N° d'ordre: 03/2018-D/INF

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Électronique et d'Informatique
Département d'Informatique

Thèse présentée pour l'obtention du grade de Docteur en Sciences

Option: Informatique

Spécialité: Génie logiciel

Thème

Approche de réutilisation des patrons de procédés logiciels

Par: Asma HACHEMI

Soutenue publiquement le 17 Octobre 2018, devant le jury composé de

Mme Ioualalen Boukala Malika	Professeur	à l'USTHB	Président
Mr Ahmed-Nacer Mohamed	Professeur	à l'USTHB	Directeur de Thèse
Mme Alimazighi Zaia	Professeur	à l'USTHB	Examinateur
Mr Ahmed-Ouamer Rachid	Professeur	à l'U. de Tizi Ouzou	Examinateur
Mr Balla Amar	Professeur	à l'ESI	Examinateur
Mr Sahnoun Zaidi	Professeur	à l'U. de Constantine	Examinateur

À mes parents;

Remerciements

Louange à Allah, Le Seigneur de l'univers.
Seigneur, nous ne savons que ce que Tu nous as enseigné; Toi le Savant, le Sage. Seigneur, enseigne-nous ce qui nous sera utile; fais en sorte que ce que Tu nous enseignes nous soit utile, et augmente notre savoir.

Cette thèse est l'aboutissement d'un long travail ne pouvant être possible sans le soutien de plusieurs personnes, que je souhaite remercier à travers ces quelques mots.

D'abord, je suis honorée que Madame *IOUALALEN BOUKALA Malika*, Professeur à l'USTHB, ait accepté de présider mon jury de soutenance. Je tiens à la remercier fortement.

Je tiens à remercier infiniment Madame *ALIMAZIGHI Zaia*, Professeur à l'USTHB, de m'avoir fait l'honneur d'être examinatrice de cette thèse. Mes vifs remerciements vont également à l'endroit de Monsieur *AHMED-OUAMER Rachid* Professeur à l'université Mouloud MAMERI de Tizi Ouzou, de Monsieur *BALLA Amar* Professeur à l'ESI, et de Monsieur *SAHNOUN Zaidi* Professeur à l'université Abdelhamid MEHRI de Constantine. Je les remercie très chaleureusement pour avoir bien voulu m'honorer en participant à mon jury de thèse.

Je tiens à exprimer ma vive reconnaissance à mon directeur de thèse Monsieur *AHMED-NACER Mohamed*, Professeur à l'USTHB, pour sa confiance, son attention, ainsi que pour ses compétences scientifiques qui m'ont permis de mener à bien ce travail. La réussite de cette thèse doit énormément à ses encouragements.

Merci à tous mes collègues et amis qui ont contribué à orienter mes travaux, qui m'ont soutenu et qui ont pris soin de moi.

Je tiens aussi à adresser ma gratitude et ma sympathie à tous ceux qui, de près ou de loin, ont contribué avec foi et conscience à mon enseignement, et à faire de moi ce que je suis. J'espère qu'ils retrouvent dans ces quelques mots l'expression de ma profonde reconnaissance et mon respect.

Enfin, mes remerciements ne pouvaient se conclure sans une pensée pour ma famille. C'est avec beaucoup de tendresse et d'affection que mon salut s'adresse à mes parents ainsi qu'à mon frère et à ma sœur, qui m'ont couvert avec leur soutien, amour et générosité. Je suis particulièrement très reconnaissante envers *Mama*, pour tous ses efforts et ses sacrifices irréfutables. Je resterai certaine que sans elle, je ne serais arrivée à ce point.

Meriama, grand cœur plein d'affection, merci encore.

Alger, le 05 Mai 2018
Asma HACHEMI

Table des matières

Liste des figures

Liste des tables

CHAPITRE

INTRODUCTION GENERALE

Ce chapitre introduit de manière concise le travail présenté dans cette thèse. Il commence par donner le contexte de notre travail (cf. § I). La réutilisation de patrons dans la modification de modèles de procédés est ensuite présentée (cf. § II). Après cela, un aperçu de nos principales contributions est donné (cf. § III). Enfin, le plan de la présente thèse est exposé (cf. § IV).

I. Contexte

Les produits logiciels deviennent de plus en plus riches et complexes et leur utilisation atteint de plus en plus de domaines; cette tendance est résumée succinctement dans [AND11]: "les logiciels dévorent le monde". Le double défi de l'ingénierie logicielle est celui de la qualité et de la productivité. Cela crée un contexte où les patrons logiciels émergent comme un moyen puissant, offrant des solutions prouvées et réutilisables pour relever ce double défis. La montée en échelle des patrons atteint cependant un point où le besoin d'outils pour trouver, comprendre et appliquer des patrons devient une exigence critique [HEN07].

Parmi ces patrons logiciels se trouvent les patrons de procédés. Le concept de patron de procédés a d'abord été présenté par Coplien lors de la conférence PLoP de 1994 [COP94] [KNE02] [TAS07] [BIG12], dont l'article est considéré comme "un premier document clé sur les patrons de procédés" [AMB16]. Ce concept a été introduit dans le sens de patron qui capte la connaissance de procédés. Plus tard, Ambler [AMB98] définit un patron de procédés comme suit: "un patron de procédés décrit une approche et/ou une série d'actions prouvée et réussie pour développer un logiciel".

Par ailleurs, il est connu en ingénierie logicielle que pour améliorer le produit développé, il est important de maîtriser son procédé de développement [OST87] [MON99] [EST05]. Cela conduit à l'apparition du concept de modèle de procédés de développement logiciel (que nous appellerons dans la suite de ce document *modèle de procédés*); un tel modèle est une représentation abstraite des activités du monde réel de développement logiciel, en utilisant un langage de modélisation de procédés. Dès lors, "les méthodes et les procédés ainsi que les outils les supportant prennent place au cœur de l'ingénierie logicielle" [DIT16], et il est admis aujourd'hui que "il faut améliorer les modèles de procédés pour améliorer la qualité des logiciels développés" [MAR16]. En effet, "même les petites entreprises s'intéressent de nos jours aux modèles de procédés pour améliorer la qualité de leurs produits" [GAR16]; et les patrons de procédés (relatifs aux modèles de procédés) deviennent par conséquent des éléments pertinents de l'ingénierie logicielle. Ainsi, nous nous intéressons dans le présent travail aux patrons de procédés, que nous appellerons dans la suite de ce document *patrons*.

D'autre part, "la réutilisation est un moyen prometteur pour améliorer la qualité et la productivité des modèles de procédés" [WAN10], en particulier la réutilisation de patrons qui sont eux-mêmes une source précieuse de connaissances prouvées. Une telle réutilisation peut prendre les formes suivantes: (a) la réutilisation de patrons pour créer un nouveau modèle de procédés (from scratch), ou un fragments de modèle à un modèle de procédés existant; ou (b) la réutilisation de patrons pour modifier un modèle de procédés existant suivant un patron.

Cette dernière forme de réutilisation de patrons se base sur un mécanisme de fusion, et est très utile lorsque les concepteurs de procédés doivent enrichir ou restructurer un modèle de procédés existants afin de le corriger, de satisfaire certaines contraintes ou d'améliorer son efficacité [TRA07b] conformément au patron réutilisé. De plus, cette forme de réutilisation est parfois utile même lors de la création de nouveaux modèles de procédés. En effet, "pour

obtenir un modèle de procédés spécifique à un projet, un modèle de procédés standard est d'abord créé, un patron sera ensuite choisi en fonction du contexte et du problème posé dans le projet, puis ce patron sera fusionné avec le modèle de procédés standard" [WAN10]. Nous appellerons plus tard cette forme de réutilisation de patrons, la réutilisation pour modifier des modèles de procédés existants, qu'il s'agisse de la modification d'un modèle de procédés dans sa totalité ou de la modification d'un fragment de celui-ci.

Cependant, sans des outils spécifiques traitant cette forme de réutilisation, les concepteurs de procédés doivent comprendre les connaissances représentées dans un patron, puis modifier manuellement le modèle de procédés en fonction de ce patron, ce qui constitue une manière de réutilisation inefficace et susceptible d'erreurs [HEE09]. Malheureusement, la réutilisation de patrons pour modifier des modèles de procédés existants demeure la forme de réutilisation de patrons la plus délaissée, en raison du manque d'outils automatiques qui la soutiennent, mais aussi parce qu'elle implique la fusion de modèles de procédés qui constitue elle même un problème à part entière.

II. Problématique

La réutilisation de patrons dans la modification de modèles de procédés implique en réalité deux modèles de procédés: le modèle de procédés à modifier et la solution proposée par le patron, qui est elle-même un modèle de procédés; le résultat de la réutilisation est un modèle de procédés obtenu par la fusion de ces deux modèles. Cependant, la fusion peut produire un modèle cohérent, comme elle peut produire un résultat incohérent dans le cas où des conflits (incohérences) apparaissent entre les deux modèles fusionnés. Malheureusement, il y a très peu de travaux de recherche qui abordent la résolution des conflits lors de la fusion des modèles [DAM16], et donc très peu de travaux qui traitent la réutilisation de patrons pour modifier des modèles de procédés.

L'objectif de ce travail de thèse est de proposer une approche de réutilisation, muni d'un opérateur de réutilisation capable de gérer les conflits potentiels qui résultent de la réutilisation de patrons pour modifier les modèles de procédés, en particulier lors de la procédure de fusion. L'opérateur recevra comme entrées le modèle de procédés à modifier et le patron à réutiliser; il procèdera à leur fusion avec la vérification et la résolution des conflits; le résultat sera le modèle de procédés modifié en état cohérent. Notons que nombreux conflits peuvent survenir lors de la procédure de fusion. En conséquence, pour une meilleure maitrise de l'étude des conflits nous limitons le présent travail aux conflits de premier ordre. Nous appellerons ainsi dans la suite de ce document un résultat cohérent le modèle de procédés modifié exempt de conflits de premier ordre.

Par ailleurs, nous focalisons notre intérêt dans le présent travail sur la solution proposée par le patron à réutiliser, puisque cette dernière constitue la partie du patron qui sera impliquée dans la fusion. Nous omettons ainsi de développer la partie relative à la représentation des patrons. En effet, la représentation des patrons exige d'une part un formalisme pour structurer le contenu des patrons, et d'autre part un langage et donc un méta modèle pour représenter la

partie solution (qui est elle-même un modèle de procédés). Le choix d'un formalisme ne sera pas traité dans le présent travail.

Enfin, il est important de rappeler que la fusion des modèles de procédés n'est pas un problème spécifique à la réutilisation de patrons, mais est plutôt un problème impliqué dans la forme de réutilisation à laquelle nous nous intéressons; c'est ce qui justifie notre grand intérêt pour ce problème de fusion. Il nous semble pour cela important de préciser que la problématique posée dans le présent travail de thèse reste la réutilisation de patrons pour modifier les modèles de procédés, mais que nous ne pouvons aborder cette problématique que si nous abordons la fusion des modèles de procédés.

III. Contributions

Le présent travail de thèse commence par une étude de l'état de l'art relativement: aux modèles de procédés, aux patrons de procédés [HAC17] et à la réutilisation de patrons de procédés. Un focus est ensuite porté sur les travaux de réutilisation, où un cadre de comparaison est défini pour les comparer. Cette comparaison montre en particulier que la réutilisation de patrons pour modifier des modèles de procédés a été rarement traitée dans la littérature.

Vient après l'élaboration de notre approche, qui a suivi le parcours suivant.

Le choix du méta modèle: ce choix est effectué après multiples observations des méta modèles présents dans la littérature. En effet, de nombreux méta modèles sont trop complexes pour être compris et utilisés [BEN07]; alors qu'un méta modèle basé sur des concepts couramment adoptés facilite son exploitation, ses utilisateurs n'auront pas à apprendre un nouveau langage et pourront tirer parti des outils existants [BEN07]. Ces raisons ont orienté notre choix du méta modèle.

L'étude des conflits: cette étude à constitué une phase maîtresse de notre parcours. En effet, les conflits dans un modèle de procédés sont des relations contradictoires qui peuvent exister entre les éléments du modèle. Pour identifier les éventuels conflits, nous avons étudié toutes les combinaisons de relations qui peuvent relier chaque couple d'éléments. Nous avons observé cependant que les conflits peuvent se produire à plusieurs niveaux, selon les relations impliquées dans le conflit. Pour cette raison, et pour mieux maîtriser notre problème d'identification des conflits, nous nous sommes focalisés sur un niveau de conflits que nous avons appelé le premier ordre. Cependant, il existe d'autres niveaux de conflits que nous définissons dans le présent travail, à savoir le deuxième niveau (conflits de second ordre) et le troisième niveau (conflits de troisième ordre) ainsi que le niveau des conflits de cycles indirects.
Nous avons donc identifié de manière exhaustive les dix-huit conflits de premier ordre.

La proposition d'un opérateur de réutilisation: pour donner la sémantique opérationnelle de cet opérateur, nous avons proposé un algorithme de réutilisation de patrons pour modifier des modèles de procédés, avec la gestion des conflits de premier ordre. Les entrées de l'algorithme

sont: (a) le modèle de procédés à modifier, (b) le patron à réutiliser (nous rappelons qu'il s'agit de la solution du patron qui sera mise en œuvre dans la réutilisation), (c) le mode de réutilisation sélectionné. Le produit de l'algorithme résulte de la fusion du modèle de procédés à modifier et de la solution du patron réutilisé.

La mise en place d'une plateforme de réutilisation: notre approche est mise en œuvre à travers une plateforme de réutilisation de patrons, qui nous a permis de la tester et d'exécuter entre autres les exemples donnés dans le présent document. La plateforme repose sur la représentation graphique des patrons et des modèles de procédés, et prend en charge leur sauvegarde et leur restauration à travers une base de données. La plateforme est conçue en six modules collaborant pour fournir les fonctionnalités requises.

Enfin, une évaluation de l'approche que nous décrivons dans le présent travail est établie, et une comparaison pour montrer sa valeur ajoutée est dressée.

IV. Organisation du document

Pour exposer notre approche, le présent document est organisé comme suit:

Le chapitre premier, *État de l'art*, présente l'état de l'art de l'univers des modèles de procédés et des patrons de procédés. Les principales notions manipulées tout au long de cette thèse sont introduites dans ce chapitre, et une étude bibliographique des travaux relatifs à la réutilisation de patrons est présentée.

Le second chapitre, *Réutilisation de patrons sur des modèles de procédés existants*, aborde la problématique de modification d'un modèle de procédés par réutilisation de patrons, aussi bien que l'étude des conflits engendrés par cette forme de réutilisation. Le méta modèle régissant les modèles de procédés et les patrons manipulés à travers notre approche est également abordé dans ce chapitre.

Le troisième chapitre, *Contributions pour la réutilisation de patrons*, présente les contributions de ce travail de thèse. Un aperçu de notre approche ainsi que nos objectifs sont donnés en premier lieu. Ensuite, les conflits de premier ordre traités par l'opérateur de réutilisation sont présentés. Ce chapitre se termine par l'évaluation de notre approche, où une comparaison est établie pour montrer sa valeur ajoutée.

Le quatrième chapitre, *Plateforme de réutilisation de patrons*, est consacré principalement à l'architecture de notre plateforme de réutilisation et à sa réalisation. Il présente une étude de cas, montrant la réutilisation de patrons à travers notre plateforme.

Enfin, dans le chapitre *Conclusion Générale*, une analyse des apports du présent travail de thèse est établie. D'autre part, des perspectives de recherche, relatives à notre approche ainsi qu'au domaine des patrons de procédés, qui pourront êtres poursuivies dans de futures travaux sont adressées.

CHAPITRE I

ETAT DE L'ART

Ce chapitre est consacré aux concepts qui seront exploités dans notre travail. Il aborde dans un premier temps les modèles de procédés (cf. § 1). Dans un second lieu, une introduction au domaine des patrons de procédés est donnée (cf. § 2). Troisièmement, la réutilisation de patrons est abordée, et une synthèse des travaux qui s'y réfèrent est donnée (cf. § 3).

1. Les modèles de procédés logiciels

1.1 L'ingénierie dirigée par les modèles

L'ingénierie dirigée par les modèles (IDM) est une vue abstraite des systèmes, devenue un sous domaine central du génie logiciel [CHA09]. Elle est appliquée à diverses préoccupations du génie logiciel, et contribue à rendre les modèles de procédés logiciels rigoureux et sujets à la manipulation automatique, afin de devenir plus productifs que contemplatifs [BEZ04].

En focalisant le raisonnement sur les modèles, l'IDM permet de travailler à un niveau d'abstraction supérieur et de vérifier divers aspects d'un système. Ainsi, l'IDM est en passe de lever le verrou qui consistait à ne pouvoir tester un système que tardivement dans le cycle de vie [DIA11].

Dans le monde de l'IDM, on distingue quatre niveaux de modélisation, montrés dans la figure I.1 par la pyramide de modélisation de l'OMG (Object Management Group [OMG16]): le niveau *modèle* M1, le niveau *méta modèle* M2, le niveau *méta méta modèle* M3, et le niveau *instance* M0 à la base de la pyramide représentant un système en cours d'exécution [FLE10].

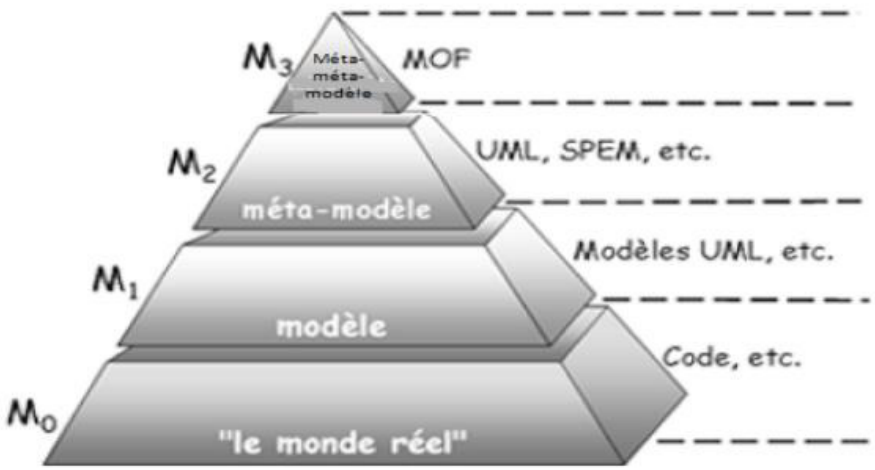


Figure I.1. Pyramide de modélisation de l'OMG [BEZ03]

Un élément de niveau M1 décrit les traits partagés de plusieurs éléments de niveau M0 [FLE10]. La différence entre un élément de niveau M0 et un élément de niveau M1, est semblable à celle qui existe dans les langages de programmation entre une classe et une instance de la classe [OST87].

Un modèle de niveau M1 est dit conforme à un méta modèle de niveau M2, s'il respecte les règles spécifiées par ce dernier [FLE10]. Meta Object Faclity (MOF) [MOF11] est un méta méta modèles (niveau M3), c'est à dire un formalisme pour décrire des langages de modélisation (des méta modèles). Il s'auto décrit pour pouvoir limiter l'architecture pyramidale à quatre niveaux, et éviter une progression infinie des niveaux méta [DIA11].

1.2 Procédé logiciel

Un cycle de vie offre une trame à partir de laquelle on peut bâtir une stratégie de développement, propre à chaque projet. Il décrit le *quoi faire* (les tâches à mener et les résultats attendus), laissant de côté une bonne part du *comment* (les méthodes, les techniques), du *qui* (les ressources humaines, matérielles et logicielles) et du *quand* (les dates de début et de fin des tâches et leurs relations de dépendance). Un cycle de vie n'est donc pas utilisable tel quel dans un projet. De ces lacunes est née la notion de procédé (de développement) logiciel [FLE10].

Un procédé logiciel a pour but de supporter la production de logiciels [KAB09]. Il se situe au niveau M0 de la pyramide de l'OMG, et est définie comme "une suite d'opérations ou d'évènements, faisant intervenir des équipes de personnes, des outils et des techniques pour assurer le développement et la maintenance de logiciels" [COM08]. C'est en fait un processus réel de développement, adapté au contexte d'un projet donné, matérialisé par un planning prévisionnel avec affectation des rôles et des ressources, et lancé au démarrage du projet [FLE10].

1.3 Modèle de procédés

Un modèle est une description d'un système du monde réel. Il se situe au niveau M1 de la pyramide de l'OMG. C'est l'abstraction de ce qui est intéressant d'une réalité, afin de faciliter sa compréhension et la simulation de son fonctionnement [GAR07].

La modélisation de procédés est l'opération qui consiste à concevoir un modèle de procédés. Cette conception est l'abstraction et la généralisation des constituants des procédés logiciels du monde réel, sous forme d'éléments abstraits et de relations abstraites entre ces éléments. La conception de modèles de procédés peut se faire également à partir d'autres modèles de procédés.

Selon l'usage qui lui est attribué, un modèle peut être descriptif ou prescriptif [FLE10].

- Descriptif: le modèle offre une représentation d'un système existant, mais c'est le système qui tient lieu de référence. On fait en sorte que le modèle puisse se substituer à lui pour un usage prévu, par exemple, la compréhension en phase de maintenance, ou la vérification de propriétés en phase de test.

- Prescriptif: le modèle est la référence. Il représente un système qui n'existe pas encore ou qui doit évoluer. Il pose des contraintes que le système devra respecter.

L'objectif de la modélisation des procédés n'est pas seulement de capitaliser un savoir faire en matière de développement logiciel, sous forme d'un modèle de procédés, mais de s'en servir pour comprendre, analyser et exécuter le procédé [DIA11]. Ainsi, la modélisation de procédés permet entre autres l'analyse et l'amélioration des procédés [GAR06]. Elle facilite également la gestion de procédés, puisqu'on peut développer des outils d'assistance pour la

planification, le contrôle et la surveillance de procédés [GAR06]. Enfin, un modèle de procédés sert à faciliter la communication entre les participants d'un procédé [GAR06].

1.4 Méta modèle de procédés

Depuis qu'il a été établi que la qualité des procédés logiciels pouvait avoir un impact considérable sur la qualité des produits logiciels [HUM88], la communauté du génie logiciel ne cesse de voir émerger des langages de modélisation de procédés [DIA11]. En effet, un langage de modélisation de procédés (Process Modeling Language) est un formalisme pour la modélisation de procédés logiciels, textuel ou graphique, qui permet de décrire des modèles de procédés [KAB09].

Les principales caractéristiques de ces langages [KAB09] sont principalement: l'expressivité, qui est la capacité de représenter tous les éléments du procédé par des dispositifs du langage; la compréhension, qui est le degré de facilité avec lequel on arrive à comprendre le modèle décrit à travers le langage; l'abstraction et la modularité, qui représentent la capacité du langage de supporter les mécanismes d'abstraction et de modularité afin de structurer les modèles; et l'exécutabilité, qui est la capacité du langage de définir des modèles opérationnels (exécutables).

Un méta modèle, défini au niveau M2 de la pyramide de l'OMG, est "un modèle définissant le langage pour exprimer des modèles" [BEN07]. Il fait des déclarations au sujet de ce qui peut être exprimé dans les modèles valides d'un certain langage de modélisation [SEI03]. Autrement dit tout modèle valide doit respecter la structure définie par son méta modèle [BLA05].

La relation liant un modèle à son méta modèle est la relation "conforme à". Un modèle est dit conforme à son méta modèle si les éléments définis dans le modèle, ainsi que les relations entre ces éléments, sont définis dans le méta modèle [BEN07].

1.5 Représentation graphique

Un langage graphique possède non seulement un alphabet composé de symboles de type caractère mais également de symboles de type figure géométrique, et possède aussi plusieurs opérateurs de placement pour exprimer par exemple la proximité, l'inclusion, et la connexion [FLE10]. Un tel langage se distingue par le fait que le placement des symboles nécessite de s'appuyer sur des opérateurs de placement, et que l'on use d'un panel plus vaste de symboles caractérisés par leurs formes, tailles, orientations et couleurs [FLE10].

L'utilisation d'un langage graphique dans la modélisation de procédés logiciels, appelée représentation graphique de modèles de procédés, a le mérite de pouvoir être lue d'un simple regard et de pouvoir véhiculer l'information indépendamment d'une langue particulière ou d'un niveau d'expérience particulier. Une telle utilisation permet aux concepteurs de procédés de bénéficier d'un support expressif et compréhensible, facilitant la communication.

Les outils de représentation graphique de modèles de procédés offrent la possibilité de représenter un modèle sur une surface de dessin, juste en ramenant les différents concepts de procédés, substitués en symboles, par des actions glisser/déposer et des agencements. Ainsi, la plupart des outils de modélisation, dans différentes disciplines de l'informatique, adoptent le mode graphique pour représenter la structure d'un système.

2. Les patrons de procédés logiciels

2.1 Les patrons logiciels

Un patron logiciel (software pattern) est un concept destiné à résoudre un problème récurrent du domaine du génie logiciel, en apportant une solution prouvée et abstraite.

L'origine des patrons remonte aux années soixante-dix avec les travaux de l'architecte Christopher Alexander, qui a remarqué que la phase de conception en architecture laisse apparaître des problèmes récurrents. Il a cherché alors à résoudre l'ensemble de ces problèmes, et a établi une collection de 253 patrons [ALE77] couvrant tous les aspects de la construction.

La première fois où des patrons sont apparus dans le monde du génie logiciel fut en 1987, avec le fameux papier de Kent Beck et Ward Cunningham intitulé *Using Pattern Languages for Object-Oriented Programs* [BEC87]. Ce travail décrit cinq patrons relatifs à des problèmes de conception de fenêtres en *Small talk*.

La communauté logicielle a suivi l'exemple de Beck et Cunningham, et des patrons ont commencé à apparaitre et à investir plusieurs domaines. Parmi les premiers travaux fut celui des patrons organisationnels de Coplien [COP94]; mais le travail le plus célèbre dans le monde des patrons logiciels demeure l'ouvrage publié en 1995 par le Gang of Four [GAM95], sur les patrons de conception (Design Patterns) qui font aujourd'hui référence dans le monde de l'informatique. Un patron de conception décrit une solution prouvée à un problème récurrent dans la conception de logiciels, indépendamment des langages de programmation.

La communauté s'est rendu compte de la puissance des patrons, et d'autres types de patrons ont commencé à émerger couvrant des aspects divers, tels que les patrons de logiciels d'entreprises [FOW02], les patrons de workflow [ALA03], les patrons de sécurité informatique [FER07], les patrons d'applications web [VOR09] ... Et surement, on ne cessera de voir fleurir des patrons.

L'utilisation des patrons offre de nombreux avantages [HAC11]:

D'abord, les patrons permettent de mettre en avant les bonnes pratiques, en répondant à un problème via une solution prouvée et validée. Ainsi, on gagne en rapidité et en qualité.

De plus, l'abstraction du problème et de la solution dans un patron le rend réutilisable et capable de s'appliquer dans divers cas de figure. Il pourra ainsi être instancié dans plusieurs situations et pourra être enrichi si nécessaire par des informations complémentaires.

Aussi, les patrons (notamment les patrons de conception) étant largement documentés et connus d'un grand nombre de spécialistes, permettent de faciliter la communication. Par exemple, si un développeur annonce que sur un point du projet il utilise le patron *Observer*, il sera compris des informaticiens sans pour autant rentrer dans les détails de la conception.

Enfin, de même que la qualité d'un code source peut être jugée par un regard rapide aux normes de codage qui ont été suivies, la qualité d'une solution peut maintenant être estimée d'après les patrons utilisés.

2.2 Les patrons de procédés logiciels

Une fois familier avec les patrons logiciels, on est capable d'inférer ce qu'est un patron de procédés à partir de son nom. Le terme *patron de procédés* (*Process Pattern*) a été introduit la première fois par Coplien [COP94], dans le sens d'un patron qui capture des connaissances de procédés de développement logiciel. Alors que les patrons de conception représentent une issue aux problèmes récurrents dans la phase de conception des logiciels, les patrons de procédés montrent comment accomplir efficacement les travaux quelque soit la phase du développement logiciel. Les patrons de conception s'intéressent au produit de la conception, et non pas au processus de conception, qui est du domaine des patrons de procédés.

Dans la littérature, les définitions données pour la notion de patrons de procédés sont encore imprécises, et ne font pas l'objet d'un consensus. Un patron de procédés est défini comme : un modèle fournissant "une conception répandue (a widespread design) pour les procédés" [IID02]; un modèle représentant "un procédé prouvé qui résout un problème récurent de développement" [DIT02]; un modèle qui capture "une structure commune de procédés" [STO01]; "un patron qui capture un procédé prouvé, applicable pour résoudre un problème récurent de développement" [HAG04]; ou "un fournisseur de meilleures pratiques (best practices) pour manipuler des problèmes récurrents durant la phase de développement logiciel" [BUN15].

Les patrons de procédés offrent de nombreux avantages [OSM12]: ils fournissent des approches solides pour résoudre des problèmes en utilisant des solutions prouvées. Ils peuvent être expressifs et agir comme un vocabulaire qui facilite la communication. Ils fournissent des solutions généralisées, de sorte qu'elles peuvent être réutilisées et adaptées à différents besoins. La réutilisation de patrons aide à prévenir les difficultés mineures qui causent des problèmes majeurs dans le processus de développement.

De plus, les patrons de procédés offrent des avantages spécifiques à la modélisation de procédés [TRA07]. En effet, ils fournissent une description modulaire et réutilisable de procédés, puisqu'ils peuvent modéliser un fragment de procédés comme un composant réutilisable correspondant à un problème spécifique. S'ajoute à cela que l'utilisation des

patrons de procédés de manière formelle a le potentiel d'automatiser le procédé de développement logiciel, et de réduire les coûts de développement [ENG16]. Enfin, comme les patrons sont fréquemment utilisés, ils peuvent être améliorés au fil du temps en exploitant l'expérience collective de leurs utilisateurs et en contribuant en retour à la communauté des patrons [OSM12].

