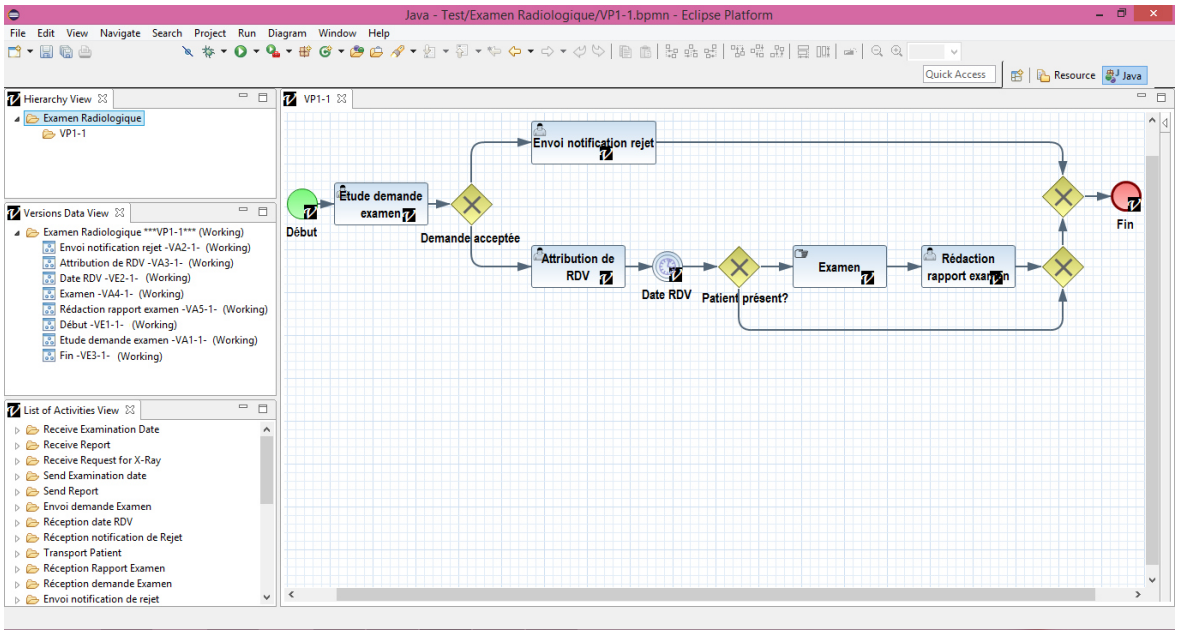


Figure V.17. Résultat de la création de la première version du processus *Examen Radiologique*

III.2. Validation de la première version du processus *Examen Radiologique*

Tout au long de la création de VP1-1 (la première version du processus *Examen Radiologique*), l'état de cette version est « *En travail* ». A partir du moment où la version de processus devient définie complètement, elle peut être validée et donc passer à l'état « *Stable* » pour ne plus subir de modifications. La validation de VP1-1 se fait par la commande « Validate » du menu contextuel « Handle vensions » comme le montre la partie ❶ de la Figure V.18.

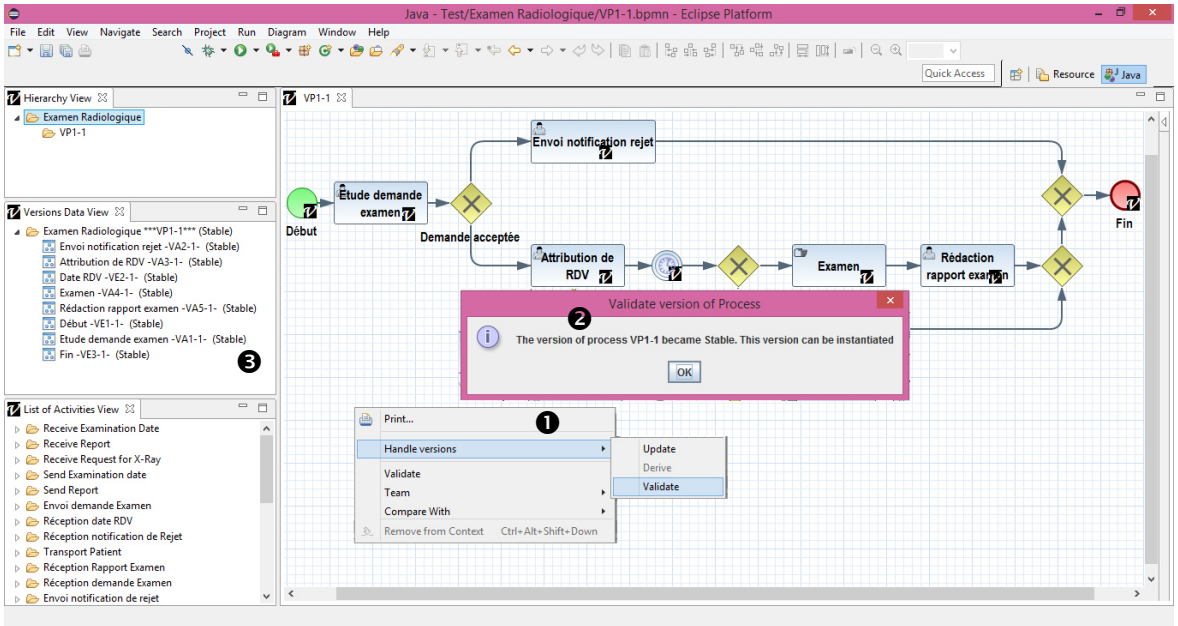


Figure V.18. Validation de la première version du processus *Examen radiologique*

Lors de la validation de VP1-1, le message affiché dans la partie ❷ de la Figure V.18 apparaît. Ce message indique si la version du processus a été validée ou non. Dans le cas où la validation est faite, la version du processus VP1-1 ainsi que tous ces composant versionnables (*i.e.,* les versions d'activités et d'événements) passent à l'état « *Stable* ». La vue « Versions Data View » présentée dans la partie ❸ de la Figure V.18 donne le résultat de la validation de VP1-1.

III.3. Dérivation de la deuxième version du processus *Examen Radiologique* à partir de la première version

Pour créer la deuxième version du processus *Examen Radiologique* (identifiée par VP1-2), le concepteur doit dériver la première version (VP1-1) puis effectuer les changements nécessaires. Pour cette dérivation, il peut soit utiliser le menu contextuel « Handle versions », soit l'interface de dérivation présentée dans la partie ❶ de la Figure V.19. Dans cette interface, le concepteur sélectionne le bouton radio « Derive », le nom du processus (*Examen Radiologique*) et la version à dériver (VP1-1). Puis il clique sur le bouton « Finish » pour effectuer la dérivation de VP1-2 à partir de VP1-1. Cette dérivation donne lieu à la création d'un nouveau diagramme dans la fenêtre d'édition nommé VP1-2 correspondant à la deuxième version du processus *Examen Radiologique* qui est alors dans l'état « *En travail* ». Au début, la version VP1-2 est une copie de la définition de la version dont elle est dérivée. Le résultat de la dérivation est présenté dans la partie ❷ de la Figure V.19.

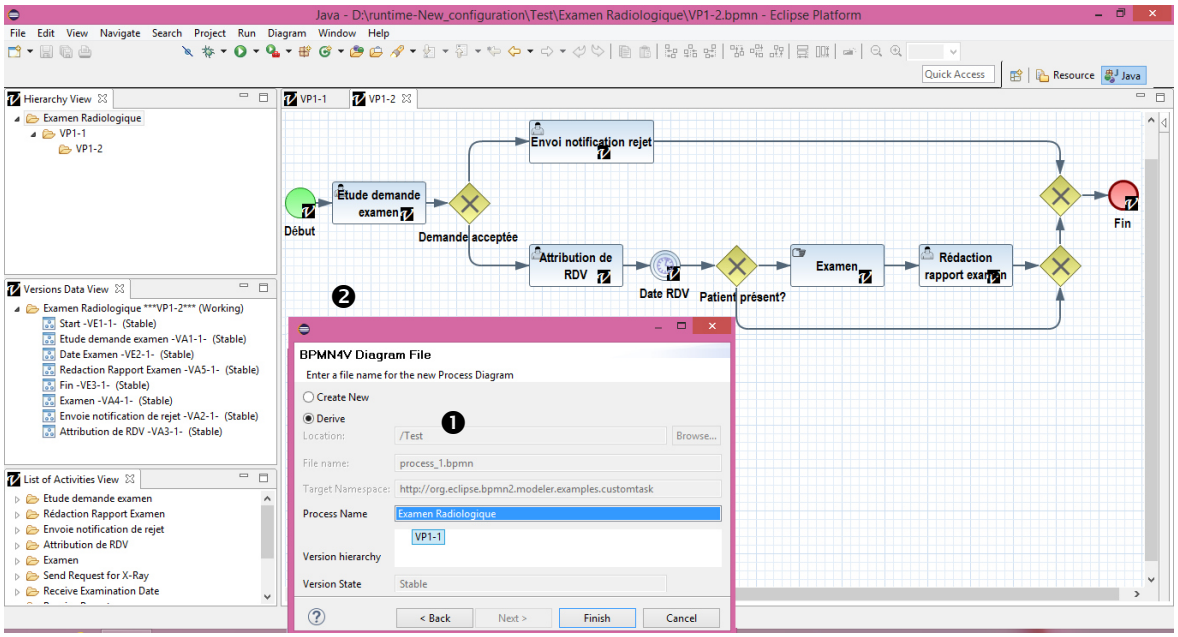


Figure V.19. Dérivation de la deuxième version du processus *Examen Radiologique*

III.4. Modification de la deuxième version du processus *Examen Radiologique*

Une fois dérivée, VP1-2 doit être modifiée pour être en concordance avec la définition de la deuxième version du processus *Examen Radiologique* présentée dans le chapitre III (*cf.* §I.4). Cette modification peut être effectuée par (i) l'adjonction de versions d'activités ou d'événements, (ii) la suppression de versions existantes, (iii) la modification de versions existantes, et / ou (iv) la modification du type de routage utilisé.

Par exemple, l'adjonction de la première version de l'activité « Examen complémentaire » (identifiée par VA6-1) à VP1-2 peut se faire en utilisant l'interface « Define Process Version of Component » présentée dans la Figure V.20. Cette interface s'affiche suite à l'exécution de la commande « Update » du menu contextuel « Handle versions ». Elle donne le détail des versions d'activités et d'événements préalablement modélisées (nom, hiérarchie de dérivation, ressources etc.) et permet l'insertion de la version sélectionnée dans la version de processus à modifier.

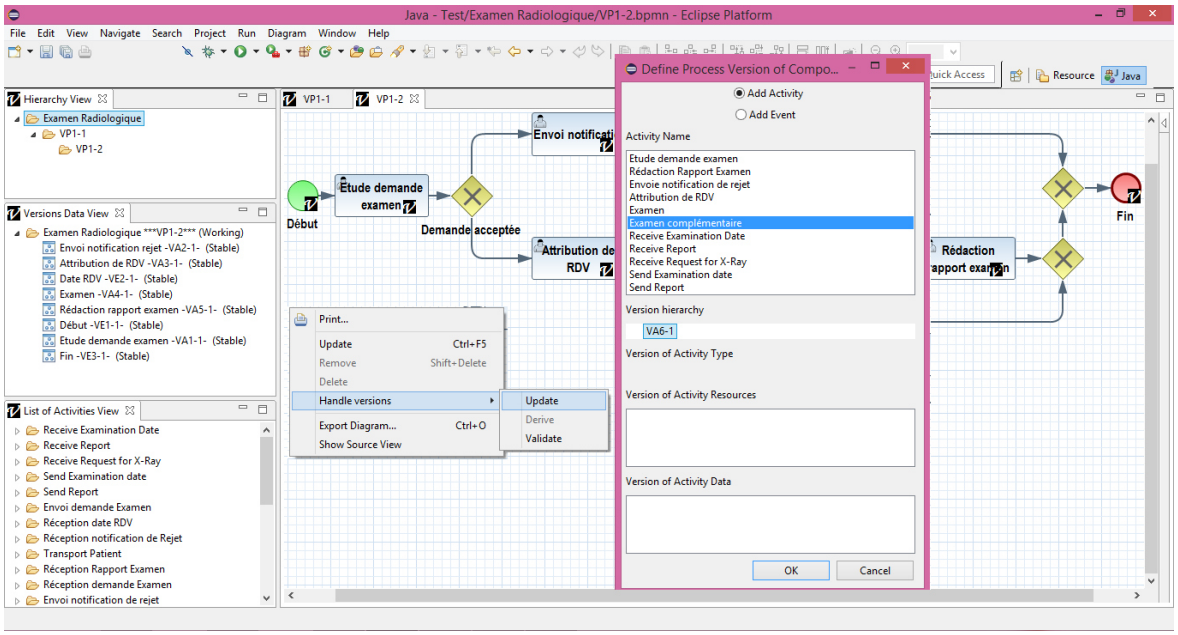


Figure V.20. Utilisation de l'interface « Define Process Version of Component » pour la modification de VP1-2

VP1-2 doit être aussi modifiée en supprimant l'activité *Attribution de RDV* (VA3-1), l'événement *Date Examen* (VE2-1) et en modifiant le routage utilisé. Le résultat de la modification et de la validation de VP1-2 est donné en Figure V.21.

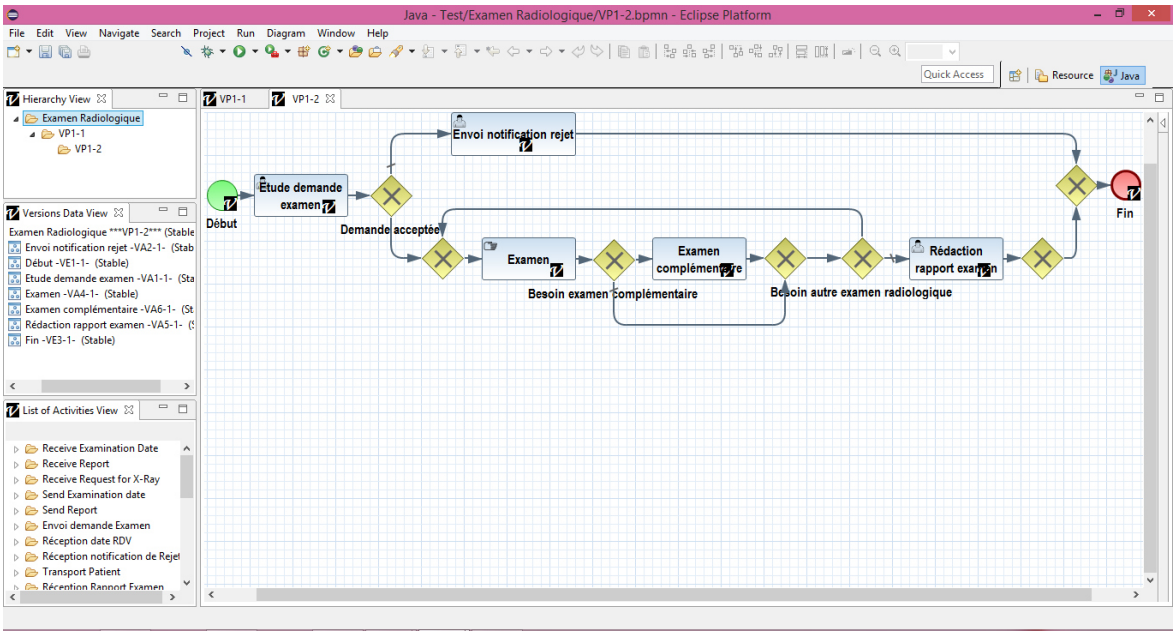


Figure V.21. Deuxième version du processus *Examen Radiologique* VP1-2

IV. Modélisation des versions de la collaboration *Examen Radiologique*

Dans cette section, nous montrons comment un concepteur peut utiliser l'éditeur BPMN4V-Modeler pour modéliser les versions de la collaboration *Examen Radiologique* présenté précédemment dans le chapitre VI (*cf.* §I.3). La vidéo[6] est une démonstration en ligne de cette modélisation.

IV.1. Création de la première version de la collaboration *Examen Radiologique*

La création de la collaboration *Examen Radiologique* ainsi que sa première version identifiée par VC1-1 est à réaliser à travers l'utilisation de l'interface de création présentée dans la partie ❶ de la Figure V.22. Cette interface permet au concepteur de saisir le nom de la collaboration ainsi que les noms des processus qui la composent. Lorsque le concepteur clique sur le bouton « Finish » un nouveau diagramme, nommé VC1-1, est créé et affiché dans la fenêtre d'édition. Ce diagramme, présenté dans la partie ❷ de la Figure V.22, correspond à la première version de la collaboration *Examen Radiologique* composée par la première version du processus de la *Clinique* (identifiée par VP2-1) et la première version du processus du *Centre de Radiologie* (identifiée par VP3-1).