Ainsi, un patron de procédés représente une approche réutilisable d'un procédé dont il est prouvé de rapporter des résultats efficaces. Parmi les premiers patrons de procédés qui ont émergé, on cite les patrons d'Ambler [AMB98a] [AMB99]. D'autres auteurs ont suivi l'exemple, et des patrons de procédés ne cessent d'éclore. Une riche introduction au monde des patrons de procédés logiciels est présentée dans [HAC17].

2.3 Travaux de recherche sur les patrons de procédés

Les travaux de recherche sur les patrons de procédés peuvent se classer globalement en les quatre catégories suivantes, qui représentent les principales thématiques de recherche sur les patrons.

2.3.1 Réutilisation de patrons

Les deux principales aptitudes qui rendent possible la définition d'artéfacts réutilisables sont l'abstraction et la généralisation. L'abstraction est l'aptitude à ne retenir d'une chose que l'essence utile pour un contexte donné. Elle est utilisée pour documenter efficacement des artéfacts réutilisables [FLE10]. La généralisation est l'aptitude à identifier dans diverses situations des éléments communs et des points de variation. Elle permet d'augmenter la capacité de réutilisation, en explicitant pour tout artéfact une partie fixe et une partie variable [FLE10].

Ainsi, l'ingénierie de réutilisation des patrons de procédés, comme pour tous les artefacts réutilisables, s'appuie sur deux cycles dépendants de développement d'artefacts. Un cycle de développement *par* la réutilisation, produisant des artéfacts propres à chaque projet en s'appuyant sur des artéfacts préexistants et mis à disposition dans des bibliothèques, des marchés, des ouvrages, etc. [FLE10]. Un cycle de développement *pour* la réutilisation, qui produit des artéfacts réutilisables offrant un certain niveau de généricité, et mis ensuite à disposition dans des bibliothèques, des marchés, des ouvrages, etc [FLE10].

Les patrons de procédés n'échappent pas à cette règle. Leur réutilisation est également régi par ces deux cycles, *par* et *pour* la réutilisation.

2.3.2 Organisation de patrons

Les relations liant des patrons entre eux constituent une base pour la composition de patrons en solutions complexes. Une telle composition permet de résoudre des problèmes complexes, qui ne sont pas traités par un patron tout seul. Il est nécessaire donc de savoir quels patrons peuvent fonctionner ensemble pour les faire collaborer. Les contextes des patrons doivent pour cela être exprimés avec plus d'exactitude, et les relations entre patrons doivent être définies plus précisément [HAG02].

Partant de cela, plusieurs chercheurs ont abordé les relations entre patrons. Certains ont travaillé sur la définition de ces relations, avec éventuellement des notations graphiques et des langages pour les représenter. D'autres travaux plus évolués ont tenté de faire ressortir ces relations, dans le cas où elles ne sont pas explicites ou lorsqu'elles concernent différents catalogues de patrons. Quelques exemples de travaux sont donnés dans ce qui suit.

Nous citons parmi les travaux de définition de relations entre patrons celui de Gnatz *et al.* [GNA03], qui ont proposé les deux relations suivantes:

- *Alternative*: cette relation est définie entre deux patrons qui résolvent le même problème.
- *Raffine*: cette relation est définie entre un patron supérieur, et des patrons réalisant des activités contenues dans le patron supérieur.

Nous citons également le travail de Henney [HEN99], qui stipule que "un riche ensemble de relations entre patrons est le sang qui permet à un langage de patrons de devenir vivant". Il définit les relations suivantes entre patrons.

- *Solution is context*: cette relation signifie que la solution d'un patron est une part du contexte d'un autre patron. Ceci comprend le cas où un patron fournit un support à un autre patron.
- *Specialization*: cette relation signifie qu'un patron résout un problème plus spécifique que celui résolu par l'autre patron.
- *Pattern pair*: cette relation signifie que deux patrons résolvent le même problème, mais avec des solutions complémentaires.
- *Inverted patterns*: cette relation signifie que deux patrons proposent des solutions à des problèmes similaires, mais avec des structures (solutions) presque inverses.

Quant aux travaux d'analyse de relations entre patrons, nous citons le travail de Prabhakar *et al.* [PRA10], qui proposent un modèle graphique DDTM (Design Decision Topology Model), pour analyser les relations entre patrons de conception de manière automatique. Ces relations sont assimilées aux relations entre les graphes représentant les patrons.

Nous citons également le travail de Kubo *et al.* [KUB05], qui se trouve parmi les premiers travaux à aborder les relations entre patrons de différents catalogues. Ce travail présente une approche automatique d'analyse des relations entre patrons.

Nous citons aussi le travail ce Hachemi *et al.* [HAC12], qui s'inspire de [KUB05] pour proposer une approche automatique d'analyse des relations primaires entre patrons. Par relations primaires, il est entendu les relations *Uses* et *Refines*. Ce travail se base sur un analyseur d'inclusion et une mesure de similarité entre patrons, pour détecter les éventuelles relations existantes entre eux.

2.3.3 Modélisation de patrons

Dans cet axe de recherche, deux problèmes sont distingués: la proposition de formalismes pour représenter les patrons, et le développement de langages de description de ces patrons.

Un formalisme est une structure syntaxique comportant un ensemble de rubriques structurant le contenu des patrons, où chaque rubrique représente une information particulière du patron. Les formalismes de description de patrons sont généralement tous basés sur le triplet (problème, contexte, solution), pour exprimer respectivement un certain problème récurrent, le contexte d'application du patron et la solution pour ce problème.

Le besoin d'un langage pour modéliser les patrons émerge de plus en plus. Un tel langage précise la sémantique des concepts qu'il véhicule et fournit éventuellement les notations graphiques. Un tel langage devrait être basé sur un méta modèle, qui décrit de manière précise tous les éléments de modélisation du patron et la sémantique de ces éléments. L'aspect formel d'un tel langage limite les ambiguïtés et les incompréhensions.

Parmi les principaux travaux dans le domaine des langages on cite PROPEL (PROcess Patterns dEscription Language) [HAG04] [HAG04a]. Il s'agit d'un langage de description de patrons de procédés semi formel et basé sur UML. Un procédé dans PROPEL est modélisé comme un diagramme d'activité UML, qui décrit le déroulement d'un ensemble d'activités et le flux d'objets entre ces activités; un rôle est responsable de l'exécution du procédé.

Dans PROPEL, un patron (ProcessPattern) propose un procédé prouvé (Process), pour résoudre un problème récurrent de développement logiciel (Problem), dans un contexte (Context) définissant les conditions qui doivent être vérifiées avant et après l'application du patron. Un patron est un composant de ProcessPatternCatalog, c'est à dire qu'un patron est toujours assigné à un catalogue de patrons par l'association catalogue (Catalog).

En termes de formalismes, nous citons parmi les principaux travaux dans ce domaine celui d'Ambler [AMB98]. Le formalisme d'Ambler comporte les six rubriques suivantes.

- *Nom*: nom qui décrit brièvement le patron.
- *Type*: prend l'une des valeurs *Tâche*, *Phase* ou *Étape*, pour signifier si le patron décrit une tâche, une phase ou une étape.
- *Forces*: buts à atteindre et contraintes à respecter en appliquant le patron.
- *Initial contexte*: conditions nécessaires pour déclencher l'application du patron.
- *Solution*: décrit en détail comment exécuter les actions du patron pour résoudre le problème.
- *Final contexte*: conditions qui doivent être satisfaites après l'application du patron.

Il est à noter que des travaux de recherche se sont intéressés à la correspondance entre formalismes de patrons (patterns forms matching). De tels travaux ont pour but de faire collaborer des patrons hétérogènes, c'est à dire des patrons exprimés dans des formalismes différents. Nous citons par exemple [*HAC12a*] qui vise les patrons dont le formalisme ne contient pas de contexte final.

2.3.4 Identification de patrons

L'identification de patrons constitue un très important axe de recherche sur les patrons. Elle consiste d'abord à identifier les sources de connaissances susceptibles de contenir des

problèmes récurrents, ensuite à identifier ces problèmes pour enfin spécifier les solutions à offrir dans les patrons [GZA00].

Les travaux dans cet axe de recherche tentent de proposer des ensembles de patrons extraits d'expériences réelles et prouvées, pour fournir à leurs exploitants des solutions efficaces. Pour représenter de tels ensembles de patrons, on emploi souvent le terme *catalogue* de patrons, qui désigne un groupement de patrons répondants à une problématique commune. D'autres termes sont aussi utilisés pour désigner de tels ensembles, comme *collection*, *famille*, *système* ou même *langage*.

Nous donnons dans ce qui suit, à titre d'exemple, quelques collections de patrons logiciels.

Le catalogue de Staudt *et al.* [STA10] est constitué de patrons de haut niveau, traitants les problèmes communs de gestion des exceptions. Ces patrons s'intéressent à l'utilisation du mécanisme de gestion des exceptions, dans différents contextes de procédés. L'objectif étant de permettre aux concepteurs de procédés de réfléchir en terme de ces patrons et de distinguer quand est ce qu'ils sont utiles dans un procédé.

Le catalogue de patrons de Demeyer *et al.* [DEM03] intitulé Object Oriented Reengineering Patterns (OORP), est une collection de patrons s'intéressant à la réingénierie (Reengineering) et à la rétro ingénierie (Reverse engineering) de logiciels orientés objet.

3. La réutilisation de patrons de procédés

3.1 Principaux travaux de réutilisation de patrons

Le double défi auquel est confrontée l'ingénierie des logiciels est celui de la qualité et de la productivité. La réutilisation est par conséquent considérée comme un moyen efficace pour surmonter ce double défi. Dans ce contexte, les patrons s'imposent comme un excellent outil, offrant des solutions prouvées et réutilisables pour la modélisation de procédés. Plusieurs travaux de recherche ont donc été menés autour des patrons, et l'importance de pouvoir automatiser la procédure de réutilisation de patrons est bien ressenti. Le but d'une telle réutilisation étant d'offrir à la communauté du génie logiciel le moyen de tirer profit de la connaissance prouvée offerte par les patrons, pour réduire l'effort et le temps de modélisation de procédés et pour améliorer la qualité de ces modèles.

Les principaux travaux de réutilisation de patrons pour la modélisation de procédés sont récapitulés dans ce qui suit.

3.1.1 L'environnement PROMENADE

PROMENADE (PROcess oriented Modelling and ENActment of software DEvelopments) [RIB00] [RIB02] est un environnement qui fournit les mécanismes de réutilisation de modèles de procédés. Un patron de procédés dans PROMENADE est considéré comme un modèle de procédés. Un patron dans PROMENADE peut être paramétrable et/ou contient des éléments

non raffinés. Il peut être personnalisé pour répondre aux besoins d'un projet particulier et/ou combiné avec d'autres patrons.

L'application des patrons dans PROMENADE se fait via l'opérateur *Binding*. Cet opérateur crée une relation entre un élément du modèle de procédés fournisseur, et un élément du modèle client qui l'instancie. Ce lient est établi par le biais des paramètres effectifs qui sont liés aux paramètres formels du modèle fournisseur. Le méta modèle régissant PROMENADE est montré dans la figure I.2.

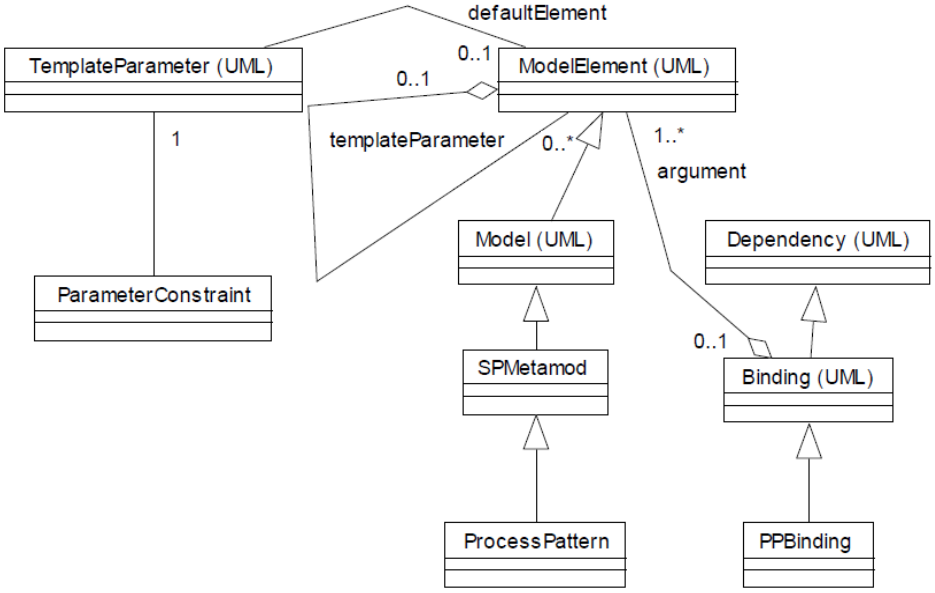


Figure I.2. Méta modèle régissant les patrons de procédés dans PROMENADE [RIB02]

Dans PROMENADE, la notion de réutilisation de procédés *process reuse* est mise en œuvre à travers deux activités: (1) *Harvesting* qui transforme un ou plusieurs modèles de procédés en un patron réutilisable, et (2) *Reuse* qui transforme un ou plusieurs modèles (à vrai dire un ou plusieurs patrons, qui sont eux même des modèles de procédés) en un modèle exécutable.

Ces deux activités sont réalisées grâce aux mécanismes suivants.

- *Abstraction*: Pour se débarrasser de certains détails spécifiques ou indésirables d'un modèle de procédés. Ce mécanisme est utilisé dans l'activité *Harvesting*.
- *Adaptation*: Pour rendre un modèle de procédés (ici, un patron de procédés) plus spécifique, ou convenable pour être réutilisé dans un projet particulier. Ce mécanisme est utilisé dans l'activité *Reuse*.
- *Composition*: Pour combiner un ensemble de modèles de procédés, afin de créer un nouveau. Ce mécanisme est utilisé à la fois dans *Harvasting* et *Reuse*.
- *Delayed binding*: Pour retarder jusqu'au moment de l'exécution la sélection d'une partie spécifique d'un modèle de procédés.

L'approche PROMENADE est intéressante, pourtant elle ne traite pas des patrons de procédés proprement dit, mais plutôt des modèles de procédés paramétrables. De plus, elle manque d'un guidage méthodologique pour une réutilisation systématique des patrons dans la

modélisation de procédés. Aussi, restructurer un modèle de procédés existant selon un patron n'est pas possible dans cette approche. Plusieurs aspects de PROMENADE sont incomplets, tels que le manque d'outil graphique pour modéliser des procédés logiciels.

3.1.2 L'environnement RHODES

L'environnement RHODES [COU01] [COU02] propose une approche basée sur un méta modèle, et sur des composants de procédés réutilisables. Il est constitué des éléments suivants:

- Le méta modèle, montré par la figure I.3, qui formalise la sémantique des entités de procédés ainsi que la composition des différents composants.
- La base des composants de procédés, qui est un dépositoire permettant aux concepteurs de procédés de sauvegarder et de puiser dedans des entités réutilisables.
- L'éditeur des composants de procédés, comprenant un ensemble d'outils pour manipuler les composants de procédés.
- Le noyau d'exécution avec le compilateur PBOOL.
- Les instances exécutables de RHODES, correspondant aux procédés logiciels.

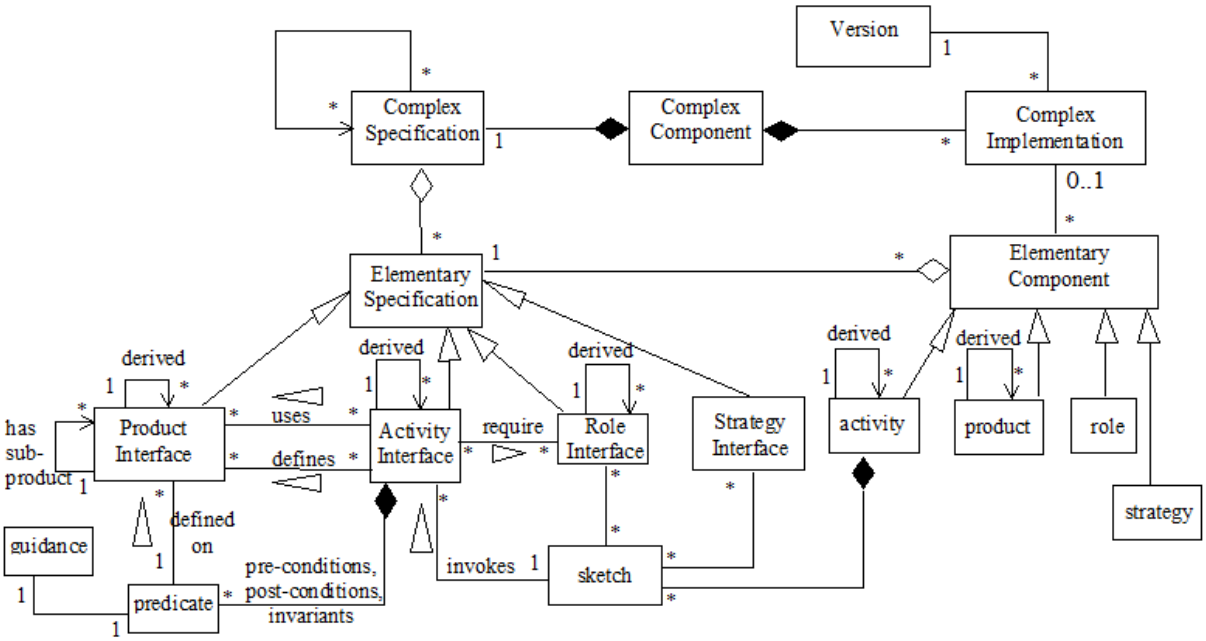


Figure I.3. Le méta modèle de RHODES [COU02]

Dans RHODES, deux niveaux de composants de procédés sont distingués: les composants élémentaires et les composants complexes. Les composants élémentaires sont des entités basiques de procédés (activités, produits, rôles et stratégies). Les composants complexes sont utilisés pour grouper les composants élémentaires selon un sujet d'application particulier. Un patron dans RHODES est un composant complexe.

Dans l'environnement RHODES, on parle de patrons de procédés, alors qu'il s'agit plutôt de composants complexes, regroupés par domaine d'application (analyse, conception, test …). RHODES offre un guidage pour la réutilisation de patrons à travers un méta procédés, mais ce guidage reste non détaillé. De plus, l'environnement RHODES qui ne peut manipuler que

ses propres patrons écrits en PBOOL, ne permet pas de restructurer un modèle de procédés existant en réutilisant un patron.

3.1.3 L'approche d'Iida

L'approche d'Iida [IID02] est basée sur des composants de procédés réutilisables, qui seront assemblés selon des patrons compositionnels (*Compositional Patterns*). Cette approche suppose les composants de procédés capables de se connecter entre eux de manière flexible, et décrit les patrons compositionnels comme des modèles pour la connexion de ces composants.

Chaque composant de procédés dans l'approche d'Iida encapsule une série d'activités, et possède les caractéristiques suivantes:

- Définit explicitement une interface qui peut être inspectée de l'extérieur, pour que des modules de procédés externes puissent savoir comment accéder au composant. Les composants de procédés peuvent être connectés entre eux via les informations de leurs interfaces.
- A des objets (artefacts/produits) en entrée et en sortie.
- Spécifie explicitement un objectif et un responsable.

Un patron compositionnel est défini comme un ensemble d'éléments de procédé et de connexions entre ces éléments. Il peut être vu comme un contexte dans lequel les composants sont utilisés. La Figure I.4 montre un exemple d'un patron compositionnel.

La figure I.5 montre les relations entre les classes du procédé composite, les composants de procédés et le patron compositionnel. Chaque procédé composite comporte un *composition manager,* qui organise les composants de procédés selon le patron organisationnel choisi. Le *composition manager* possède obligatoirement un patron compositionnel par défaut, qu'il utilisera si aucun patron ne lui est spécifié.

Pour permettre l'exploitation de leur approche, Iida *et al.* ont développé un outil graphique permettant la définition d'un procédé, en faisant assembler des composants graphiquement représentés sur une fenêtre.

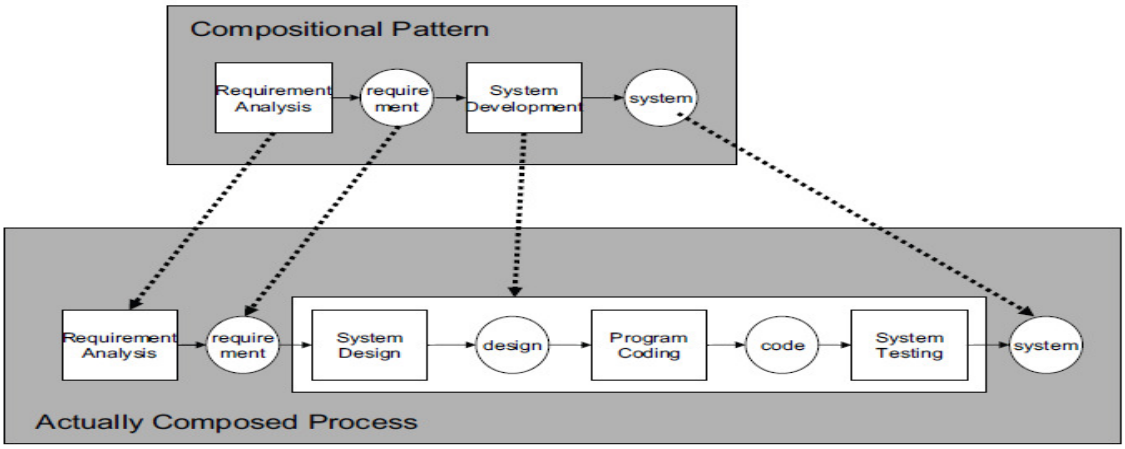


Figure I.4. Exemple d'un patron compositionnel [IID02]

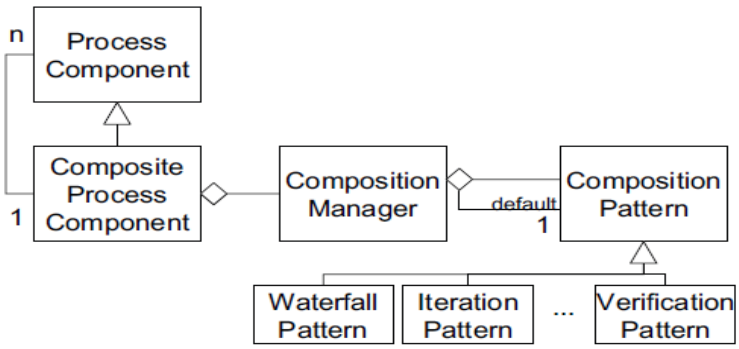


Figure I.5. Méta modèle de l'approche d'Iida [IID02]

Malgré l'intérêt de l'approche d'Iida, elle présente l'inconvénient suivant: on ne peut réutiliser un patron *compositional* que pour créer un nouveau modèle de procédés, il n'est pas possible par contre de modifier un modèle de procédés en réutilisant des patrons.

3.1.4 La réutilisation dans SPEM

SPEM (Software and Systems Process Engineering Meta-model) [OMG08] est une norme de l'OMG dédiée à la description de procédés logiciels. Le modèle conceptuel de SPEM repose sur l'idée qu'un procédé de développement logiciel est une collaboration entre des rôles, qui effectuent des activités sur des produits.

SPEM 1.1 est structurée en deux paquetages: *SPEM-Foundation* qui est une extension d'un sous ensemble d'UML, fournissant les concepts de base pour définir le méta-modèle SPEM, et *SPEM-Extension* qui décrit des concepts de base pour décrire les procédés logiciels.

Le concept central de SPEM 1.1 est *WorkDefinition*, une unité de travail réalisée par un responsable *ProcessPerformer,* pour produire ou modifier des artéfacts *WorkProduct*. SPEM1.1 supporte l'organisation modulaire et la réutilisation de procédés via le concept de composant de procédés *ProcessComponent,* qui est spécialisé en discipline *Discipline* et procédé *Process*. *Discipline* regroupe des activités ayant le même thème, alors que *Process* est considéré comme un composant de procédés autonome, contenant un ensemble de *WorkDefinition* et gouverné par un cycle de vie.

Dans SPEM 2.0, le contenu d'une méthode (*MethodContent*) est un élément réutilisable, il fournit des explications décrivant la réalisation des buts spécifiques du développement, de façon indépendante d'un cycle de vie concret. Une activité (*Activity*) dans SPEM 2.0 est aussi bien une unité de travail dans un procédé qu'un procédé lui-même. Elle prend des éléments *MethodContent* et les relie selon l'ordre d'exécution adapté aux besoins d'un projet.

La réutilisation des éléments *MethodContent* est réalisée grâce au concept de *Method Content Use*. Ce dernier référencie un *Method Content Element* et fournit sa représentation dans un procédé. Ainsi, un *Method Content Element* peut être représenté par différents *Method Content Use,* chacun dans le contexte d'un procédé et avec son propre ensemble de relations avec d'autres éléments de ce procédé.

SPEM 2.0 propose le concept de patron de procédés *ProcessPattern* pour la capitalisation des bonnes pratiques et la création de nouveaux procédés. Un patron est représenté simplement comme un procédé ayant le type *ProcessPattern*. La réutilisation de patrons peut être réalisée de deux façons: (1) via des opérations de type *copier-modifier* pour permettre aux concepteurs d'adapter les patrons selon leurs besoins; ou (2) via le mécanisme *Activity Use* qui permet l'établissement d'une relation entre une activité à définir (qui est aussi bien une activité qu'un procédé) et le patron à réutiliser (lui aussi étant une activité), en spécifiant le type d'application. Les types d'application proposés par SPEM 2.0 sont: *extention*, *localContribution* et *localReplacement*.

SPEM 2.0 considère les patrons comme des procédés réutilisables, et fournit les concepts et les mécanismes généraux pour les réutiliser. Cependant, l'adaptation des éléments réutilisables fournie par SPEM2.0 est encore limitée et informellement définie, et la possibilité de modifier un modèle de procédés existant en réutilisant un patron n'est pas offerte par SPEM. Il est à noter également que SPEM ne donne aucun guidage pour la réutilisation de patrons.

3.1.5 L'approche de Tran

L'approche de Tran *et al.* [TRA11] a pour but la réutilisation efficace de patrons dans la modélisation de procédés logiciels. Cette approche comporte deux volets: (1) d'une part un langage de description de procédés, qui intègre le concept de patron de procédés; (2) d'une autre part une méthode de modélisation de procédés à base de patrons, constituée d'un ensemble d'opérateurs de réutilisation de patrons, et d'un méta procédé décrivant quand et comment réutiliser les patrons pour élaborer des modèles de procédés.

Le langage de description de procédés proposé dans l'approche de Tran *et al.* est défini par le méta modèle UML-PP (UML for Process Patterns). Ce dernier permet de décrire les éléments principaux des procédés logiciels, et de représenter les procédés basés sur les patrons. Il est organisé en trois paquetages: le paquetage *ProcessStructure* décrit les éléments constituant un procédé; le paquetage *ProcessPattern* décrit la structure d'un patron de procédés; le paquetage *PatternRelationship* décrit les mécanismes d'application de patrons, ainsi que les relations entre patrons.

Les operateurs de réutilisation de patrons offerts par l'approche de Tran *et al.*, montrés par la figure I.6, se déclinent en trois catégories: *opérateurs de recherche*, *opérateurs d'adaptation* et *opérateurs d'imitation*. L'approche définit ces opérateurs et leur donne une sémantique opérationnelle pour les rendre exécutables.

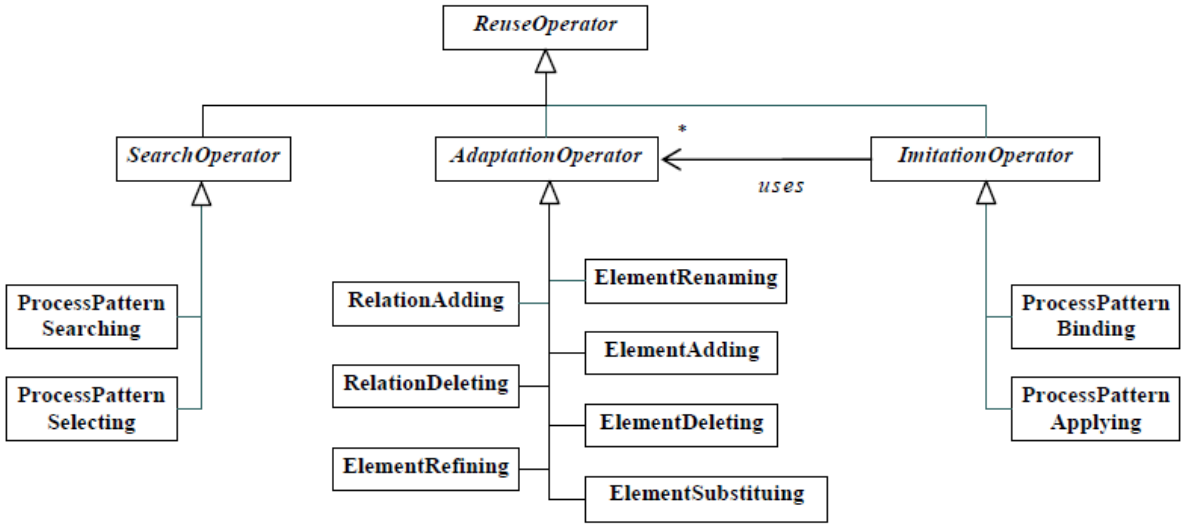


Figure I.6. Opérateurs de réutilisation [TRA07]

Le méta procédé proposé par l'approche de Tran *et al.*, nommé PATPRO, a pour objectif la construction ou l'amélioration de modèles de procédés, en guidant les concepteurs dans la réutilisation de patrons. PATPRO décrit les méta tâches pour construire statiquement des modèles de procédés, et supporte le raffinage de modèles pendant la modélisation. PATPRO se présente sous la forme des quatre méta tâches suivantes, organisées selon un cycle de vie inspiré de la cascade.

- *Analyser les Besoins*: cette méta tâche analyse les caractéristiques du procédé à modéliser, et identifie les besoins de modélisation.
- *Modéliser un Procédé*: cette méta tâche élabore le modèle de procédés à partir de sa spécification. C'est la tâche centrale de la modélisation.
- *Simuler un Procédé*: cette méta tâche simule l'exécution d'un modèle de procédés, par l'intermédiaire d'un moteur d'exécution de procédés.
- *Évaluer un Procédé*: cette méta tâche évalue un modèle de procédés en le comparant avec des critères prédéfinis.

Dans le but de fournir une assistance outillée aux concepteurs, l'approche de Tran *et al.* avait pour but de développer un environnement de modélisation de procédés en UML-PP, et de réutiliser des patrons au cours de la modélisation. Toutefois, cet environnement n'a pas été entièrement implémenté [TRA11].

L'approche de Tran *et al.* est une approche assez riche et complète dans le domaine de la réutilisation de patrons. Cependant, lors de l'application de l'opérateur *ProcessPatternApplying* qu'elle défini, des conflits structurels peuvent apparaître dans le résultat, ce qui nécessite l'intervention du concepteur de procédés pour la détection et la résolution de ces conflits. De plus, un modèle de procédés peut comporter des informations incomplètes qui seront raffinées ou complétées ultérieurement (lors de l'exécution de l'instance). Dans le méta procédé de Tran *et al.*, ce raffinage est effectué avant l'exécution de

l'instance. Pourtant, pour être plus flexible et efficace, ce méta procédé devra dans l'avenir prendre en compte l'adaptation dynamique des modèles de procédés [TRA07].

3.1.6 L'approche OPF

OPF (OPEN Process Framework) [FIR01] est un moyen de construire des procédés logiciels, à travers une collection de composants de procédés réutilisables. OPF est constitué d'un méta modèle, d'un répertoire de composants et d'un guidage d'utilisation. Un procédé OPF est créé en sélectionnant des composants, ou en créant des instances de composants à parti des éléments du méta modèle. Le méta modèle d'OPF décrit les différents éléments d'un procédé, ainsi que la manière dont ils interagissent.

Pour construire un procédé, il faut d'abord instancier les entités du méta modèle. Les composants de procédés sont ensuite stockés dans le dépositoire d'OPF. Ces composants pourront après être mis ensemble pour construire un modèle de procédés. Lorsqu'un modèle de procédés est appliqué à un projet spécifique, il est en réalité entrain d'être instancié. Le résultat sera un procédé spécifique au projet (*project-specific process*), dans lequel des personnes nommées jouent des rôles précis, avec des échéances du calendrier et la livraison effective du produit.

La structure d'OPF, montrée par la figure I.7, comporte les types suivants. *Producer*: représente les différents types d'agents qui réalisent le développement. *Work Unit*: représente les travaux à effectuer pour manipuler des artefacts de développement. *Work Product*: représente les artéfacts utilisés ou créés durant le développement. *Stage*: représente les éléments d'organisation de développement. *Language*: représente les différents langages utilisés pour développer ou pour documenter certains artefacts. *Guidelines*: servent pour (a) instancier le Framework et choisir les composants à instancier ou à sélectionner à partir de la collection, (b) adapter les composants sélectionnés, et (c) étendre la collection de composants.

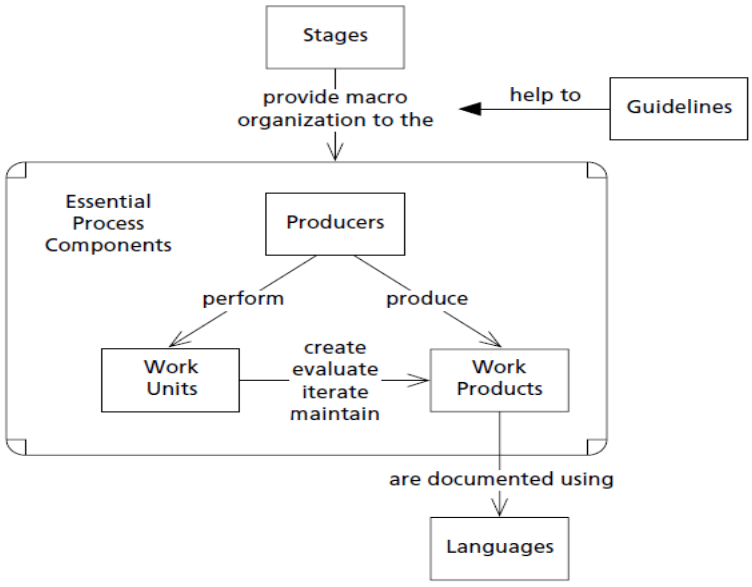


Figure I.7. Le méta modèle d'OPF [FIR01]

L'approche OPF ne traite pas des patrons de procédés proprement dits, mais propose la réutilisation de composants de procédés. Toutefois, la réutilisation dans OPF nécessite beaucoup de travail vu qu'il s'agit d'une réutilisation de composants élémentaires (tâches, rôles, produits), et ne permet pas de tirer profit de la connaissance offerte par des patrons. De plus, l'approche n'offre pas une méthode détaillée de réutilisation (choix, création, adaptation de composants), et ne donne pas de moyen pour la restructuration de modèles de procédés existants.

3.1.7 La procédure de Gholami

En constatant l'état de l'art des patrons de procédés, Gholami *et al.* ont posé la question suivante: *Comment des ingénieurs de procédés peuvent ils extraire des patrons de procédés à partir des modèles de procédés, et les organiser de telle sorte à obtenir une connaissance distillée sur les modèles de procédés?* [GHO10]. Comme réponse, ils ont proposé leur procédure descriptive, qui a pour but d'extraire des patrons de procédés à partir des méthodes de développement, pour former une librairie de fragments de méthodes (method chunks).

Ces fragments seront par la suite assemblés pour former une méthode de développement personnalisée, selon les besoins d'un projet, ou pour améliorer une méthode existante. Les auteurs représentent leur procédure descriptive sous forme d'une carte, montrée par la figure I.8, où chaque intention de la carte correspond à une étape de la procédure. Les étapes se résument comme suit:

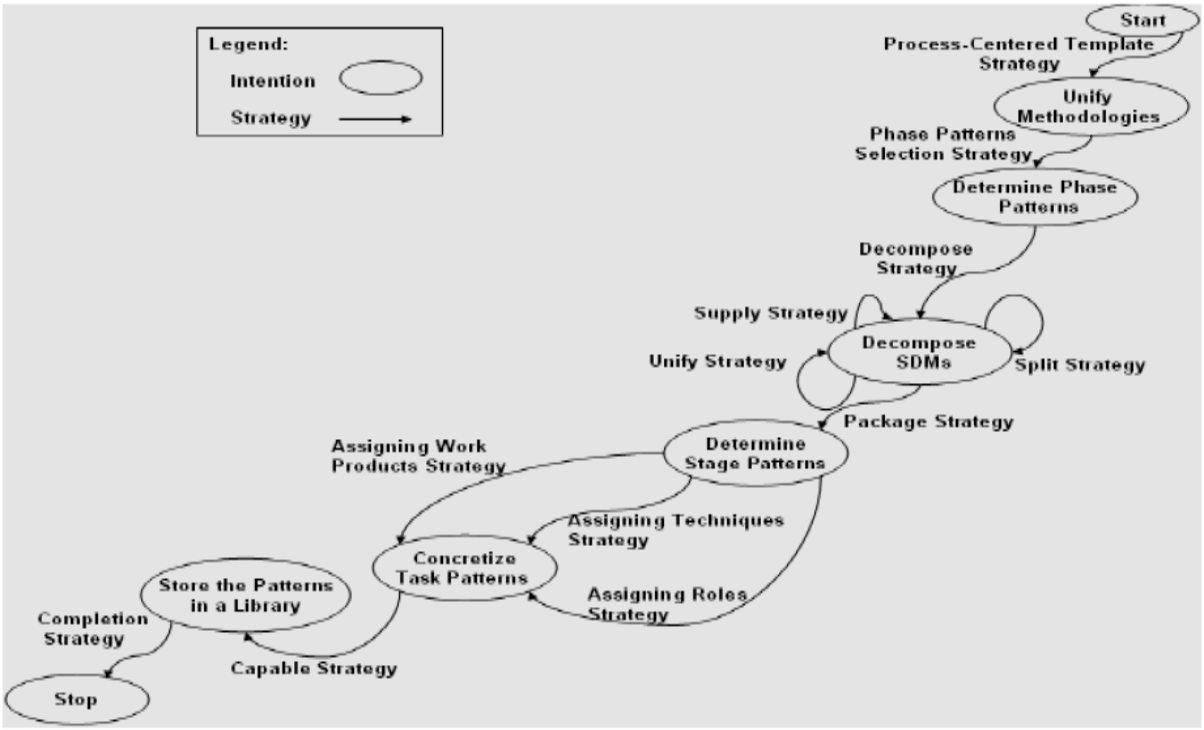


Figure I.8. Procédure d'extraction de patrons [GHO10]

Étape 1. *Unify Methodologies:* représente les modèles de procédés d'une manière unifiée [RAM08] en utilisant la stratégie *process-centered template*, pour faciliter la comparaison.

Étape 2. *Determine Phase Patterns:* utilise la stratégie *phase patterns selection* pour extraire les phases des modèles de procédés.

Étape 3. *Decompose SDMs:* utilise la stratégie *décompose* pour fragmenter un modèle de procédés, en activités candidates à former des patrons de tâche. Cette étape regroupe également les activités dans les patrons de phase correspondants.
Pour éliminer la redondance et améliorer la cohésion des activités, les stratégies suivantes sont utilisées dans cette phase selon le besoin.

- *Unify Strategy*: plusieurs activités peuvent avoir des noms différents, mais être identiques sémantiquement. Dans ce cas, l'unification des activités va conserver une seule activité, qui constituera un patron de tâche.
- *Supply Strategy*: si un certain nombre d'activités sont synonymes, mais que l'une d'elles est plus complète, alors cette stratégie va combiner ces activités pour construire une activité plus complète.
- *Split Strategy*: cette stratégie est pertinente lorsque certaines activités sont surdimensionnées pour être considérées comme des patrons. Par conséquent, ces activités seront décomposées en activités plus appropriées.

Étape 4. *Determeine Stage Patterns:* les activités ayant une proximité sémantique sont groupées pour former un patron d'étape, en utilisant la stratégie *Package*.

Étape 5. *Concretize Task Patterns:* chaque patron de tâche sera complété selon le formalisme de [GHO10]. Les stratégies *Assigning roles*, *Assigning work product* et *Assigning techniques* permettent de renseigner les différentes rubriques du patron.

Étape 6. *Store the Patterns in a Library:* la stratégie *Capable* suggère d'importer les patrons extrais dans un outil d'aide à l'ingénierie des méthodes, pour enrichir la librairie des patrons. Enfin, la stratégie *Completion* est utilisée lorsque tous les patrons sont importés dans l'outil.

La procédure de Gholami *et al.* est une procédure intéressante. Son inconvénient majeur est le fait qu'elle soit encore manuelle, et ne peut être opérée automatiquement. Les operateurs utilisés par cette procédure ne sont pas automatisables, mais reposent sur le raisonnement humain. Cependant, ce travail reste prometteur vu qu'une fois automatisé, il pourra être intégré comme plug-in aux plateformes d'ingénierie des procédés.

3.1.8 L'approche ASAP

ASAP (Automatic Structuring and Analysis of process Patterns) [JLA12a] implémente une approche sémantique pour l'unification des patrons de procédés. Les auteurs estiment que l'hétérogénéité des formalismes de patrons est un obstacle majeur dans la réutilisation de ces derniers. Dès lors, ASAP analyse et restructure la description des patrons, dans le but de faciliter et améliorer leur réutilisation dans l'ingénierie des logiciels.

ASAP consiste en un processus d'extraction d'informations à partir des patrons hétérogènes, et surtout de reconnaissance et d'annotation des rubriques pertinentes de chaque patron; de

telle sorte à ce que les patrons puissent être décrits de manière unifiée. Un mappage [JLA11] entre les formalismes de patrons s'avère nécessaire pour ASAP.

Ainsi, une description de patron sur six facettes a été adoptée [JLA12b]: (1) la facette d'identification encapsule les informations d'identification du patron, telles que nom, auteur, mots clés, classification (type, catégorie, niveau d'abstraction), origine, artefacts. (2) La facette d'information représente la partie principale du patron, et englobe le contexte, le problème et la solution. (3) La facette des relations indique comment le patron peut interagir avec d'autres patrons. (4) La facette d'assistance représente un support à l'utilisation du patron (utilisations connus, exemples, illustrations ...). (5) La facette d'évaluation fournit un feedback sur le patron (discussion, maturité …). (6) La facette de gestion fournit les informations générales sur le patron (version, date de création …).

ASAP est une approche linguistique appliquant une technique de NLP (Natural Language Processing), pour identifier les segments clés dans la description des patrons, les annoter et les restructurer en XML selon un format unifié. L'approche utilise l'outil d'extraction d'information de GATE (General Architecture for Text Engineering) [GAT16], en l'occurrence ANNIE (A Nearly-New Information Extraction system) [ANN16].

L'approche ASAP comprend deux principales phases, montrées par la figure I.9. Une première phase d'analyse, qui réalise les analyses lexicale, syntaxique et sémantique des différentes descriptions des patrons. Une seconde phase de structuration, qui converti les patrons analysés en patrons décrits de manière unifiée, selon le modèle adopté par l'approche. En résultat, un fichier XML est généré pour chaque patron fournit en entrée.

ASAP est un pas important vers l'élargissement de la réutilisation de patrons de procédés. Cependant, ce travail dans son présent état ne traite pas la réutilisation de patrons proprement dite; il n'offre ni outil de réutilisation ni méthode de modélisation de procédés à base de patrons.

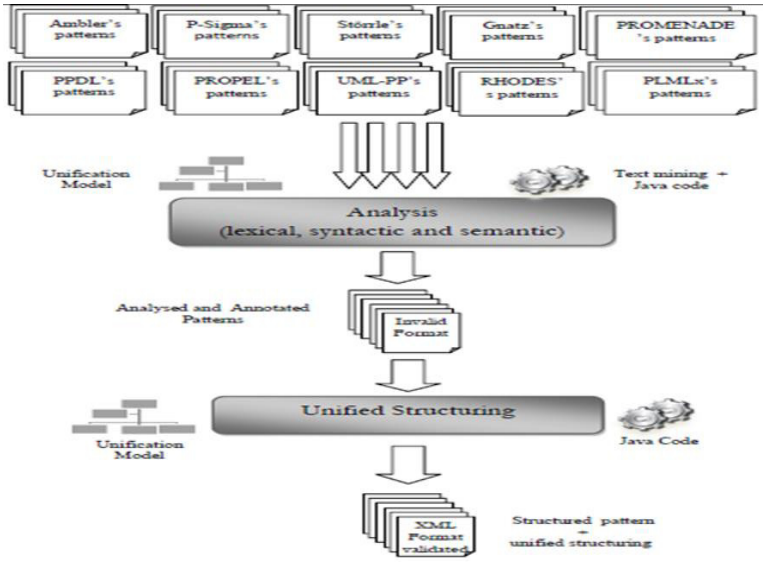


Figure I.9. L'approche ASAP [JLA12a]

3.1.9 L'approche de Wang

L'approche de Wang *et al.* [WAN10] vise la réutilisation de patrons sur des modèles de procédés existants. Dans leur méta modèle régissant les patrons (Figure I.10), Wang *et al.* considèrent la solution du patron comme un ensemble de nœuds exécutables *ExecutableNodes*, représentant les activités et leurs relations.

L'ensemble des nœuds exécutables est représenté à travers une arborescence, appelée *Process Pattern Structure Tree* (PPST). Chaque nœud exécutable est soit une activité simple *Activity*, soit une activité composée *StructedActivity*. Cette dernière agence ses activités subordonnées par le biais de cinq relations: *Sequence*, *Parallel*, *And*, *Choice* et *Cycle*.

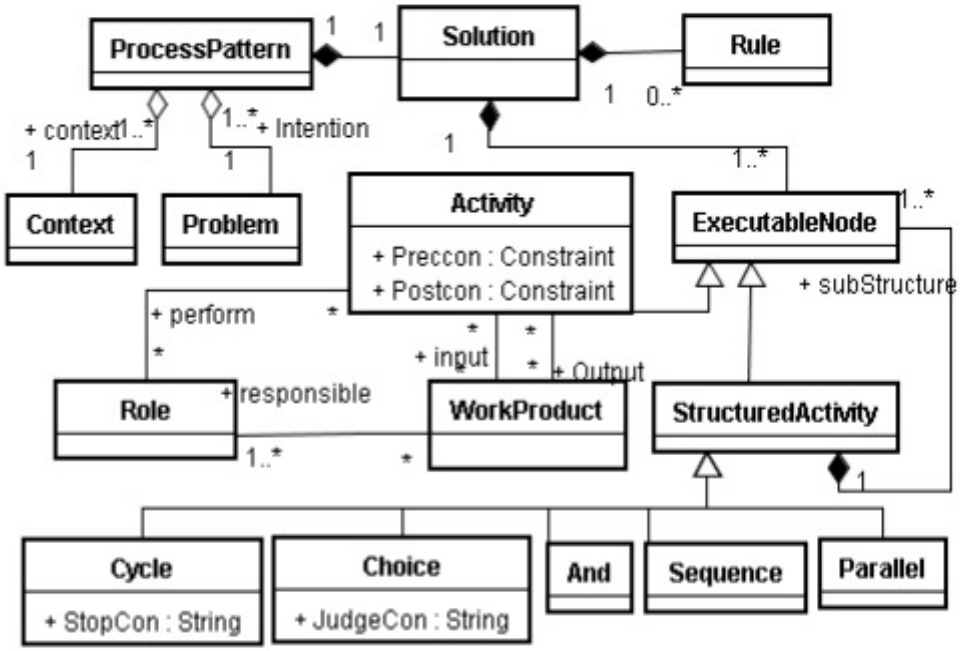


Figure I.10. Méta modèle de Wang *et al.* [HEE09]

En effet, l'application d'un patron sur un modèle de procédés existant signifie fusionner deux structures en une seule. Le modèle de procédés avant la fusion est appelé modèle source, alors que le modèle obtenu après application du patron est appelé modèle cible. Dans l'approche de Wang *et al.*, trois principes doivent être respectés. *Principe1*: s'assurer que toutes les contraintes du patron sont satisfaites dans le modèle cible. *Principe2*: s'assurer que le comportement du modèle source est préservé dans le modèle cible, tout en respectant le *Principe1*. *Principe3*: veiller à ce que des relations de précédence inutiles ne soient pas introduites dans le modèle cible.

La réutilisation d'un patron dans cette approche consiste en cinq phases. *Phase1*: transforme le modèle source en PST (*Process Structure Tree*). PST est une PPST qui ne contient pas la relation *And*. *Phase2*: compare la solution du patron avec le modèle source, en établissant une correspondance entre leurs éléments; le concepteur de procédés détermine à cet effet les comportements similaires à partir des deux espaces de noms. *Phase3*: fusionne la solution du

patron et le modèle source, en se basant sur leurs structures. Les activités et les relations du modèle source qui divergent du patron sont corrigées, et les autres informations sont conservées. En outre, les parties incomplètes du patron peuvent également être complétées par des informations incorporées à partir du modèle source. *Phase4*: la PST obtenue est simplifiée pour éliminer les éléments et relations redondants. *Phase5*: restaure la PST dans le modèle cible, et le convertir en un graphe de workflow.

L'approche de Wang *et al.* est très prometteuse, vu qu'elle cible la réutilisation de patrons sur des modèles de procédés existants. Cependant, on peut lui reprocher le fait que son méta modèle ne couvre pas toutes les relations possibles entre les éléments de procédés. Ainsi, ce méta modèle est inadéquat pour représenter différents patrons et modèles de procédés, qui peuvent être impliqués dans la réutilisation de patrons. De plus, l'approche ne fournit pas une procédure concrète pour transformer un modèle de procédés en une PST, et inversement.

Néanmoins, les inconvénients majeurs de ce travail restent: (a) l'absence d'un algorithme concret pour fusionner un patron avec un modèle de procédés. (b) *La phase3* qui favorise le patron lors de la fusion, mais qui ajoute cependant au modèle cible les éléments du modèle source qui ne sont pas dans la fusion (en plus des éléments du patron qui ne sont pas dans la fusion). De plus, cette phase ne fournit aucun moyen pour favoriser le modèle source, ni pour prendre juste les éléments commun au modèle source et au patron. (c) Les règles pour l'application de la *phase4* ne sont pas définies.

3.2 Synthèse des travaux de réutilisation

Une synthèse des travaux de recherche sur la réutilisation de patrons est donnée dans cette section, à travers la comparaison de ces travaux. À cet effet, un cadre de comparaison est établi, fournissant les différentes facettes de comparaison ainsi que leurs attributs.

3.2.1 Cadre de comparaison

Nous comparons dans la suite de ce chapitre les travaux de réutilisation de patrons. Nous nous appuyions pour cela sur un cadre de comparaison inspiré du cadre de [TRA07], lui même basé sur la classification de Prieto [PRI87], auquel nous avons ajouté un attribut. Cet attribut *Réutilisation pour modifier un modèle existant* est ajouté dans la facette de comparaison *Opérateurs de manipulation d'entités réutilisables*. Il signifie si oui ou non le travail comparé permet la réutilisation de patrons pour modifier des modèles existants.

Pour les facettes de comparaison choisies, les attributs ainsi que leurs valeurs possibles, nous donnons les explications dans ce qui suit.

a. Représentation des connaissances réutilisables de procédés

Expressivité: caractérise la capacité d'un formalisme à représenter la connaissance de procédés. Deux niveaux sont possibles.

- *Éléments de procédés*: un formalisme à ce niveau permet de représenter les éléments de procédés et leurs relations.

- *Éléments réutilisables de procédés*: un formalisme à ce niveau permet de représenter non seulement les éléments de procédés et leurs relations, mais aussi les éléments réutilisables ainsi que la manière de les réutiliser dans la description de procédés.

Formalisation: caractérise le degré de formalisation d'un formalisme, avec trois valeurs possibles.

- *Informel*: le formalisme est intuitivement défini.
- *Semi formel*: le formalisme dispose d'une syntaxe formelle.
- *Formel*: le formalisme dispose d'une syntaxe formelle et d'une sémantique formelle (exécutable).

b. Organisation des connaissances réutilisables de procédés

Type d'encapsulation des modules de connaissance: deux conteneurs de connaissance réutilisables sont possibles.

- *Composant de procédés*.
- *Patron de procédés*.

c. Opérateurs de manipulation d'entités réutilisables

Mécanisme de réutilisation: trois façons de réutiliser des connaissances de procédés sont identifiées.

- *Spécialisation*: les connaissances fournies sont adaptées par spécialisation pour le procédé en cours de modélisation.
- *Paramétrisation*: l'entité à réutiliser est associée à un ensemble de paramètres, et la réutilisation est réalisée par le choix de valeurs pour ces paramètres.
- *Composition*: les connaissances fournies sont associées les unes aux autres de façon cohérente, afin de former un (fragment de) procédé.

Réutilisation pour modifier un modèle existant: cet attribut n'est applicable que sur les travaux ayant: (a) la valeur *Patrons* pour l'attribut *Type d'encapsulation* de la facette *Organisation des connaissances de procédé réutilisables*, et (b) *Composition* ou *Paramétrisation* comme valeur ou parmi les valeurs de l'attribut *Mécanisme de réutilisation* de la facette *Opérateurs de manipulation d'entités réutilisables*.

- *Oui* si le travail concerné permet la réutilisation de patrons pour modifier des modèles de procédés existants.
- *Non* dans le cas contraire.

d. Guidage méthodologique

Type de guidage: reflète la nature du guidage proposé. Deux types sont possibles.

- *Conseils pour la réutilisation d'une entité*: décrit le contexte d'application d'une entité réutilisable et les situations recommandées pour la réutiliser.
- *Démarche de réutilisation*: propose une véritable démarche pour mettre en œuvre la réutilisation.

e. Outils de support

Catégorie: deux catégories d'outils supportant la réutilisation sont identifiées.

- *Bibliothèque*: fournit une collection d'entités réutilisables, gère la persistance de ces entités et offre des interfaces de recherche. La sélection d'entités réutilisables, leur adaptation et leur intégration dans un procédé sont quant à elles à la charge du concepteur.
- *Environnement de modélisation par la réutilisation*: fournit non seulement la gestion d'une collection d'entités réutilisables, mais aussi des fonctions pour sélectionner, adapter et appliquer ces entités réutilisables à un modèle de procédés.

Mode de support: reflète le niveau d'automatisation des outils de support. Trois degrés d'automatisation sont possibles.

- *Manuel*: l'outil permet d'accéder aux entités réutilisables, mais leur réutilisation est entièrement à la charge de l'utilisateur.
- *Semi automatique*: l'outil fournit des fonctions assistant l'application d'entités réutilisables, mais nécessite cependant l'intervention de l'utilisateur.
- *Automatique*: l'application d'entités réutilisables est entièrement automatique, et peut donc se faire sans intervention de l'utilisateur.

3.2.2 Comparaison des approches

Nous comparons dans la table I.1 les travaux de réutilisation de patrons. Une synthèse de ces travaux montre que la problématique de réutilisation de patrons, pour modéliser des procédés logiciels, prend deux aspects différents. D'une part la gestion des patrons qui serviront à modéliser des procédés, qui inclut: la création de patrons, la modification de patrons, la recherche de patrons selon des critères, la sélection d'un patron à réutiliser, ainsi que la détection des patrons liés (dépendants). D'une autre part, la gestion des modèles de procédés créés en réutilisant des patrons, qui inclut: la création d'un modèle en réutilisant un patron, la création d'un modèle dont une partie seulement est conçue en réutilisant un patron, et la modification d'un modèle ou d'une partie d'un modèle en réutilisant un patron.

Ces deux aspects de la problématique ne sont pas indépendants. Ils interagissent, vu qu'un modèle de procédés peut être conçu via des patrons, et que sa conception nécessitera ainsi de faire appelle à la gestion des patrons. De même, un patron contient un modèle de procédé (dans sa rubrique Solution), et donc fera appelle à la gestion des modèles de procédés.

L'étude bibliographique effectuée nous a permis de constater que les travaux de réutilisation de patrons n'en sont qu'à leur début. Les démarches proposées sont souvent informelles et présentées avec un niveau de détail insuffisant. Pour la plupart de ces démarches, elles ne permettent pas un guidage fin des activités de modélisation de procédés via des patrons. De plus, elles ne tiennent pas suffisamment compte de l'adaptation de la solution modélisée. Par conséquent, les patrons sont encore peu réutilisés à cause de ces lacunes, et surtout à cause du manque d'outils automatiques de réutilisation.

De tels outils doivent permettre de sélectionner les patrons en fonction du problème à résoudre, de les réutiliser et de les adapter éventuellement en fonction du contexte. Aussi, ces outils doivent permettre de détecter les patrons liés (dépendants), pour pouvoir les combiner en solutions complexes si besoin y est.

L'étude bibliographique nous a permis en plus de constater qu'une manière d'appliquer les patrons est encore délaissée: il s'agit de la réutilisation de patrons pour modifier un modèle ou un fragment de modèle de procédés. En effet, restructurer ou enrichir un modèle pour satisfaire certaines contraintes s'avère de plus en plus nécessaire. La réutilisation de patrons doit permettre de répondre à ce besoin, en offrant un mécanisme de fusion et d'adaptation qui agit simultanément sur le modèle de procédés existant et sur le patron à réutiliser.

Travaux		Promenade	Rhodes	Approche d'Iida	SPEM2	Approche deTran	Approche OPF	Procedure de Gholami	Approche ASAP	Approche de Wang
Facettes	*Attributs*									
Représentation des connaissances	**Expressivité**	éléments réutilisables	éléments réutilisables	éléments réutilisables	éléments réutilisables	éléments réutilisables	éléments de procédé	éléments de procédé	éléments de procédé	éléments de procédé
	Formalisation	formelle	formelle	semi formelle	formelle	formelle	semi formelle	informelle	informelle	semi formelle
Organisation des connaissances	**Type d'encapsulation**	patrons	composant	composant - patron	composant	patron	composant	patron	patron	patron
Opérateurs de manipulation	**Mécanisme de réutilisation**	Spécialisation, Composition, et Paramétrisation	Spécialisation et Composition	Composition	Spécialisation	Spécialisation, Composition, et Paramétrisation	Spécialisation et Composition	NA	NA	Composition
	Réutilisation pour modifier	NON	NA	NA	NA	OUI	NA	NA	NA	OUI (n'est pas concrète : ne fournit pas d'algorithme)
Guidage méthodologique	**Type de guidage**	NA	Démarche	Conseils	NA	Démarche	Conseils	NA	NA	Démarche (incomplete)
Outils de support	**Catégorie**	Gestion de bibliothèque	Gestion de bibliothèque	Gestion de bibliothèque	Environnement de modélisation	Environnement de modélisation	Gestion de bibliothèque	Gestion de bibliothèque	Gestion de bibliothèque	Environement (en cours de developpement)
	Mode de support	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Semi-automatique	Manuel	Manuel	NA

NA : non applicable.

Table I.1. Synthèse des travaux de réutilisation de patrons

3.3 Principaux travaux de fusion de modèles de procédés

Une forme particulière de réutilisation de patrons, à savoir la réutilisation de patrons sur des modèles de procédés déjà existants, exécute obligatoirement une procédure de fusion entre le patron à réutilisé et le modèle existant. Cela nous a conduits à mener une revue de la littérature, relativement au domaine de la fusion des modèles.

Nous avons constaté qu'il existe une panoplie de techniques et d'outils pour la différenciation et la fusion de modèles, qui font partie d'une large littérature sur la gestion des versions des modèles [BRO12] [CVS16] [DAM16]. Les défis dans ce domaine sont: l'identification des changements entre les versions d'un modèle, et la fusion de ces modifications pour créer une version consolidée [RUB13] [KES13] [DAM16].