6 https://www.youtube.com/watch?v=-Xazjz9XcA4&feature=youtu.be

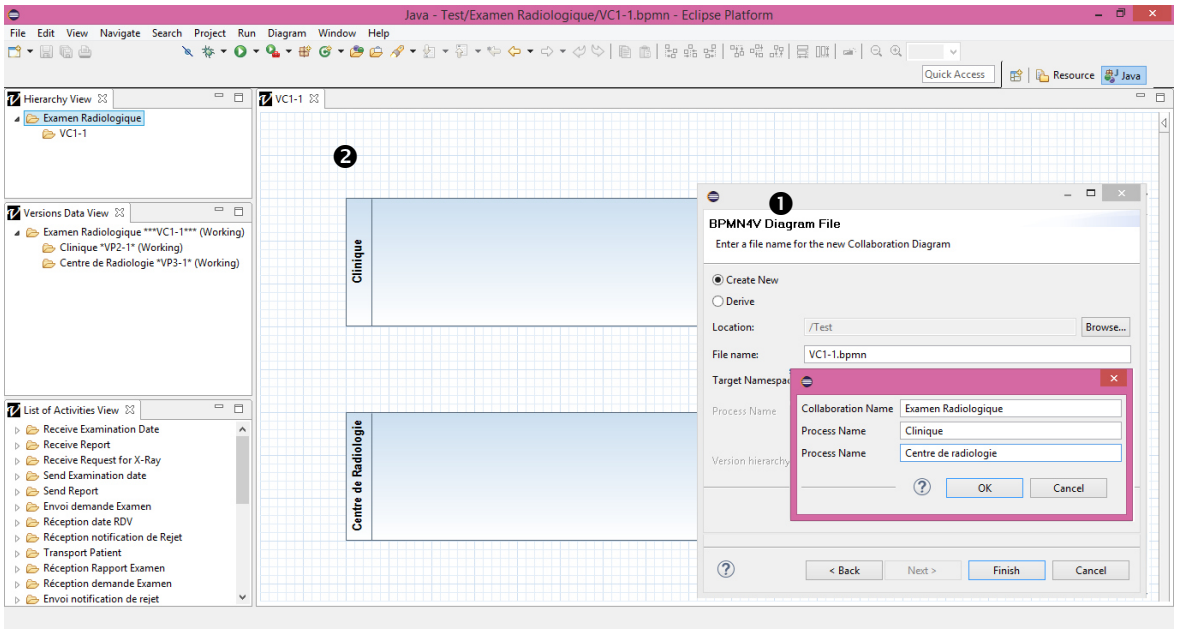


Figure V.22. Création de la première version de la collaboration *Examen Radiologique* à l'aide de l'interface de création

Le concepteur doit maintenant compléter la définition des versions des processus VP2-1 et VP3-1 par l'insertion des versions des activités, des versions des événements, des Gateways et des SequenceFlows qui les composent. La Figure V.23 visualise le résultat de la création de la première version de la collaboration *Examen Radiologique*.

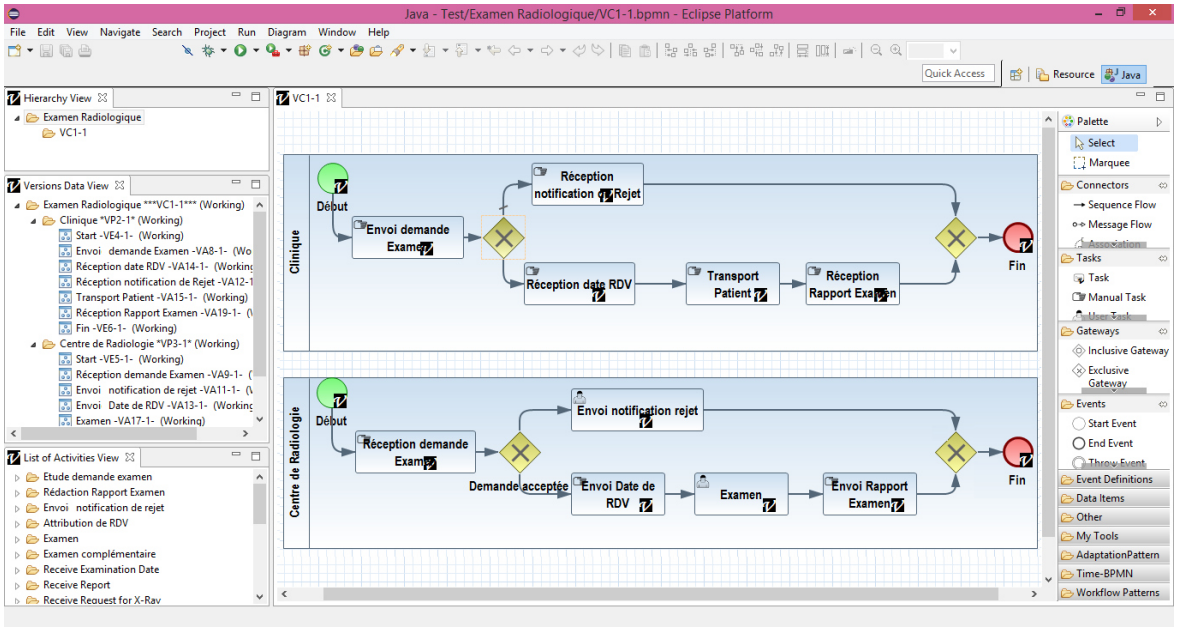


Figure V.23. Résultat de la création de la première version de la collaboration *Examen Radiologique*

IV.2. Validation de la première version de la collaboration *Examen Radiologique*

La première version de la collaboration *Examen Radiologique* VC1-1 doit être validée pour passer à son état final « *Stable* ». De même que les versions des processus privées, la validation des versions de collaboration se fait en utilisant la commande « Validate » du menu contextuel « Handle Versions ». La validation de VC1-1 entraine la validation des versions de processus, des versions des activités et des versions des événements qui la composent, comme le montre la vue « Versions Data View » de la Figure V.24 présentée ci-dessous.

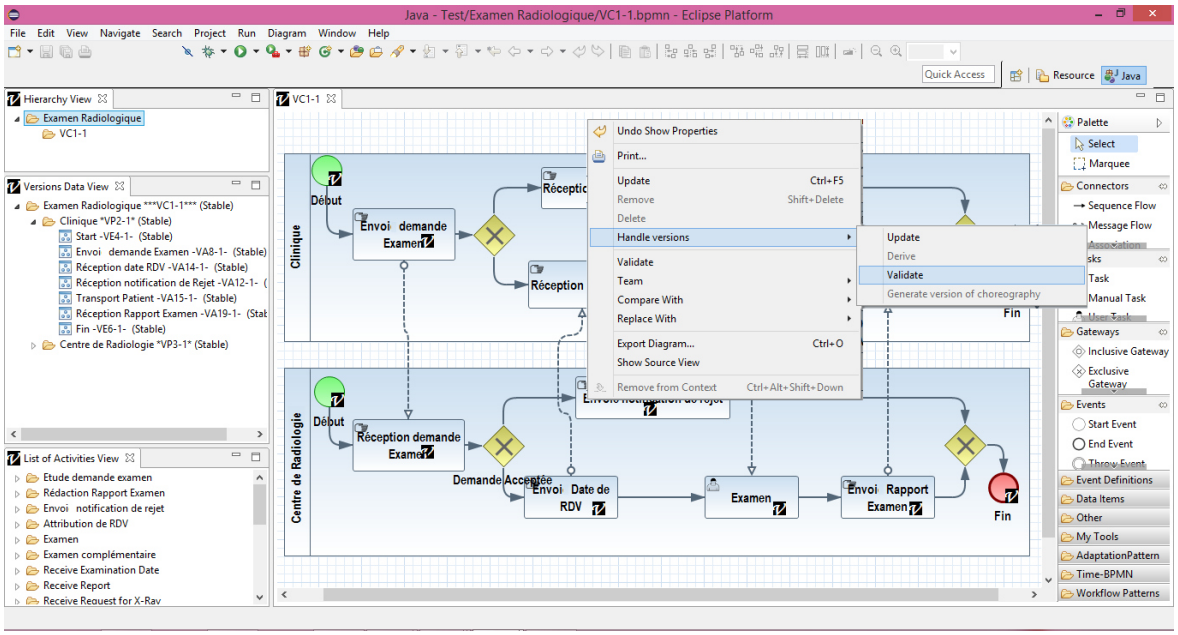


Figure V.24. Validation de la première version de la collaboration *Examen Radiologique*

IV.3. Dérivation de la deuxième version de la collaboration *Examen Radiologique* à partir de la première version

La deuxième version de la collaboration *Examen Radiologique* (identifiée par VC1-2) est définie par dérivation à partir de la première version VC1-1. Pour ce faire, le concepteur utilise la commande « Derive » du menu contextuel « Handle Versions ». Une fois la commande exécutée, un nouveau diagramme, nommé VC1-2, est créé. Ce diagramme correspond à la deuxième version de la collaboration *Examen Radiologique* ; c'est une copie de la définition de la version VC1-1. La Figure V.25 montre le résultat de la dérivation de la version VC1-2 à partir de VC1-1.

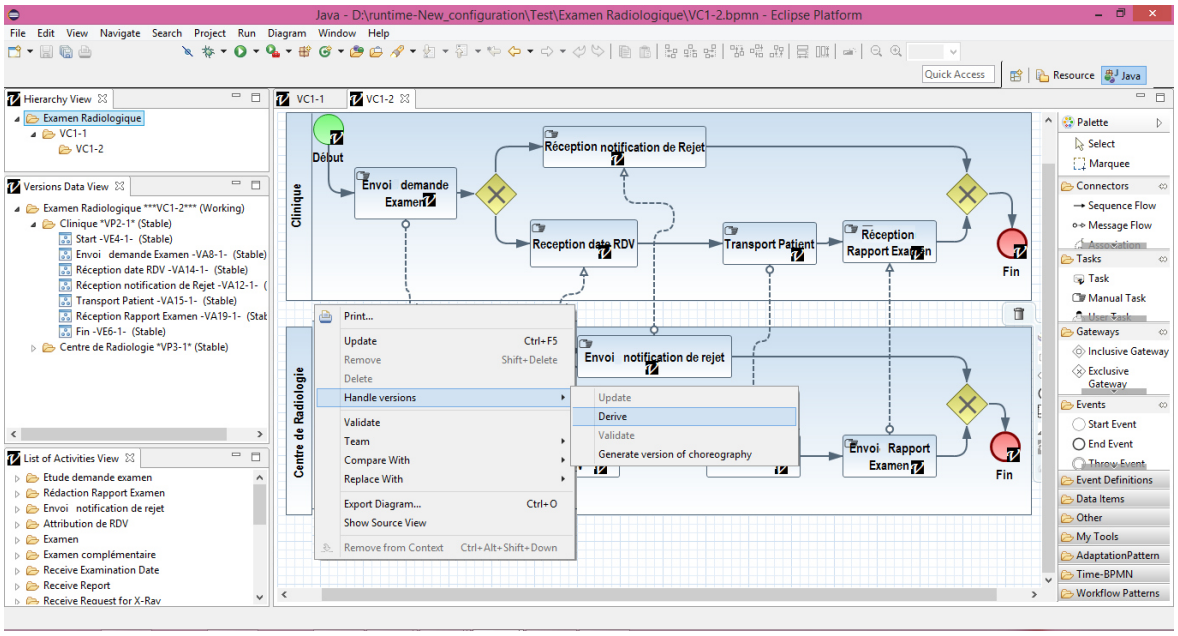


Figure V.25. Résultat de la dérivation de la deuxième version de la collaboration *Examen Radiologique* à partir de la première version

IV.4. Modification de la deuxième version de la collaboration *Examen Radiologique* en utilisant les patrons d'adaptation

A sa dérivation, VC1-2 possède la même définition que VC1-1. Cette version doit être par la suite modifiée pour qu'elle soit conforme à la deuxième version de la collaboration *Examen Radiologique* présentée dans le chapitre IV (*cf.* §I.3). Cette modification consiste en : (i) supprimer l'interaction entre la première version de la tâche *Transport Patient* et la première version de la tâche *Examen*, et (ii) ajouter une nouvelle interaction entre la première version de la tâche *Transport Matériel* et la première version de la tâche *Réaliser clichés*.

Comme nous l'avons expliqué dans le chapitre IV, la modification d'une version de collaboration peut être effectuée en utilisant les patrons d'adaptation. Nous détaillons dans ce qui suit la modification de VC1-2 en utilisant les patrons d'adaptation « supprimer une interaction » et « ajouter une interaction ».

IV.4.1. Suppression d'interaction en utilisant le patron « supprimer interaction »

Nous montrons dans cette partie comment le concepteur peut utiliser le patron « supprimer une interaction » pour modifier la deuxième version de la collaboration VC1-2 par la suppression de l'interaction entre la première version de la tâche *Transport Patient* et la première version de l'activité *Examen*. Pour ce faire, l'utilisateur de l'éditeur doit sélectionner le *MessageFlow* reliant ces deux versions de tâches puis taper la touche supprimer du clavier. Cette action déclenche l'exécution des méthodes *canDelete* et *delete* détaillées précédemment dans la section II.2.4.

Etant donné que les deux versions des processus de la *Clinique* (VP2-1) et du *Centre de Radiologie* (VP3-1) sont à l'état « *Stable* » (voir la vue Versions Data View de la Figure V.26), la méthode *canDelete* affiche à l'écran une boîte de message demandant à l'utilisateur de l'éditeur de confirmer la dérivation de ces deux versions. La partie ❶ de la Figure V.26 présente le message de confirmation affiché suite à la demande de suppression de l'interaction en question. Lorsque le concepteur confirme la dérivation, les versions de processus sont dérivées puis l'interaction est supprimée comme le montre la partie ❷ de la Figure V.26.

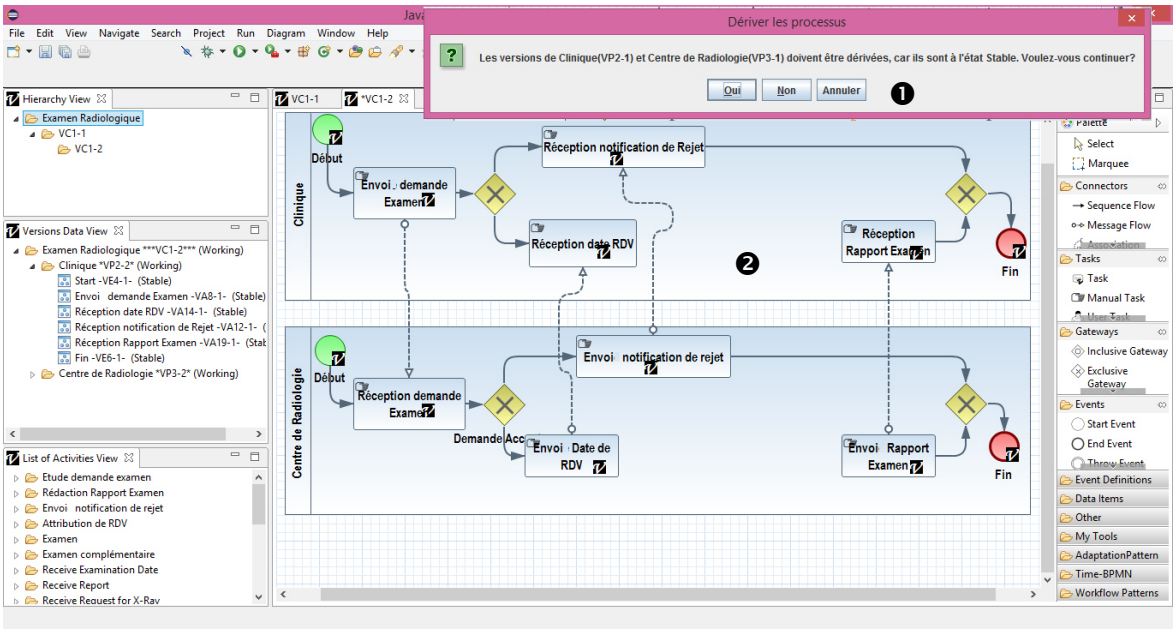


Figure V.26. Utilisation du cas#1 du patron « supprimer une interaction » pour modifier VC1-2

IV.4.2. Adjonction d'interaction en utilisant le patron « ajouter une interaction »

En plus de la suppression d'interaction, VC1-2 doit être modifiée en ajoutant une nouvelle interaction entre la première version de la tâche *Transport Matériel* (appartenant à la deuxième version du processus du *Centre de Radiologie*) et la première version de la tâche *Réaliser les clichées* (appartenant à la deuxième version du processus de la *Clinique*). Cette modification peut être effectuée en insérant la nouvelle interaction à travers l'utilisation du cas#1 du patron d'adaptation « ajouter une interaction ». Nous détaillons dans ce qui suit les étapes nécessaires à la réalisation de cette insertion.

La première étape consiste à faire un *glisser-déposer* de l'élément « Add Interaction Cas#1 » à partir de la palette. Pour ce faire, il faut cliquer sur cet élément avec le bouton gauche de la souris, maintenir ce bouton enfoncé et ne le relâcher que lorsque le curseur atteint le processus qui va contenir la tâche source de l'interaction à insérer. Dans le présent exemple, l'utilisateur de l'éditeur doit relâcher le bouton de la souris lorsqu'il atteint le *Pool* contenant la deuxième version du processus du *Centre de Radiologie*. Lorsque l'utilisateur relâche le bouton de la souris dans l'emplacement

spécifié, un contrôle sera effectué afin de vérifier l'état du processus qui va contenir la tâche source de l'interaction (« *En travail* » ou « *Stable* »). Si l'état de la version de processus est « *Stable* », une interface est affichée pour demander la dérivation de cette version. Sinon, assistant (*i.e.,* un Eclipse Wizard[7]) s'affiche pour guider le concepteur dans la réalisation des étapes qui suivent.

La deuxième étape consiste à spécifier le type de routage utilisé pour coordonner les nouvelles tâches à insérer. Rappelons que l'insertion de ces tâches peut se faire selon trois types de routage différents ; un routage séquentiel, un routage parallèle ou un routage conditionnel (*cf.* les options d'insertion présentée dans le chapitre IV §III.2). Plus précisément, la première interface donnée par le Wizard permet au concepteur de sélectionner le bouton représentant le type de routage qu'il a choisi, puis de cliquer sur le bouton Next pour passer à l'étape suivante. La partie droite de la Figure V.27 présente l'interface de choix du type de routage alors que la partie gauche de cette Figure donne un extrait du code implémentant cette interface.

Extrait du code de la première interface du Wizard

```
public class WizardSelectionOption extends WizardPage
 implements IConstants {
public WizardSelectionOption(ISelection selection) {
 super("wizardPageSP"); //$NON-NLS-1$
 setTitle("Select the option of add");
 setDescription("Select the option");
 setPageComplete(false);}

public void createControl(Composite parent) {

  Composite container = new Composite(parent,
SWT.NULL);
  GridLayout layout = new GridLayout(2, false);
  GridData data;
  Boolean newInteraction = false;
  container.setLayout(layout);

//Créer un bouton pour une insertion en séquence
  Button button = new Button(container, SWT.RADIO |
SWT.PUSH);
  button.setLayoutData(
     new
GridData(SWT.FILL,SWT.CENTER,false,false,1,1));
  button.setImage(Activator.getDefault().
getImage(IConstants.ICON_Seq));

  Label processLabel = new Label(container,
     SWT.WRAP | SWT.NONE);
  data = new
GridData(SWT.LEFT,SWT.FILL,false,false,1,1);
  processLabel.setLayoutData(data);
  processLabel.setText("\n\n\n\n"+"Insert an new task
in sequence between two existing Tasks");
  setControl(container);}
}
```

La première interface du Wizard

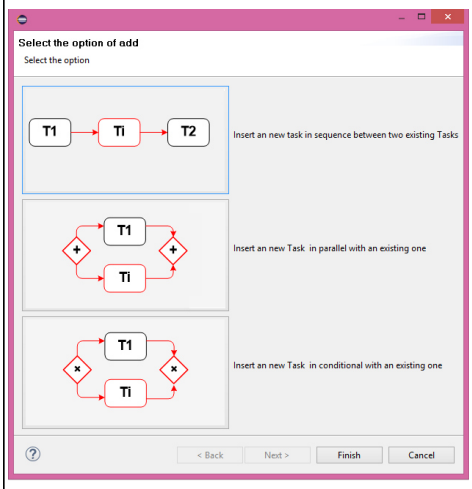


Figure V.27. Interface pour le choix du type de routage

La troisième étape permet la spécification de la version de la tâche source de l'interaction à ajouter à l'aide de la deuxième interface offerte par le Wizard. Cette interface permet (i) d'afficher la version de processus qui va contenir la nouvelle tâche, et (ii) de donner la main au concepteur pour définir la version de la tâche à insérer.

[7]Un Wizard est un composant Eclipse utilisé pour implanter un traitement sous la forme d'une succession d'étapes à l'aide d'interfaces (nommées pages) comportant un bouton Next pour passer d'une étape à une autre.

Lors de cette définition, le concepteur peut soit choisir une version existante, *i.e.,* qui a été définie dans une autre version de processus, soit créer une nouvelle tâche et sa première version. Concernant l'interaction à ajouter dans VC1-2, nous supposons que la tâche d'envoi n'existe pas auparavant. Ainsi, le concepteur doit saisir le nom de la tâche puis cliquer sur Next pour passer à l'étape suivante. La partie droite de la Figure V.28 présente l'interface de définition de la tâche source de l'interaction à ajouter alors que la partie gauche de cette figure donne un extrait du code implémentant cette interface.

Extrait du code de la deuxième interface du Wizard	Interface pour la définition de la tâche d'envoi

```
public class WizardSource extends WizardPage
implements IConstants {
@Override
public void createControl(Composite parent) {

Composite container = new Composite(parent,
SWT.NULL);
GridLayout layout = new GridLayout(3, false);
newInteraction = false;
container.setLayout(layout);
Button button = new Button(container, SWT.BORDER);
button.setImage(Activator.getDefault().
getImage(IConstants.ICON_S1));
            button.setLayoutData(new
GridData(GridData.FILL, GridData.CENTER, true, true,
3, 1));
ContainerShape cp = context.getTargetContainer();
Object b[] = Graphiti.getLinkService()
      .getAllBusinessObjectsForLinkedPictogramElemen
t(
cp.getGraphicsAlgorithm().getPictogramElement());

Participant p = (Participant) b[0];
vprocess = new Label(container, SWT.NONE);
vprocess.setText("Process and version of process");
containerText = new Text(container, SWT.BORDER |
SWT.SINGLE);
containerText.setText(p.getProcessRef().getName() +
"("
      + p.getProcessRef().getId() + ")");
idprocess = p.getProcessRef().getId();
containerText.setEditable(false);
```

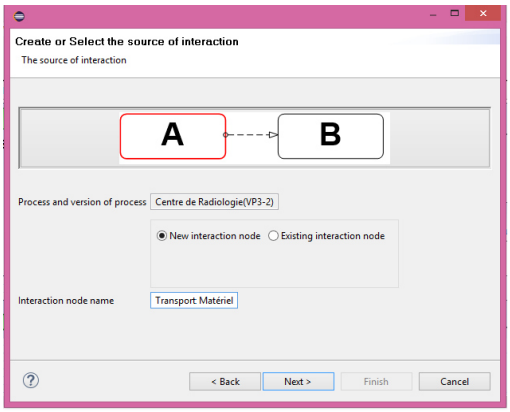


Figure V.28. Interface pour la définition de la tâche source de l'interaction à ajouter

La quatrième étape consiste à définir la tâche de réception de l'interaction à ajouter à l'aide de la troisième interface du wizard. Cette interface comporte une zone de liste qui permet au concepteur de choisir la version de processus qui va contenir la tâche de réception. Une fois le processus choisi, le concepteur procède à la définition de la version de la tâche de réception. De même que dans l'étape précédente, le concepteur peut créer une nouvelle tâche (et sa première version) ou bien il peut sélectionner une version de tâche modélisée préalablement. Dans le cas de VC1-2, nous supposons que la tâche de réception de l'interaction à ajouter a été modélisée dans une autre version de processus. Ainsi, la troisième interface du wizard offre deux zones de liste qui permettent la recherche et la sélection de la première version de la tâche *Réaliser les clichés*. Une fois la tâche de réception définie, l'utilisateur de l'éditeur clique sur le bouton « Finish » pour terminer le wizard. La partie droite de la Figure V.29 présente l'interface de définition de la tâche cible de l'interaction alors que la partie gauche de cette figure présente un extrait du code implémentant cette interface.

Extrait du code de la troisième interface du Wizard

```
public class WizardSource extends WizardPage implements IConstants {
@Override
public void createControl(Composite parent) {

final Composite container = new Composite(parent, SWT.NULL);
GridLayout layout = new GridLayout(3, false);
container.setLayout(layout);
Button button = new Button(container, SWT.BORDER);
button.setImage(Activator.getDefault().getImage(IConstants.ICON_S2));
button.setLayoutData(new GridData(GridData.FILL, GridData.CENTER, true,
true, 3, 1));
ContainerShape cp = context.getTargetContainer();
PictogramElement sourcep = cp.getGraphicsAlgorithm()
                .getPictogramElement();
Object b[] = Graphiti.getLinkService()
                .getAllBusinessObjectsForLinkedPictogramElement(
cp.getGraphicsAlgorithm().getPictogramElement());
Participant p = (Participant) b[0];
Collection<PictogramElement> c = Graphiti.getPeService()
                .getAllContainedPictogramElements(
diagram.getGraphicsAlgorithm().getPictogramElement());
Iterator<PictogramElement> ii = c.iterator();
PictogramElement[] ciblep = new PictogramElement[4];
while (ii.hasNext())

{
PictogramElement pp = ii.next();
if (pp.toString().contains("ContainerShapeImpl") && sourcep != pp
                        && pp.eContainer() == cp.eContainer()) {
ciblep[i] = pp;
i++;}}
vprocess = new Label(container, SWT.NONE);
vprocess.setText("Process and version of process");
containerText = new Combo(container, SWT.BORDER);
int j = 0;
final Participant[] par = new Participant[4];
while (ciblep[j] != null) {
Object bp[] = Graphiti.getLinkService()
        .getAllBusinessObjectsForLinkedPictogramElement(ciblep[j]);
      if (bp[0].toString().contains("Participant_")) {
            Participant pt = (Participant) bp[0];
            containerText.add(pt.getProcessRef().getName() + "("
                        + pt.getProcessRef().getId() + ")");
                  par[j] = pt;
            }
            j++;
      }
```

Interface pour la définition de la tâche de réception

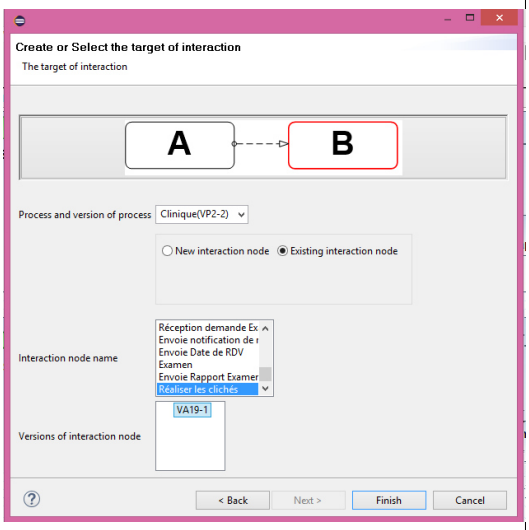


Figure V.29. Interface pour la définition de la tâche cible de l'interaction à ajouter

La Figure V.30 visualise le résultat obtenu suite à la modification de VC1-2 en utilisant les patrons d'adaptation.

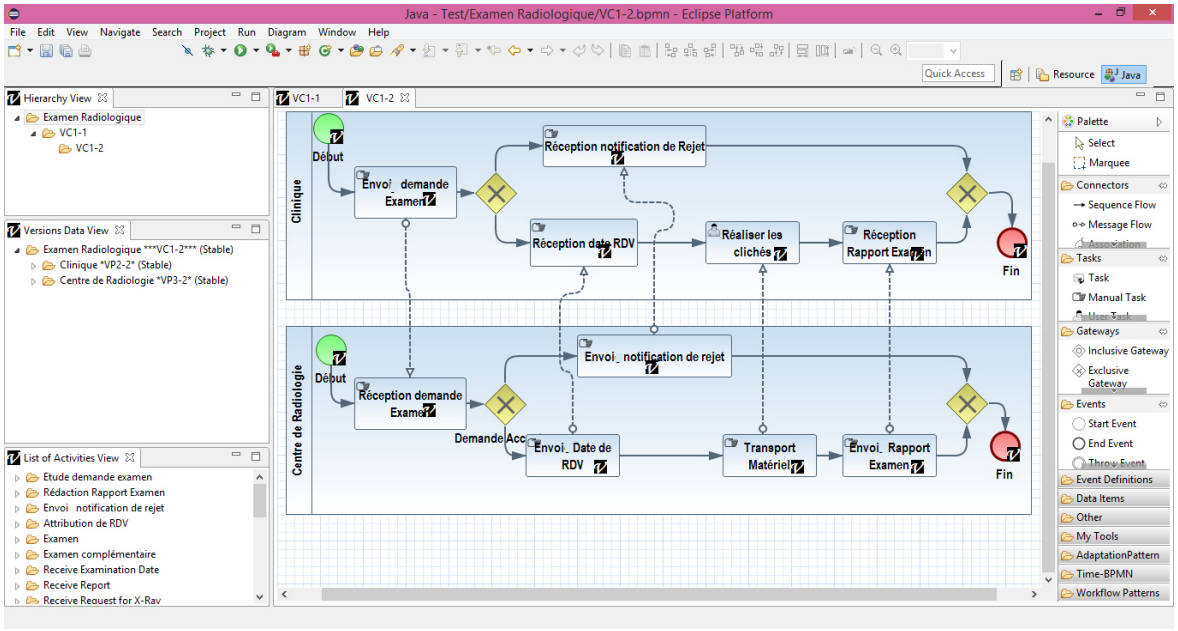


Figure V.30. Deuxième version de la collaboration *Examen Radiologique*

Finalement, VC1-2 peut servir de base pour déduire automatiquement le diagramme de chorégraphie correspondant à cette version. Ce diagramme est créé suite à l'exécution de la commande « Generate version of choreography » comme nous l'avons détaillé en section II.2.2. Le résultat de cette génération est affiché dans la Figure V.31.

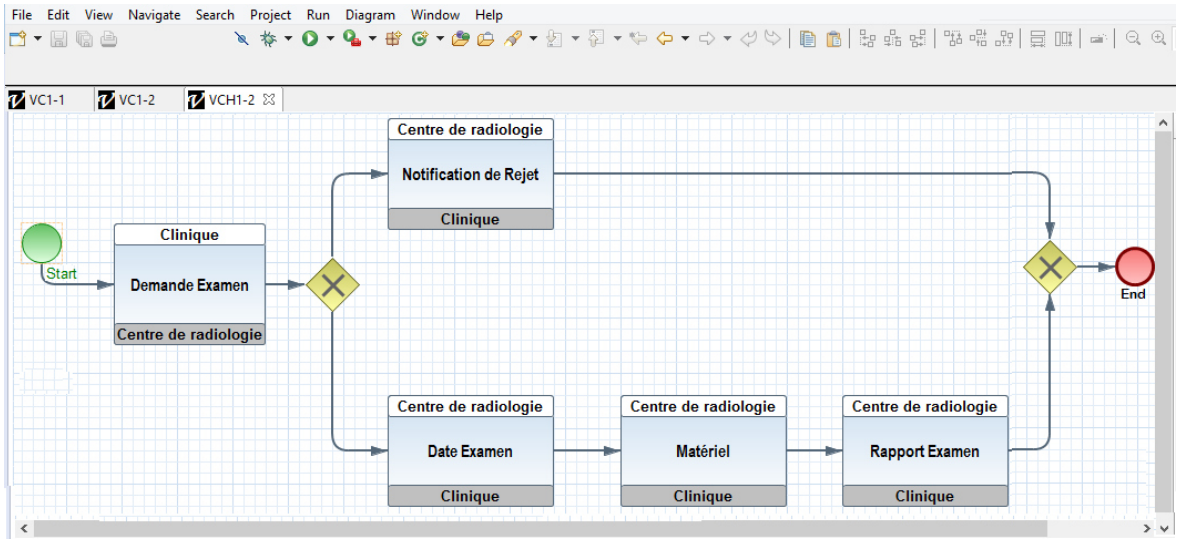


Figure V.31. Version de choreographie générée automatiquement à partir de VC1-2

V. Evaluation

Afin d'évaluer nos contributions, notamment BPMN4V-PP et BPMN4V-CC, nous avons fait appel à un échantillon de 20 doctorants et étudiants en master. Les membres de notre échantillon ont suivi un cours autour de la flexibilité des processus, dans lequel nous avons introduit nos contributions. Ils sont répartis en trois classes comme suit :

- Classe1 composée de quatre membres qui sont des experts dans le domaine des systèmes d'information orientés processus et BPMN 2.0.
- Classe2 composée de six membres qui sont formés en méta-modélisation.
- Classe3 composée de dix membres qui sont formés dans le domaine des systèmes d'information généralisés.

L'évaluation que nous avons proposé a deux objectifs : (i) évaluer nos contributions, et plus précisément BPMN4V-PP et BPMN4V-CC, la dynamique des versions, les patrons d'adaptation et l'éditeur BPMN4V-Modeler et (ii) faire une comparaison entre le plug-in Eclipse BPMN2 Modeler et notre éditeur BPMN4V-Modeler. Dans ce qui suit nous détaillons ces deux points.

V.1. Evaluation des contributions

Durant cette évaluation, nous nous sommes basés sur les cinq critères d'évaluation suivants : la compréhension de l'approche, l'utilisation des patrons d'adaptation, la facilité d'utilisation de BPMN4V-Modeler, l'ergonomie de BPMN4V-Modeler et la concordance entre l'approche proposée et BPMN4V-Modeler (*i.e.,* le degré d'accord entre nos contributions et l'éditeur que nous avons développé). Le résultat de cette évaluation est donné dans la Figure V.32 ci-dessous présentée.

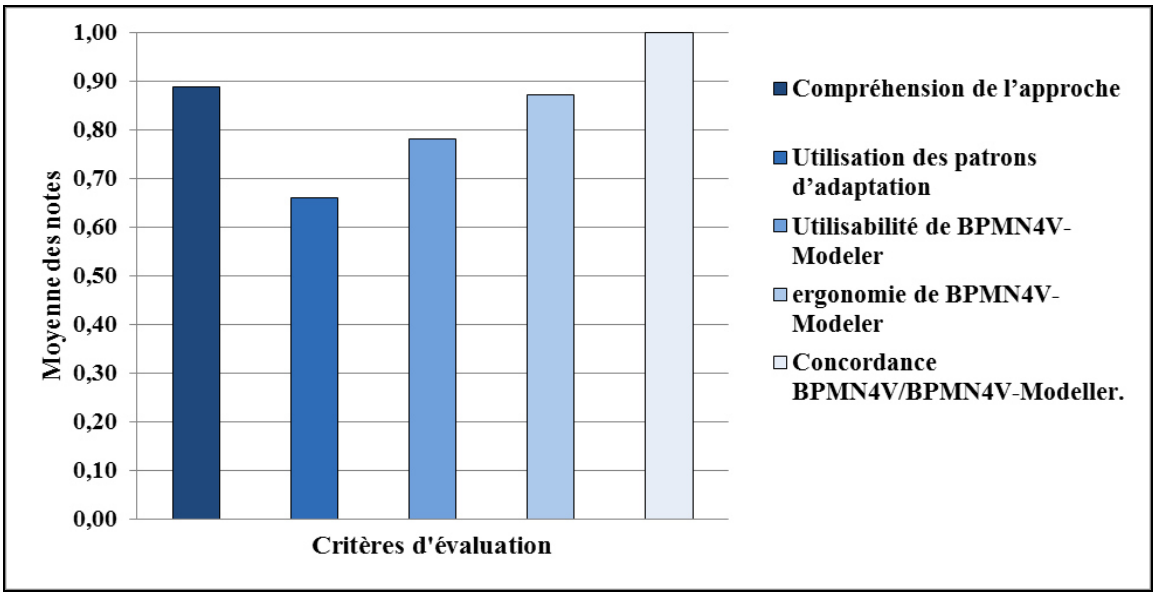


Figure V.32. Résultat de l'évaluation de BPMN4V-PP et BPMN4V-CC

Le premier critère porte sur la **compréhension de l'approche de modélisation** des versions de processus (les processus privés et les collaborations) que nous proposons. En effet, nous avons demandé à chaque membre interviewé de modéliser des versions du processus privé *Examen Radiologique* ainsi que des versions de la collaboration *Examen Radiologique* selon différents scénarii, et nous avons mesuré leur degré de compréhension en fonction des éléments de mesure suivant :

- Versions identifiées : chaque version identifiée est notée sur une échelle allant de 0 (aucun élément de la version n'est identifié) à 1 (tous les éléments de la version sont identifiés). Selon cet élément de mesure, nous avons obtenu une moyenne de 1 pour les membres de la classes1 et de la classe2, une moyenne de 0.8 pour les membres de la classe3, et une moyenne générale de 0.93.