Cependant, les techniques utilisées dans ces travaux ont uniquement porté sur la détection des incohérences entre versions [SAB08] [BRO12a], et il y a eu par contre très peu de travaux qui résolvent ces incohérences [DAM16]. Ces travaux de résolution d'incohérences peuvent être classés en deux principales catégories [DAM16]: la première catégorie confie la résolution des incohérences à l'utilisateur, tel que [EMF16]; ou guide l'utilisateur et lui suggère des résolutions possibles, tel que [GER13]. La deuxième catégorie tente d'automatiser la procédure de résolution des incohérences; certains travaux dans cette catégorie permettent de différer la résolution, tel que [WIE12] par exemple.

Un travail majeur dans le domaine de la fusion des modèles est l'approche présentée dans [DAM14], résumée dans la figure I.11. Cette approche fusionne automatiquement les versions d'un modèle, si elles sont exemptes d'incohérences. Cependant, la résolution d'une incohérence peut créer d'autres incohérences (incohérences en cascade) [DAM16], et le nombre d'alternatives de résolution augmente avec la complexité de la règle de cohérence et avec le nombre d'éléments du modèle impliqués dans l'incohérence [RED12] [DAM16]. Pour ces raisons, l'approche [DAM14] notifie les concepteurs du modèle au cas où il existe des incohérences, et les aide à travers un calcul de compromis dans leur résolution.

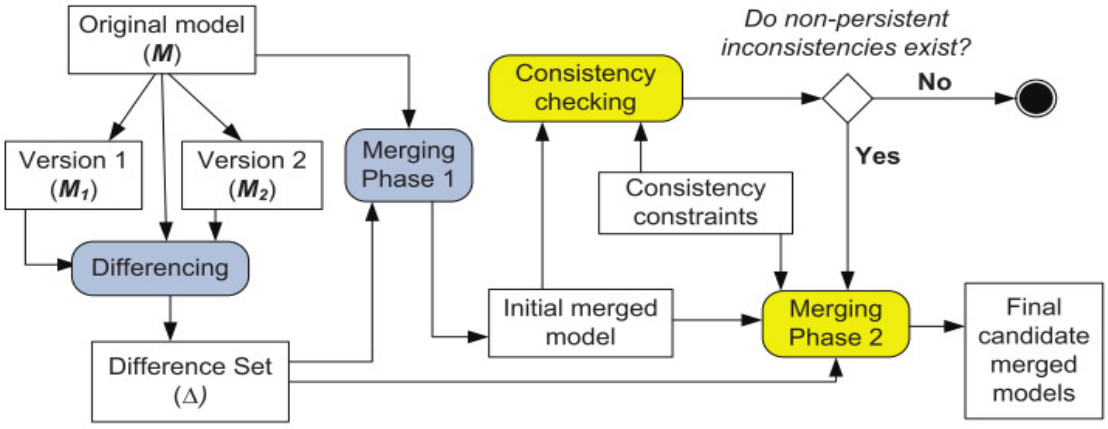


Figure I.11. La procédure de fusion [DAM14]

Le principe de l'approche [DAM14] est le suivant: elle considère le modèle original (M) et ses versions (M1) et (M2), pour calculer l'ensemble (Δ) des différences entre versions. Elle procède ensuite à la fusion en utilisant M et Δ dans une première phase. Le modèle fusionné est après cela vérifié, en utilisant les contraintes de cohérence définies par l'utilisateur. De telles contraintes spécifient la consistance syntaxique et sémantique requise pour un modèle.
Les incohérences (si elles existent) sont détectées en utilisant un vérificateur incrémental d'incohérences. Ce vérificateur identifie les éléments du résultat qui ne respectent pas les contraintes de cohérence, pour procéder ensuite à la seconde phase qui est la prise en charge des incohérences détectées.

3.4 Synthèse des travaux de fusion

Les approches proposées dans la littérature relativement à la fusion de modèles, se limitent à la fusion de versions issues d'un même modèle initial. De telles approches ne peuvent donc pas être appliquées pour fusionner deux modèles indépendants.

L'approche [DAM14] n'est pas dédiée aux modèles de procédés logiciels, mais est plutôt applicable (tel que stipulé par ses auteurs) à tous les langages de modélisation, tant que ces derniers respectent un méta modèle bien défini. Ce fait peut sembler être un avantage, mais en réalité il ne l'est pas. Ce fait sous entend qu'il faut définir pour chaque langage de modélisation, un ensemble de règles de cohérence et de bonne formalisation (well-formedness), qui seront vérifiées lors de la fusion. Sachant que l'approche [DAM14] ne permet pas la génération automatique d'un tel ensemble de règles, elle devient ainsi contraignante et réalisable uniquement pour un langage de modélisation dont l'ensemble des règles (de cohérence et de bonne formalisation) est défini.

4. Conclusion

Nous avons présenté dans ce chapitre un état de l'art relativement aux domaines des modèles de procédés logiciels, des patrons de procédés et de la réutilisation de ces patrons. La modélisation de procédés est une discipline du génie logiciel qui vise la maîtrise des projets de développement, en leur fournissant les moyens de modéliser les procédés de support [DIA11]. Quant aux patrons, ils fournissent des solutions prouvées et destinées à la réutilisation dans la modélisation de procédés. L'étude bibliographique que nous avons menée s'est focalisée essentiellement sur cette réutilisation.

Au regard de cette étude, deux cycles de réutilisations sont mis en évidence: un cycle de développement de patrons pour la réutilisation (dans la modélisation de procédés); et un cycle de développement de procédés par réutilisation de patrons. L'étude bibliographique révèle également que les démarches proposées jusque là ne permettent pas un guidage fin des activités de modélisation de procédés via des patrons. Elle révèle aussi que la réutilisation de patrons pour modifier un modèle ou un fragment de modèle de procédés est encore négligée. En effet, cette forme de réutilisation se base sur un mécanisme de fusion de modèles, dont une partie de notre étude bibliographique a été consacrée pour son étude.

CHAPITRE II

REUTILISATION DE PATRONS SUR MODELES DE PROCEDES EXISTANTS

Nous abordons dans ce chapitre une problèmatique particuliere de la réutilisation de patrons. Il s'agit de la réutilisation pour modifier des modèles de procédés déjà existants (cf. § 1). À cet effet, nous commençons par présenter le méta modèle que nous avons adopté (cf. § 2), notamment ses volets structurel et celui relatif à la réutilisation. Nous aborderons après cela les conflits (cf. § 3), et leurs différents niveaux ainsi que les incohérences (cf. § 4).

1. La modification d'un modèle de procédés

La réutilisation de patrons pour modéliser des procédés logiciels est une approche prometteuse. Elle consisite en la création de modèles de procédés, ou de fragments de modèles de procédés, en se basant sur des patrons. Par exemple, si un patron de procédés répond à la question: quelle est la définition de P? alors il pourra être employé en tant que modèle (cet emploi est appelé réutilisation de patron) pour produire le contenu de P, sachant que P peut être un élément de procédés ou un modèle de procédés dans son intégralité. En effet, il s'agit ici de la création de P en réutilisant le patron; le modèle de procédés capturé dans ce patron sera imité pour générer la description de P.

La réutilisation de patrons pour modéliser des procédés logiciels consiste également en la modification de modèles de procédés existants, ou de fragments de modèles, en se basant sur des patrons. En effet, les concepteurs de procédés doivent parfois restructurer ou enrichir un modèle de procédés ou un fragment de modèle, pour accroître son efficacité ou pour satisfaire certaines contraintes. Plus précisément, ils peuvent vouloir appliquer de nouvelles relations sur un groupe d'éléments de procédés existants, ou ajouter de nouvelles informations à ce groupe. Dans de tels cas, une manière spécifique de réutiliser les patrons de procédés devient nécessaire, il s'agit de la réutilisation de patrons pour modifier des modèles de procédés déja existants.

Par exemple, si un patron décrivant un modèle P constitué des éléments $\{p_1, p_2, ... p_N\}$, alors il pourra être appliqué à un modèle de procédés E déja existant, constitué des éléments $\{e_1, e_2, ... e_M\}$, pour restructurer le modèle E ou pour lui ajouter de nouvelles informations (cette application est aussi appelée réutilisation de patron). Donc, ici il ne s'agit pas de création, puisque le modèle E est déja existant. Dans cette réutilisation, une correspondance 0..1 à 0..1, avec au moins une relation 1 à 1, sera établie entre les p_i et les e_i de meme nature (un élement p_i ne peut correspondre qu'à un seul élément e_i). Par conséquent, les relations définies sur p_i et e_i seront fusionnées. Pourtant, cette forme de réutilisation est encore appliquée de façon limitée en raison du manque d'outils de support. Il devient ainsi interessant de proposer des approches et des environnements, pour réutiliser de manière systématique des patrons dans la modélisation de procédés.

La réutilisation de patrons pour modifier un modèle de procédés existant a été abordée dans [IID99], mais l'auteur n'a défini aucun mécanisme pour supporter cette forme de réutilisation de patrons. Alors, l'idée a été reprise par [TRA07a] qui a proposé comme mécanisme la relation *ProcessPatternApplying*. Cette relation est définie entre un patron et un modèle de procédés. Elle lie chaque élément du patron défini comme paramètre formel, avec un élément du modèle défini comme paramètre effectif. Pourtant, la relation *ProcessPatternApplying* peut générer un résultat incohérent.

En effet, cette forme de réutilisation necessite une procédure de fusion entre le patron (à vrai dire, la solution de ce dernier qui est elle même un modèle de procédés) et le modèle à modifier. La fusion consiste en la production d'un modèle de procédés résultant à partir des

éléments et des relations des deux modèles fusionnés. Le problème se pose alors lorsque dans ce résultat deux relations ou plus sont contradictoires; le modèle résultant présentera ainsi un ou plusieurs conflits et sera par conséquent inutilisable.

Le défit pour cette forme de réutilisation étant donc de pouvoir fusionner des modèles de procédés, tout en garantissant un résultat exempt de conflits. Il devient dès lors necessaire d'étudier les conflits éventuels, pour pouvoir les détécter lors d'une procédure de fusion et prendre en conséquence les mesures correctives adequates.

Ainsi, nous souhaitons proposer une approche de réutilisation de patrons, pour modifier des modèles de procédés existants.

2. Le méta modèle adopté

Avant de poursuivre le travail sur la problématique de réutilisation de patrons sur des modèles de procédés existants, la structure de ces patrons et modèles doit être fixée. À cet effet, l'adoption d'un méta modèle pour régir la sémantique statique des modèles et patrons s'impose à ce point du parcours.

Nous commençons par exposer dans ce qui suit notre choix d'un tel méta modèle. Ensuite, nous présentons en détail les différents constituants de ce méta modèle. Nous terminerons cette section avec un exemple illustratif d'un modèle de procédés régi par le méta modèle adopté.

Cependant, nous ne nous intéressons dans ce travail qu'à l'attribut Solution des patrons, qui rentre en jeux de manière directe dans la réutilisation et qui n'est autre qu'un modèle de procédés. Nous omettons ainsi d'aborder entièrement la représentation des patrons dans le présent travail. En effet, la représentation d'un patron exige d'une part un formalisme pour structurer son contenu, et d'une autre part un langage et donc un méta modèle pour exprimer sa solution. Le choix d'un formalisme reste au dela du cadre du présent travail, et ne sera donc pas abordé dans ce document.

2.1 Choix du méta modèle

Le choix de notre méta modèle à été fait après plusieurs observations des méta modèles présentés dans la littérature. En effet, plusieurs travaux ont depuis longtemps classifié les différents constituants des modèles de procédés, à l'instar de [FUG00], et ont montré que les plus importants sont les suivants.

- Activité: c'est une étape de procédés opérant sur des artefacts, et couplée à un agent humain ou à un outil de production. Elle peut être à différents niveaux d'abstraction, c'est à dire qu'elle peut être décomposée. D'un méta modèle à un autre, les synonymes d'activité sont Tâche, Étape, Définition de travail, etc. Cependant, nous pouvons avoir tous ou partie de ces synonymes comme concepts d'un même méta modèle.

- Artefact: c'est un produit créé ou modifié pendant un procédé, soit comme un résultat requis soit pour faciliter le procédé. C'est l'entrée et la sortie des activités. Un artefact peut être simple ou composite, et peut avoir des dépendances avec d'autres artefacts. Les synonymes d'artefact sont Produit [CON99], WorkProduct [ISO06] et Resource [CAS00].

- Rôle: c'est une définition des droits (autorisations), des obligations et des responsabilités de l'agent impliqué dans une activité logicielle. Un rôle est un concept statique qui peut être joué par plusieurs agents; et inversement, un agent peut jouer plusieurs rôles.

Néanmoins, plusieurs méta modèles présentés dans la littérature sont trop complexes pour la compréhension et l'utilisation [BEN07]. En effet, parmi les critères d'un bon méta modèle se trouve l'expressivité (expressiveness [BEN07]), qui signifie que le méta modèle doit être assez riche en termes de notions, pour pouvoir exprimer toutes les informations qu'il peut être nécessaire de véhiculer à travers un modèle de procédés. Ainsi, plusieurs auteurs de méta modèles ont tendance à encombrer leur méta modèles par des notions non très courantes et parfois même ambigües.

Cependant, l'expressivité n'est pas le seul critère d'un bon méta modèle. Il y a aussi la compréhensibilité (understandability [BEN07]), qui signifie que les notions du méta modèles doivent êtres facilement et directement compréhensible (assez communes), et surtout sans ambigüité. Il y a également le critère d'abstraction (abstraction [BEN07]), qui est un mécanisme permettant de se focaliser sur l'aspect le plus important d'un système, et de cacher les détails non pertinents. Ce mécanisme est important dans la modélisation de procédés, parce qu'il aide à maitriser la complexité du procédé, en permettant au concepteur et à l'utilisateur de se focaliser sur ce qui est important. Les notions du méta modèles doivent aussi êtres assez abstraites, et doivent se focaliser sur les informations les plus importantes d'un modèle en omettant les détails.

Pour satisfaire ces critères, nous avons opté pour un méta modèle simple, facile à comprendre et à utiliser, et se limitant aux éléments de procédés les plus importants cités ci avant. Bien évidement, le méta modèle choisi contient aussi les relations nécessaires pour lier ces éléments de procédés. En effet, le choix d'un méta modèle basé sur des notions communément adoptées le rend plus facile à utiliser, et exploitable à moindre coût. Les concepteurs de procédés n'auront donc pas à apprendre un nouveau langage, et pourront ainsi tirer parti des outils de modélisation existants [BEN07].

2.2 Éléments et relations de procédés et de contrôle

La structure choisie pour représenter les modèles de procédés est composée de trois catégories de concepts: (a) les éléments de procédés, (b) les relations de procédés, ainsi que (c) les éléments et relations de contrôle. Nous détaillerons chacune de ces catégories dans ce qui suit.

2.2.1 Éléments de procédés

Il s'agit de trois concepts de base, permettant de constituer un modèle de procédés. Ces concepts sont définis comme suit.

- Tâche (*Task*): c'est une unité de travail contrôlée et réalisée dans le but de créer ou de modifier un produit. Ce concept remplace les concepts de *WorkDefinition* et *Activity* de SPEM [TRA07].

- Rôle (*Role*): c'est un concept décrivant un ensemble de responsabilités et de compétences nécessaires pour effectuer une activité de développement. Ce concept remplace les concepts de *ProcessPerformer* et *ProcessRole* de SPEM [TRA07].

- Produit (*Product*): c'est un artefact créé, consommé, ou modifié durant le procédés de développement. Il remplace le concept de *WorkProduct* de SPEM [TRA07].

2.2.2 Relations de procédés

Il s'agit de sept liens, permettant d'associer des éléments de procédés. Ces liens sont définis comme suit.

- Agrégation (*Aggregation*): c'est une relation de contenance entre deux éléments de procédés de même nature [TRA07].

- Spécialisation (*Refinement*): c'est une relation de spécialisation entre deux éléments de procédés de même nature, où l'élément spécialisé possède toujours les mêmes propriétés que l'élément générique, en plus de ses propriétés spécifiques [TRA07].

- Précédence (*TaskPrecedence*): cette relation relie deux tâches, et exprime l'ordre de leurs réalisations par le biais d'un attribut *kind*. Cet attribut peut prendre l'une des valeurs: *FinishStart* (la tâche successeur doit attendre la terminaison de la tâche prédécesseur pour commencer), *FinishFinish* (le successeur doit attendre la terminaison du prédécesseur pour terminer), *StartFinish* (le successeur doit attendre le début du prédécesseur pour terminer) et *StartStart* (le successeur doit attendre le début du prédécesseur pour commencer) [TRA07].

- Réalisation de tâche (*TaskPerformance*): cette relation exprime la participation d'un rôle à la réalisation d'une tâche, soit en effectuant cette tâche (en étant *Performer*) soit en assistant sa réalisation (en étant *Assistant*) [TRA07].

- Paramètre (*TaskParameter*): cette relation relie une tâche à un produit pour exprimer, à travers l'attribut *direction*, le fait que ce produit est soit: requis par la tâche (*IN*), fourni par la tâche (*OUT*), ou mis à jour par la tâche (*IN-OUT*) [TRA07].

- Impacte de produits (*ProductImpact*): cette relation lie deux produits, pour indiquer que la modification de l'un entraine la modification de l'autre [TRA07].

- Responsabilité de produit (*ProductResponsability*): cette relation établie entre un rôle et un produit signifie que le rôle est responsable du produit [TRA07].

2.2.3 Éléments et relations de contrôle

Les éléments de contrôle ont pour but de faciliter la lecture du modèle de procédés, ou de contrôler certaines de ses parties. Dans notre méta modèle, les éléments de contrôle donnés dans ce qui suit sont inspirés de ceux d'*UML 2.5 SuperStructure* [UML15]. La table II.1 résume les contraintes sur ces éléments de contrôle.

- Nœud initial (*InitialActivity*): indique le début du modèle de procédés. Il ne peut être précédé par aucun élément, et est suivi par un seul élément de type *Task* ou *Choice*. Dans un modèle de procédés, il ne peut y avoir qu'un seul nœud initial.

- Nœud final (*FinalActivity*): indique la fin du modèle de procédés. Il ne peut être suivi par aucun élément, et est précédé par un ou plusieurs éléments de type *Task* ou *Choice*. Dans un modèle de procédés, il peut y avoir un ou plusieurs nœuds finaux.

- Choix (*Choice*): permet de donner plusieurs possibilités de sortie après l'évaluation d'une condition. Le choix possède un lien entrant le liant à un nœud initial, à une tâche ou bien à un autre choix. Il possède au moins deux liens sortants, le liant à un nœud final, à une tâche ou bien à un autre choix; ces liens sont accompagnés de conditions de garde pour conditionner le choix.

Éléments de contrôle	Nombre d'occurrences	Éléments précédents	Éléments suivants
InitialActivity	Un	Aucun	Un élément de type *Task* ou *Choice*
FinalActivity	Un ou plusieurs	Au moins un élément de type *Task* ou *Choice*	Aucun
Choice	Aucun, un ou plusieurs	Un élément de type *InitialActivity*, ou bien un ou plusieurs éléments de type *Task* ou *Choice*	Aucun, un ou plusieurs éléments de type *InitialActivity*, *Task* ou *Choice*

Table II.1. Contraintes sur les éléments de contrôle

Les relations de contrôle quant à elles, sont des liens reliant soit des éléments de contrôle, soit un élément de procédés avec un élément de contrôle. Dans notre méta modèle, une seule relation de contrôle est définie; il s'agit de la relation *Link*. La table II.2 récapitule les contraintes de contrôle sur *Link*.

- Lien (*Link*) est une relation de contrôle permettant d'introduire éventuellement une condition de garde, de telle sorte que si cette condition est évaluée à vrai, alors le lien sera franchi en direction de l'élément suivant dans l'enchainement du modèle de procédés.

Relations de contrôle	Nombre d'occurrences	Éléments précédents	Éléments suivants
Link	Deux ou plusieurs	*InitialActivity*, *Task* ou *Choice*	*FinalActivity*, *Task* ou *Choice*

Table II.2. Contraintes sur *Link*

2.3 Volet structurel du méta modèle

Notre méta modèle peut être vu comme constitué de deux volets: (a) un volet structurel relatif aussi bien aux modèles de procédés qu'à la rubrique Solution des patrons (étant elle même un modèle de procédés). (b) Un volet relatif à la réutilisation de patrons. Nous aborderons chacun de ces volets dans ce qui suit.

Le volet structurel de notre méta modèle, présenté à travers le diagramme de classes de la figure II.1, régie les modèles de procédés. Ses deux majeurs constituants sont les nœuds et les arêtes, représentés par les classes *Node* et *Edge*. Ces classes composent un modèle de procédés, lui même représenté par la classe *ProcessModel*.

Les nœuds du modèle de procédés sont les éléments de procédés *ProcessElement* (*Task*, *Role* et *Product)*, ainsi que les éléments de contrôle *ControlNode* (*InitialActivity*, *FinalActivity* et *Choice)*. Les arêtes sont les relations entre les éléments de procédés *ProcessRelation* (*Aggregation*, *Refinement*, *TaskPrecedence*, *TaskPerformance*, *TaskParameter*, *ProductImpact*, *ProductResponsability)*, ainsi que les relations de contrôle *ControlFlow* (*Link)*.

Link est une relation liant deux éléments de contrôle ou un élément de contrôle et une tâche. La relation *Aggregation* lie deux éléments de procédés de même nature, dont l'un est considéré comme *Aggregate* et l'autre comme *Component*. La relation *Refinement* lie également deux éléments de même nature, dont l'un est considéré comme *Original Element* et l'autre comme *Refined Element*. La relation *TaskPrecedence* quant à elle lie deux tâches, dont l'une est considérée comme *Predecessor* et l'autre comme *Successor*. *ProductImpact* est une relation liant deux produits, où l'un est considéré comme *Impacting Element* et l'autre comme *Impacted Element*. *TaskPerformance* est une relation liant une tâche *Performed Task* à un rôle *Performer*. Cette relation dispose d'un attribut dont la valeur est *Performer* ou *Assistant*. La relation *TaskParameter* lie une tâche à un produit considéré comme son paramètre. Enfin, la relation *ProductResposibility* lie un produit à son rôle responsable.

2.4 Volet relatif à la réutilisation de patrons

Le second volet de notre méta modèle, régissant les relations entre modèles et patrons, est donné à travers le diagramme de classes de la figure II.2. Ce volet est constitué principalement des classes: modèle de procédés *ProcessModel*, patron *ProcessPattern*, opérateur de réutilisation *ReuseOperator*, et substitution de paramètres *Substitution* elle même composée d'éléments de procédés *ProcessElement*. Les deux concepts *ReuseOperator* et *Substitution* seront définis dans le chapitre III.

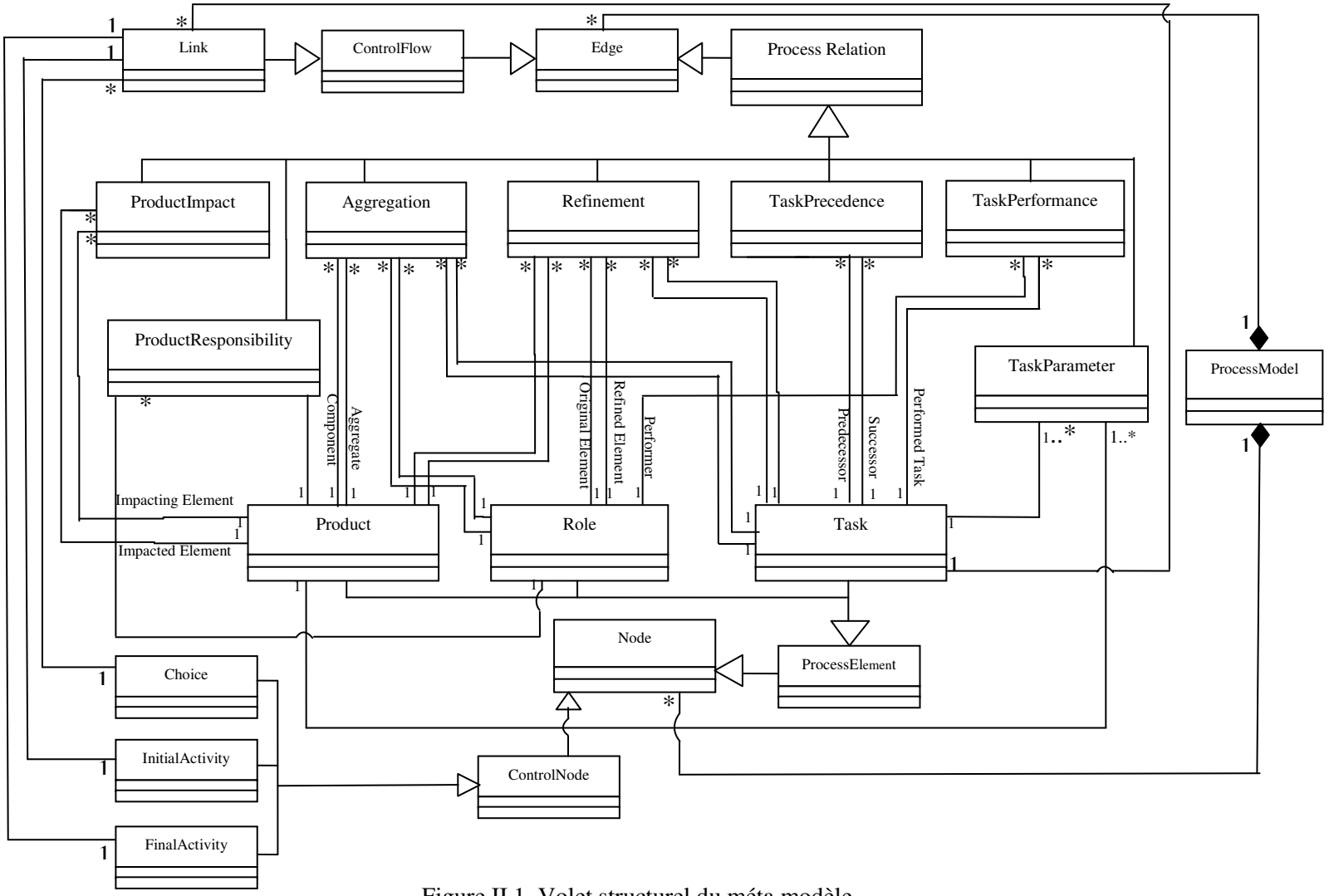


Figure II.1. Volet structurel du méta modèle

Notons que pour la classe *ProcessPattern*, nous ne montrons que l'attribut Solution, qui rentre en jeu de façon principale dans la réutilisation. La figure II.2 montre qu'un patron est composé d'un modèle de procédés, jouant le rôle de *Solution*. Cette figure montre également que l'opérateur *ReuseOperator* fait appel à une instance de *ProcessPattern* représentant le patron à réutiliser, et à une instance de *ProcessModel* représentant le modèle à modifier; et que l'opérateur est composé d'au moin une instance de *Substitution*.

En effet, nous imposons au moins une substitution de paramètres à l'opérateur, pour qu'on puisse parler réellement de fusion entre un patron et un modèle de procédés. Une substitution, montrée par la figure II.2, est constituée de deux éléments de procédés, l'un jouant le rôle de paramètre formel et provenant de la solution du patron, et l'autre jouant le rôle de paramètre effectif et provenant du modèle à modifier. Le paramètre effectif remplacera le paramètre formel, et de cette manière, une correspondance sera établie entre le patron et le modèle pour concrétiser leur fusion.

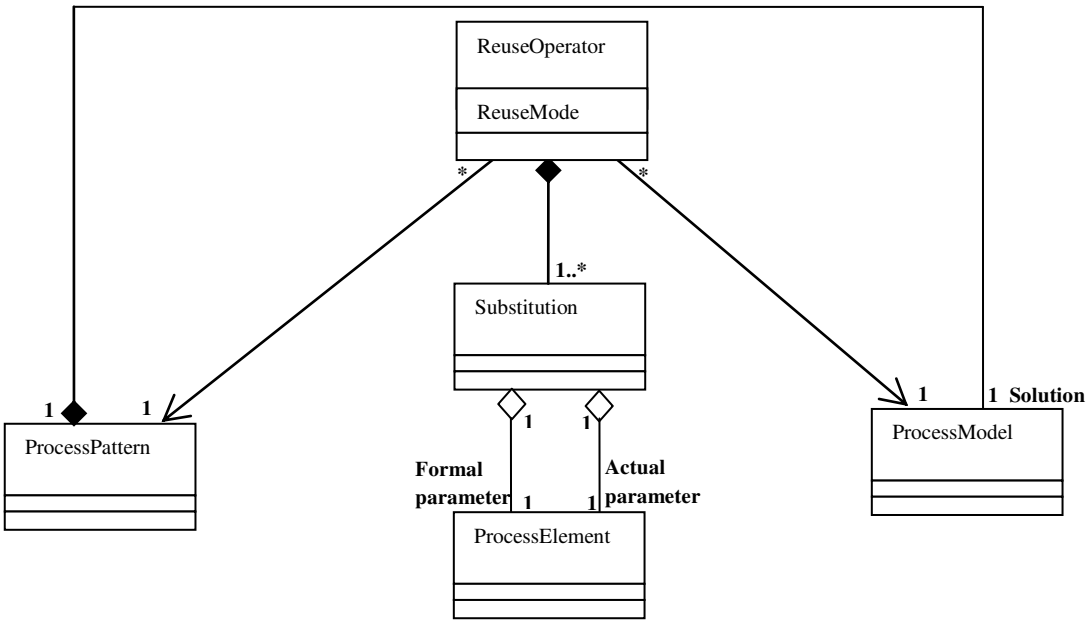


Figure II.2. Volet du méta modèle régissant les relations entre modèles et patrons

2.5 Exemple illustratif d'un modèle conforme au méta modèle

Dans cette section, nous avons choisi de décrire un exemple simpliste pour illustrer les concepts du méta modèle. L'exemple montré par la figure II.3, "Le monde réel du développement" [AOU12], est un modèle de procédés qui consiste en un ensemble de tâches (Analyse des besoins, Conception, ...) définies à travers le concept de *Task*.