- Hiérarchie de dérivation : il s'agit de mesurer la faculté des membres à identifier les relations de dérivation entre les différentes versions. Chaque relation identifiée est noté sur une échelle allant de 0 (relation non identifiée) à 1 (relation correctement identifiée). Selon cet élément de mesure, nous avons obtenu une moyenne de 1 pour tous les membres de notre échantillon ce qui montre que tous les membres ont réussi à définir la hiérarchie de dérivation du processus privé et de la collaboration *Examen Radiologique*.

- Kit de versionnement : il s'agit de mesurer le degré d'assimilation du kit de versionnement utilisé pour versionner des classes du méta-modèle BPMN 2.0. Pour ce faire, nous avons proposé des QCM portant sur la signification des classes et des relations de ce kit. Chaque bonne réponse est notée 1. Selon cet élément de mesure, nous avons obtenu : une moyenne de 0.9 pour les membres de la classe1, une moyenne de 0.93 pour les membres de la classe2, une moyenne de 0.79 pour les membres de la classes3 et une moyenne générale de 0.87.

- Complexité des Méta-modèles BPMN4V-PP, BPMN4V-CC : la complexité des méta-modèles est notée à 1 si le membre de notre échantillon voit que les méta-modèles sont simples, 0.5 s'ils sont moyennement compliqués et 0 s'ils sont compliqués. Selon cet élément de mesure nous avons obtenu une moyenne de 0.87 pour les membres de la classe2, 0.75 pour les membres de la classe1 et la classe3 et une moyenne générale 0.79.

Etant donnés les quatre éléments de mesure cités préalablement, nous avons mesuré la moyenne des notes attribuées pour notre premier critère d'évaluation et nous avons obtenu une moyenne de 0.91 pour les membres de la classe1, une moyenne de 0.95 pour les membres de la classe2, une moyenne de 0.83 pour les membres de la classe3 et une moyenne générale de 0.89 ce qui montre que notre approche est simple et compréhensible aussi bien pour les membres formés en modélisation que pour les autres.

Le deuxième critère d'évaluation porte sur l'utilisation des **patrons d'adaptation**. En effet, nous avons mesuré la faculté des membres à identifier les patrons d'adaptation nécessaires pour la modification des schémas des versions de la collaboration *Examen Radiologique*. Chaque patron identifié est noté sur une échelle allant de 0 (patron non identifié) à 1 (patron correctement identifiée). Selon ce critère d'évaluation, nous avons obtenu une moyenne de 0.83 pour les membres de la classe1, une moyenne de 0.52 pour les membres de la classe2, une moyenne de 0.4 pour les membres de la classe3 et une moyenne générale de 0.66. Ce résultat montre que le pourcentage de réussite de l'identification des patrons est important pour les membres experts en BPMN alors qu'il plus que moyen pour les autres.

Les troisième et quatrième critères d'évaluation portent sur le BPMN4V-Modeler. Nous avons donné la main aux membres pour tester l'éditeur en modélisant les versions du processus privé *Examen Radiologique* et les versions de la collaboration *Examen Radiologique*. Plus précisément, à travers le troisième critère d'évaluation nous avons mesuré l'**Utilisabilité** de l'éditeur alors qu'avec le quatrième critère nous avons mesuré son **ergonomie**. Pour mesurer l'utilisabilité de l'éditeur, nous avons tenu compte des éléments d'évaluation suivants :

- Modélisation des versions : il s'agit de mesurer la faculté des membres à créer, valider, dériver et modifier les versions de l'exemple. Chaque version modélisée est notée sur une échelle allant de 0 (version non modélisée) à 1 (version correctement modélisée). Selon cet élément de mesure, nous avons obtenu une moyenne de 0.69.

- Appréciations des difficultés : il s'agit de mesurer le degré de difficulté de chaque opération effectuée lors de la modélisation d'une version. Le degré de difficulté est noté à 1 si le membre voit que l'opération est simple, 0.5 si l'opération est moyennement complexe et 0 si l'opération est complexe.

En fonction de ces deux éléments d'évaluation, nous avons calculé la moyenne des notes attribuées pour le troisième critère d'évaluation et nous avons obtenu une

moyenne de 0.78. Le quatrième critère d'évaluation portant sur l'ergonomie du modeler est mesuré en fonction des appréciations données par les membres sur les éléments suivants : les écrans, les menus, les vues, les boites de dialogue, la palette, etc. Un élément d'évaluation de l'ergonomie est noté à 1 si le membre voit que l'élément est bien défini (*i.e.,* claire, non ambigu et facile à utiliser), 0.5 s'il est moyen et 0 s'il est insuffisant. La moyenne des notes attribuées pour le quatrième critère est de 0.87.

Le cinquième critère d'évaluation porte sur la **concordance entre l'approche de modélisation et le modeler**. En effet, nous avons demandé aux membres de donner leurs appréciations concernant la proximité entre l'approche de modélisation proposée (méta-modèles, la dynamique des versions, etc.) et BPMN4V-Modeler. Ce critère est noté à 1 si le membre voit que l'outil et l'approche sont concordants, 0.5 s'ils sont moyennement concordants et 0 s'il n'y a aucune concordance. La moyenne des notes attribuées pour ce critère est de 1, ce qui montre que tous les membres pensent qu'il y a une forte concordance entre notre approche et l'éditeur que nous avons développé.

V.2. Comparaison des éditeurs

Dans le but de souligner l'avantage de l'utilisation de notre éditeur BPMN4V-Modeler par rapport au plug-in Eclipse BPMN2 Modeler, nous avons proposé une comparaison entre ces deux éditeurs. Pour ce faire, nous avons demandé aux membres de notre échantillon de modéliser, dans un premier temps, les différents digrammes correspondant au processus privé *Examen Radiologique* et à la collaboration *Examen Radiologique*. Dans un second temps, nous avons demandé aux mêmes membres de modéliser les versions de ces deux processus en utilisant notre éditeur BPMN4V-Modeler. Dans ces deux cas, nous avons mesuré, pour chaque membre, le temps mis dans la modélisation en utilisant les deux éditeurs. Nous avons, par la suite, calculé la moyenne du temps écoulé en distinguant les membres experts en BPMN 2.0 (*i.e.,* les membres de la classe1) des autres membres.

La Figure V.33 donne le résultat de cette évaluation. D'après ce résultat, nous pouvons remarquer que les membres de la classe1 ont mis un temps moyen de 10.5 minutes pour la modélisation des versions de ces deux processus avec BPMN4V-Modeler, alors qu'ils ont mis un temps moyen de 13.5 minutes pour la modélisation des diagrammes avec BPMN 2.0 Modeler, soit un gain de 3 minutes. Concernant les membres des classes 2 et 3, ont mis un temps moyen de 14.75 minutes pour la modélisation des versions des processus avec BPMN4V-Modeler et un temps moyen de 20 minutes pour la modélisation des versions avec BPMN2 Modeler, soit un moyen de 5.25 minutes.

Le résultat de cette comparaison peut être interprété comme suit : Les concepteurs passent moins de temps à modéliser les schémas des processus grâce à l'utilisation des versions et des patrons d'adaptation. En effet, en utilisant BPMN4V-Modeler le concepteur n'est pas amené à modéliser tous les schémas des processus à partir de zéro (from scratch). Il doit juste modéliser la première version, les autres

versions sont créées par dérivation à partir des versions antérieures. De plus, l'utilisation des patrons d'adaptation facilite la modification des schémas des versions de processus puisque ces patrons combinent un ensemble d'opérations élémentaires.

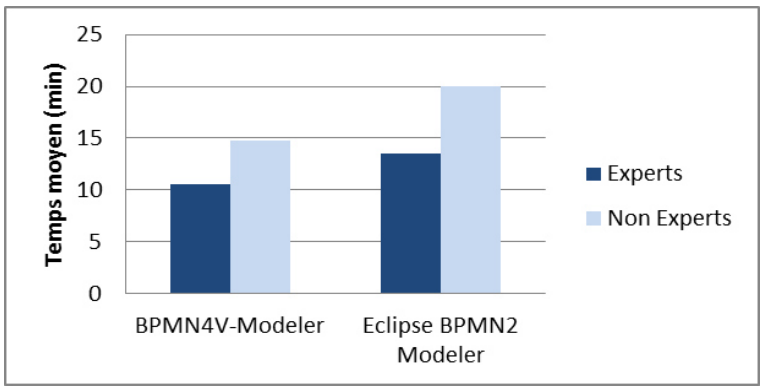


Figure V.33. Résultat de la comparaison entre le plug-in Eclipse BPMN2 Modeler et BPMN4V-Modeler

VI. Conclusion

Le plug-in BPMN4V-Modeler, présenté dans ce chapitre, est un outil dédié à la modélisation des processus intra- et inter-organisationnels flexibles. A travers ce plug-in, nous avons pu montrer la faisabilité de nos contributions que nous avons décrites dans les chapitres III et IV. BPMN4V-Modeler offre les fonctionnalités suivantes :

- Modélisation des versions des processus. Ce plug-in implémente les extensions apportées à BPMN permettant de concevoir des versions de processus privées conformément au méta-modèle BPMN4V-PP.
- Modélisation des versions de collaborations. L'outil permet la modélisation des versions de collaboration qui sont des instances du méta-modèle BPMN4V-CC.
- Manipulation des versions modélisées en offrant la possibilité de créer, modifier, valider et dériver des versions. Cette fonctionnalité est assurée à travers la définition de menus contextuels et d'interfaces graphiques facilitant la gestion des versions.
- Modification des versions de collaborations en utilisant les patrons d'adaptation. Cette fonctionnalité est assurée à travers : (i) l'extension de la palette du plug-in BPMN2 Modeler pour offrir une nouvelle catégorie d'éléments assurant ces patrons, et (ii) la définition des Wizards qui assistent le concepteur lors de l'utilisation d'un patron.
- Déduction automatique des versions de chorégraphies à partir des versions de collaborations.

Suite à une évaluation de nos contributions faite par des membres qui ont suivi une formation sur la flexibilité, nous avons pu s'assurer que les solutions que nous avons proposées sont simples et compréhensibles. De plus, cette évaluation a pu montrer que le plug-in BPMN4V-Modeler est un outil commode et capable de bien concrétiser nos contributions.

Conclusion Générale

De nos jours, la bonne gestion des processus métier représente un facteur de réussite des organisations. En effet, la modélisation et l'automatisation des processus participent grandement à l'amélioration de la performance des entreprises. Cependant, ces processus ont besoin d'être souvent modifiés de façon à s'adapter à l'environnement fortement fluctuant dans lequel les entreprises sont plongées. Les processus se doivent donc d'être flexibles.

I. Bilan général

Cette thèse a par conséquent abordé la problématique des processus flexibles. Nous avons adopté la définition des processus flexibles de [Chaâbane, 2012] et la taxonomie des besoins de flexibilité de [Reichert et Weber, 2012]. En outre, comme la majorité des systèmes de gestion de processus adopte le paradigme guidé par les activités pour modéliser les processus qu'ils exécutent, nous nous intéressons plus particulièrement au problème de la flexibilité des processus dont le schéma est décrit selon ce paradigme. Enfin, nous avons utilisé la notion de version pour répondre au problème de la flexibilité des processus.

Nous avons opté pour l'utilisation de la technique de versionnement puisque tout d'abord les versions constituant une réponse à plusieurs besoins de flexibilité définis dans la taxonomie de [Reichert et Weber, 2012]. Ensuite les contributions existantes autour des versions de processus sont incomplètes puisque la granularité du versionnement concerne essentiellement la perspective comportementale des processus ; les autres perspectives, dédiées à la description des ressources impliquées dans les processus et des données circulant dans ces processus, sont laissées de côté. De plus, la perspective contextuelle, qui permet de représenter les situations dans lesquelles les versions modélisées peuvent être utilisées, n'est pas prise dans ces contributions. Enfin, ces contributions ne se basent pas sur une notation standard pour la modélisation des processus.

L'objectif de cette thèse est donc la proposition d'une solution pour modéliser les processus intra- et inter-organisationnels flexibles. L'approche que nous avons adoptée pour atteindre cet objectif repose sur les éléments suivants :

- La notation BPMN 2.0 pour la modélisation des processus. Ce choix est justifié du fait que BPMN est considéré comme un standard permettant la modélisation graphique non seulement des intra-organisationnels (à travers le diagramme de processus privé) mais aussi des processus inter-organisationnels (à travers les diagrammes de collaboration et de chorégraphie).
- La technique de versionnement pour traiter le problème de la flexibilité des processus.

I.1. Résumé des contributions

Les contributions de cette thèse concernent la modélisation des versions des processus intra- et inter-organisationnels en utilisant les diagrammes de BPMN 2.0

pour traiter la flexibilité de ces processus. Plus précisément, la thèse apporte les contributions suivantes : BPMN4V-PP pour la modélisation et la manipulation des versions des processus privés, BPMN4V-CC pour la modélisation et la manipulation des versions de collaborations, six patrons d'adaptation pour la modification des versions de collaboration, une démarche de déduction des versions de chorégraphies et l'outil BPMN4V-Modeler implémentant BPMN4V-PP et BPMN4V-CC .

I.1.1. BPMN4V-PP pour la modélisation et la manipulation des versions de processus privés

Cette contribution consiste à étendre BPMN pour qu'il puisse supporter la modélisation des versions de processus privés. Il s'agit d'apporter des extensions au méta-modèle de BPMN 2.0 en utilisant la technique de versionnement. Le méta-modèle obtenu, nommé BPMN4V-PP (BPMN for Versions of Private Processes) comporte sept classes versionnables permettant de capturer les changements relatifs à un processus, aux activités (atomiques ou composites) et aux événements qui le composent, aux ressources invoquées pour l'accomplissement de ces activités et aux informations manipulées. En outre, il est nécessaire de pouvoir manipuler les versions de processus modélisées conformément à BPMN4V-PP. Aussi, nous avons proposé un diagramme d'états-transitions décrivant les états d'une version et les transitions possibles (définies sous forme d'opérations) permettant de passer d'un état à un autre. Ces opérations correspondent aux opérations que l'on peut appliquer sur les versions définies comme instances du méta-modèle BPMN4V-PP : elles correspondent aux opérations de création, modification, suppression, validation et dérivation de versions.

L'avantage de BPMN4V-PP est qu'il permet de modéliser les processus privés flexibles sous forme de versions en considérant les principales perspectives de modélisation de processus [Aalst *et al.*, 2003(a)]. En effet, dans ce méta-modèle la granularité du versionnement concerne des concepts issus des différentes perspectives : il ne se limite pas à versionner des processus ou des sous processus, mais considère le versionnement d'éléments constitutifs de ces perspectives, telles que les activités, les ressources impliquées dans la réalisation de ces activités et les informations qu'elles manipulent. Cependant, cette solution ne prend pas en considération la perspective contextuelle [Nurcan et Edme, 2005] qui permet de décrire le contexte dans lequel est utilisée chaque version de processus.

Le méta-modèle BPMN4V-Context vient remédier à cette limite. En effet, nous avons apporté une deuxième extension à BPMN 2.0 permettant de décrire le contexte des versions de processus modélisées conformément à BPMN4V-PP. Cette notion de contexte est importante pour aider le concepteur (i) à définir pourquoi une version de processus a été modélisée et (ii) à sélectionner la version qui convient à une situation particulière.

I.1.2. BPMN4V-CC pour la modélisation et la manipulation des versions de collaborations

Cette contribution consiste à étendre BPMN pour qu'il puisse supporter les versions de collaborations. Ainsi, nous avons proposé le méta-modèle BPMN4V-CC (BPMN for Versions of Collaborations and Choreographies) intégrant des classes versionnables permettant de suivre les changements d'une collaboration, des processus qui la composent et des messages échangés lors de l'interaction entre ces processus.

De même que pour les processus privés, nous avons défini les opérations permettant la manipulation des versions de collaborations. Nous avons ainsi défini des opérations supportant la création, la modification, la suppression, la validation et la dérivation de versions de collaborations modélisées conformément à BPMN4V-CC.

I.1.3. Six patrons d'adaptation pour la modification des versions de collaboration

Afin de remédier à la complexité des opérations de création et de modification des versions de collaborations, nous avons proposé six patrons d'adaptation correspondant à des primitives de haut niveau. Un patron d'adaptation est le regroupement d'une série d'actions élémentaires (dites aussi primitives de bas niveau) permettant de faire passer une version de collaboration d'un schéma à un autre. Ainsi les patrons définis permettent :

- d'ajouter une nouvelle interaction au schéma d'une version de collaboration (le patron « ajouter interaction »),
- de supprimer, remplacer, déplacer ou modifier une interaction existante (les partons « supprimer interaction », « remplacer interaction », « déplacer interaction » et « modifier interaction »), et
- d'échanger deux interactions existantes dans une version de collaboration (le patron « échanger interaction »).

I.1.4. Démarche de déduction des versions de chorégraphies

Cette contribution consiste à proposer une démarche permettant la génération automatique d'une version de chorégraphie à partir d'une version de collaboration. La démarche proposée est formée de quatre étapes basées sur la construction d'arbres assurant la transformation d'une version de collaboration en une version de chorégraphie. Nous avons explicité l'algorithme implémentant chacune de ces étapes.

Cette déduction est envisageable, car en BPMN, la notion de collaboration englobe la notion de chorégraphie. En effet une chorégraphie n'est qu'une autre représentation d'une collaboration dans laquelle on se focalise sur l'orchestration des messages transmis entre les différents partenaires. La limite de cette contribution est qu'il faut toujours passer par la version de collaboration pour modéliser la version de chorégraphie correspondante.