Ces tâches sont ordonnées par le biais des relations de précédences *TaskPrecedence*, qui les relient en précisant la nature de la précédence (FS: *FinishStart*, FF: *FinishFinish*, ...). Les nœuds de début et de fin du modèle, I et F respectivement de types *InitialActivity* et

FinalActivity, accroissent la lisibilité du modèle. Ces nœuds de contrôle sont liés aux tâches à travers des liens de type *Link.*

Plusieurs rôles participent dans ce modèle de procédés (Analyste, Concepteur, ...). Ils sont définis à travers la notion de *Role*, et sont liés aux tâches par des relations de type *TaskPerformance* dont la nature est pour cet exemple *Performer* (P), pour signifier que le rôle est responsable de la réalisation de la tâche.

Différents produits sont manipulés à travers ce modèle de procédés (Cahier des charges, Rapport de conception, ...), et sont définis par la notion de *Product*. Ces produits sont associés aux tâches à travers les relations *TaskParameter*, qui précisent le sens du paramètre par rapport à la tâche (IN ou OUT dans le présent exemple).

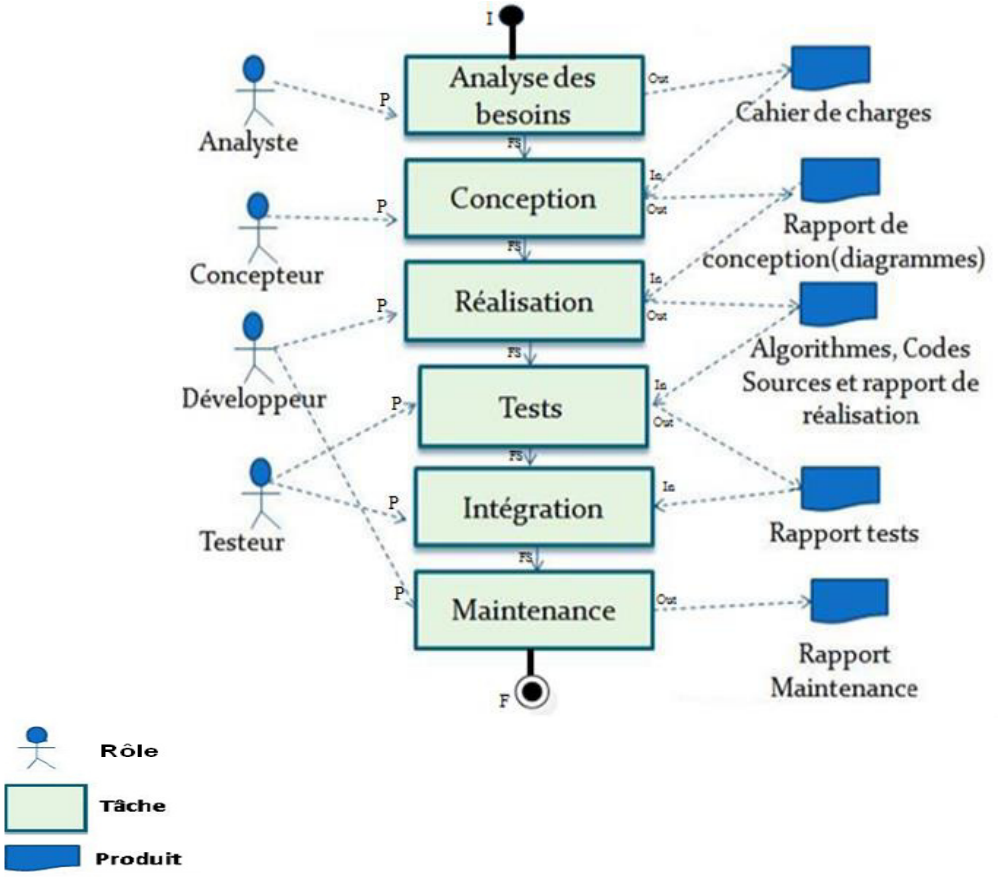


Figure II.3. « Le monde réel du développement » [AOU12]

3. Les conflits de la réutilisation de patrons pour modifier des modèles de procédés

Nous consacrons cette section à la présentation des conflits de la réutilisation de patrons. Nous commençons par définir la notion de conflit, et donner un exemple pour l'illustrer. Puis, nous abordons les modes de réutilisation, et leur influence sur la gestion des conflits.

3.1 Définition d'un conflit

Il est important de définir la notion de conflits dans notre contexte, avant d'aller plus loin dans la thèse. Nous avons donc choisi dans cette section de répondre à la question: qu'est ce qu'un conflit ?

Les définitions du mot *conflit* données par le dictionnaire [DIC16] sont: *opposition d'interet*, *antagonisme entre deux parties* et *rivalité*. Ces définitions demeurent vraies dans notre contexte relatif à la fusion de modèles de procédés. En effet, la fusion donne lieu à une coexistance entre des relations de procédés en provenance des deux modèles fusionnés. Cette coéxistance n'est pas toujours possible, vu qu'elle peut manquer de logique si les relations impliquées ont des sémantiques contradictoires. Pour résumer donc cette notion, nous pouvons dire qu'un conflit dans un modèle de procédés est traduit par l'existence de relations contradictoires entre les éléments de ce modèle [HAC18].

Il est maintenant important de répondre à la question suivante: comment se forme un conflit dans un modèle de procédés? À première vue, nous pouvons dire qu'un conflit se forme dès qu'une relation est ajoutée à un modèle de procédés cohérent, et que cette relation est contradictoire avec l'une des relations de ce modèle. Pourtant, cette réponse reste insuffisante si on veut étudier des conflits de façon exhaustive. À cet effet, nous commençons par donner dans la table II.3, pour chaque type de relation de procédés, l'élément considéré comme source et celui considéré comme cible de la relation.

Chaque relation de procédés est un lien impliquant un élément source et un élément cible. La relation est par conséquent considérée comme *sortante* par rapport à son élément source, et comme *entrante* par rapport à son élément cible. Ainsi, pour pouvoir étudier de façon exhaustive les relations susceptibles d'êtres impliquées dans des conflits, deux point de vue s'offrent à nous: soit le point de vue de l'élément source où la relation est *sortante*, soit du point de vue de l'élément cible où la relation est *entrante*.

Relation	Élément source	Élément cible
Refinement	L'élément raffiné	L'élément général
Aggregation	Le composite	Le composant
ProductImpact	L'élément impactant	L'élément impacté
ProductResponsability	Le rôle	Le produit
TaskPerformance	Le rôle	La tâche
TaskParameter	La tâche	Le produit
TaskPrecedence	La tâche prédécesseur	La tâche successeur

Table II.3. Sources et cibles des relations de procédés

Nous choisissons dans le présent travail le point de vue de l'élément source, et nous considérons donc pour chaque élément de procédés, toutes ses relations *sortantes*. Les relations *entrantes* d'un élément donné seront quant à elles prises en considérations dans leurs éléments sources (où elles sont des relations *sortantes*). Ce choix nous emmène donc à nous focaliser sur les conflits causés par les relations sortantes.

Ainsi, un conflit pourra se former dans un modèle de procédés initialement cohérent, si après modification de ce modèle, l'un de ses éléments sera affecté par au moins deux relations sortantes contradictoires.

3.2 Exemple illustratif de conflit

La figure II.4 présente un conflit lié à aux produits en entrée/sortie et aux dépendances *TaskPrecedence* entre tâches. En effet, considérons les deux modèles de procédés PM1 constitué des tâches A et B et du produit P; et PM2 constitué des tâches C et D. ces deux modèles sont initialement cohérents. Nous allons alors les fusionner, en faisant correspondre la tâche C à la tâche A, et la tâche D à la tâche B. Le résultat de cette fusion sera le modèle de procédés PM3 constitué des tâches A et B et du produit P; où la tâche B précèdera la tâche A, et où le produit P en sortie de la tâche A sera en entrée de la tâche B.

Cette configuration de relations entre tâches et produit constitue un conflit. C'est le fait qu'un produit ne peut jamais être un paramètre d'entrée pour une tâche antérieure à sa création (la création étant assurée par la tâche pour laquelle le produit constitue un paramètre de sortie). Dans cet exemple, la tâche A crée le produit P. Ce produit est ensuite mis en entrée de la tâche B, qui précède la tâche A, chose qui est impossible.

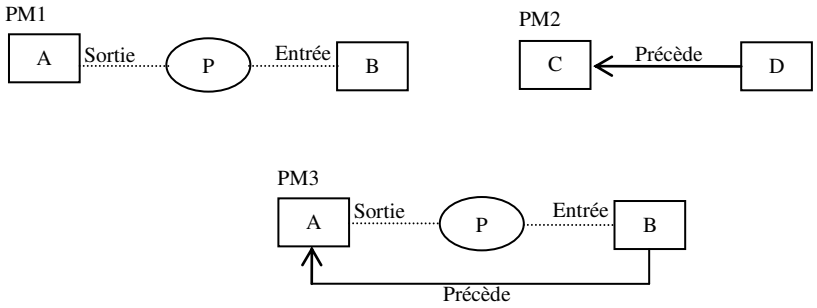


Figure II.4. Exemple illustratif d'un conflit

3.3 Modes de réutilisation et gestion de conflits

La réutilisation de patrons pour modifier un modèle de procédés fusionne deux modèles, celui à modifier et celui donné à travers la solution du patron à réutiliser. Ainsi, deux possibilités s'offrent au concepteur de procédés lors de la fusion: il pourra favoriser l'un ou l'autre des deux modèles fusionnés lors de la production du résultat. Dans [TRA07a], l'opérateur *PatternApplying* a traduit ces possibilités de fusion à travers les trois modes d'application de l'opérateur. Ces modes sont *Change*, *Extend* et *Replace*, où les deux derniers sont relatifs à la fusion.

La réutilisation de patrons en mode *Change* remplace le modèle de procédés à modifier par la solution du patron, et donc n'opère pas de fusion. La réutilisation en mode *Replace* fusionne les deux modèles en favorisant le patron. C'est à dire que lorsque le modèle à modifier contient des éléments et relations différents de ceux du patron, ces éléments et relations ne seront conservés dans le résultat que s'ils ne sont pas en contradiction avec ceux définis dans le patron. Le cas échéant, ils seront omis alors que l'on concervera ceux du patron. La réutilisation en mode *Extend* quant à elle, fusionne en favorisant le modèle à modifier. C'est à dire que les éléments et relations du modèle à modifier resteront intacts, alors que les éléments et relations en provenance du patron seront ignorés dans le cas où ils sont contradictoirs avec ceux du modèle à modifier.

Ainsi, les mesures correctives à prendre lorsqu'un conflit est susceptible de se former, dépendront du mode de réutilisation adopté par le concepteur de procédés. En effet, lorsqu'un conflit est susceptible de se former entre deux relations (où l'une provient du patron et l'autre du modèle à modifier), alors l'une de ces deux relations sera omise et l'autre relation sera conservée, selon que le mode de réutilisation adopté favorise le patron ou le modèle à modifier.

4. L'étude des conflits

Les conflits apparaissent dans un modèle de procédés, lorsque l'un de ses éléments est affecté par au moins deux relations sortantes contradictoires. Pour identifier tous les conflits potentiels dans un modèle de procédés, nous avons étudié les combinaisons de relations sortantes qui peuvent relier chaque couple d'éléments de procédés. Nous avons observé alors que les conflits peuvent survenir à plusieurs niveaux, selon les relations impliquées. Nous avons classé ces niveaux comme suit: le premier niveau dont les conflits sont appelés conflits de premier ordre; le deuxième niveau dont les conflits sont appelés conflits de second ordre; le troisième niveau dont les conflits sont appelés conflits de troisième ordre; ainsi que le niveau des cycles indirects.

Nous nous intéressons dans le présent travail aux conflits de premier ordre. Le principe de traitement des conflits relatifs aux autres nivaux sera semblable à celui du premier ordre. L'idée est que les conflits d'un niveau particulier doivent être identifiés de manière exhaustive, pour pouvoir ensuite être détectés et résolus dans l'éventualité où ils se présentent dans un modèle de procédés. Néanmoins, les niveaux de conflits hormis le premier, restent au-delà de la portée du présent travail.

Il est également important de mentionner que notre but étant l'étude des conflits dus à la réutilisation de patrons pour modifier des modèles de procédés. Cependant, ces conflits ne sont pas les seuls susceptibles de se produire sur un modèle de procédés. Il y à en effet les conflits indépendants de cette forme de réutilisation, et donc indépendants de la fusion de modèles de procédés, que nous appellerons dans ce qui suit *incohérences*. Ces incohérences peuvent se produire lors de la création ou la modification d'un modèle de procédés, et seront abordées en fin de la présente section.

Quant aux niveaux de conflits, pour faciliter leur compréhension, nous schématisons dans la figure II.5 une configuration d'éléments et de relations de procédés. Nous représentons un élément de procédés par un rectangle, et une relation de procédés par une arête reliant deux rectangles. Cette configuration nous permettra de définir dans ce qui suit les différents niveaux de conflits.

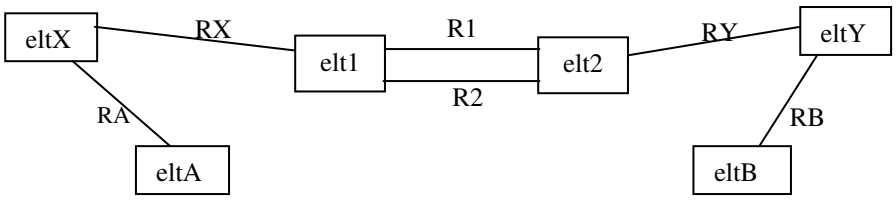


Figure II.5. Configuration d'éléments et de relations de procédés

4.1 Conflits de premier ordre

Nous définissons un conflit de premier ordre comme suit, en faisant référence à la figure II.5: soient elt1 et elt2 deux éléments de procédés (une tâche, un rôle ou un produit), et soit R1 une relation de procédés (TaskPerformance, ProductResponsability, TaskParameter, TaskPrecedence, ProductImpact, Agregation ou Refinement) reliant elt1 et elt2. Un conflit de premier ordre est une inconsistance provoquée par l'ajout d'une relation R2 entre elt1 et elt2 qui soit contradictoire (incohérente) avec R1 [HAC18].

Le nom *premier ordre* découle du fait que ces conflits sont causés par des relations contradictoires, reliant directement (relation de premier niveau) un même couple d'éléments de procédés. Par exemple, soient deux tâches T1 et T2 liées par *Aggregation* (R1), où T1 est l'agrégat de T2. L'ajout d'une seconde *Aggregation* (R2) entre T1 et T2, où T2 est l'agrégat de T1, constitue un conflit de premier ordre.

4.2 Conflits de deuxième ordre

Un conflit de second ordre apparaît lorsqu'une relation R1, liant deux éléments de procédés elt1 et elt2, entre en conflit avec une autre relation RX ou RY. RX ou RY est une relation qui implique l'un des éléments elt1 ou elt2, mais sans pour autant lier ces deux éléments [HAC18]. La figure II.5 montre une telle configuration d'éléments et de relations.

Par exemple, soit le rôle R responsable du produit P (donc R et P sont liés par la relation *ProductResponsibility* (R1)). L'ajout d'un second role R' responsable de ce même produit P (donc R' et P sont liés par la relation *ProductResponsibility* (RX)) crée un conflit de deuxième ordre. En effet, cette configuration va donner deux rôles R et R' tous les deux responsables du même produit P.

4.3 Conflits de troisième ordre

Un conflit de troisième ordre apparaît lorsqu'une relation R1 liant deux éléments de procédés elt1 et elt2, entre en conflit avec une autre relation RA (ou respectivement RB). Cette relation

RA (respectivement RB) implique l'élément de procédés eltX (respectivement eltY), qui est déjà lié à l'élément elt1 (respectivement elt2), sans pour autant être le moyen de liaison entre eltX et elt1 (respectivement entre eltY et elt2) [HAC18]. La configuration donnée par la figure II.5 montre un tel cas de conflit.

Par exemple, soient deux tâches T1 et T2 liées par la relation R1, qui est de type *TaskPrecedence* où T1 précède T2. Soient également PA et PB deux produits liés respectivement aux tâches T1 et T2 à travers les relations *TaskParameter*, avec PA *IN* pour T1et PB *OUT* de T2. L'ajout d'une relation RA entre PA et PB *ProductImpact*, où PB impacte PA, constitue un conflit de troisième ordre.

4.4 Conflits des cycles indirects

Les cycles indirects sont des relations de même nature, mais de sens opposés, reliant de manière transitive deux éléments de procédés [HAC18].

Comme exemple, prenons un cycle indirect de la relation *ProductImpact*, montré par la figure II.6. Le produit P1 impacte le produit P2, le P2 impacte le produit P3, le produit P3 impacte le produit P4, et enfin le P4 impacte le produit P1. Cette configuration donne un cycle de la relation *ProductImpact*, reliant de manière indirecte les deux produits P1 et P4.

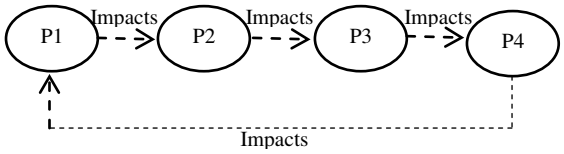


Figure II.6. Cycle indirecte de la relation *ProductImpact*

4.5 Incohérences

Des incohérences indépendantes de la réutilisation de patrons pour modifier des modèles de procédés, et donc indépendantes de la fusion, sont susceptibles de se produire lors de la création ou la modification d'un modèle de procédés. Ces incohérences, que nous citons dans la suite de cette section, doivent être interceptées et traitées lors de la création ou de la modification d'un modèle de procédés pour garantir un résultat cohérent.

- Les incohérences des redondances d'éléments ou de relations. Aucune redondance d'éléments ou de relations n'est tolérée dans un modèle de procédés, mais également aucune redondance de relations par transitivité. Une telle redondance concerne le cas où une relation lie deux éléments de procédés de façon directe, et relie également ces deux même éléments une seconde fois mais de façon transitive (indirecte).

  Par exemple, soit un produit P1 qui impacte un produit P2, et ce dernier impacte un autre produit P3. La relation qui fait que P1 impacte P3 est dans ce cas une relation redondante par transitivité, comme le montre la figure II.7.

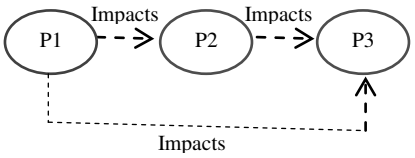


Figure II.7. Exemple de relation redondante par transitivité

- Les incohérences des éléments *inaccessibles*, des éléments *puits* et des éléments *isolés*. Un élément *inaccessible* est soit un élément de type *Task* soit un élément de contrôle hormis l'activité initiale, où aucune relation *TaskPrecedence* ou *Link* dans le modèle de procédés ne peut conduire vers lui. Un élément *puits* est soit un élément de type *Task* soit un élément de contrôle hormis l'activité finale, à partir duquel aucune relation *TaskPrecedence* ou *Link* ne conduit vers le restant du modèle de procédés. Un élément *isolé* est élément de procédés ou de contrôle, hormis les éléments *inaccessibles* ou *puits*, qui n'est pas correctement lié au modèle de procédés et donc isolé par rapport à ce dernier.

  Par exemple, dans le modèle de procédés montré par la figure II.8, T4 est une tâche inaccessible, T3 est une tâche "puits", et le produit P2 est un produit isolé puisqu'il n'est lié à aucune tâche.

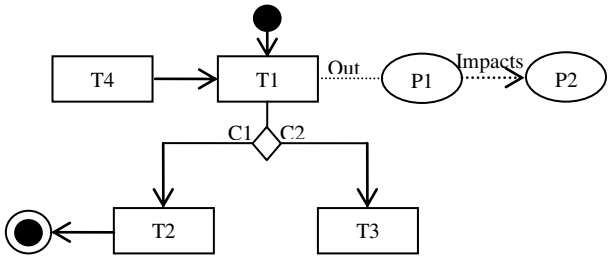


Figure II.8. Exemple d'un modèle de procédés avec incohérences

- Les incohérences des relations réflexives. Une relation réflexive est une relation de procédés qui lie un élément de procédés à lui même.

  Par exemple, un produit qui s'impacte lui-même dans un modèle de procédés, constitue une incohérence dans ce modèle.

- Les incohérences des relations opposées, directes ou par transitivité. Il s'agit de relations de même nature mais de sens inverses, liant un même couple d'éléments de façon directe ou par transitivité.

  Par exemple, un produit P1 impacte un produit P2 et ce dernier impacte à son tour le produit P1. Là, il s'agit de relations opposées de façon directe.
  Un autre exemple de relation opposées par transitivité: un produit P3 qui impacte un produit P4 et ce dernier impacte un autre produit P5, et le P5 impacte à son tour le produit P3.
  La figure II.9 illustre ces exemples.

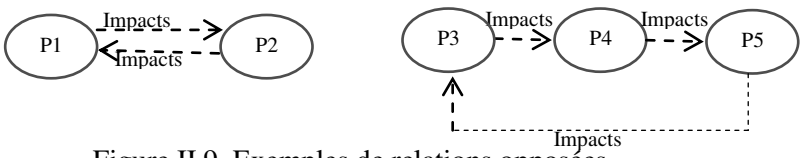


Figure II.9. Exemples de relations opposées

Cependant, nous précisons que les incohérences des relations opposées directes sont des conflits de premier ordre; elles sont cependant indépendantes de la réutilisation de patrons et donc de la fusion, ce qui justifie notre choix de les aborder parmi les incohérences. Il en est de même pour les relations opposées indirectes, qui sont des conflits de cycles indirectes.

- Les incohérences des retours temporels. Un tel retour est une relation de procédés de type *TaskParameter*, mettant un produit en entrée d'une tâche qui précède sa création.

  Un exemple de cela, que nous montrons à travers la figure II.10, est la présence d'un produit P sortant d'une tâche T2 pour entrer dans une tâche T1 qui la précède.

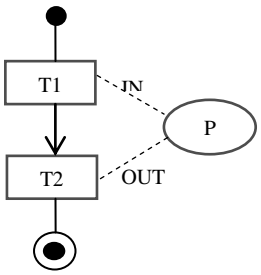


Figure II.10. Exemple d'un retour temporel

- Les incohérences de contrôle. De telles incohérences sont dues aux éléments et relations de contrôle, qui sont mal placés dans un modèle de procédés.
  Par exemple, la non unicité de l'élément initial dans un modèle de procédés, ou la présence de deux liens sortants d'un choix et allant vers une même tâche.

5. Conclusion

Dans ce chapitre, nous avons présenté la réutilisation de patrons pour modéliser des procédés logiciels, et avons abordé une forme particulière de cette réutilisation, qui est la réutilisation de patrons pour modifier des modèles de procédés déjà existants. Nous avons également présenté notre méta modèle à travers ses deux volets. Ce méta modèle va régir la structure des modèles de procédés et patrons, qui seront manipulés grâce à notre approche, ainsi que leurs relations.

Par ailleurs, nous avons défini dans ce chapitre les conflits engendrés par la réutilisation de patrons pour modifier des modèles de procédés. Nous avons montré les différents niveaux de conflits (conflits de premier ordre, de second ordre, de troisième ordre, les cycles indirects), ainsi que les incohérences. Notre intérêt porte principalement dans le présent travail sur les conflits de premier ordre; les conflits liés aux autres niveaux restent cependant au delà du cadre de cette thèse.

CHAPITRE III

CONTRIBUTIONS POUR LA REUTILISATION DE PATRONS

Ce chapitre introduit les contributions du présent travail de thèse, et commence par donner un aperçu de notre approche ainsi que des objectifs que nous nous sommes fixés (cf. § 1). Ensuite, les conflits de premier ordre sont abordés (cf. § 2). L'opérateur de réutilisation de patrons est après cela présenté (cf. § 3). Enfin, notre approche est évaluée; la configuration expérimentale utilisée ainsi que l'étude de cas élaborée sont alors présentées (cf. § 4).

1. Aperçu et objectifs de notre approche

Après l'étude de la problématique de réutilisation de patrons pour modifier des modèles de procédés existants, nous rappelons brièvement que cette problématique implique en fait deux modèles de procédés: celui à modifier et la solution offerte par le patron à réutiliser (qui est elle-même un modèle de procédés). Le résultat d'une telle réutilisation est un modèle de procédés obtenu en fusionnant ces deux modèles. Cependant, la fusion peut produire un modèle cohérent, comme elle peut conduire à un résultat incohérent si des conflits apparaissent entre les modèles fusionnés.

Le but du présent travail est de proposer une approche résolvant cette problèmatique. Les objectifs de cette approche sont: (a) le choix du méta modèle pour régir les modèles de procédés et patrons à manipuler (cf. section 2 du chapitre II). (b) L'identification des conflits potentiels dus à la réutilisation de patrons pour modifier des modèles de procédés, en particulier lors de la procédure de fusion. (c) La proposition à travers un algorithme de la sémantique opérationnelle d'un opérateur de réutilisation. Cet opérateur procèdera à la fusion du modèle de procédés à modifier avec le patron à réutiliser, tout en assurant la detection et la résolution des conflits éventuels. Le résultat de cet opérateur sera le modèle modifié en état cohérent. (d) L'approche à proposer a également comme objectif la proposition d'une plateforme de réutilisation de patrons pour modifier des modèles de procédés existants. Enfin, une évaluation concernera cette approche pour montrer sa valeur ajoutée.

De nombreux conflits sont suceptibles de survenir lors de la réutilisation de patrons. Pour une meilleure maitrise de l'étude des conflits, nous limitons alors le présent travail de thèse aux conflits de premier ordre. En conséquence, nous appellerons dans la suite de ce document *un résultat cohérent*, le modèle de procédés modifié exempt de conflits de premier ordre.

2. Conflits de premier ordre

Nous consacrons la présente section aux conflits de premier ordre. Nous présentons d'abord l'analyse qui nous a permis de cerner ces conflits; et nous listons ces derniers par la suite.

2.1 Analyse des conflits de premier ordre

Nous nous focalisons dans le présent travail sur les conflits de premier ordre. Ces conflits sont causés par des relations sortantes contradictoires, reliant de façon directe un même couple d'éléments de procédés. Les autres niveaux de conflits quant à eux, ne font pas l'objet du présent travail.

Pour identifier les conflits de premier ordre de façon exhaustive, nous avons analysé toutes les combinaisons possibles de relations sortantes, entre deux éléments de procédés. Cette analyse a été appliquée pour tous les couples d'éléments de procédés.

Pour ce faire, nous avons commencé par énumérer touts les couples d'éléments de procédés, que nous montrons à travers la table III.1.

Eléments de procédés	Task	Role	Product
Task	{*Task, Task*}	{*Task, Role*}	*{Task, Product*}
Role	{*Role, Task*}	{*Role, Role*}	*{Role, Product*}
Product	{*Product, Task*}	*{Product, Role*}	*{Product, Product*}

Table III.1. Couples d'éléments de procédés

Nous avons pris en considération le fait que chaque élément peut être la source ou la cible d'une relation, donc les couples inversés ont tous les deux été considérés pour l'analyse. Par exemple, dans le couple {*Task, Role*} on considère *Task* comme source de la relation et *Role* comme cible, tandis que dans l'autre couple inversé {*Role, Task*} on considère l'inverse.

Nous avons ainsi obtenu une liste de neuf couples d'éléments de procédés. Ensuite, pour chacun de ces neuf couples, nous avons appliqué toutes les combinaisons de deux relations possibles et sortantes (cf. table II.3 du chapitre II). Les relations possibles sont les relations qu'on peut appliquer entre un couple d'éléments. Par exemple, la relation *Refinement* est une relation possible entre le couple {*Task, Task*}; cette même relation n'est pas possible entre le couple {*Task, Product*}. Nous ciblons les relations sortantes, puisque nous avons choisi dans ce travail d'adopter le point de vue des éléments sources (cf. 3.1 du chapitre II).

Par exemple, en considérant le couple {*Role, Role*}, nous montrons à travers la table III.2 le sens des relations de procédés applicable à l'élément *Role*. Certaines relations sont définies comme relations sortantes par rapport à cet élément (*Aggregation*, *Refinement*, *TaskPerformance*, *ProductResponsability*), alors que d'autres sont entrantes (*Aggregation*, *Refinement*); certaines relations sont quant à elles non applicables à *Role* (*TaskPrecedence, TaskParameter, ProductImpact*). Parmi les relations sortantes, nous allons alors garder que les relations pouvant avoir *Role* comme cible, en se référant toujours à la table III.2. Nous obtenons donc les deux relations *Aggregation* et *Refinement*, à partir desquelles nous allons construire toutes les combinaisons possibles ayant *Role* comme source et *Role* comme cible. Ces combinaisons sont montrées à travers la table III.3.

Relations de procédés	Sens par rapport à l'élément *Role*
Aggregation	Entrante ou sortante
Refinement	Entrante ou sortante
TaskPrecedence	Non applicable à l'élément *Role*
TaskPerformance	Sortante
TaskParameter	Non applicable à l'élément *Role*
ProductResponsability	Sortante
ProductImpact	Non applicable à l'élément *Role*

Table III.2. Sens des relations applicables à l'élément *Role*

Relations de procédés	Aggregation	Refinement
Aggregation	[*Aggregation, Aggregation*]	[*Aggregation, Refinement*]
Refinement	[*Refinement, Aggregation*]	[*Refinement, Refinement*]

Table III.3. Combinaisons ayant comme source et cible l'élément Role

Nous avons donc considéré, après élimination des doublons, trente-six combinaisons de relations à analyser. Pour expliquer la notion de doublon, nous reprenons l'exemple précédent relatif au couple {*Role, Role*}. Nous avons obtenu pour ce couple quatre combinaisons, de deux relations possibles et sortantes, montrées par la table III.3. Parmi ces combinaisons, la combinaison [*Aggregation, Refinement*] est considérée comme un doublon de [*Refinement, Aggregation*]. Par conséquent, une seule combinaison sera considérée pour étude.