I.1.5. L'outil BPMN4V-Modeler

Afin de mettre en œuvre nos contributions, nous avons développé l'outil BPMN4V-Modeler en se basant sur le plug-in Eclipse BPMN2 Modeler. Cet outil offre des vues Eclipse, des menus contextuels et des Wizards permettant (i) la modélisation et la manipulation des versions de processus privés conformément à BPMN4V-PP, (ii) la modélisation et la manipulation des versions de collaborations conformément à BPMN4V-CC, et (iii) la déduction automatique des versions de chorégraphies.

I.2. Comparaison des contributions

Le tableau suivant reprend les critères d'évaluation évoqués dans le chapitre II pour comparer les travaux de l'état de l'art. Il permet de mesurer le degré de satisfaction de nos contributions par rapport à ces critères.

Critères		**Nos contributions**
Besoins de flexibilité	Variabilité	(+) est assurée par la définition des versions alternatives dont chacune convient à un contexte bien déterminé
	Adaptation	(+/-) est assurée par la définition de versions alternatives prenant en compte les changements prévisibles pouvant être anticipées au moment de la modélisation
	Incomplétude	(-) n'est pas prise en compte
	Evolution	(+) est assurée par la définition des versions dérivées par évolution pour tracer l'historique d'évolution des processus
Perspectives	Comportementale	(+) considérée dans le méta-modèle BPMN4V-PP
	Informationnelle	(+) considérée dans les méta-modèles BPMN4V-PP et BPMN4V-CC
	Organisationnelle	(+) considérée dans le méta-modèle BPMN4V-PP
	Contextuelle	(+) considérée dans le méta-modèle BPMN4V-Context pour tenir compte du contexte des processus privés (-) non considérée pour les processus inter-organisationnels
Type de Processus	Processus intra-organisationnels	(+) modélisés avec le diagramme de processus privé en BPMN 2.0
	Processus inter-organisationnels	(+) modélisés avec les diagrammes de collaborations et de chorégraphies de BPMN 2.0
Traçabilité de l'évolution		(+) est assurée par l'utilisation de la notion de version
Utilisation d'un standard		(+) toutes les contributions proposées sont basées sur le standard BPMN 2.0

D'après ce tableau, nous pouvons constater que notre travail présente des originalités qui le distinguent des travaux existants :

- La première originalité est que ce travail défend l'idée que les versions sont une technique qui ne se limite pas à faciliter l'évolution des processus, mais peut prendre en compte d'autres besoins de flexibilité. Ainsi, comme illustré dans le chapitre II, et si l'on considère la taxonomie définie dans [Reichert et Weber, 2012], nous avons montré que les versions sont utiles pour traiter les besoins de variabilité, d'évolution et d'adaptation des processus.

- La deuxième originalité repose sur l'approche multi-perspective que nous adoptons pour la modélisation des versions de processus. D'une part, nous étendons les travaux de l'état de l'art en considérant, en plus de la perspective comportementale considérée dans les propositions de l'état de l'art ([Weber *et al.,* 2013], [Ekanayake *et al.,* 2011] et [Zhao et Liu, 2013]), les perspectives informationnelle et organisationnelle sur lesquelles la notion de version porte également. D'autre part, nous introduisons la perspective contextuelle qui permet de définir le contexte d'utilisation des versions de processus, et donc d'aider les concepteurs dans leur choix des versions de processus à utiliser. Bien que le travail de [Chaâbane, 2012] prenne en considération ces quatre perspectives pour traiter la flexibilité des processus, ce travail ne se base pas sur une notation standard dans les solutions qu'il propose et ne considère que les processus intra-organisationnels.

- La troisième originalité réside dans la prise en compte non seulement des processus intra-organisationnels flexibles, mais aussi des processus inter-organisationnels. En effet, les solutions de l'état de l'art basées sur les versions ont traité uniquement la flexibilité des processus intra-organisationnels alors que dans le présent travail nous avons abordé la flexibilité des processus intra- et inter-organisationnels.

II. Perspectives de recherche

Notre travail a des limites, et par conséquent des perspectives à court et à long termes sont envisageables. Ces perspectives sont évoquées ci-dessous.

II.1. Perspectives à court terme

Nos perspectives à court terme sont les suivantes :

- Proposer une démarche méthodologique qui permet l'instanciation et l'exploitation des versions de processus modélisées conformément aux méta-modèles proposés.

- La perspective contextuelle, telle qu'elle est proposée, est incomplète en l'état. En effet, BPMN4V-Context ne permet de définir que le contexte d'utilisation des processus intra-organisationnels. Nous n'avons pas traité la définition du contexte d'utilisation des processus inter-organisationnels. Etudier la notion de contexte dans un cadre inter-organisationnel, où différentes organisations autonomes et hétérogènes sont amenées à coopérer pour coordonner leurs processus, est très intéressant. Ce sera la première perspective que nous aborderons à l'issue de cette thèse.

- Il est nécessaire de définir un langage pour l'interrogation des versions. Ce langage est important pour retrouver une version (de processus, de collaboration, d'activités, etc.) afin de l'instancier ou de la réutiliser dans la définition d'une nouvelle version.

- Nous n'avons pas investigué l'aspect vérification des versions de processus modélisées. Cet aspect est particulièrement intéressant pour les versions de collaborations, notamment après utilisation des patrons d'adaptation. Cette étape de vérification permettrait de s'assurer que les schémas obtenus respectent les propriétés de compatibilité et de cohérence énoncées dans [Decker et Weske, 2007].

- Il serait aussi utile d'étendre BPMN pour pouvoir modéliser directement des versions de chorégraphies sans passer par les versions de collaboration. Il s'agit de proposer des extensions au méta-modèle de BPMN pour la modélisation des versions de chorégraphies et de définir les opérations qui permettent la manipulation de ces versions.

II.2. Perspectives à long terme

Notre travail peut être prolongé pour prendre en compte les aspects suivants :

- Nous avons dans ce travail examiné la modélisation des versions de processus. Les éléments constituant notre proposition ont été mis en œuvre dans l'outil BPMN4V-Modeler. Cependant il faut aussi trouver des solutions pour étendre les moteurs d'exécution afin qu'ils puissent exécuter les versions de processus issues de l'outil BPMN4V-Modeler. Ces moteurs doivent être capables d'exécuter des instances de versions de processus, de suspendre l'exécution des instances auxquelles on apporte des modifications, et de sauvegarder dans un fichier spécial, appelé log de changement, le détail des changements effectués en cours de l'exécution des versions.

- Exploiter le fichier log d'exécution (qui comporte les traces d'exécution des versions de processus) et le fichier log de changement (qui comporte le détail des changements effectués en cours d'exécution) afin de faire des analyses. En effet, il est possible d'extraire à partir de ces fichiers des connaissances permettant d'optimiser la modélisation des versions de processus.

- La technique de versionnement que nous utilisons pour aborder la flexibilité des processus permet la prise en compte des besoins de variabilité, d'évolution et d'adaptation de la taxonomie de Reichert [Reichert et Weber, 2012]. Néanmoins, il est intéressant de considérer aussi le besoin d'incomplétude de cette taxonomie. En effet, nous pensons qu'il est envisageable d'adapter les modèles de décision *Liaison tardive* et *Modélisation tardive* en utilisant la technique de versionnement pour supporter ce besoin.

Références

[Aalst *et al.*, 2003(a)] W.M.P. van der Aalst, M.H. Weske and G. Wirtz, "Advanced topics in workflow management: issues, requirements and solutions". In the International Journal of Integrated Design and Process Science 7 (3), 2003, pages 47-77.

[Aalst *et al.*, 2003(b)] W.M.P van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, "Workflow Patterns". In the International Journal on Distributed and Parallel Databases, 14(1), 2003, July 2005, pages 5-51.

[Aalst *et al.,* 2004] W.M.P. van der Aalst, L. Aldred, M. Dumas, A.H.M. ter Hofstede, "Design and Implementation of the YAWL system". In the International Conference on Advanced Information Systems Engineering, LNCS 3084, Riga, Latvia, June 2004, pages 142–159.

[Aalst *et al.,* 2005] W.M.P. van der Aalst, M. Weske, and D. Grünbauer. "Case Handling: A New Paradigm for Business Process", In the international journal of Support. Data and Knowledge Engineering, 53(2), 2005, pages 129–162.

[Aalst *et al.,* 2009] W. van der Aalst, M. Pesic, H. Schonenberg, "Declaration Workflow: Balancing between Flexibility and Support". In the International Journal on Computer Science – Research and Development, 23(2), 2009, pages 99–113.

[Aalst et Berens, 2001] W. van der Aalst, P. Berens, "Beyond Workflow Management: Product-Driven Case Handling". In the International Conference on Supporting Group Work, Boulder, Colorado, USA, October 2001, pages 42–51.

[Aalst et Kumar, 2000] W.M.P van der Aalst, A. Kumar, "XML Based Schema Definition for Support of Inter-organizational Workflow". In the Technical Report/University of Colorado and University of Eindhoven, 2000.

[Aalst, 1999] W. M. P. van der Aalst. "Process-Oriented Architectures for Electronic Commerce and Interorganizational Workflow". In Information Systems Journal, 1999, pages 639-671.

[Aalst, 2004] W.M.P. van der Aalst, "Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management". Lectures on Concurrency and Petri Nets, Advances in Petri-Net, LNCS 3098, 2004 pages 1–65.

[Adams *et al.,* 2006] M. Adams, A. ter Hofstede, D. Edmond, W.M.P. van der Aalst, "Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows". In the International Conference on Cooperative Information Systems, LNCS 4275, Montpellier, France, November 2006, pages 291–306.

[Adams *et al.,* 2007] M. Adams, A.H.M. ter Hofstede, W.M.P. van der Aalst, D. Edmond, "Dynamic, extensible and context-aware exception handling for workflows". In the International Conference on Cooperative Information Systems, Springer, New York, 2007, pages 95–112.

[Adams, 2013] M. Adams, "Usability Extension for the Worklet Service". YAWL Symposium, Sankt Augustin, Germany, 2013, pages 69-75.

[Andonoff *et al.,* 2009] E. Andonoff, W. Bouaziz, C. Hanachi, L. Bouzguenda, "An Agent-based Model for Autonomic Coordination of Inter-Organizational Business Processes". Informatica, 20(3), September 2009, pages 323–342.

[Baresi *et al.,* 1999] L. Baresi, F. Casati, S. Castano, MG. Fugini, I. Mirbel, B. Pernici, "WIDE workflow development methodology". In the International joint conference

on Work activities coordination and collaboration, San Francisco, California, USA, February 1999, pages 19-28.

[Ben said *et al.,* 2014(a)] I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz, "Extending BPMN 2.0 meta-models for Process Version Modelling". In the International Conference on Enterprise Information Systems, Portugal, April 2014, pages 384-393.

[Ben said *et al.,* 2014(b)] I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz, "Context-Aware Adaptive Process Information Systems: the Context-BPMN4V Meta-model". In the East-European Conference on Advances in Databases and Information Systems, Macedonia, September 2014, pages 366-382.

[Ben said *et al.,* 2015(a)] I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz. "BPMN4V - an Extension of BPMN for Modelling Adaptive Processes using Versions". In the International Conference on Enterprise Information Systems, Spain, April 2015, pages 258-267.

[Ben Said *et al.,* 2015(b)] I. Ben Said, M.A. Chaabane, E. Andonoff, R. Bouaziz, "Flexibility of collaborative processes using versions and adaptation patterns". In the International Conference on Research Challenges in Information Science, Greece, May 2015, pages 400-411

[Ben Said *et al.,* 2016] I. Ben Said, M.A. Chaabane, E. Andonoff, R. Bouaziz, "A Version-based Approach to Address Flexibility of BPMN Collaborations and Choreographies". In the International Conference on e-Business, July 2016, Lisbon Protugal pages 31-42.

[Bouaziz et Andonoff, 2013] W. Bouaziz, E. Andonoff, "Autonomic Protocol-based Coordination in Dynamic Inter-Organizational Workflow". In the International Conference on Research Challenges in Information Science, Paris, May 2013, pages 555–566.

[Bouaziz, 2010] W. Bouaziz, "La coordination à base de protocoles dans les systèmes multi-agents : application au workflow inter-organisationnel", Rapport de thèse, Université Toulouse 1, 2010.

[Boukadi, 2009] K. Boukadi, "Coopération interentreprises à la demande : Une approche flexible à base de services adaptables". Rapport de thèse, École Nationale Supérieure des Mines de Saint-Étienne, Novembre, 2009.

[Boukhbouze *et al.,* 2009] M. Boukhebouze, Y. Amghar, AN. Benharkat, Z. Maamar, "A Rule-Based Modeling for the Description of Flexible and Self-healing Business Processes". In the 13th East European Conference on Advances in Databases and Information Systems, Riga, Latvia, September 2009, pages 15-27.

[Boukhedouma *et al.,* 2013(a)] S. Boukhedouma, M. Oussalah, Z. Alimazighi, and D. Tamzalit, "Adaptation patterns for Service-based Inter-Organizational Workflows". In the International Conference on Research Challenges in Information Science, Paris, France, May 2013, pages 567–576.

[Boukhedouma *et al.,* 2013(b)] S. Boukhedouma, M. Oussalah, Z. Alimazighi, and D. Tamzalit, "Flexible Loosely Coupled Inter-Organizational Workflows using SOA". In the International Conference on Computer Systems and Applications, Ifrane, Morocco, May 2013, pages 1–8.

[Bouzguenda, 2006] L. Bouzguenda, " Coordination Multi-Agents pour le workflow Inter-Organisationel Lâche". Rapport de thèse, Université Toulouse 1, May 2006.

[Casati *et al.,* 1996] F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Workflow Evolution". ER 1996, pages 438-455.

[Casati *et al.,* 1998] F. Casati, S. Ceri, B. Pernici, G. Pozzi, "Workflow evolution". In the International Journal on Data & Knowledge Engineering, 24(3), 1998, pages 211–238.

[Chaâbane *et al.*, 2010] M.A. Chaâbane, E. Andonoff, R. Bouaziz, L. Bouzguenda, "Modélisation Multidimensionnelle des Versions de Processus". Revue Ingénierie des Systèmes d'Information, Numéro spécial Modélisation d'entreprise, 15(5), December 2010, pages 89-114.

[Chaâbane, 2012] M.A. Chaâbane, "De la modélisation à la spécification des processus flexibles : Une approche basée sur les versions". Rapport de thèse, Université Toulouse 1, Septembre, 2012.

[Chan et Franklin, 2003] S.W.K. Chan, J. Franklin, "Dynamic Context Generation for Natural Language Understanding: A Multifaceted Knowledge Approach", In the IEEE Transaction on Systems, Man Cybernetics Part A: Systems and humans, 33(1), 2003, pages 23-41.

[Chebbi *et al.,* 2006] I. Chebbi, S. Dustdar, S. Tata, "The view-based approach to dynamic inter-organizational workflow cooperation". In the International Journal on Data & Knowledge Engineering, 56(2), 2006, pages 139-173.

[Cohn et Hull, 2009] D. Cohn, R. Hull Business Artifacts, "A Data-centric Approach to Modeling Business Operations and Processes". In the IEEE Data Engineering Bulletin, 32(1), March 2009, pages 3-9.

[Conforti *et al.,* 2015] R. Conforti, M. Dumas, M. La Rosa, A. Maaradji, H. Nguyen, A. Ostovar, S. Raboczi; "Analysis of Business Process Variants in Apromore". In the International Conference on Business Process Management (Demo), Innsbruck, Austria, pages 16-20

[Dadam et Reichert, 2009] P. Dadam, M. Reichert, "The ADEPT project: a decade of research and development for robust and flexible process support". In the International Journal of Computer Science-Research and Development, 23(2), 2009, pages 81–97.

[Damaggio *et al.,* 2011], E. Damaggio, A. Deutsch, R. Hull, V. Vianu, "Automatic Verification of Data-Centric Business Processes". In the International Conference on Business Process Management BPM, Clermont-Ferrand, France, August September 2011, pages 3-16.

[Daoudi et Nurcan, 2007] F. Daouadi, S. Nurcan, "A Benchmarking Framework for Methods to Design Flexible Business Processes". In the International Journal on Business Process Management (Special Issue Development and Support of the Software Process Improvement and Practice), 12(1), 2007, January 2007, pages 51-63.

[Decker et Weske, 2007] G. Decker, and M. Weske, "Behavioral Consistency for B2B Process Integration". In the Internatinal Conference on Advanced Information Systems Engineering, Norway, June 2007, pages81–95.

[Dey, 2000] A. K. Dey. "Providing Architectural Support for Building Context-Aware Applications". Rapport de thèse, Institue de technologie de Georgia, Décembre, 2000.

[Dey, 2001] A. K. Dey. “Understanding and using context”. Personal and ubiquitous computing, 5(1), 2001, pages 4-7.

[Divitini *et al.,* 2001] M. Divitini, C. Hanachi, C. Sibertin-Blanc., “Inter-Organizational Workflow for Enterprise Coordination”. Book, Coordination of Internet Agents, chapter 15, Tolksdorf Editors, Springer Verlag, 2001.

[Döhring et Zimmermann, 2011] M. Döhring, B. Zimmermann, “vBPMN: Event-Aware Workflow Variants by Weaving BPMN2 and Business Rules”. In the 12th International Conference on Enterprise, Business-Process and Information Systems Modeling, London, UK, June 2011, pages 332-341.

[Dumas *et al.,* 2013] M. Dumas, M. La Rosa, J. Mendling, H. Reijers, “Fundamentals of Business Process Management”. Book, Springer, 2013.

[Ekanayake *et al.,* 2011] C.C. Ekanayake, M. La Rosa, A. H. M. ter Hofstede, M.C. Fauvet, “Fragment-Based Version Management for Repositories of Business Process Models”. In the International Conference on Cooperative Information Systems, Hersonissos, Crete, Greece, Octobre 2011, pages 20-37

[Ellis *et al.,* 1995] C.A. Ellis, K. Keddara, G. Rozenberg, “Dynamic change within workflow systems”. In the International Conference on Organizational Computing Systems, Milpitas, 1995, pages 10–21.