Parmi ces trente-six combinaisons, certaines sont cohérentes ou représentent des cas de redondance de relations. Une combinaison cohérente est une combinaison de deux relations dont la présence simultanée entre deux éléments de procédés ne constitue pas un conflit. Une telle combinaison sera alors exclue de la liste des conflits. Une combinaison de redondance quant à elle, est une combinaison constituée de deux relations de même nature. Une telle combinaison représente la duplication d'une relation entre deux éléments de procédés, et ne constitue pas un cas de conflit. Elle sera alors exclue de la liste des conflits. En se référant aux combinaisons relatives au couple {*Role, Role*}, données précédemment dans la table III.3, la combinaison [*Aggregation, Aggregation*] est une combinaison de redondance, aussi bien que la combinaison [*Refinement, Refinement*].

Ainsi, parmi les trente-six combinaisons étudiées, et en dehors des combinaisons cohérentes ou de redondance de relation, le restant des combinaisons constitue des conflits. C'est ainsi que nous avons établi notre liste des dix-huit conflits de premier ordre, que nous expliquerons dans la section suivante

2.2 Liste des conflits de premier ordre

L'étude des conflits nous a permis d'identifier tous les conflits de premier ordre, et cela grâce à l'analyse de toutes les combinaisons de relations entre les couples d'éléments de procédés. Nous avons ainsi pu établir la liste suivante de dix-huit conflits de premier ordre [HAC18].

2.2.1 Conflits des cycles

Les conflits provoqués par des cycles sont inspirés du problème des cycles dans les relations homogènes [TRA07]. Les relations homogènes sont des relations de procédés qui lient deux éléments de procédés de même nature; ces relations sont: *TaskPrecedence*, *ProductImpact*, *Aggregation* et *Refinement*. Les conflits des cycles, montrés par la figure III.1, sont alors comme suit:

- *Cas 1 :* Soient elt1 et elt2 deux éléments de procédés liés par la relation *Aggregation*, où elt1 représente le tout et elt2 représente la partie. L'ajout d'une seconde *Aggregation* entre ces deux éléments, dans laquelle elt1 représente la partie et elt2 représente le tout, crée un conflit. C'est l'interprétation du fait qu'un élément ne peut pas être simultanément le tout et la partie d'un autre élément.

- *Cas 2 :* Soient elt1 et elt2 deux éléments liés par la relation *Refinement*, où elt1 spécialise elt2. L'ajout d'une seconde relation de *Refinement* entre ces deux éléments, dans laquelle elt2 spécialise elt1 crée un conflit. Ceci est dû au fait qu'un élément ne peut pas généraliser et spécialiser simultanément un autre élément.

- *Cas 3 :* Soient P1 et P2 deux produits connectés par la relation *ProductImpact*, où P1 a un impacte sur P2. L'ajout d'une seconde relation *ProductImpact* entre ces deux produits, où P2 a un impacte sur P1, crée un conflit. C'est l'interprétation du fait qu'un produit ne peut pas impacter un autre produit et être simultanément impacté par ce dernier.

- *Cas 4 :* Soient T1 et T2 deux tâches liées par la relation *TaskPrecedence*, où T1 précède T2. L'ajout d'une seconde relation *TaskPrecedence* entre ces deux tâches, dans laquelle T2 précède T1, crée un conflit que nous appelons *la boucle TaskPrecedence*. C'est l'interprétation du fait qu'une tâche ne peut pas précéder son prédécesseur.
  Il est à noter que ce scénario peut dans certaines situations ne pas être considéré comme un conflit, mais plutôt comme un retour arrière.

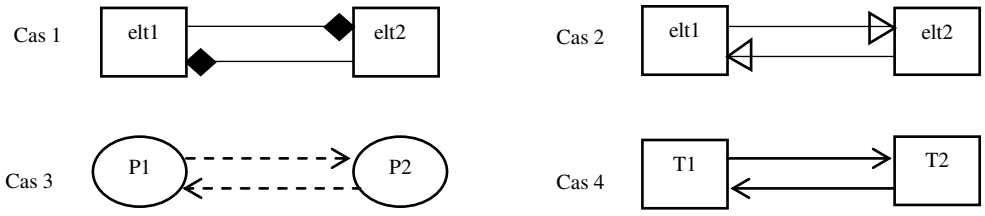


Figure III.1. Conflits des cycles

2.2.2 Conflits entre les relations *Aggregation* et *Refinement*

Les conflits causés par les relations *Aggregation* et *Refinement*, montrés par la figure III.2, sont les suivants:

- *Cas 5 :* Soient elt1 et elt2 deux éléments liés par la relation *Aggregation*, où elt1 représente le tout et elt2 représente la partie. L'ajout de la relation *Refinement* entre ces deux éléments, où elt1 spécialise elt2 crée un conflit. C'est l'interprétation du fait que *le tout* n'est pas le raffinement de *la partie*. En outre, l'ajout de la relation *Refinement* où elt2 spécialise elt1 crée un conflit. C'est l'interprétation du fait que *la partie* ne peut pas spécialiser *le tout*.

- *Cas 6 :* Inversement, si une relation *Refinement* existe entre deux éléments elt1 et elt2 où elt1 spécialise elt2, alors l'ajout de la relation *Aggregation* entre ces deux éléments (soit avec elt1 représentant le tout et elt2 représentant la partie soit avec elt2 qui représente le tout et elt1 qui représente la partie), crée un conflit pour les mêmes raisons citées dans le *cas 5*.

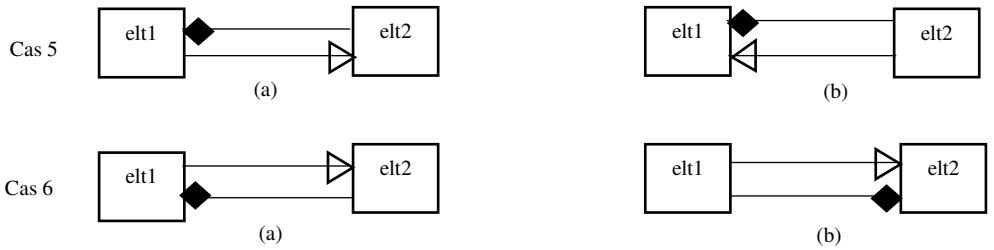


Figure III.2. Conflits entre les relations *Aggregation* et *Refinement*

2.2.3 Conflits de la relation *TaskParameter*

Les conflits dus à la relation *TaskParameter,* montrés par la figure III.3, sont comme suit:

- *Cas 7 :* Soient P un produit et T une tâche, connectés par la relation *TaskParameter* où P est un paramètre *IN* pour T. L'ajout d'une seconde relation *TaskParameter* entre ces deux éléments, où P est soit un paramètre *OUT* soit un paramètre *IN-OUT* pour T, ne crée pas un conflit mais crée plutôt une redondance. Ainsi, nous gardons une seule relation *TaskParameter* entre P et T, où P sera un paramètre *IN-OUT* pour T.

- *Cas 8 :* Soient un produit appelé P et une tâche appelée T liés par la relation *TaskParameter*, où P est un paramètre *IN-OUT* pour T. L'ajout d'une seconde relation *TaskParameter* entre ces deux éléments, où P est un paramètre *IN* ou bien un paramètre OUT pour T, ne crée pas de conflit mais crée plutôt une redondance. Ainsi, nous gardons une seule relation *TaskParameter* entre P et T, dans laquelle P sera un paramètre *IN-OUT* pour T.

- *Cas 9 :* Soient un produit appelé P et une tâche appelée T liés par la relation *TaskParameter*, où P est un paramètre *OUT* pour T. L'ajout d'une seconde relation *TaskParameter* entre ces deux éléments, où P est un paramètre *IN* pour T crée un conflit. C'est l'interprétation du fait qu'un produit ne peut pas être un paramètre d'entrée (*IN*) pour la tâche qui le crée (tâche pour laquelle il est *OUT*).

- *Cas 10 :* Soient un produit appelé P et une tâche appelée T liés par la relation *TaskParameter*, où P est un paramètre *OUT* pour T. L'ajout d'une seconde relation *TaskParameter* entre ces deux éléments, où P est un paramètre *IN-OUT* pour T crée un conflit. C'est l'interprétation du fait qu'un produit ne peut pas être mis à jour (*IN-OUT*) par la tâche qui est entrain de le créer.

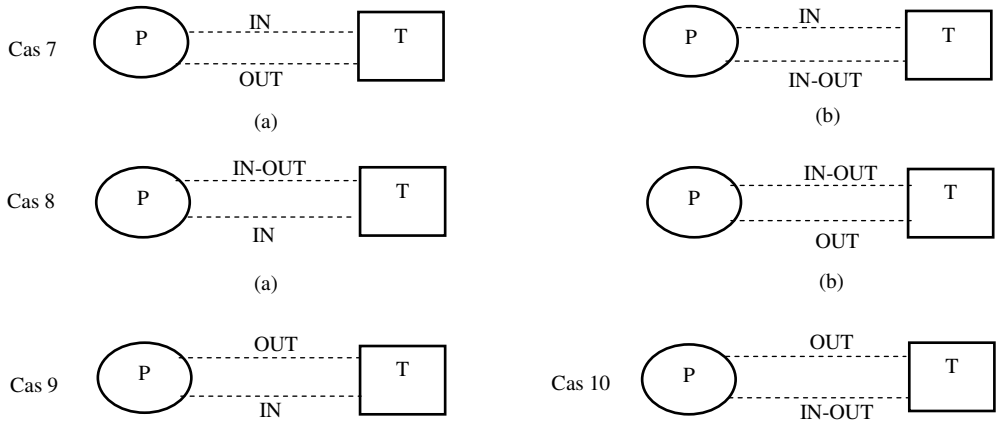


Figure III.3. Conflits de la relation *TaskParameter*

2.2.4 Conflits de la relation *TaskPerformance*

Les conflits causés par la relation *TaskPerformance* sont des problèmes abordés mais non traités dans l'approche [TRA07]. Ces conflits sont montrés par la figure III.4.

- *Cas 11 :* Soient un rôle R et une tâche T connectés par la relation *TaskPerformance,* où R est *Performer* pour T. L'ajout d'une seconde relation *TaskPerformance* entre ces deux éléments, où R est *Assistant* pour T, crée un conflit. C'est l'interprétation du fait qu'un rôle ne peut pas simultanément accomplir et aider dans l'accomplissement d'une même tâche.

- *Cas 12 :* Inversement, si R et T sont connectés par la relation *TaskPerformance* où R est *Assistant* pour T; alors l'ajout d'une seconde relation *TaskPerformance* entre ces deux éléments, où R est *Performer* pour T, crée un conflit pour la même raison citée dans le *cas 11*.

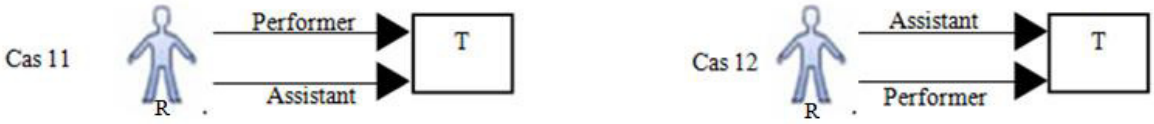


Figure III.4. Conflits de la relation *TaskPerformance*

2.2.5 Conflits entre les relations *Aggregation* et *TaskPrecedence*

Les conflits dus aux relations *Aggregation* et *TaskPrecedence,* montrés par la figure III.5, sont comme suit:

- *Cas 13:* Soient T1 et T2 deux tâches liées par la relation *Aggregation*, où T1 représente le tout et T2 représente la partie. L'ajout d'une relation *TaskPrecedence* entre ces deux tâches (soit avec T1 qui précède T2, soit avec T2 qui précède T1) crée un conflit. C'est l'interprétation du fait qu'une tâche ne peut pas précéder ou succéder à l'une de ses parties.

- *Cas 14:* Inversement, si une relation *TaskPrecedence* existe entre deux tâches T1 et T2, où T1 précède T2; alors l'ajout d'une relation *Aggregation* entre ces deux tâches (soit avec T1 représentant le tout et T2 représentant la partie ou inversement avec T2 représentant le tout et T1 la partie), crée un conflit pour la même raison citée dans le *cas 13*.

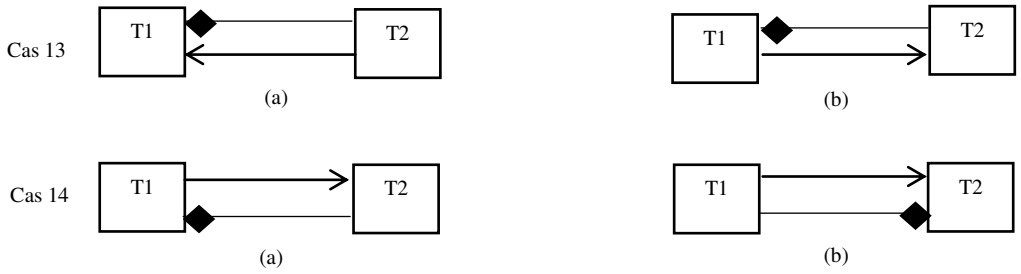


Figure III.5. Conflits entre les relations *Aggregation* et *TaskPrecedence*

2.2.6 Conflits entre les relations *Refinement* et *TaskPrecedence*

Les conflits dus aux relations *Refinement* et *TaskPrecedence,* montrés par la figure III.6, sont comme suit:

- *Cas 15:* Soient T1 et T2 deux tâches liées par la relation *Refinement*, où T1 spécialise T2. L'ajout d'une relation *TaskPrecedence* entre ces deux tâches (soit avec T1 précède T2, soit avec T2 précède T1) crée un conflit. C'est l'interprétation du fait qu'une tâche ne peut pas précéder ou succéder à l'une de ses spécialisations.

- *Cas 16:* Inversement, si une relation *TaskPrecedence* existe entre deux tâches appelées T1 et T2, où T1 précède T2, alors l'ajout d'une relation *Refinement* entre ces deux tâches (soit avec T1 qui spécialise T2 ou inversement avec T2 qui spécialise T1), crée un conflit pour la même raison citée dans le *cas 15*.

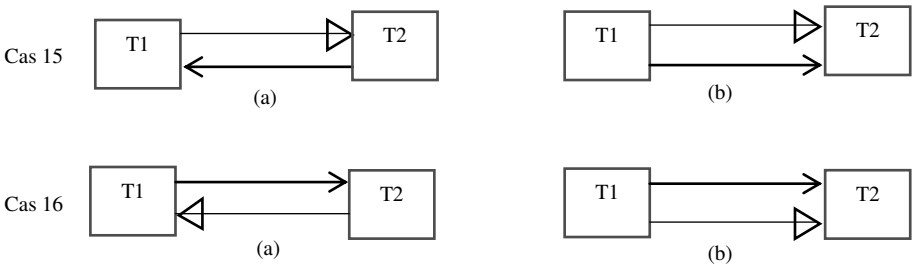


Figure III.6. Conflits entre les relations *Refinement* et *TaskPrecedence*

2.2.7 Conflits entre les relations *ProductImpact* et *Refinement*

Les conflits causés par les relations *ProductImpact* et *Refinement,* montrés par la figure III.7, sont les suivants:

- *Cas 17:* Soient P1 et P2 deux produits connectés par la relation *ProductImpact*, où P1 impacte P2. L'ajout de la relation *Refinement* entre ces deux produits, où P1 spécialise

P2 crée un conflit. C'est l'interprétation du fait que le produit spécialisé ne peut pas impacter son produit original.

- *Cas 18:* Inversement, si une relation *Refinement* existe entre deux produits P1 et P2 où P1 spécialise P2; alors l'ajout de la relation *ProductImpact* entre ces deux produits où P1 impacte P2, crée un conflit pour la même raison citée dans le *cas 17.*

Figure III.7. Conflits entre les relations *ProductImpact* et *Refinement*

3. L'opérateur de réutilisation

Cette section est dédiée à notre opérateur de réutilisation. Mais avant d'aborder en détail le fonctionnement de cet opérateur, nous commençons par donner les choix que nous avons adoptés pour son élaboration.

3.1 Principe et choix adoptés

Notre but est de proposer un opérateur de réutilisation de patrons, que nous appellerons *ReuseOperator*, pour modifier des modèles de procédés existants avec gestion des conflits de premier ordre. La sémantique opérationnelle de notre opérateur est donnée à travers l'algorithme implémentant cet opérateur. La présente section donne le principe de fonctionnement de l'algorithme, ainsi que les choix adoptés pour assurer ce fonctionnement.

L'algorithme est montré par la figure III.8, sous forme d'un organigramme simplifié pour faciliter sa lecture. Les entrées de l'algorithme sont:

- Le modèle de procédés à modifier, appelé *PM*.
- Le patron à réutiliser, appelé *Sol* (en réalité, il s'agit de la section Solution du patron).
- Le mode de réutilisation sélectionné, à savoir *extend*, *replace* ou *change*.

Le résultat, appelé *Result* dans l'algorithme, est le modèle de procédés résultant de la fusion du modèle de procédés *PM* avec la solution du patron *Sol*.

Nous rappelons qu'un modèle de procédés est constitué d'éléments et de relations. Chaque relation est un lien binaire entre deux éléments, qui sont la source et la cible de cette relation. Nous avons choisi dans le présent travail d'adopter le point de vue des éléments sources, donc nous nous intéressons aux relations sortantes.

Soit *e* un élément de procédés appartenant à *Result*. Nous appellerons r_x une relation sortante de *e*, qui est contradictoire avec une autre relation de procédés *r* (également sortante de *e*) à ajouter éventuellement (*r*) au résultat.

Nous utiliserons dans l'algorithme trois listes de relations ainsi que trois listes de conflits, que nous définissons dans la table III.4.

Nom de la liste	Type d'éléments	Contenu
Liste 1	Relations de procédés	*TaskPrecedence, Aggregation, Refinement, TaskParameter*
Liste 2	Relations de procédés	*Refinement, Aggregation, TaskPerformance*
Liste 3	Relations de procédés	*Refinement, Aggregation, TaskParameter, ProductImpact*
Liste 4	Cas de conflits	cas 1, cas 2, cas 4, cas 5, cas 6, cas 7, cas 8, cas 9, cas 10, cas 13, cas 14, cas 15, cas 16
Liste 5	Cas de conflits	cas 1, cas 2, cas 5, cas 6, cas 11, cas 12
Liste 6	Cas de conflits	cas 1, cas 2, cas 3, cas 5, cas 6, cas 7, cas 8, cas 17, cas 18

Table III.4. Listes exploitées dans l'algorithme

Nous donnons dans la figure III.8 un synopsis de l'algorithme qui véhicule la sémantique opérationnelle de notre opérateur de réutilisation. Pour produire le résultat de réutilisation, l'algorithme commence par copier la solution du patron dans ce résultat. Ensuite, si le mode de réutilisation requiert une fusion (*Extend* ou *Replace*), alors cette dernière sera opérée en intégrant d'abord les éléments du modèle à modifier dans le résultat, puis ses relations. L'intégration de ces dernières passera par plusieurs vérifications, relativement à la nature de la relation à intégrer et aux cas de conflits susceptibles d'être engendrés par cette relation.

Dans notre approche, nous considérons chaque patron à réutilisé comme cohérent, puisque les patrons offrent par définition des solutions prouvées [HAG04]. Ainsi, aucune vérification n'est appliquée sur le patron dans notre opérateur, et il est réutilisé tel quel. Nous supposons également dans notre approche que le modèle de procédés à modifier est cohérent. Donc, aucune vérification ne lui est appliquée avant la fusion. Cependant, si une incohérence existe dans le modèle à modifier, elle sera détectée lors de la procédure de fusion et sera traitée exactement comme une incohérence due à cette procédure.

Nous choisissons dans notre approche de considérer la boucle *TaskPrecedence* (cf. la section 2.2.1 du chapitre III) comme un conflit. Cependant, cette boucle peut réellement être un conflit aussi bien qu'elle peut être un retour arrière effectué exprès. Le concepteur de procédés étant le seul capable de décider si une telle boucle est un conflit ou non. Il convient cependant de noter que si la boucle *TaskPrecedence* existe dans le patron, elle ne sera pas détectée par notre opérateur, car le patron est réutilisé tel quel dans la procédure de fusion.

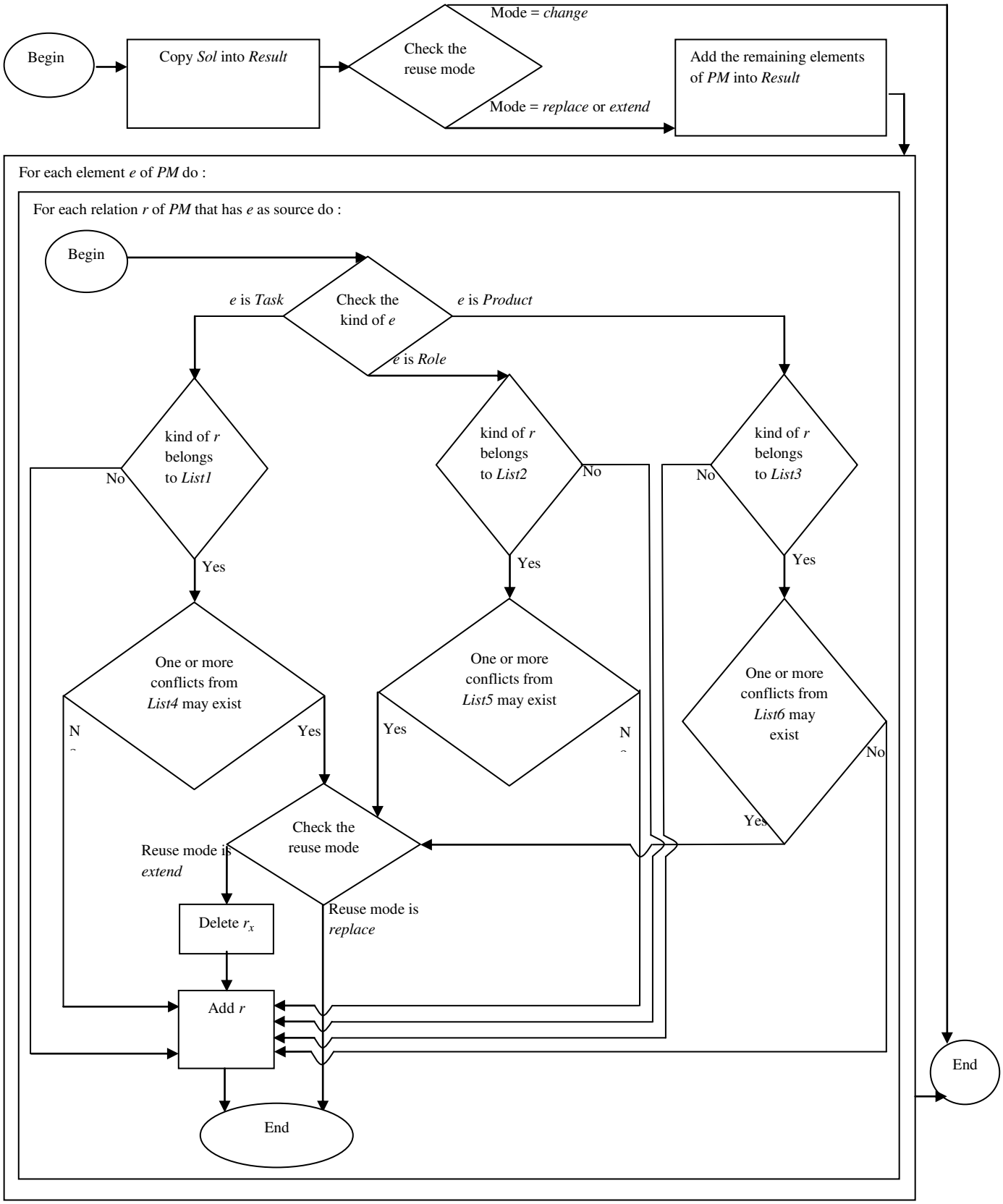


Figure III.8. Synopsis de l'algorithme [HAC18]

3.2 Détail de l'algorithme

Le détail de l'algorithme véhiculant la sémantique opérationnelle de notre opérateur de réutilisation de patrons, est donné à l'annexe A de ce document.

4. Évaluation de l'approche

Cette partie du chapitre concerne l'évaluation de l'approche. Elle vise à présenter la configuration expérimentale utilisée pour l'évaluation, ainsi que la comparaison du présent travail pour montrer sa valeur ajoutée.

Afin de vérifier que notre approche est capable de réutiliser avec succès des patrons sans produire d'incohérences dans le résultat, nous avons préparé la configuration expérimentale suivante: nous avons sélectionné une collection de cas de test, que nous avons divisée en trois ensembles correspondant à trois objectifs à atteindre. L'évaluation de ces cas de test est présentée dans la section 4.1. Dans la section 4.2, nous avons essayé de comparer notre approche avec les travaux connexes, présentés dans la section 3 du chapitre I. Il s'est avéré que l'approche la plus appropriée pour la comparaison était [TRA07], elle a donc été choisie à cette fin.

4.1 Configuration expérimentale

Notre configuration expérimentale consiste en trois ensembles de cas, correspondant aux trois objectifs que nous voulons atteindre par le biais de notre approche.

Le premier objectif est de réussir à travers notre opérateur de réutilisation, à procéder à la fusion (du modèle à modifier avec le patron à réutiliser) sans introduire de nouveaux conflits dans le résultat. Par nouveau conflit, nous entendons un conflit qui n'était auparavant ni dans le modèle à modifier, ni dans le patron à réutiliser.

Pour ce premier objectif, nous avons conçu le premier ensemble de tests. Cet ensemble contient les cas où le modèle de procédés à modifier est cohérent ainsi que le patron à réutiliser, et où le résultat de la réutilisation manuelle est également cohérent. Nous entendons par réutilisation manuelle, celle qui produit le résultat en intégrant tous les éléments et relations de procédés des deux modèles fusionnés.

En effet, nous procédons d'abord à la réutilisation manuelle qui produit un résultat cohérent, pour comparer ce dernier avec le résultat de la réutilisation automatique basée sur notre opérateur. Les résultats manuels et automatiques sont ensuite comparés pour chacun des cas de test. Nous précisons cependant que ce premier objectif a également été vérifié à travers les deux autres ensembles de test cités ci-après.

Un exemple de cas de test du premier ensemble est donné à travers la table III.5. Il montre la réutilisation du patron donné dans cette table, en appliquant les substitutions suivantes: (T1, T1.1) (T2, T2.2) (T3, T3.3) (P1, P1.1). Cette réutilisation appliquée manuellement produit un

résultat cohérent. Les résultats automatiques sont également cohérents, comme le montre la table, pour les deux modes de réutilisation (*Extend* et *Replace*).

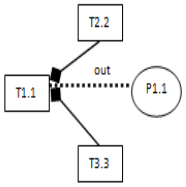
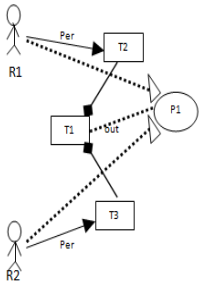
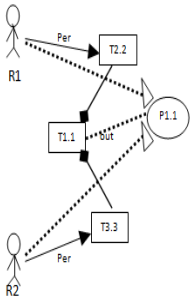
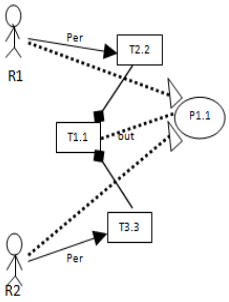
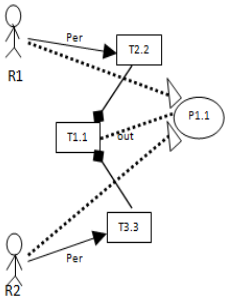
Modèle	**Patron**	**R. manuelle**	**R. automatique en mode *Extend***	**R. automatique en mode *Replace***
				

Table III.5. Exemple d'un cas de test du premier ensemble

Le deuxième objectif de l'opérateur est de réussir à détecter et à gérer chacun des dix-huit conflits de premier ordre, dans l'éventualité où un tel cas est engendré par la fusion. Pour ce deuxième objectif, nous avons conçu le deuxième ensemble de tests, qui contient des cas où le modèle de procédés à modifier ainsi que le patron à réutiliser sont cohérents, mais où le résultat de la réutilisation manuelle du patron devrait produire au moins un conflit de premier ordre. Dans ce second ensemble de tests, nous avons couvert chacun des dix-huit conflits de premier ordre.

Un cas de test du deuxième ensemble est donné à travers la table III.6. Il montre la réutilisation du patron donné dans cette table, en appliquant les substitutions suivantes: (T1, T1.1) (T2, T2.2) (R1, R1.1) (R2, R2.2) (P1, P1.1) (P2, P2.2). Cette réutilisation appliquée manuellement produit un conflit de premier ordre (cycle *TaskPrecedence*), entre les tâches T1.1 et T2.2. Les résultats automatiques sont par contre cohérents, comme le montre la table, pour les deux modes de réutilisation.

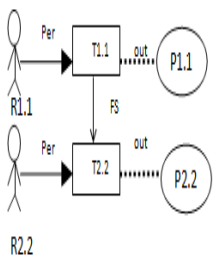
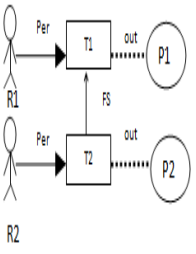
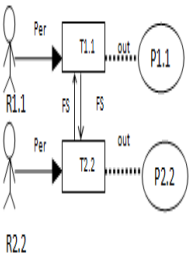
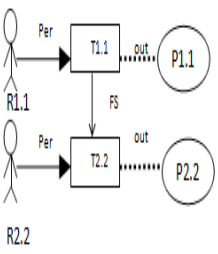
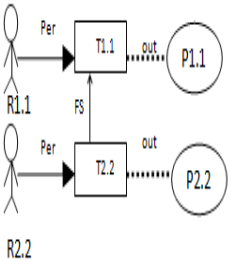
Modèle	**Patron**	**R. manuelle**	**R. automatique en mode *Extend***	**R. automatique en mode *Replace***
				

Table III.6. Exemple d'un cas de test du deuxième ensemble

Le troisième objectif est de détecter et de gérer avec succès les conflits de premier ordre préexistants dans le modèle de procédés à modifier, et qui se répercutent dans le résultat de la réutilisation manuelle. Pour ce troisième objectif nous avons conçu le troisième ensemble de test. L'ensemble regroupe les cas où le modèle de procédés à modifier contient au moins un conflit de premier ordre, et où la répercussion du conflit sur le résultat est attendue. Ce troisième ensemble couvre également chacun des dix-huit cas de conflits.