[Estanol *et al.,* 2012] M. Estanol, A. Queralt, MR. Sancho, E. Teniente, “Artifact-Centric Business Process Models in UML”. In the International Conference on Business Process Management (Workshop), 2012, pages 292-303.

[Fdhila *et al.,* 2012] W. Fdhila, S. Rinderle-Ma, and M. Reichert, “Change Propagation in Collaborative Processes Scenarios”. In the International Conference on Collaborative Computing: Networking, Applications and Worksharing. Pittsburg, Pennsylvania, USA, October 2012, pages 452–461.

[Fdhila *et al.,* 2015] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert, “Dealing with Change in Process Choreographies: Design and Implementation of Propagation Algorithms”. In the International Journal on Information Systems, 49, 2015, pages 1–24.

[Fellmann *et al.,* 2011] M. Fellmann, F. Hogrebe, O. Thomas, M. Nüttgens, “Checking the Semantic Correctness of Process Models - An Ontology-driven Approach Using Domain Knowledge and Rules”. In the International Journal on Enterprise Modelling and Information Systems Architectures, 6(3), 2011, pages 22-35.

[Gottschalk *et al.,* 2008] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, M. La Rosa, “Configurable workflow models”. In the International Journal of Cooperative Information Systems, 17(2), 2008, pages 177–221

[Grenier *et al.,* 2004] U. Greiner, J. Ramsch, B. Heller, M. Löffler, R. Müller, E. Rahm, “Adaptive guideline-based treatment workflows with AdaptFlow”. In the Symposium on Computerized Guidelines and Protocols, 2004, pages 113-117.

[Hallerbach *et al.,* 2010(a)] A. Hallerbach, T. Bauer, M. Reichert, “Capturing variability in business process models: The Provop approach”. In the International Journal of Software Maintenance and Evolution: Research and Practice, 22(6/7), 2010, pages 519–546.

[Hallerbach *et al.,* 2010(b)] A. Hallerbach, T. Bauer, M. Reichert, "Configuration and Management of Process Variants", In the International Handbooks on Information Systems, Springer, New York, 2010, pages 237 -255.

[Hull *et al.,* 1999] R. Hull, J. Su, "The Vortex Approach to Integration and Coordination of Workflows". In the International Join Conference on Work Activities Coordination and Collaboration, San Francisco, 1999.

[Kanzow, 2004] S. Kanzow, "Approche pour l'ordonnancement distribué de workflows dans le contexte d'entreprises virtuelles", Rapport de thèse, Université Paris XII - Val de Marne, 2004.

[Keidl et Kemper, 2004] M. Keidl, A. Kemper, "A framework for context-aware adaptable Web services". In the International Conference on Extending Database Technology (EDBT'04), Heraklion, Crete- Creece, March 2004, pages 826-829.

[Klingemann, 2000] J. Klingemann, "Controlled flexibility in workflow management". In the International Conference on Advanced Information Systems Engineering, New York, 2000, pages 126–141.

[Korherr *et al.,* 2007] B. Korherr, B. List, "Extending the EPC and the BPMN with Business Process Goals and Performance Measures". In the International Conference on Enterprise Information Systems, Portugal, June 2007, pages 287-294.

[Kradofler et Geppert, 1999] M. Kradofler, A. Geppert, "Dynamic Workflow Schema Evolution based on Workflow Type Versioning and Workflow Migration". In the International Conference on Cooperative Information Systems, Edinburgh, Scotland, 1999, pages 104–114.

[Kumar *et al.,* 2006] K. Kumar, MM. Narasipuram, "Defining Requirements for Business Process Flexibility". In the International Conference on Business Process Modelling, Development, and Support, Luxemburg, June 2006, pages 137-148.

[La Rosa *et al.,* 2009] M. La Rosa, M. Dumas, A.H.M., ter Hofstede, "Modelling business process variability for design-time configuration". In the Handbook of Research on Business Process Modeling Idea Group Inc, 2009, pages 204–228.

[Lanz *et al.,* 2010] A. Lanz, M. Reichert, P. Dadam, "Robust and flexible error handling in the AristaFlow BPM Suite". In the International Conference on Advanced Information Systems Engineering (Forum), Tunisia, June 2010, pages 174–189.

[Lerner et al., 2010] B. Lerner, S. Christov, L.J. Osterweil, R. Bendraou, U. Kannengiesser, A.E. Wise, "Exception handling patterns for process modeling". In the IEEE Transactions on Software Engineering. 36(2), 2010, pages 162–183.

[Lezoche *et al.,* 2008] M. Lezoche, M. Missikof, L. Tininii, "Business Process Evolution: a Rule-based Approach". In the International Conference on Advanced Information Systems, Montpellier, France, June 2008, pages 407–414.

[Li, 2009] C. Li, "Mining process model variants: Challenges, techniques, examples". Rapport de thèse, University of Twente, Netherlands, 2009.

[Ly *et al.,* 2011] L.T. Ly, S. Rinderle-Ma, D. Knuplesch, P. Dadam, "Monitoring business process compliance using compliance rule graphs". In the International Conference on Cooperative Information Systems. Crete, Greece, 2011, pages 82–99.

[Makni *et al.,* 2012] L. Makni, N. Haddar, H. Ben-Abdallah, "Detection of Semantic Relations between Business Process Activity Labels". In the International

Conference on Enterprise Information Systems, Wroclaw, Poland, June 2012, pages 273-277.

[Milner *et al.,* 1997] R. Milner, M. Tofte, D. Macqueen, "The Definition of Standard ML". In the MIT Press, Cambridge, MA, USA, 1997.

[Müller *et al.,* 2004] R. Müller, U. Greiner, E. Rahm, "AGENTWORK: A Workflow-System Supporting Rule-Based Workflow Adaptation". In the International Journal on Data and Knowledge Engineering, 51, Elsevier, 2004, pages 223–256.

[Müller *et al.,* 2007] D. Müller, M. Reichert, J. Herbst, F. Poppa, "Data-Driven Design of Engineering Processes with COREPROModeler". In the 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, Paris, France, June 2007, pages 376-378.

[Müller *et al.,* 2008(a)] D. Müller, M. Reichert, J. Herbst, D. Köntges, A. Neubert, "COREPROSim: A Tool for Modeling, Simulating and Adapting Data-Driven Process Structures". In the International Conference Business Process Management, Milan, Italy, September 2008, pages 394-397.

[Müller *et al.,* 2008(b)] D. Müller, M. Reichert, J. Herbst, "A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures". In the International Conference on Advanced Information Systems Engineering, Montpellier, France, June 2008, pages 48-63.

[Narasipuram *et al.,* 2008] MM. Narasipuram, G. Regev, K. Kumar, A. Wegmann, "Business process flexibility through the exploration of stimuli". In the International Journal of Business Process Integration and Management, 3(1), 2008, pages 36-46.

[Nurcan et Edme, 2005] S. Nurcan, MH. Edme, "Intention Driven Modelling for Flexible Workflow Applications". In the International Journal on Software Process: Improvement and Practice, 10(4), 2005, pages 363–377.

[Nurcan, 2008] S. Nurcan, "A Survey on the Flexibility Requirements related to Business Process and Modelling Artifacts". In the International Conference on System Sciences, Waikoloa, Big Island, Hawaii, USA, January 2008, pages 378–387.

[Nurcan, 2011] S. Nurcan, "IS Olympics: Information Systems in a Diverse World". In the International Conference on Advanced Information Systems Engineering (Forum), London, UK, June 2011.

[OMG, 2008] OMG, "Business Process Model and Notation (BPMN) Version 1.1", Document Number: formal/2008-01-17, OMG, January 2008.

[OMG, 2009] OMG, "Business Process Model and Notation (BPMN) Version 1.2", Document Number: Formal/ 2009-01-03, OMG, January 2009.

[OMG, 2011] OMG, "Business Process Model and Notation (BPMN) Version 2.0", Document Number: formal/2011-01-03, available at: http://www.omg.org/spec/BPMN/2.0, 2011.

[Park et Choi, 2004] C. Park, I. Choi, "Management of business process constraints using BPTrigger". In the International Journal on Computers in Industry, 55(1), 2004, pages 29-51.

[Pesic et Aalst, 2006] M. Pesic, W. van der Aalst, "A Declarative Approach for Flexible Business Processes". In the International Workshop on Dynamic Process

Management, held at International Conference on Business Process Management, Vienna, Austria, September 2006, pages 169–180.

[Pesic *et al.,* 2007(a)] M. Pesic, H. Schonenberg, W.M.P. van der Aalst, "DECLARE: Full Support for Loosely-Structured Processes". In the International Conference on Entreprise Distributed Object Computing Conference, Annapolis, Maryland, USA, 2007, pages 287-300.

[Pesic *et al.,* 2007(b)] M. Pesic, M. Schonenberg, N. Sidorova, W.M.P van der Aalst, "Constraint-based workow models: Change made easy". In the International Conference on Cooperative Information Systems, November 2007, pages 77-94.

[Pesic, 2008] M. Pesic, "Constraint-based workflow management systems: Shifting control to users". Rapport de thèse, Université technique d'Eindhoven, 2008.

[Polyvyanyy *et al.,* 2012] A. Polyvyanyy, L. Garcia-Banuelos, M. Dumas, "Structuring Acyclic Process Models". In the International Journal on Information Systems, 37(6), pages 518–538.

[Reichert *et al.,* 2005] M. Reichert, S. Rinderle, U. Kreher, P. Dadam, "Adaptive process management with ADEPT2". In the International Conference on Data Engineering, IEEE Computer Society, 2005, pages 1113–1114.

[Reichert *et al.,* 2015] M. Reichert, A. Hallerbach, T. Bauer, "Lifecycle Management of Business Process Variants". In the Handbook on Business Process Management (1), 2015, pages 251-278.

[Reichert et Dadam, 1998] M. Reichert, P. Dadam. "ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control". In the International Journal on Intelligent Information Systems, 10(2), 1998, pages 93-129.

[Reichert et Weber, 2012] M. Reichert, and B. Weber, "Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies". Book, Springer, 2012.

[Rinderle et Reichert, 2010] S. Rinderle-Ma, M. Reichert, "Advanced migration strategies for adaptive process management systems". In the international conference on Commerce and Enterprise Computing, New York, 2010, pages 56–63.

[Robert, 2002] "Le petit ROBERT", Dictionnaire de la langue Française, Edition 2002.

[Rolland *et al.,* 2010] C. Rolland, "Fitting System Functionality to Business Needs: Alignment Issues and Challenges". In the International Conference on Software Methodologies, Tools and Techniques, Yokohama City, Japan, September 2010, pages 137–147.

[Rosemann et Aalst, 2005] M. Rosemann, W.M.P. van der Aalst, "A configurable reference modelling language". In the International Journal on Information Systems, 32(1), 2005, pages 1–23.

[Rosemann *et al.,* 2008] M. Rosemann, R. Jan, F. Christian, "Contextualisation of business processes". International Journal of Business Process Integration and Management, 3(1), 2008, pages 47-60.

[Sadiq *et al.,* 2005] S. Sadiq, W. Sadiq, M. Orlowska, "A framework for constraint specification and validation". In the International Journal on Information Systems, 30(5), 349–378 (2005)

[Saidani *et al.,* 2015] O. Saidani, C. Rolland, S. Nurcan, "Towards a Generic Context Model for BPM". In the International Conference on System Sciences, Kauai, Hawaii, USA, January 2015, pages 4120-4129.

[Saidani et Nurcan, 2006(a)] O. Saidani, S. Nurcan, "FORBAC: A Flexible Organisation and Role-Based Access Control Model for Secure Information Systems". In the International Conference on Advances in Information Systems, Turkey, October 2006, pages 364-376.

[Saidani et Nurcan, 2006(b)] O. Saidani, S. Nurcan, "A Role-Based Approach for Modeling Flexible Business Processes". In the Workshop on Business Process Modeling, Development and Support, Luxemburg, June 2006, pages 111-120.

[Saidani et Nurcan, 2009] O. Saidani, S. Nurcan, "Context-Awareness for Adequate Business Process Modelling". In the IEEE International Conference on Research Challenges in Information Science (RCIS'09), Fès, Morocco, April 2009, pages 177-186.

[Saleemi *et al.,* 2011] M. Saleemi, N.D. Rodriguez, J. Liliusn I. Porres. "A framework for context-aware applications for smart spaces". In the International Conference on Smart Spaces and Next Generation Wired/Wireless Networking, Russia, August 2011, pages 14-25.

[Schmidt *et al.,* 1999] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, W. V.d. Velde, "Advanced Interaction in Context". In the International symposium on Handheld and Ubiquitous Computing, London, UK, September 1999, pages 89-101.

[Schonenberg *et al.,* 2008] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, W.M.P. van der Aalst, "Process Flexibility: A Survey of Contemporary Approaches". In the International Workshop on CIAO/EOMAS at International Conference on Advanced Information Systems, France, June 2008, pages 16–30.

[Sciore, 1994] E. Sciore, "Versioning and Configuration Management in Object-Oriented Databases". In the International Journal on Very Large Databases, 3(1), 1994, pages 77–106.

[Simonin *et al.,* 2013] J. Simonin, S. Nurcan, J. Barrios, "Evolution organisationnelle fondée sur la cohérence des relations entre acteurs avec les buts métier". In the 31e congrès Informatique des organisations et systèmes d'information et de décision, Paris, May 2013, pages 225–240.

[Theodorakis et Spyratos, 2002] M. Theodorakis, N. Spyratos, "Context in artificial intelligence and information modeling". In the 2nd Hellenic Conference on AI, SETN-2002, Proceedings Companion Volume, Thessaloniki, Greece, April 2002, pages 27-38.

[Weber *et al.,* 2008] B. Weber, M. Reichert and S. Rinderle-Ma, "Change Patterns and Change Support Features -Enhancing Flexibility in Process-Aware Information Systems". In the International Journal on Data and Knowledge Engineering, 66(3), September 2008, pages 438-466.

[Weber *et al.,* 2013] B. Weber, S. Rinderle, M. Reichert, "Change Patterns and Change Support Features in Process-Aware Information Systems". In the Seminal Contributions to Information Systems Engineering, 2013, pages 381-395.

[Weske, 2012] M. Weske, “Business Process Management Concepts, Languages, Architectures”. Book, Springer, 2012.

[WfMC, 1994] The Workflow Management Coalition, “The Workflow Reference Model”. Technical Report WFMC-TC-1003, Novembre, 1994.

[WfMC, 1999] The Workflow Mangement Coalition. Interoperability abstract specification, Technical Report WfMC-TC-1012, 1999.

[Zhao et Liu, 2007] X. Zhao, C. Liu, “Version Management in the Business Change Context”. In the International Conference Business Process Management, Australia, September 2007, pages 198–213.

[Zhao et Liu, 2013] X. Zhao, C. Liu, “Version management for business process schema evolution”. In the International Conference on Information Systems, 38(8), 2013, pages 1046-1069.

Publications de l'auteur

✓ I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz, "Extending BPMN 2.0 meta-models for Process Version Modelling". In the International Conference on Enterprise Information Systems, Portugal, April 2014, pages 384-393.

✓ I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz, "Context-Aware Adaptive Process Information Systems: the Context-BPMN4V Meta-model". In the East-European Conference on Advances in Databases and Information Systems, Macedonia, September 2014, pages 366-382.

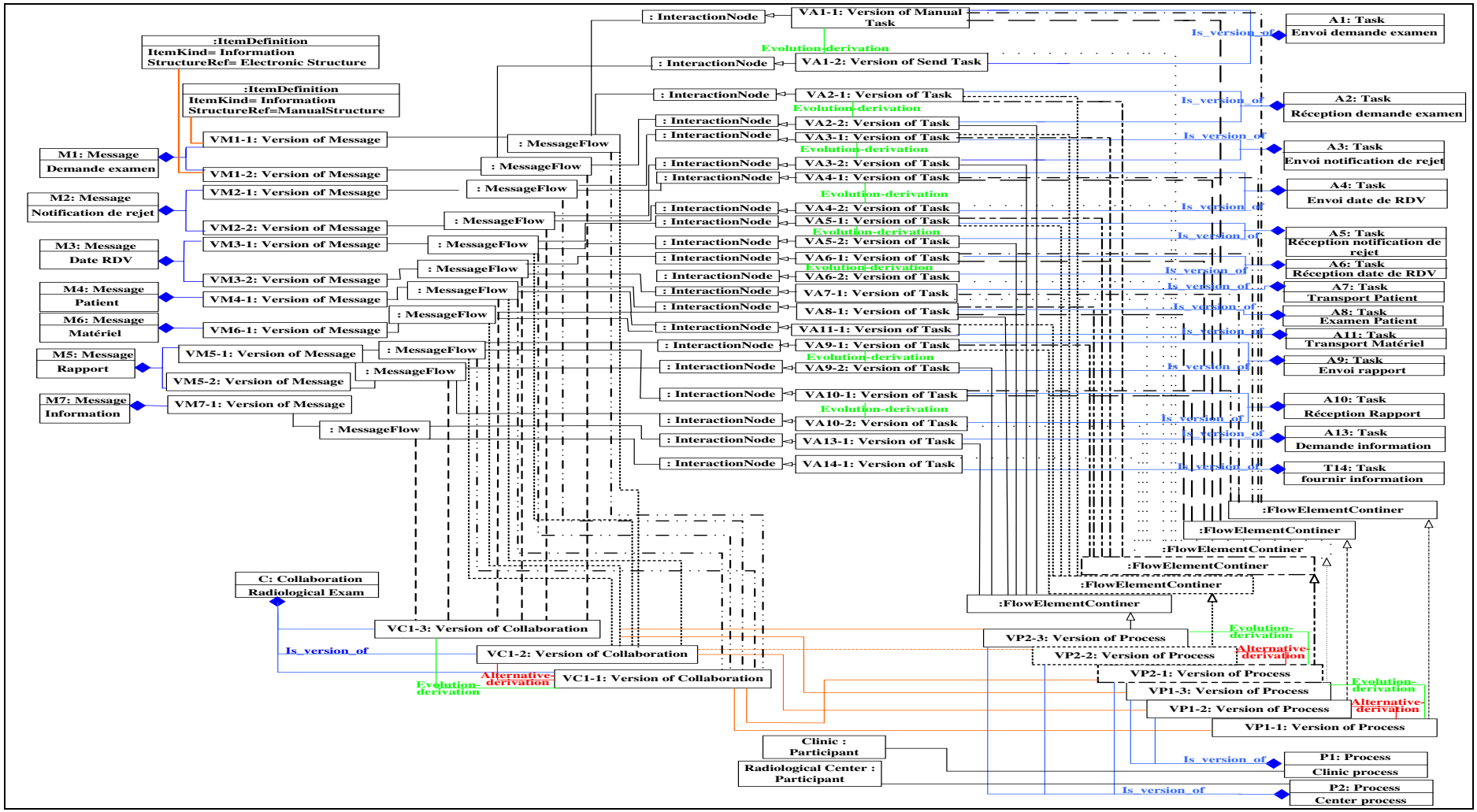
✓ I. Ben Said, MA. Chaabane, E. Andonoff, R. Bouaziz. "BPMN4V - an Extension of BPMN for Modelling Adaptive Processes using Versions". In the International Conference on Enterprise Information Systems, Spain, April 2015, pages 258-267.