Un exemple pris dans le troisième ensemble est donné par la table III.7. Il présente la réutilisation d'un patron sur un modèle de procédés qui contient préalablement un conflit de premier ordre, entre les tâches T1.1 et T2.2 (cycle *TaskPrecedence*). Les substitutions suivantes sont appliquées: (T1, T1.1) (T2, T2.2) (R1, R1.1) (R2, R2.2) (P1, P1.1) (P2, P2.2). Ce conflit sera répercuté sur le résultat de la réutilisation manuelle, comme le montre la table. Quant aux résultats de la réutilisation automatique, ils sont cohérents pour les deux modes de réutilisation.

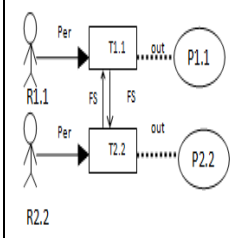
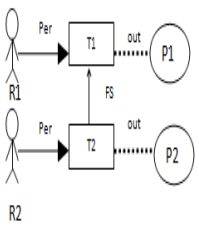
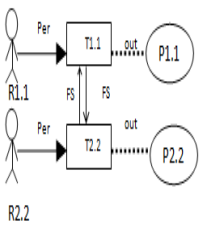
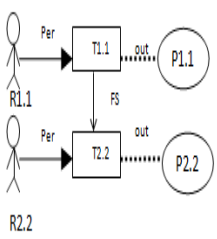
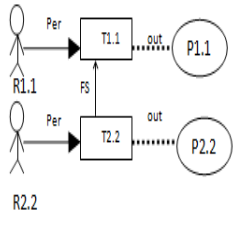
Modèle	**Patron**	**R. manuelle**	**R. automatique en mode *Extend***	**R. automatique en mode *Replace***
				

Table III.7. Exemple d'un cas de test du troisième ensemble

Les trois ensembles de tests ont été appliqués automatiquement grâce à l'opérateur de réutilisation, à travers notre plateforme (que nous allons présenter en détail dans le chapitre IV). Les conflits de premier ordre ont été vérifiés par la suite sur les résultats obtenus.

Pour les trois ensembles, tous les tests réalisés ont fourni des modèles de procédés cohérents. L'opérateur a réussi à produire un résultat exempt de conflits de premier ordre, même si de tels conflits étaient susceptibles de se produire dans le résultat, ou qu'ils existaient auparavant dans le modèle à modifier. Cela est dû à la construction progressive du résultat, tout en gérant les conflits.

En effet, l'opérateur commence par intégrer les relations du patron dans le résultat de la fusion. Ces dernières étant cohérentes entre elles, le résultat est donc cohérent. Ensuite l'opérateur traite les relations du modèle, une par une, et procède à l'ajout de chaque relation qui ne crée aucun conflit avec celles déjà présentes; sinon l'opérateur procède à la gestion du conflit dans le cas d'une relation susceptible d'en créer. Au final, un résultat cohérent est obtenu.

4.2 Comparaison et analyse

Nos avons donné dans la section 3.2 du chapitre I une synthèse des travaux de réutilisation présentés dans la littérature, par le biais d'une évaluation de ces travaux à travers un cadre de comparaison. Ce cadre nous a fournit les différentes facettes de comparaison ainsi que leurs attributs, que nous allons exploiter dans ce qui suit, pour évaluer notre approche.

a. Représentation des connaissances réutilisables de procédés

Expressivité: Les éléments réutilisables de procédés.
En effet, le méta modèle de notre approche permet de représenter non seulement les éléments de procédés et leurs relations, mais aussi les éléments réutilisables (dans notre cas les patrons), ainsi que la manière de les réutiliser (l'opérateur *ReuseOperator* pour notre cas).

Formalisation: Formel.
Notre méta modèle dispose d'une sémantique précise, et d'une syntaxe définie à travers une représentation graphique des différentes notions de ce méta modèle. Les modèles de procédés modélisés à travers notre plateforme, donc régis par notre modèle et écrits à travers notre syntaxe graphique, peuvent être exploités de façon automatique dans la plateforme. Cependant, l'exécution de ces modèles de procédés n'est pas prévue dans le cadre du présent travail. Ainsi, en faisant abstraction de ce volet d'exécution, nous pouvons donner la valeur Formel à l'attribut *Formalisation*.

b. Organisation des connaissances réutilisables de procédés

Type d'encapsulation de modules de connaissance: Patron de procédés.
En effet, les éléments réutilisables dans notre approche sont les patrons de procédés.

c. Opérateurs de manipulation d'entités réutilisables

Mécanisme de réutilisation: Paramétrisation et Composition.
Notre opérateur de réutilisation procède à une correspondance entre les paramètres du patron (considérés comme paramètres formels), et ceux du modèle à modifier (considérés comme paramètres effectifs). De plus, les connaissances fournies par le modèle à modifier ainsi que par le patron à réutiliser sont associées les unes aux autres de façon cohérente (à travers la procédure de fusion), afin de former le résultat.

Réutilisation pour modifier un modèle existant: Oui.
Il s'agit de la fonctionnalité "pilier" de notre approche.

d. Guidage méthodologique

Type de guidage: Conseils pour la réutilisation.
Nous ne proposons pas une véritable démarche pour mettre en œuvre la réutilisation de patrons pour modifier des modèles de procédés.

e. Outils de support

Catégorie: Environnement.
Notre approche est concrétisée à travers un environnement de réutilisation de patrons, offrant les fonctionnalités basiques de gestion de ces derniers, en plus d'autres fonctionnalités relatives à la modélisation de procédés.

Mode de support: Automatique.
Notre environnement permet la réutilisation de patrons de façon entièrement automatique, qui peut donc se faire sans intervention du concepteur de procédés pour la détection et la résolution de conflits par exemple. Cependant, l'environnement demeure semi automatique sur le choix des substitutions de paramètres et sur la résolution de certaines incohérences, puisque cela nécessite l'intervention du concepteur de procédés.

Nous avons essayé de comparer notre approche avec les travaux présentés dans la section 3 du chapitre I. Les approches qui semblaient appropriées pour la comparaison étaient: l'approche de Tran [TRA07], l'approche de Wang [WAN10] [HEE09] et l'approche de Dam *et al.* [DAM14].

Cependant, l'approche de Wang [WAN10] [HEE09] se base sur la notion de StructureTree, mais ne fournit pas de procédure concrète pour transformer un modèle de procédés en StructureTree et inversement. Son méta modèle ne couvre pas toutes les relations possibles entre éléments de procédés. De plus, l'approche manque d'un algorithme pour concrètement fusionner un patron avec un modèle de procédés. Ces raisons (cf. section 3.1.9 du chapitre I) excluent alors cette approche de la comparaison.

Quant à l'approche de Dam *et al.* [DAM14] qui est dédiée à la fusion des modèles, son inconvénient majeur demeure le fait qu'elle se limite à la fusion des versions issues d'un même modèle d'origine. La raison est que cette approche repose sur le calcul de la différence entre les versions (cf. section 3.3 du chapitre I). De plus, l'approche n'est pas dédiée aux modèles de procédés. Cela signifie que pour pouvoir l'exploiter, elle nécessite la définition d'un ensemble de règles de cohérence et de bonne formalisation, relativement à notre méta modèle, afin que ces règles puissent être vérifiées lors de la fusion. Cette approche n'est donc pas adéquate pour la comparaison.

Nous comparons ainsi notre travail avec l'approche de Tran [TRA07], car cette dernière est la seule dans la littérature qui aborde de la procédure de fusion dans la réutilisation de patrons. Il s'agit d'une approche complète, qui traite deux types de réutilisation: la réutilisation de patrons pour générer le contenu d'un élément de procédés, et la réutilisation de patrons pour enrichir ou pour (ré)organiser un groupe d'éléments de procédés existants. Cette approche est donc concernée par la fusion entre patrons et modèles, et est par conséquent appropriée pour la comparaison.

4.2.1 Description du cas de comparaison

Nous considérons le cas de réutilisation suivant, sur lequel nous appliquons chacune des approches [TRA07] (dans la section 4.2.2) et la notre (dans la section 4.2.3), pour comparer les résultats. Un cas simple est intentionnellement choisi, pour faciliter sa compréhension d'une part et pour une meilleure focalisation sur la réutilisation d'une autre part.

Le modèle de procédés montré par la figure III.9 représente une procédure de révision d'un artefact logiciel. Ce modèle se compose de deux tâches principales, *Analyze* qui permet d'analyser l'artefact, suivi par *Improve* qui permet d'améliorer l'artefact si nécessaire. La tâche *Improve* est l'agrégation de trois tâches, à savoir: *Rework* permettant éventuellement de modifier l'artefact, *Inspect* pour vérifier les éventuelles modifications, et *Evaluate* qui permet d'évaluer l'artefact après des changements éventuels.

Ce modèle de procédés nécessite d'être modifié. En effet, si après évaluation l'artefact s'avère insatisfaisant, la procédure d'évaluation se termine et aucun moyen de retravailler ce même artefact n'est possible. Cela est dû au fait qu'il n'y a aucun moyen de boucler pour relancer la procédure d'évaluation. Le patron *ArtefactRevision* [DAN01] sera alors réutilisé pour pallier à la lacune de ce modèle. Ce patron, simplifié pour le besoin de clarté et montré par la figure III.10, décrit une approche générale pour réviser un artefact. Il se compose principalement des tâches: *Analyze* qui analyse l'artefact, *Improve* permettant de l'améliorer et *Evaluate* pour l'évaluer. Les tâches de ce patron s'agencent les unes après les autres grâce aux relations *TaskPrecedence*. La tâche *Evaluate* peut être suivie par la tâche *Analysze* (pour permettre un retour arrière), dans le cas où l'artefact n'est pas approuvé.

Comme indiqué précédemment, il existe trois modes de réutilisation parmi lesquels deux opèrent une fusion, à savoir les modes *Replace* et *Extend*. Ainsi, nous réutiliserons dans ce qui suit le patron dans chacun de ces deux modes.

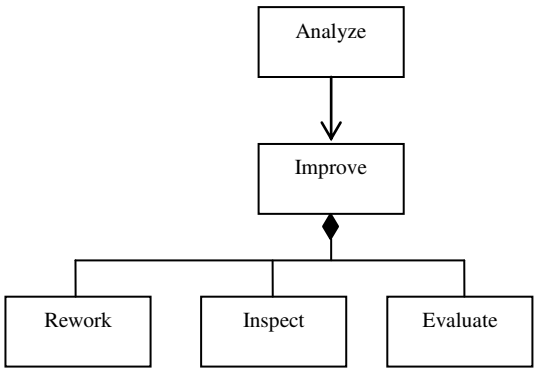


Figure III.9. Le modèle à modifier

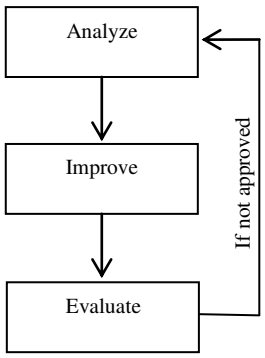


Figure III.10. Le patron *Artefact Revision* [DAN01]

4.2.2 La réutilisation selon l'approche [TRA07]

L'approche [TRA07] propose comme mécanisme de réutilisation l'opérateur *PatternApplying*, qui procède en trois étapes principales: la première étape est la duplication (de la solution) du patron, ce qui donne le résultat montré par la figure III.11. La deuxième étape est l'ajout des éléments du modèle au résultat (figure III.12). La troisième étape est l'ajout des relations du

modèle au résultat. Cependant, cette dernière étape est réalisée indépendamment du mode de réutilisation (*Extend* ou *Replace*), et sans prise en considération des conflits. Par conséquent, toutes les relations du modèle seront intégrées au résultat, comme le montre la figure III.13.

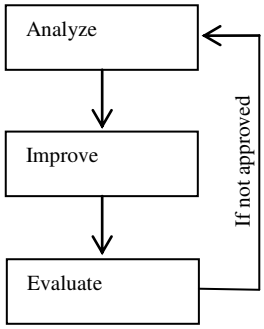


Figure III.11. Resultat de la première étape

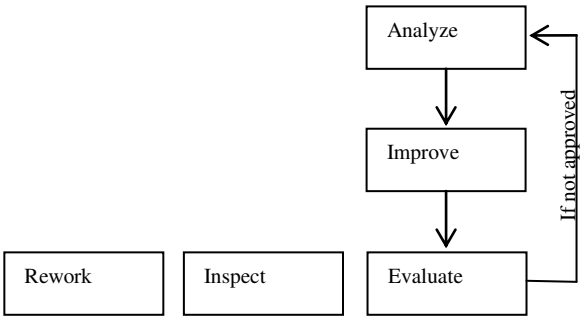


Figure III.12. Resultat de la seconde étape

Évidemment, cette fusion résulte en un modèle de procédés incohérent. En effet, dans le résultat la tâche *Evaluate* succède à la tâche *Improve* (par le biais de la relation *TaskPrecedence*), comme indiqué par le patron. Cependant, la tâche *Improve* est constituée entre autres de la tâche *Evaluate* (par le biais de la relation *Aggregation*), comme indiqué par le modèle. Le problème est que cette relation *Aggregation* est intégrée sans contrôle au résultat. Ce dernier présentera donc un conflit de premier ordre, dû à l'ajout de la relation *Aggregation* entre deux éléments déjà liés par la relation *TaskPrecedence* (*cas 14* de la section 2.2.5).

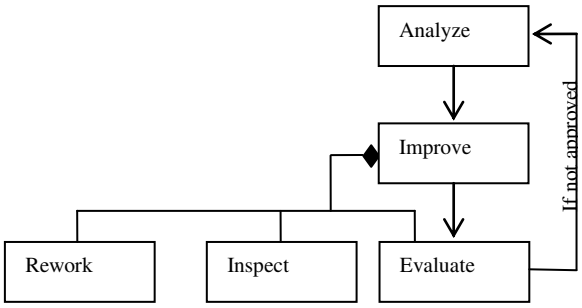


Figure III.13. Le modèle de procédés résultant

4.2.3 La réutilisation selon notre approche

Notre opérateur procède à la fusion comme suit: d'abord, il duplique la solution du patron ce qui donne le résultat montré par la figure III.14. La phase suivante intègre les éléments restants du modèle dans le résultat, comme le montre la figure III.15. Le mode de réutilisation (*Extend* ou *Replace*) est alors pris en considération, pour permettre d'ajouter les relations du modèle au résultat.

Si le mode choisi est *Replace*, alors pour chaque relation à ajouter, notre opérateur vérifie si l'ajout crée un conflit avec les relations déjà existantes dans le résultat ou non. Si la relation à ajouter créera un conflit, elle est alors supprimée (à vrai dire elle ne sera pas ajoutée au

résultat). La Figure III.16 montre le résultat de notre opérateur de réutilisation en mode *Replace*.

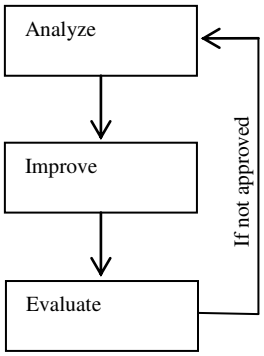


Figure III.14. Le résultat intermédiaire

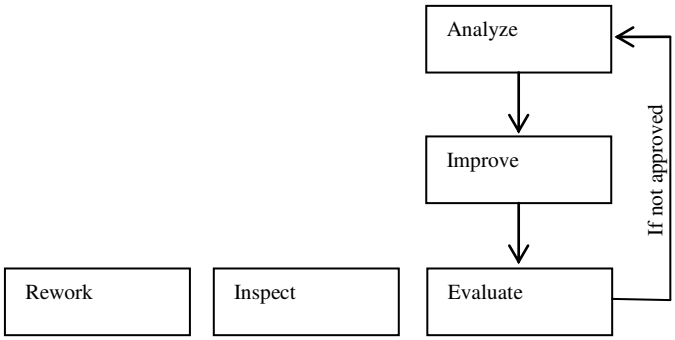


Figure III.15. Les éléménts restants intégrés au résultat

En effet, lors de l'ajout des relations du modèle au résultat, notre opérateur détecte que l'ajout d'une relation *Aggregation* entre *Improve* et *Evaluate* créera un conflit de premier ordre avec la relation déjà existante entre ces deux tâches, à savoir la relation *TaskPrecedence*. Ainsi, comme le mode de réutilisation est *Replace*, qui privilégie en cas de conflit les relations du patron par rapport à celles du modèle de procédés, alors la relation *Aggregation* n'est pas ajoutée. En conséquence, *Improve* consistera uniquement en deux tâches, *Rework* et *Inspect*, tandis que la tâche *Evaluate* succèdera à la tâche *Improve*, ce qui produit à un modèle de procédés cohérent.

Maintenant, étudions le cas où le mode choisi est *Extend*. La politique inverse sera adoptée en cas de conflit; c'est-à-dire que si la relation à ajouter créera un conflit avec une relation déjà existante dans le résultat, alors cette relation déjà existante sera supprimée afin que l'ajout puisse avoir lieu sans problème. La figure III.17 montre le résultat de notre opérateur de réutilisation en mode *Extend*.

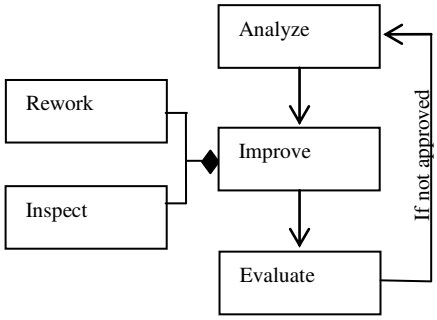


Figure III.16. Le résultat en mode *Replace*

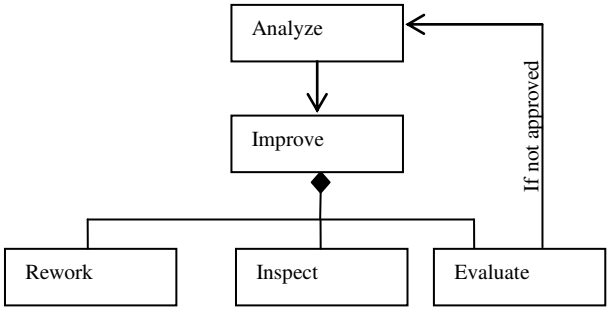


Figure III.17. Le résultat en mode *Extend*

En effet, dans le présent cas notre opérateur détecte que l'ajout de la relation *Aggregation* (relation appartenant au modèle) entre *Improve* et *Evaluate*, provoquera un conflit de premier ordre avec la relation déjà existante entre ces deux tâches, à savoir *TaskPrecedence*. Ainsi, comme le mode de réutilisation est *Extend*, qui privilégie les relations du modèle en cas de

conflit, la relation *Aggregation* sera ajoutée au résultat après suppression de la relation du patron qui provoque le conflit, à savoir *TaskPrecedence* (entre *Improve* et *Evaluate*). En conséquence, d'une part *Improve* consistera en trois tâches: *Rework*, *Inspect* et *Evaluate*, et d'une autre part il n'y aura pas de relation *TaskPrecedence* entre *Improve* et *Evaluate*, ce qui préserve la cohérence du résultat.

5. Conclusion

Dans ce chapitre, nous avons principalement présenté les objectifs de notre approche, relativement à la réutilisation de patrons dans la modification de modèles de procédés déjà existants.
Par ailleurs, nous avons présenté l'étude et l'élaboration de la liste exhaustive des conflits de premier ordre, et avons donné de manière détaillée l'algorithme de réutilisation de patrons, qui prend en charge ces conflits.

La dernière partie de ce chapitre a été dédiée à l'évaluation de notre approche. À cet effet, nous avons présenté la configuration expérimentale utilisée pour l'évaluation, ainsi que la comparaison de notre approche, par rapport aux travaux de l’état de l’art, montrant la valeur ajoutée de l'approche.

CHAPITRE IV

PLATEFORME DE REUTILISATION DE PATRONS

Ce chapitre présente notre plateforme de réutilisation de patrons. Il commence par donner ses fonctionnalités, ainsi que la notation graphique qu'elle utilise (cf. § 1). Ensuite, il présente son architecture de façon détaillée (cf. § 2)**.** L'implémentation de la plateforme est après cela abordée (cf. § 3). Enfin, le chapitre présente une étude de cas à travers cette plateforme (cf. § 4).

1. Description de la plateforme

Notre approche permet de décrire la structure des modèles de procédés et patrons, ainsi que le comportement d'un opérateur de réutilisation. Pour évaluer cette approche, mais aussi pour assister les concepteurs de procédés dans la réutilisation de patrons, nous avons développé une plateforme qui fournit un environnement de mise en œuvre de la réutilisation de patrons pour modifier des modèles de procédés existants.

Cette plateforme offre ainsi différentes fonctionnalités que nous allons détailler dans la suite de cette partie, et se base sur la représentation graphique des notions du méta modèle adopté dans le présent travail. Cette représentation est donnée en détail dans la suite de ce chapitre. La plateforme supporte également la sauvegarde et la restauration de modèles de procédés et de patrons, à travers une base de données dédiée à cet effet.

1.1 Fonctionnalités offertes

Cette section présente les fonctionnalités essentielles proposées par la plateforme, qui sont principalement offertes à travers l'éditeur graphique, mais également à travers l'ensemble d'outils de vérification, de sauvegarde et restauration, de fusion et de réutilisation. Ces fonctionnalités peuvent se résumer dans ce qui suit.

1. *Représentation graphique des modèles et patrons*.
La plateforme propose un éditeur graphique de modèles de procédés logiciels, offrant un ensemble d'outils d'édition. Cet éditeur a pour but d'outiller le langage de modélisation défini à travers le volet structurel de notre méta modèle. L'éditeur permettra ainsi de créer ou de modifier des modèles de procédés et patrons, conformément au méta modèle. La modification consiste en l'ajout, la suppression ou le changement d'éléments ou de relations de procédés.

2. *Sauvegarde et restauration des modèles et patrons*.
La plateforme est dotée d'une base de données, et offre la fonctionnalité de sauvegarde et de restauration des modèles de procédés manipulés ainsi que des patrons. Cette fonctionnalité permet également de créer et de gérer une source de connaissances pour la réutilisation de patrons.

3. *Vérification des modèles de procédés*.
La vérification des modèles de procédés s'effectue principalement vis à vis de la sémantique statique du méta modèle, mais également vis à vis de l'ensemble des incohérences que nous avons définies dans la section 4.5 du chapitre II.
Deux modes de vérification sont possibles: la vérification à la demande qui peut être lancée à tout moment par le concepteur de procédés, et la vérification pendant la modélisation qui est une vérification automatique.

Quant à la vérification des conflits de premier ordre, elle s'applique automatiquement sur le modèle résultant d'une opération de réutilisation de patrons. Les conflits seront

automatiquement résolus pour produire un modèle cohérent. Cette vérification s'appliquera également de façon automatique, lors de l'appel de la fonctionnalité de fusion de deux modèles de procédés.

En plus de cela, la vérification vis-à-vis de la sémantique statique du méta modèle s'exécute automatiquement, au fur et à mesure de la création ou de la modification d'un modèle de procédés, de telle sorte que le concepteur de procédés ne puisse créer, modifier ou supprimer des éléments ou relations de procédés que de manière conforme au méta modèle.

La vérification à la demande concerne les incohérences (cf. section 4.5 du chapitre II), et peut être lancée à tout instant par le concepteur de procédés. Les éventuelles incohérences lui seront alors signalées pour qu'il puisse procéder à leur correction.

4. *Fusion de modèles de procédés*.
La plateforme permet de fusionner deux modèles de procédés. Cette fusion sera automatiquement exempte de conflits de premier ordre.

5. *Réutilisation de patrons*.
La fonctionnalité pilier de la plateforme est la réutilisation de patrons pour modifier des modèles de procédés existants. Cette réutilisation produit un modèle de procédés exempt de conflits de premier ordre.

1.2 Notation graphique et procédure de dessin

La notation graphique utilisée dans la plateforme s'inspire de la notation d'UML. La simplicité étant une exigence majeure pour cette notation, de telle sorte qu'un concepteur de procédés puisse par exemple connaître d'un coup d'œil le nom d'une tâche, ses paramètres d'entrée et de sortie, ses prédécesseurs et successeurs dans le modèle de procédés, ainsi que le rôle responsable de sa réalisation. La procédure de dessin permettant de représenter graphiquement les constituants des modèles de procédés, en se basant sur cette notation graphique, est donnée dans la présente section.

1. Notation des éléments de procédés.
La table IV.1 donne la notation des éléments de procédés adoptée dans la plateforme.

2. Notation des relations de procédés.
La table IV.2 donne la notation des relations de procédés adoptée dans la plateforme.

3. Notation des éléments et relations de contrôle.
La table IV.3 donne la notation utilisée dans la plateforme, pour représenter les éléments et relations de contrôle.

Eléments de procédés	Représentation graphique
Task	
Role	
Product	

Table IV.1. Notation des éléments de procédés

Relations de procédés	Représentation graphique
Aggregation	
Refinement	
TaskPrecedence	
TaskPerformance	
Taskparameter	
ProductResponsability	
ProductImpact	

Table IV.2. Notation des relations de procédés

Eléments et relations de contrôle	Représentation graphique
InitialActivity	
FinalActivity	
Choice	
Link	

Table IV.3. Notation des éléments et relations de contrôle

La procédure de dessin de notre plateforme se base principalement sur la gestion d'un ensemble de rectangles, contenant la représentation graphique des éléments et relations, et regroupés dans une zone de dessin. La zone de dessin, que nous appelons JCanvas, est une sorte de fenêtre sans titre ni bordures, qui sert à regrouper des réctangles dans une zone de l'écran et de les dessiner grâce à la méthode *paint*.

Un rectangle contient de nombreuses méthodes utiles pour la représentation graphique. Nous citons par exemple: *Intersects* qui permet de vérifier si un rectangle est en intersection avec un autre; *IsFree* pour vérifier si une certaine zone du JCanvas est déjà occupée par un objet; *Contains* qui permet de vérifier si le rectangle contient un point donné, etc.

JCanvas dispose d'une liste contenant tous les éléments et relations (rectangles) dessinés dans la zone. La méthode *paint* du JCanvas effectue une itération sur cette liste et appelle la méthode *draw* de chacun de ses éléments. Chaque élément ou relation de procédés doit alors implémenter une interface IDrawable, qui contient la méthode *draw* chargée de dessiner la représentation de l'objet, selon l'implémentation qu'on lui donne.

2. Architecture de la plateforme

Nous commençons par aborder dans cette partie du chapitre la structure générale de notre plateforme. Puis, nous détaillerons l'ensemble des modules qui la constituent.

2.1 Structure générale

La présente section décrit l'architecture générale de notre plateforme, conçue de façon modulaire dans la perspective d'une maintenance évolutive facile. Cette architecture s'inspire du modèle *MVC* (*Model-View-Controller*) [KRA88], et est divisée en six modules collaborant pour assurer les fonctionnalités de la plateforme.

Le module *Noyau* est la concrétisation du méta modèle dans la plateforme. Le module *Outils Graphiques* prend en charge tous les aspects de la représentation graphique dans la plateforme, en plus des interfaces de cette dernière. Le module *Listeners* se charge des écouteurs des évènements produits sur la plateforme. Le module *Réutilisation de Patrons* est responsable des fonctionnalités relatives à la réutilisation de patrons sur des modèles de procédés existants, avec la gestion des conflits de premier ordre. Le module *Incohérences* est le responsable de la gestion des incohérences (autres que les conflits de premier ordre) générées lors de la modélisation de procédés. Enfin, le module *BD* exploite un système de gestion de bases de données, pour gérer une base de données relationnelle chargée de la sauvegarde et de la restauration des modèles de procédés et des patrons.

Le modèle *MVC* permet de séparer méthodiquement les aspects métier (la partie *Model*), des aspects de présentation (la partie *View*), et des liens entre les actions utilisateurs et les traitements métier (la partie *Controller*). L'architecture de notre plateforme qui s'inspire de ce modèle est montrée par la figure IV.1 et présentée dans ce qui suit.

La partie *View* se résume en le module *Outils Graphique*. Ce dernier présente l'ensemble des interfaces graphiques à travers lesquelles le concepteur de procédés interagit avec la plateforme. Les actions du concepteur sont en effet des événements qui se produisent sur l'ensemble des interfaces de la plateforme et qui sont acheminés vers le module *Listeners*. Ce dernier représente la partie *Controller* de l'architecture.

Le module *Listeners* (la partie *Controller*) se charge de lancer le traitement adéquat, selon l'action du concepteur de procédés, en terme de traitements métiers et actions sur la base de données.