✓ I. Ben Said, M.A. Chaabane, E. Andonoff, R. Bouaziz, "Flexibility of collaborative processes using versions and adaptation patterns". In the International Conference on Research Challenges in Information Science, Greece, May 2015, pages 400-411

✓ I. Ben Said, M.A. Chaabane, E. Andonoff, R. Bouaziz, "A Version-based Approach to Address Flexibility of BPMN Collaborations and Choreographies". In the International Conference on e-Business, July 2016, Lisbon Protugal pages 31-42.

Annexes

Annexe A1 :
Instanciation des trois versions de la collaboration *Examen Radiologique*



Annexe A2
Patrons d'adaptation

A2.1 Patron d'adaptation « ajouter une interaction »

1) Cas#2 : ajouter un message, ajouter un nœud d'envoi et utiliser un nœud de réception existant

Le cas#2 du patron « ajouter une interaction » permet d'ajouter un nouveau message ainsi que le nœud d'envoi de ce message. Ainsi, nous devons, tout d'abord, ajouter le nouveau nœud dans le processus correspondant. Puis nous devons définir un nouveau message et le *MessageFlow* qui le porte reliant le nouveau nœud d'envoi inséré et un nœud de réception existant.

En fonction du type de routage utilisé pour coordonner le nouveau nœud inséré dans le processus correspondant, nous distingons les trois options suivantes : insertion en séquence, insertion parallèle et insertion conditionnelle. Nous présentons dans ce qui suit une description de ces options ainsi que les algorithmes qui les implémentent.

✓ Description des options du cas#2

La Figure Annexe2.1, illustre un exemple d'utilisation du cas#2 du patron « ajouter une interaction » en détaillant les différentes options (séquence, parallèle ou conditionnelle) d'insertion du nouveau nœud d'envoi. Chacune de ces options permet : (i) la définition de la deuxième version du processus B (identifiée par VP2-2) dérivée à partir de sa première version (identifiée par VP2-1), (ii) l'insertion de la première version de la tâche d'envoi B3 (identifiée par VA6-1), (iii) la définition de la deuxième version de la tâche A2 (identifiée par VA2-2) dérivée à partir sa première version (VA2-1) et appartenant à la deuxième version du processus A (identifiée par VP1-2) suite à la modification de son type, (iv) la définition de la première version du message M (identifiée par VM1-1), et (v) la définition d'un *MessageFlow* portant la version de message VM1-1 et reliant les versions de tâches VA2-2 (deuxième version de A2) et VA6-1 (première version de B3). La première option correspond à une insertion séquentielle de la tâche B3 dans le processus B, tandis que la deuxième correspond à une insertion parallèle et la troisième option correspond à une insertion conditionnelle. Dans la Figure Annexe2.1, les modifications effectuées à l'aide du cas#3 du patron d'adaptation « ajouter une interaction » sont présentées en rouge.

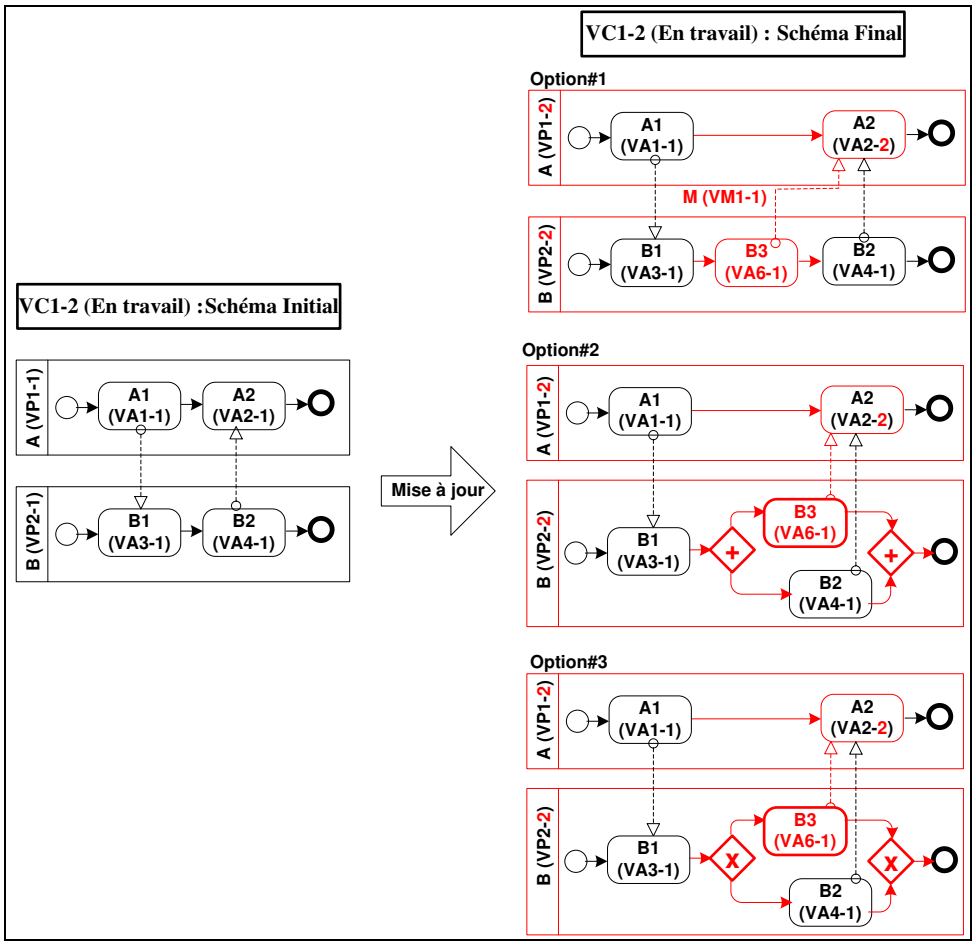


Figure Annexe2.1. Options du Cas#2 du patron « ajouter une interaction »

✓ Algorithme du cas#3

Pour mieux expliquer le cas#2 du patron « ajouter une interaction », nous proposons l'algorithme de la méthode « ajouterInteractionCas#2 » implémentant ce cas. Cet algorithme utilise les méthodes des classes du méta-modèle BPMN4V-CC décrites dans le tableau IV-3 (présenté dans le chapitreIV). Les paramètres de cette méthode sont les suivants : la version de message à insérer M, le nœud d'interaction E à insérer dans la version du processus Pe entre les deux *FlowNodes* Fnpe (le nœud qui précède E) et Fnse (le nœud qui suit E), s1 le type de routage utilisé lors de l'insertion de E, et R le nœud d'interaction appartenant à la version du processus Pr. Cet algorithme utilise, d'abord, la méthode ajouterNoeud détaillée précédemment pour insérer le nœud d'envoi « E » dans la version de processus Pe. Suite à cette insertion, la version de processus Pe doit être dérivée si elle est dans un état « *Stable* » ou si elle n'est pas déjà dérivée. Par la suite, cet algorithme procède à la dérivation de la tâche de réception R ainsi que la version de processus Pr si elles sont stables ou si elles ne sont pas déjà dérivées. Finalement, cet algorithme permet la définition d'un nouveau message dans la collaboration et l'insertion d'un nouveau *MessageFlow* qui relie les nœuds E et R.

```
Algorithme    ajouterInteractionCas#2    (M:version_of_message,    E:
InteractionNode,   Pe:version_of_process,   s1:String,   Fnpe:FlowNode,
Fnse:FlowNode, R:InteractionNode, Pr:version_of_process)
Variables
Ped,Prd:version of process
Tr :InteractionNode
Début

Si Pe.estStable() alors
    Ped = Pe.derive()
    Ped.ajouterNoeud(E,s1,Fnpe,Fnse)
Sinon
    Pe.ajouterNoeud (E,s1,Fnpe,Fnse)
Finsi

Si R.estStable() alors
    Tr=R.derive()
    Si Pr.estStable() alors
            Prd=Pr.derive()
    Finsi
Sinon
    Tr=R
Finsi

    this.ajouterMessage(M)
    this.ajouterMessageFlow(M,E,Tr)
Fin
```

2) Cas#3 : ajouter un message, ajouter un nœud de réception et utiliser un nœud d'envoi existant

Le cas#3 du patron « ajouter une interaction » permet d'ajouter un nouveau message à la collaboration ainsi que son nœud de réception. Ainsi, l'utilisation de ce cas permet : l'adjonction d'un nœud de réception selon trois options différentes (en séquence, en parallèle ou en conditionnel), l'adjonction d'un nouveau message et un nouveau *MessageFlow* qui le porte reliant un nœud d'envoi existant et le nouveau nœud de réception inséré. Dans ce qui suit, nous présentons une description des trois options d'insertion du nouveau nœud de réception ainsi que les algorithmes qui les implémentent.

✓ Description des options du cas#3

Un exemple d'utilisation du cas#3 du patron « ajouter une interaction » est présenté dans la Figure Annexe2.2. Dans cet exemple, nous distinguons trois options possibles pour insérer une nouvelle tâche de réception B3 dans le processus B. La première option correspond à une insertion séquentielle, tandis que la deuxième correspond à une insertion parallèle et la troisième option correspond à une insertion conditionnelle. Plus précisément, ce cas de patron permet de (i) définir la deuxième version du processus B (identifiée par VP2-2) dérivée à partir de sa première version (VP2-1), (ii) insérer la première version de la tâche B3 (identifiée par VA6-1) dans VP2-2, (iii) ajouter la première version du message M (identifiée par VM1-1), (iv) définir la deuxième version de la tâche A2 (identifiée par VA2-2) dérivée à partir de sa première version VA2-1 et appartenant à la deuxième version du processus A

(identifiée par VP1-2), et (v) définir le *MessageFlow* portant la version de message VM1-1 et reliant les versions de tâches VA2-2 (deuxième version de A2) et VA6-1 (première version de B3). Dans la Figure Annexe2.2, les modifications effectuées à l'aide du cas#3 du patron d'adaptation « ajouter une interaction » sont présentées en rouge.

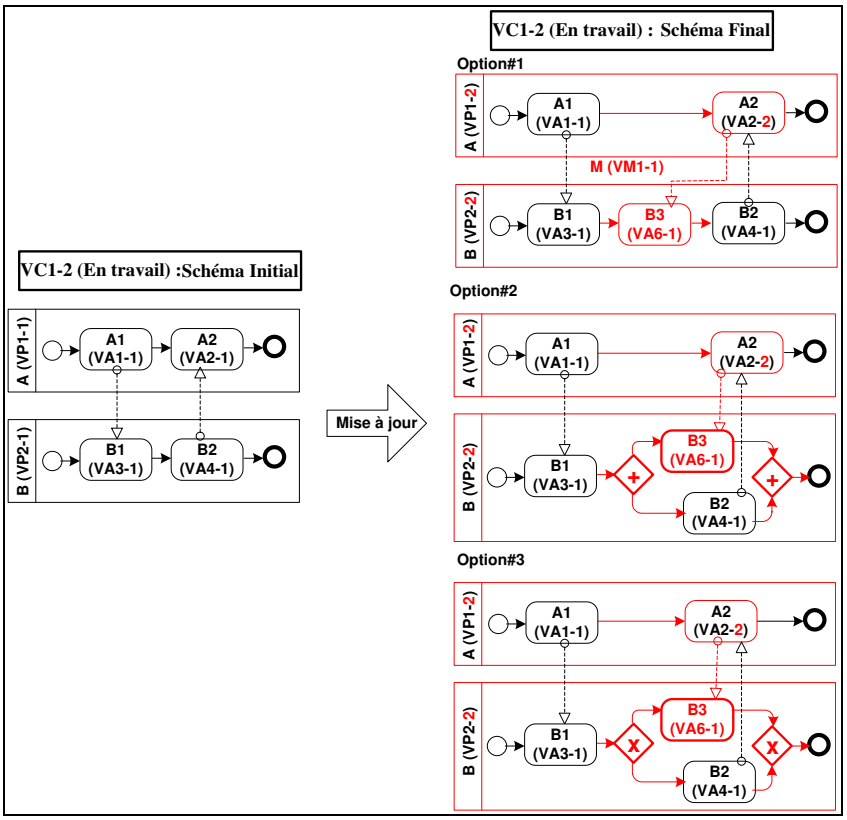


Figure Annexe2.2. Cas#3 du patron « ajouter une interaction »

✓ Algorithme du cas#3

L'algorithme de la méthode « ajouterInteractionCas#3 » comporte les paramètres suivants : la version de message à insérer M, le nœud d'interaction E appartenant à la version du processus Pe, le nœud d'interaction R à insérer dans la version du processus Pr entre les *FlowNode* Fnpr (le nœud qui précède R) et Fnsr (le nœud qui suit R), et s2 le type de routage utilisé lors de l'insertion de R. Cet algorithme permet (i) l'insertion d'un nœud R dans la version de processus Pr et la dérivation de cette version de processus (si elle est à l'état « *Stable* » ou si elle n'est pas déjà dérivée), (ii) la dérivation du nœud d'envoi E, (iii) la définition d'un nouveau message M et (iv) l'insertion d'un nouveau *MessageFlow* qui relie les nœuds E et R et qui porte le message M.

```
Algorithme ajouterInteractionCas#3 (M:version_of_message, E:
InteractionNode, Pe:version_of_process, R:InteractionNode, Pr:
version_of_process, s2: String, Fnpr: FlowNode, Fnsr: FlowNode)
 Variables
 Ped, Prd : version_of_process
 Te : InteractionNode
 Début
```

```
Si E.estStable() alors
    Te=E.derive()
    Si Pe.estStable() alors
            Ped=Pe.derive()
    Finsi
Sinon
    Te=E
Finsi

Si Pr.estStable() alors
    Prd = Pr.derive()
    Prd.ajouterNoeud (R,s2,Fnpr,Fnsr)
Sinon
    Pr.ajouterNoeud (R,s2,Fnpr,Fnsr)
Finsi
    this.ajouterMessage(M)
    this.ajouterMessageFlow(M,Te,R)
Fin
```

3) Cas#4 : ajouter un nouveau message entre des nœuds d'envoi et de réception existants

Le cas#4 du patron « ajouter une interaction » permet d'ajouter un message entre deux nœuds d'envoi et de réception existants dans la collaboration. Par conséquent, un nouveau message doit être ajouté, dans un premier temps, à la version de collaboration à modifier. Dans un second temps, un nouveau *MessageFlux* qui porte le nouveau message ajouté et qui relie les deux nœuds existants est à définir.

Dans ce qui suit, nous présentons une description de ce deuxième cas ainsi que l'algorithme qui l'implémente.

✓ Description du cas#4

La Figure Annexe2.3 montre un exemple d'utilisation du cas#4 du patron « ajouter une interaction ». Dans ce cas, nous ajoutons un nouveau message entre les tâches A2 et B2 de la collaboration C. Plus précisément, ce cas permet de (i) définir la deuxième version de la tâche A2 (identifiée par VA2-2) dérivée à partir de la version VA2-1 (première version de la tâche A2) ainsi que la deuxième version du processus A (identifiée par VP1-2), (ii) définir la deuxième version de la tâche B2 (identifiée par VA4-2) dérivée à partir de VA4-1 (première version de la tâche B2) ainsi que la deuxième version du processus B (identifiée par VP2-2), (iii) définir la première version du message M (identifiée par VM1-1) et (iv) ajouter un *MessageFlow* qui porte le message M et relie les versions de tâches VA2-2 (deuxième version de la tâche A2) et VA4-2 (deuxième version de la tâche B2) . La dérivation des tâches A2 et B2 est due du fait que ces deux tâches ont changé de type. En effet, avant l'utilisation du cas#4 du patron « ajouter une interaction », la première version de la tâche A2 (identifié par VA2-1) était une tâche de réception de message alors que la première version de la tâche B2 (identifiée par VA4-1) était une tâche d'envoi de message. L'adjonction du nouvel *MessageFlow* à la collaboration C entraine la définition de la deuxième version de la tâche A2 (VA2-2) et de la deuxième version de la tâche B2 (VA4-2) suite à la modification des types de ces deux versions. Ces deux tâches

deviennent des tâches abstraites (Task) permettant à la fois l'envoi et la réception de message. Dans la Figure Annexe2.3, les modifications effectuées à l'aide du cas#4 du patron d'adaptation « Ajouter une interaction » sont présentées en rouge.

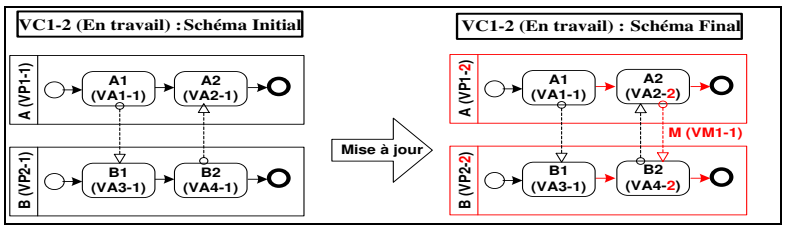


Figure Annexe2.3. Cas#4 du patron « ajouter une interaction »

✓ Algorithme du cas#4.

L'algorithme de la méthode « ajouterInteractionCas#4 » qui implémente le cas#4 du patron « ajouter une interaction » est ci-dessous donné. Plus précisément la méthode « ajouterInteractionCas#4 » utilise les paramètres suivants : M la version du message à ajouter, E le nœud d'interaction du processus Pe et R le nœud d'interaction du processus Pr. Cet algorithme permet (i) la dérivation des nœuds E et R s'ils sont stables, (ii) l'adjonction de la version de message M et (iii) l'insertion d'un nouveau *MessageFlow* entre le nœud d'envoi E et le nœud de réception R.

```
Algorithme ajouterInteractionCas#4 (M:version_of_message, E:
InteractionNode, Pe:version_of_process, R:InteractionNode, Pr:
version_of_process)
Variables
Ped, Prd : version of process
Te, Tr : InteractionNode
Début

    this.ajouterMessage(M)

Si E.estStable() alors
    Te=E.derive()
    Si Pe.estStable() alors
            Ped=Pe.derive()
    Finsi
    Sinon
    Te=E
Finsi

Si R.estStable() alors
    Tr=R.derive()
    Si Pr.estStable() alors
            Prd=Pe.derive()
    Finsi
Sinon
    Tr=R
Finsi

    this.ajouterMessageFlow(M,Te,Tr)
Fin
```

A2.2 Patron d'adaptation « supprimer une interaction »

1) Cas#2 : supprimer un message et redéfinir ses nœuds d'envoi et de réception

Le cas#2 du patron « supprimer une interaction » permet uniquement la suppression d'un message appartenant à la version de la collaboration à modifier. En effet, dans ce cas, les nœuds reliant le message à supprimer participent à d'autres *MessageFlow*. De ce fait, ces nœuds doivent être conservés dans la collaboration.

La Figure Annexe2.4 montre un exemple d'utilisation du cas#2 de ce patron. Ce cas permet (i) la suppression de la première version du message M (identifiée par VM1-1), (ii) la suppression du *MessageFlow* portant la version de message VM1-1 et reliant les tâches A2 et B2, (iii) la définition de la deuxième version de la tâche A2 (identifiée par VA2-2) et de la deuxième version de la tâche B2 (identifiée par VA5-2) dérivées respectivement à partir des premières versions de ces deux tâches VA2-1 et VA5-1 (iv) la dérivation de la première version du processus A (VP1-1) et de la première version du processus B (VP2-1). La dérivation des versions de tâches A2 et B2 est effectuée suite à la modification des types de ces deux tâches. En effet, la version de tâche VA2-2 est devenue une tâche d'envoi de message tandis que la version de tâche VA5-2 est une tâche de réception de message. Cette dérivation déclenche la dérivation des versions de processus VP1-1 et VP2-1 (*cf.* la propagation de la dérivation présentée en Figure IV.12 du chapitre IV).

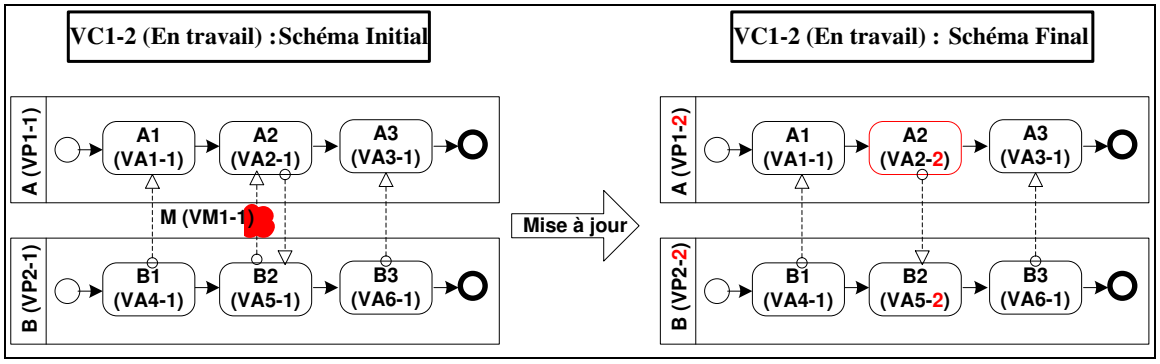


Figure Annexe2.4. Cas#2 du patron « supprimer une interaction »

2) Cas#3 et Cas#4 du patron « supprimer une interaction »

Le cas#3 et le cas#4 du patron « supprimer une interaction » permettent la suppression d'un message ainsi qu'un seul nœud relié à ce message (le nœud d'envoi ou le nœud de réception). Le cas#3 de ce patron est utilisé quand le nœud d'envoi du message à supprimer n'est pas impliqué dans d'autres *MessageFlow* alors que son nœud de réception participe dans d'autres *MessageFlow*. Par conséquent, l'utilisation de ce cas procède à la suppression du message, de son *MessageFlow* et de son nœud d'envoi. Le cas#4 se diffère du cas#3 au niveau du nœud à supprimer ; il s'agit de supprimer le nœud de réception puisque il n'est pas impliqué dans d'autres *MessageFlow*.

La Figure Annexe2.5 montre un exemple d'utilisation des cas#3 et cas#4 du patron « supprimer une interaction ». Dans cet exemple, nous illustrons la suppression du message M reliant les tâches A2 et B2 de la collaboration C. Plus précisément, le cas#3 permet : (i) la suppression du message M, (ii) la suppression du *MessageFlux* portant le message M et reliant les deux nœuds A2 et B2. (iii) la suppression de la première version de tâche B2 (identifiée par VA5-1), et par conséquent la définition de la deuxième version du processus B (VP2-2) dérivée à partir de sa première version (VP2-1), et (iv) la définition de la deuxième version de la tâche A2 (VA2-2) dérivée à partir de sa première version (VA2-1) ce qui donne suite à la modification de son type, d'où la définition de la deuxième version de processus A (VP1-2) dérivée à partir de sa première version VP1-1. Quant au cas#4, il permet : (i) la suppression du message M, (ii) la suppression du *MessageFlow* portant le message M et reliant les deux nœuds A2 et B2, (iii) la suppression de la première version de la tâche A2 (VA2-1), et par conséquent la définition de la deuxième version du processus A (VP1-2) dérivée à partir de sa première version VP1-1, et (iv) la dérivation de la première version de la tâche B2 (VA5-2) à partir de sa première version (VA5-1) suite à la modification de son type, ce qui entraine la dérivation de la première version de processus B (VP2-1).

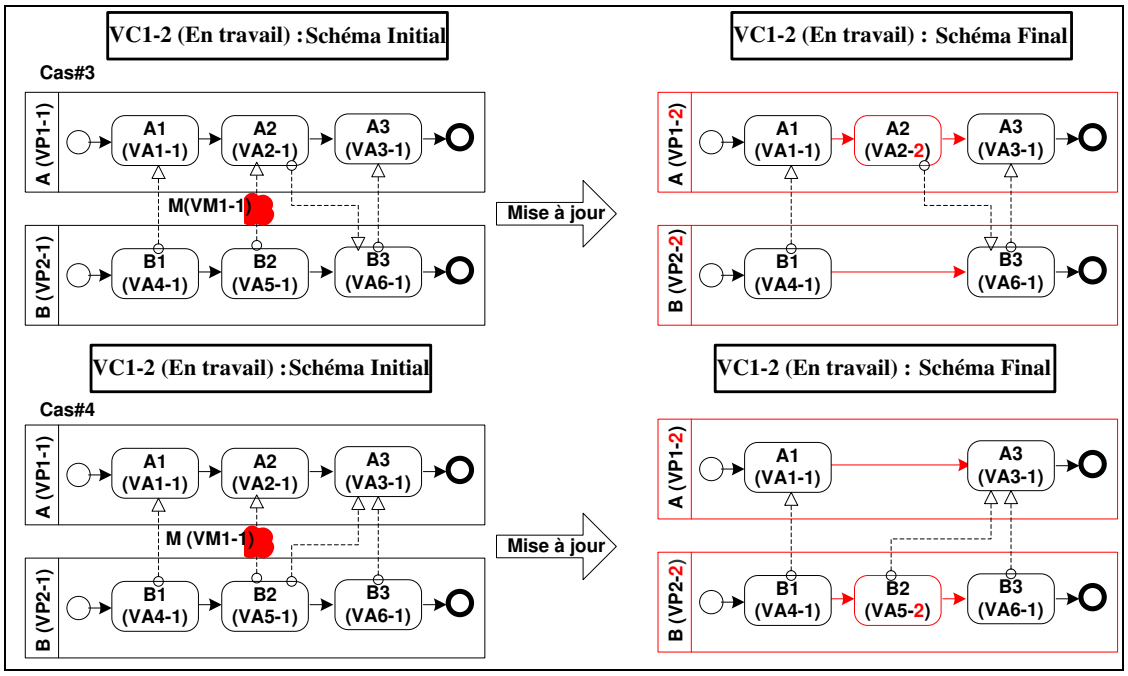


Figure Annexe2.5. Cas#3 et Cas#4 du patron « supprimer une interaction »

3) Algorithme du patron « supprimer une interaction »

Pour mieux expliquer les quatre cas du patron « supprimer une interaction », nous présentons dans ce qui suit deux algorithmes implémentant ces cas. Ces algorithmes font référence aux méthodes des classes du méta-modèle BPMN4V-CC décrites dans le tableau IV-3 du chapitre IV.

Le premier algorithme détaille la méthode « supprimerNoeud(T) » qui permet la suppression du nœud T d'une version de processus. L'algorithme de la méthode « supprimerNoeud » est présenté ci-dessous.

```
Algorithme supprimerNoeud(T: InteractionNode)
Variables
Tp,Ts : FlowNode
Début
    Tp= T.getPrecedent()
    Ts= T.getSuivant()
    this.supprimerSequenceFlow(Tp,T)
    this.supprimerSequenceFlow(T,Ts)
    this.enleverNoeud(T)
    this.ajouterSequenceFlow(Tp,Ts)
Fin
```

Le deuxième algorithme implémente la méthode « supprimerInteraction ». Les paramètres de cette méthode sont les suivants : la version du message à supprimer M, le nœud d'interaction E appartenant à la version du processus Pe et le nœud d'interaction R appartenant à la version du processus Pr. Cette méthode permet, alors, la suppression de l'interaction entre les nœuds E et R tout en considérant les quatre cas du patron « supprimer une interaction » décrits précédemment. Plus précisément, nous considérons, dans cet algorithme, les deux variables Ne et Nr qui contiennent le nombre de *MessageFlow* reliés respectivement au nœud d'envoi E et au nœud de réception R. Cet algorithme permet (i) la suppression du message M et du *MessageFlow* reliant les nœuds d'envoi et de réception de message E et R, (ii) la suppression du nœud d'envoi E si le nombre des *MessageFlow* reliés est égal 1 (*i.e.,* Ne=1) et (iii) la suppression du nœud de réception R si le nombre des *MessageFlow* reliés est égal 1 (*i.e.,* Nr=1). Dans le cas où les nœuds E et R ne sont pas supprimés (*i.e.,* Ne>1 et Nr>1) cet algorithme permet la dérivation de ces deux nœuds ainsi que leurs processus Pe et Pr (si elles sont à l'état « *Stable* » ou elles ne sont pas déjà dérivées).

```
Algorithme supprimerInteraction(M:version_of_message, E:
InteractionNode, Pe:version_of_process, R:InteractionNode; Pr:
version_of_process)
Variable
Ped,Prd:version_of_process
Te,Tr:InteractionNode
Ne,Nr:integer
Début
    Ne= E.calculerNbMsg()
    Nr=R.calculerNbMsg()
    this.supprimerMsgFlow(M,E,R)
    Selon NE
            Ne=1 :
            Si Pe.estStable() alors
                    Ped = Pe.derive ()
                    Ped.supprimerNoeud(E)
            Sinon
                    Pe.supprimerNoeud(E)
            Fin Si

            Ne>1 :
```

```
            Si E.estStable() alors
                    Te=E.derive()
                            Si Pe.estStable () alors
                              Ped = Pe.derive ()
                            Fin Si
            Finsi

    Fin Selon que

    Selon que Nr

            Nr=1 :
            Si Pr.estStable() alors
                    Prd = Pr.derive ()
                    Prd.supprimerNoeud (R)
            Sinon
                    Pr.supprimerNoeud (R)
            Fin Si

            Nr>1 :
            Si R.estStable() alors
                    Tr=R.derive()
                            Si Pr.estStable () alors
                               Prd = Pr.derive ()
                            Fin Si
            Finsi
    Fin Selon que
this.supprimerMessage(M)
Fin
```

A2.3 Patron d'adaptation « modifier une interaction »

Cas#4 : modifier le message

Le cas#4 du patron « modifier une interaction » permet la modification du message porté par une interaction d'une version de collaboration. Cette modification peut résulter des deux sous-cas suivants : le premier consiste à remplacer la version du message à modifier par une autre version de message, alors que le deuxième sous-cas permet la dérivation de cette version de message et la modification de son *ItemDefinition,* en particulier les propriétés *itemKind* et / ou *itemStructure*.

✓ Description des sous-cas du cas#4

La Figure Annexe2.6, présente un exemple d'utilisation du cas#4 du patron « modifier une interaction » tout en considérant les sous-cas énoncés précédemment. Plus précisément, dans cet exemple, il s'agit de modifier la première version du message M1 (identifiée par VM1-1) envoyé de la première version de la tâche A2 (identifiée par VA2-1) vers la première version de la tâche B2 (identifiée par VA4-1). Dans le premier sous-cas, la version de message VM1-1 est remplacée par la première version du message M2 (VM2-1) (qui n'existait pas auparavant dans la collaboration). Dans le deuxième sous-cas, cette version de message est dérivée suite au changement de son *ItemDefinition*. Dans ces deux cas, il est nécessaire de dériver les versions des tâches et les versions des processus impliquées dans cette version de message si ces versions sont à l'état « *Stable* » ou elles ne sont pas déjà dérivées.

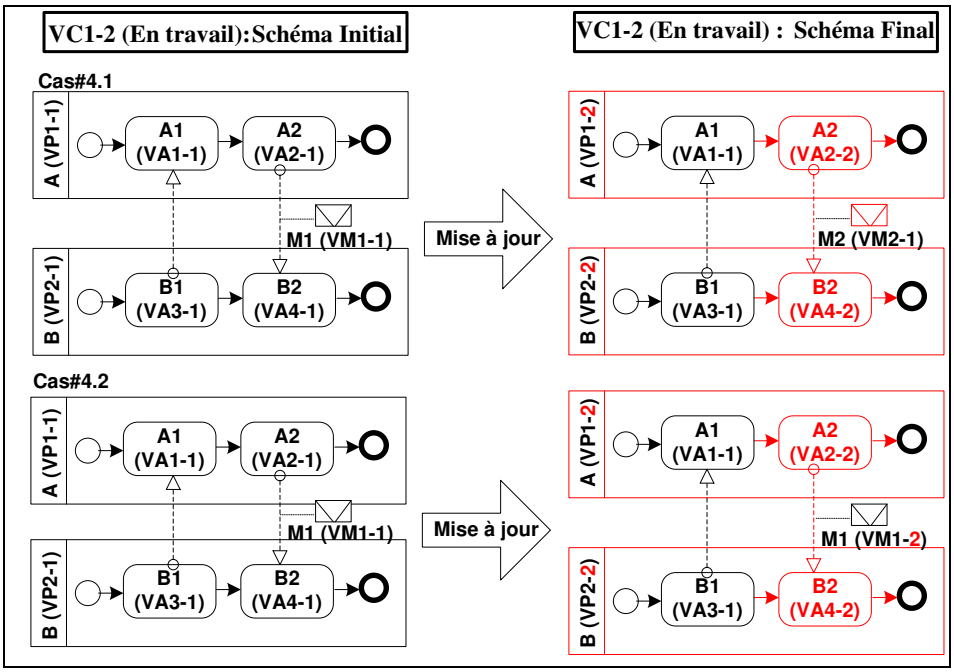


Figure Annexe2.6. Cas#4 du patron « modifier une inetraction »

✓ Algorithme du cas#4

L'algorithme de la méthode « modifierInteractionCas#4 » implémentant le cas#4 du patron « modifier une interaction » est donné ci-dessous. Cette méthode utilise les paramètres suivants : M1 la version de message à modifier, M2 la nouvelle version de message, id la nouvelle *itemDefinition* à définir, E le nœud d'interaction représentant la source du message à modifier appartenant à la version du processus Pe, et R le nœud d'interaction représentant la cible du message à modifier appartenant à la version du processus Pr. Plus précisément, cet algorithme permet de modifier la version du message M1 envoyée entre les nœuds d'interaction E et R. Cette version de message peut être modifiée soit en la remplaçant par la version de message M2, soit par la modification de la propriété *ItemDefinition*. De plus, cet algorithme permet la dérivation des versions des nœuds d'interaction et des processus impliquées dans la version de message à modifier si ces versions sont à l'état « *Stable* » ou si elle n'est pas déjà dérivée.

```
Algorithme        modifierInteractionCas#4        (M1:version_of_message,
   M2:version_of_message,      id:ItemDefinition,      E:InteractionNode,
   R:InteractionNode, Pe:Version_of_Process, Pr:Version_of_Process)
Variables
Md:version of message
Mf:MessageFlow
Début
//cas#4.1
Si M2 <> null alors
   Mf= This.getMessageFlow(M2,E, R)
   This.supprimerMessage(M1)
   Mf.setMessage(M2)
Sinon
//cas#4.2
       Si M1.estStable() alors
```

```
                Md=M.derive()
                Md.setItemDefinition(id)

        Finsi
Fin Si
Si E.estStable() alors
   E.Derive()
   Si Pe.estStable() alors
        Pe.Derive()
   Finsi
FinSi
Si R.estStable() alors
   R.Derive()
   Si Pr.estStable() alors
        Pr.Derive()
   Finsi
FinSi

Fin
```