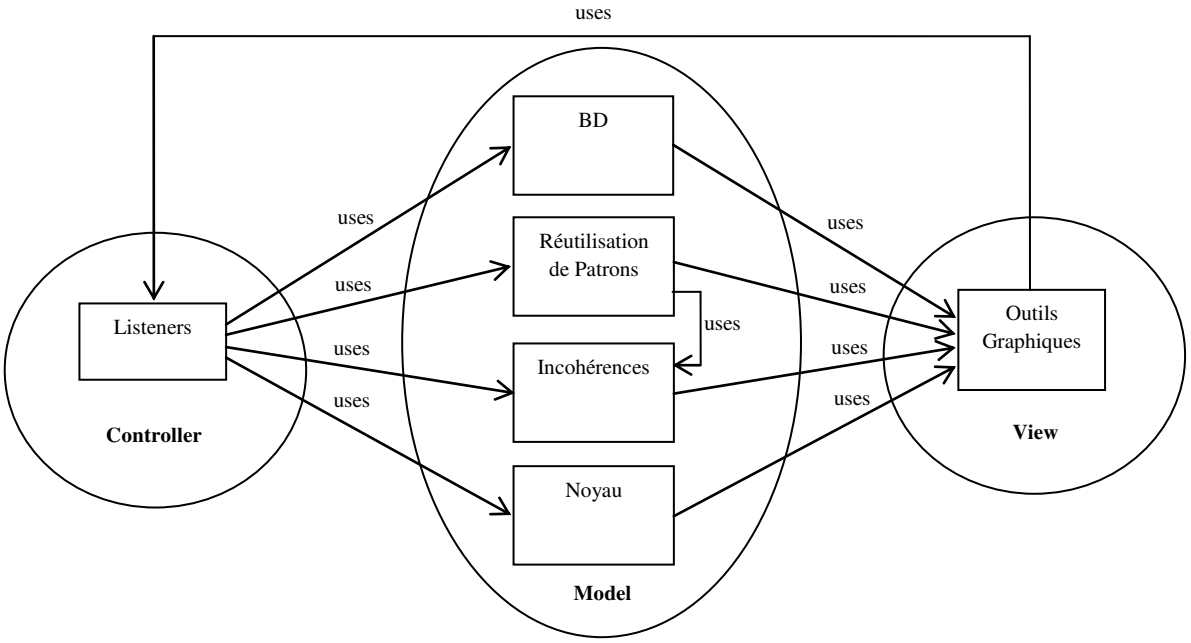


Figure IV.1. Architecture de la plateforme

La partie *Model* de l'architecture se compose des modules *Réutilisation de Patrons*, *Incohérences, Noyau* et *BD*. Ces modules, en dehors de *BD*, se partagent les traitements métiers de la plateforme. Tous les traitements relatifs à la réutilisation de patrons sont assurés par le module *Réutilisation de Patrons*. Ce dernier utilise le module *Incohérences* pour traiter des éventuelles incohérences surgissant lors de la modification d'un modèle de procédés par réutilisation de patrons. Le module *Incohérences* se charge également des traitements d'incohérences de la modélisation de procédés. Le module *Noyau* se charge de tous les traitements relatifs à la sémantique statique du méta modèle. Quant à l'ensemble des données de la plateforme (modèles et patrons), le module *BD* se charge de sa gestion. Enfin, les modifications apportées au niveau de la partie Model sont répercutées sur la partie View pour être restituées (présentées) au concepteur de procédés.

2.2 Ensemble des modules

Les six modules constituant la plateforme sont détaillés dans ce qui suit.

1. Le module *Noyau*: ce module est concrétisé à travers le package Noyau, qui englobe toutes les classes relatives au méta modèle donné dans la section 2 du chapitre II.

2. Le module *Outils Graphiques*: ce module se charge de la partie graphique de la plateforme. Il est constitué d'un ensemble de classes, interfaces et énumérations, dont les plus importantes sont montrées par la figure IV.2. Ces classes implémentent principalement l'interface d'édition et l'ensemble des fenêtres.

Nous avons utilisé le patron de conception *Observer* [GAM95] pour organiser ce module. Dans notre cas, la zone de dessin subit les actions du concepteur de procédés (création, suppression, modification); pour préserver la cohérence du système, l'instance du modèle de procédés en cours de modélisation doit être informée du changement apporté par le concepteur de procédés, afin de se mette a jour. Ainsi, nous avons considéré l'instance du modèle de procédés comme un observateur de l'objet dont l'état change, en l'occurrence la zone de dessin, qui est un observé.

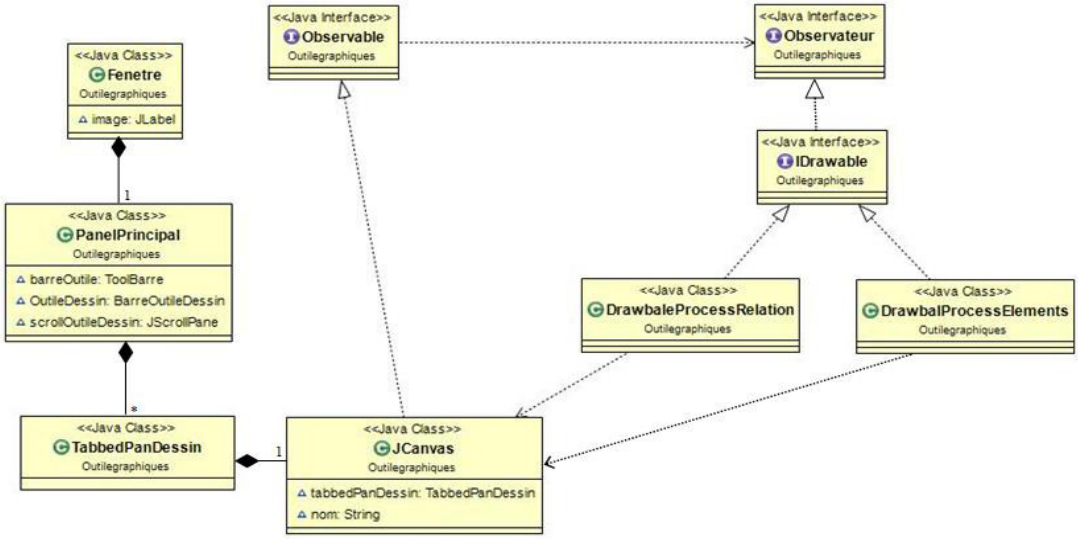


Figure IV.2. Principales classes et interfaces du module *Outils Graphiques*

3. Le module *Listeners*: ce module, montré par la figure IV.3, est dédié aux écouteurs des événements produits par le concepteur de procédés, dans le but de déclencher les traitements adéquats. Ce module est également chargé d'appeler le module de gestion des incohérences, en réponse à certains événements produits par le concepteur de procédés.

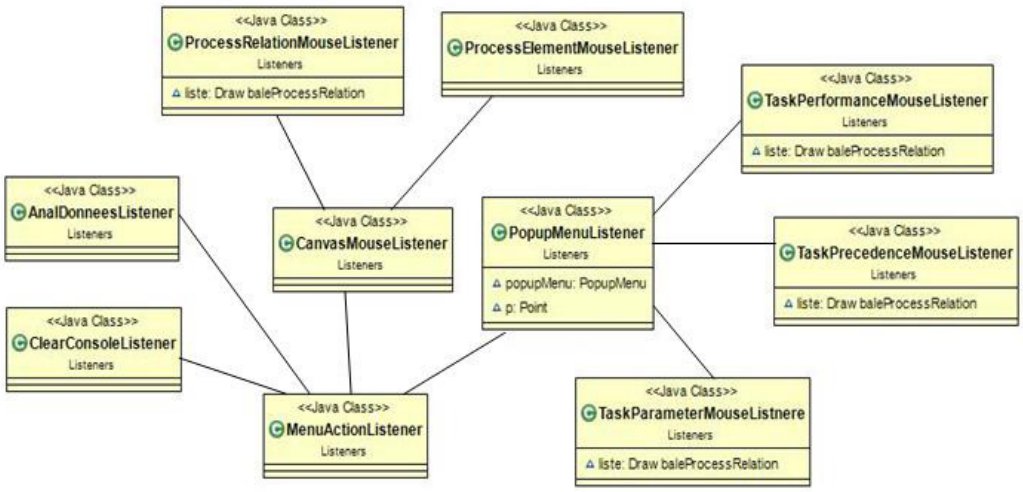


Figure IV.3. Diagramme de classes du module *Listeners*

4. Le module *Réutilisation de Patrons*: ce module est responsable des fonctionnalités relatives à la réutilisation de patrons sur des modèles de procédés déjà existants, avec la gestion des conflits de premier ordre. L'élément pilier de ce module est la classe *ReuseOperator* qui implémente la sémantique opérationnelle de notre opérateur de réutilisation (cf. figure II.2 du chapitre II).

5. Le module *Incohérences*: ce module montré par le diagramme de classes de la figure IV.4 se charge de la gestion d'un ensemble d'incohérences, qui peuvent se présenter lors de la création ou la modification d'un modèle de procédés, et qui ne sont pas des conflits de premier ordre. Ces derniers sont quant à eux traités par le module *Réutilisation de Patrons*, qui en plus de la gestion de ces conflits fait appel au module *Incohérences* pour gérer les incohérences.

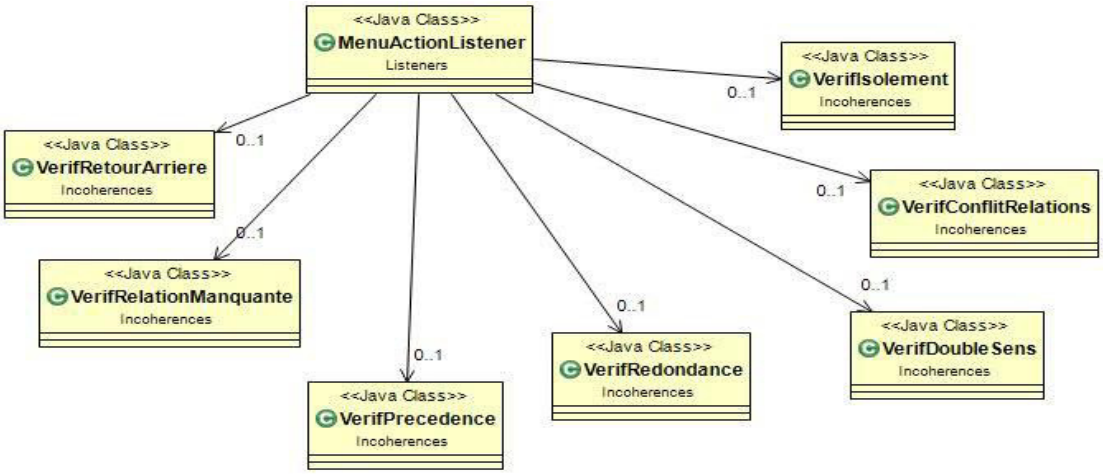


Figure IV.4. Diagramme de classes du module *Incohérences*

6. Le module BD: pour les modèles de procédés et patrons manipulés à travers notre plateforme, nous avons choisi de concevoir une base de données permettant de les sauvegarder et de les restaurer au besoin. Cette base de données relationnelle est conçue à partir d'un modèle objet, qui est le volet structurel de notre méta modèle (cf. figure II.1 du chapitre II), en appliquant les règles de passage objet-relationnel [ROY09].

Nous avons ainsi transformé les classes en relations, ayant l'identifiant de la classe comme clé primaire de la relation (primary key PK), et les propriétés de la classe comme attributs de la relation.

Concernant l'héritage, deux solutions se sont offertes à nous: (a) représenter uniquement les classes filles, par une relation chacune, ou (b) représenter uniquement la classe mère, par une seule relation. Nous avons alors adopté les deux solutions, selon le besoin de chaque partie du volet structurel de notre méta modèle.

D'abord, nous avons deux niveaux d'héritage sur la classe Node, qui possède deux classes filles ProcessElement et ControlNode; chacune possède ses classes filles: Task, Role, Product et InitialActivity, FinalActivity, Choice respectivement. Pour minimiser le nombre d'accès au niveau de la base de données ainsi que le nombre de jointures, qui

sont couteuses, et sachant que les classes filles ont les mêmes propriétés, nous avons choisi de représenter la classe mère Node par une relation contenant toutes les propriétés de ses classes filles, et de ne pas représenter par conséquent les classes ProcessElement, ControlNode, Task, Role, Product, InitialActivity, FinalActivity et Choice.

Quant aux deux niveaux d'héritage de la classe Edge: les deux classes ProcessRelation et ControlFlow héritent de la classe Edge; les classes Task_Performance, Task_Parameter, Task_Precedence, Product_responsabiliy, Aggregation, Product_Impact et Refinement héritent de la classe ProcessRelation; la classe Link hérite de la classe ControlFlow. Pour respecter la troisième forme normale, sachant que chaque classe fille a ses propres caractéristiques, nous avons choisi de ne pas représenter les classes mères Edge, ProcessRelation et ControlFlow. Tous les attributs de ces classes mères, ainsi que la clé primaire de la classe mère Edge, ont été répétés au niveau de chaque classe fille.

Nous avons également transformé les associations entre classes, selon leurs cardinalités. La table dérivée de la classe dont la multiplicité est plusieurs (*) sera considérée comme table fille. En conséquence, une copie de la clé primaire de la mère est déposée dans la table fille à titre de clé étrangère (foroeing key FK).

Dans une association ayant une cardinalité de type 1..1, l'une ou l'autre des tables dérivées des classes peut agir comme table mère. Ainsi, la clé primaire de la table mère sera ajoutée dans la table fille en qualité de clé étrangère.

La composition est traitée comme une association ayant une cardinalité 1..1, avec la table dérivée de la classe composante considérée comme table fille (qui portera comme clé étrangère la clé primaire de la table mère).

3. Mise en œuvre de la plateforme

La réalisation de la plateforme fait l'objet de la présente partie du chapitre. Nous présentons d'abord l'ensemble des outils et langage utilisés, puis nous abordons l'implémentation proprement dite.

3.1 Langages et outils de réalisation

Nous avons développé la plateforme en langage Java [JAV16] sous l'environnement Eclipse [ECL16]. Le choix du langage Java est naturel, vu qu'il facilite l'implémentation du méta modèle conçu selon l'approche orientée objet. Le choix de ce langage a aussi pour but de donner à notre plateforme une portabilité, pour qu'elle puisse s'exécuter sur déférents environnements. Quant au choix d'Eclipse, qui est un environnement de développement intégré libre et polyvalent, il se justifie par le nombre de raccourcis et d'aide à la programmation qu'il offre, ainsi qu'à la prise en charge du langage Java et de l'ensemble de ses API.

Nous avons exploité principalement des bibliothèques Java, qui nous ont facilité la réalisation de la partie graphique de la plateforme. Nous avons également exploité JDBC, qui est une API permettant aux applications Java d'accéder par le biais d'une interface commune à des sources de données, afin de communiquer avec notre base des modèles et patrons de procédés.

Pour cette base nous avons opté pour Oracle Database, qui est un système de gestion de base de données relationnel offrant un large ensemble de fonctionnalités. Il propose une version totalement gratuite *Oracle Database Express Edition*, que nous avons utilisée.

L'environnement de développement et de test utilisé pour le développement de la plateforme est le Java Runtime Environment JRE version 8 mise à jour 65 (build 1.8.0_65-b17).

3.2 Implémentation de la plateforme

Cette section décrit l'implémentation des modules de la plateforme, à travers l'ensemble suivant de packages, que montre la figure IV.5.

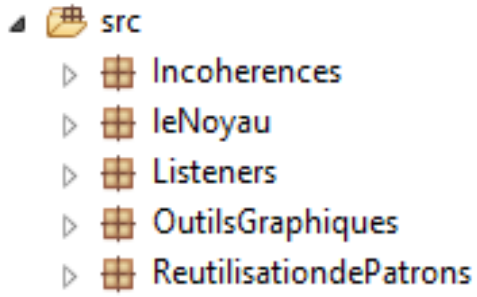


Figure IV.5. Ensemble des packages de la plateforme

Le module *Noyau* est implémenté à travers le package leNoyau. Ce dernier est constitué d'un ensemble de classes et énumérations Java montrées par la figure IV.6, qui correspondent principalement aux classes et énumérations de notre méta modèle.

En guise d'illustration, la figure IV.7 présente les fichiers Java du module *Outils Graphiques*. Ces fichiers sont des classes, interfaces et énumérations implémentant principalement l'interface d'édition et l'ensemble des fenêtres, et sont regroupés dans le package OutilsGraphiques.

La fenêtre principale de l'éditeur graphique (la classe Fenetre) est composée d'un panneau principal, lui même composé d'un onglet de dessin (la classe TabbedPaneDessin), dont la zone de dessin est représentée par la classe JCanvas. Nous avons intégré la partie chargée de la connexion avec la base de données dans la classe Fenetre, afin d'assurer une connexion globale qui sera instanciée une seule fois au lancement de la plateforme.

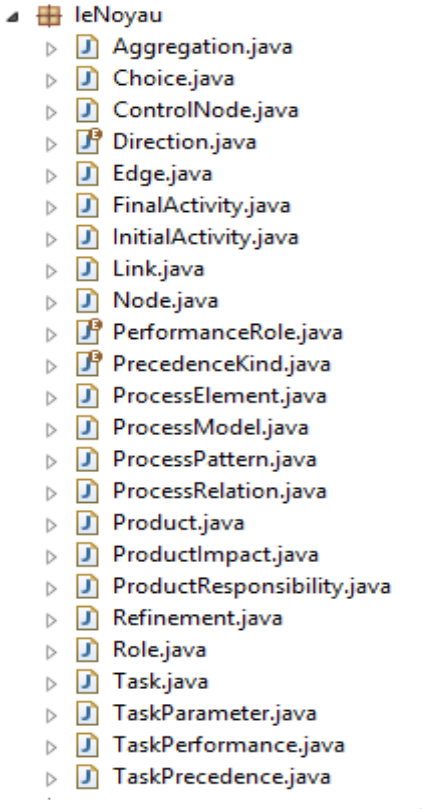


Figure IV.6. Contenu du package leNoyau

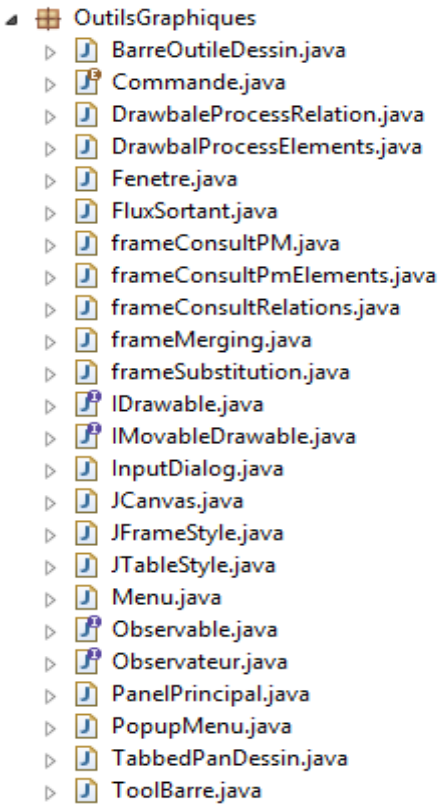


Figure IV.7. Contenu du package OutilslsGraphiques

Le package Listeners montré par la figure IV.8 implémente le module de même nom. Il est constitué d'un ensemble de classes, principalement chargées de la gestion des événements de l'utilisateur de notre plateforme.

Par exemple, les traitements en réponse aux événements des menus sont principalement regroupés dans la classe MenuActionListner, et distingués selon la valeur d'ActionCommand. On site également par exemple la partie de la sauvegarde et de la restauration des modèles de procédés qui est intégrée dans cette classe, avec ActionCommand égale à Sauvegarder ou Restaurer. Pour la duplication d'un modèle de procédés par exemple, ActionCommand est égale à Dupliquer. Pour l'impression des modèles de procédés, ActionCommand est égale à Imprimer.

Le package ReutilisationdePatrons quant à lui implémente le module *Réutilisation de Patrons*, et est montré par la figure IV.9. Ce package contient ente autres la classes *ReuseOperator*, qui possède la méthode *Merge()*. Cette dernière permet d'effectuer la fusion des modèles de procédés tout en gérant les conflits de premier ordre, ainsi que d'autres incohérences par appel au module *Incohérences*.

L'opérateur de réutilisation (cf. 2.4 du chapitre II) agit sur deux modèles de procédés à fusionner, qui sont le modèle de procédés à modifier et la solution du patron à réutiliser. Le mode de réutilisation est indiqué par une valeur de l'énumération *Mode*, affectée à l'attribut

ReuseMode, et un vecteur de *Substitution* contenant au moins une substitution de paramètres, est également nécessaire. Une substitution est composée d'un paramètre formel *FormelParameter* qui est un élément du patron, et un paramètre effectif *ActualParameter* qui est un élément du modèle de procédés; ces deux paramètres doivent être compatibles (de même type).

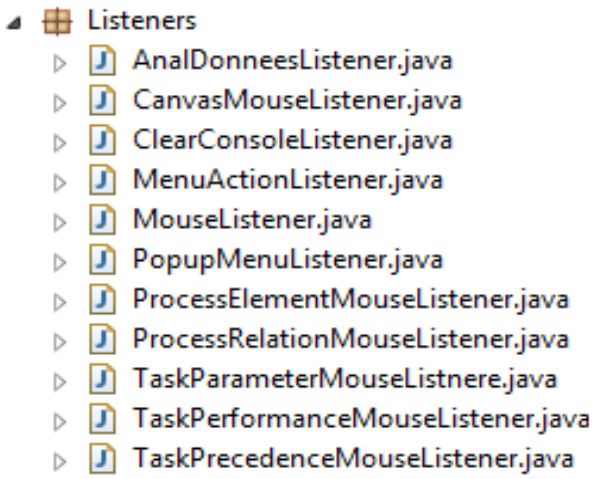


Figure IV.8. Contenu du package Listeners

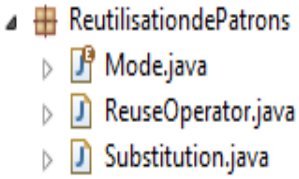


Figure IV.9. Contenu du package ReutilisationdePatrons

La figure IV.10 présente la package Incoherences qui implémente le module de même nom. Ce package regroupe les classes Java implémentant les différentes vérifications à opérer sur un modèle de procédés. L'instanciation des objets de différents classes de ce package ainsi que l'appel de leurs méthodes, se fait par les différentes classes du module *Listeners*, mais également celles du module *Reutilisation de Patrons*.

La dépendance entre le package Incohérences et le package OutilsGraphiques est due au fait que chaque incohérence détectée est rapportée au concepteur de procédés, soit pour qu'il lui apporte une correction soit juste pour l'informer.

Le module *BD* est concrétisé à travers une base de données relationnelle. Le résultat de l'application des règles de passage objet-relationnel sur le modèle objet de notre plateforme (cf. 2.3 du chapitre II) est le schéma relationnel donné dans ce qui suit, dont la décomposition respecte la troisième forme normale. L'ensemble des tables de cette base est présenté par la figure IV.11.

Process_Model (id_ProcessModel (PK), DateCreation, DateMAJ, NbModelR);

Process_Pattern (id_ProcessPattern (PK), DateCreation, DateMAJ, NbPatternR);

Node (id_Node (PK), ContentE, TypeE, PosX, PosY, id_ProcessModel (FK));

Task_Performance (id_Edge (PK), Performer_R (FK), Performed_T (FK), NatureR, PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Task_Parameter (id_Edge (PK), id_Task (FK), id_Product (FK), Direction, TypeProd, PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Task_Precedence (id_Edge (PK), Predecesseur (FK), Successeur (FK), TypePrec, PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Product_responsabiliy (id_Edge (PK), id_Role (FK), id_Product (FK), PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Product_Impact (id_Edge (PK), ImpactingE (FK), ImpactedE (FK), PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Aggregation (id_Edge (PK), Aggregate (FK), component (FK), PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Refinement (id_Edge (PK), OriginalE (FK), RefinedE (FK), PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

Link (id_Edge (PK), IdDepart (FK), IdArrivee (FK), Condition, PosXD, PosYD, PosXA, PosYA, id_ProcessModel (FK));

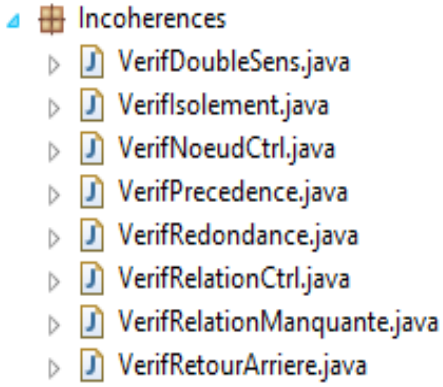


Figure IV.10. Contenu du package Incohérences

Figure IV.11. Ensemble des tables de la base de données

4. Étude de cas à travers la plateforme

Nous présentons dans ce qui suit un cas de réutilisation d'un patron pour modifier un modèle de procédés. Nous commençons par décrire ce cas, ensuite nous présentons son déroulement de manière détaillée.

4.1 Description du cas

Dans le but d'illustrer aussi bien la réutilisation de patrons pour modifier des modèles de procédés que les autres fonctionnalités offertes par la plateforme, nous nous appuyons sur la présente étude de cas, qui se résume en la modification du modèle de procédés *Inception phase* [BEN09] par la réutilisation du patron *FeedbackDevelopment* [TRA07a].

Le modèle *Inception phase* montré par la figure IV.12, est un modèle de procédés fourni par le partenaire industriel du projet européen Modelplex [IST08]. Il est constitué de trois tâches consécutives: *Elaborate UML Analysis Model* (que nous abrégerons dans ce qui suit en *Elaborate UAM*), *Validate UML Analysis Model* (que nous abrégerons en *Validate UAM*), et *Send Email*. La tâche *Elaborate UAM* a pour but de réaliser une analyse du système, et de produire le document d'analyse correspondant. La tâche *Validate UAM* vérifie le document de l'analyse en vue d'une validation. Quant à la tâche *Send Email*, elle se charge de décrire l'état d'achèvement des tâches précédentes.

Ces tâches sont réalisées par le rôle *Analyst*, qui est *Performer* pour chacune d'entre elles. *Elaborate UAM* recevra en entrée le produit *Request Document* (que nous abrégerons en *Req Doc*), et fournira en sortie le produit *UML Analysis Model* (que nous abrégerons en *UAM*). Ce dernier sera en entrée de la tâche *Validate UAM*.

Cependant, il est impossible dans ce modèle de procédés de retourner vers la tâche *Elaborate UAM*, pour apporter des modifications sur le produit *UAM*, dans le cas où celui ci ne passe pas avec succès la tâche *Validate UAM*. Dans un tel cas, le rôle *Analyst* se contente d'exécuter la tâche *Send Email*, informant de l'échec de la validation du produit *UAM*. Il s'agit là d'une lacune majeure du modèle *Inception phase*; il est donc intéressant de pouvoir retourner de la tâche *Validate UAM* vers la tâche *Elaborate UAM*, dans le cas où des changements s'avèrent nécessaires sur le produit *UAM* (qui n'a pas été validé).

La réutilisation du patron *FeedbackDevelopment* est appropriée pour palier à cette lacune. En effet, ce patron montré par la figure IV.13 a pour but de prendre en considération le feedback donné sur un produit, lors d'une procédure de développement. Ce patron est constitué de trois tâches consécutives *Task1*, *Task2* et *Task3*, où la tâche *Task2* succède à *Task1*. Un produit *Product1* en sortie de *Task1*, est un paramètre d'entrée de *Task2*. Après la tâche *Task2*, un choix est proposé: si des changements sont nécessaires sur le produit alors on retourne à la tâche *Task1*, sinon la tâche *Task3* est réalisée.

La réutilisation du patron *FeedbackDevelopment* pour modifier le modèle de procédés *Inception phase* permettra à ce dernier de bénéficier d'un retour arrière vers la tâche *Elaborate UAM*, si besoin de changement y est sur le produit *UAM*. Ce cas de réutilisation sera détaillé dans la suite de cette section.

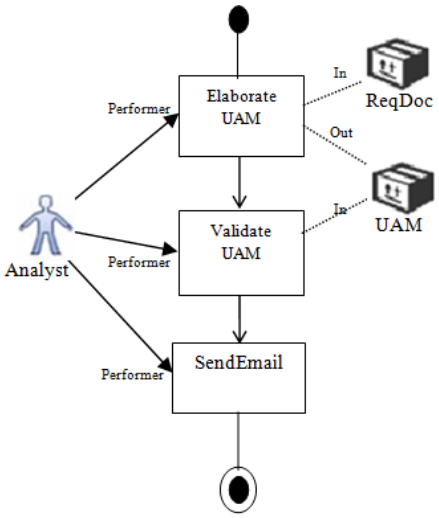


Figure IV.12. Le modèle *Inception phase*

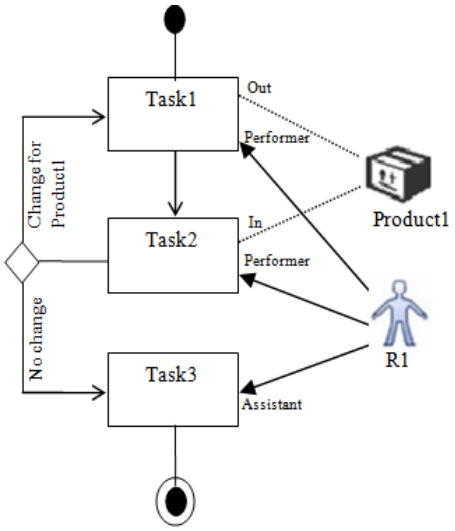


Figure IV.13. Le patron *FeedbackDevelopment*

4.2 Déroulement du cas

La création du modèle de procédés *Inception phase* (comme tout autre modèle de procédés dans notre plateforme) se fait à travers un ensemble d'outils d'édition. Ces outils permettent entre autres de créer, modifier, déplacer, dupliquer, supprimer des éléments de procédés, des relations de procédés, des éléments et relations de contrôle, ainsi que des modèles de procédés et patrons. Des outils d'affichage sont aussi disponibles dans la plateforme, pour faciliter les opérations d'édition. L'ensemble des outils est fourni à travers la barre d'outils, les menus ainsi que le panneau de création d'éléments et de relations de procédés et de contrôle (à gauche de l'espace d'édition). La figure IV.14 montre la création du modèle de procédés *Inception phase*, à travers l'interface d'édition de la plateforme.

Lors de la création du modèle, une erreur s'est glissée ! Le produit *UAM* en sortie de la tâche *Validate UAM* est mis en entrée de la tâche *Elaborate UAM*; chose qui est impossible, vu que la tâche *Elaborate UAM* précède la tâche *Validate UAM*.

Cette incohérence, produite par inattention par le concepteur de procédés, est interceptée par l'outil de détection des incohérences et est signalée au concepteur, comme montré par la figure IV.15 (message en bas de l'interface d'édition).

Nous notons que les éventuels conflits de premier ordre engendrés par la réutilisation de patrons seront également interceptés de cette même manière. Cependant, ils ne seront pas signalés au concepteur de procédés, mais seront plus tôt gérés par l'opérateur de réutilisation.

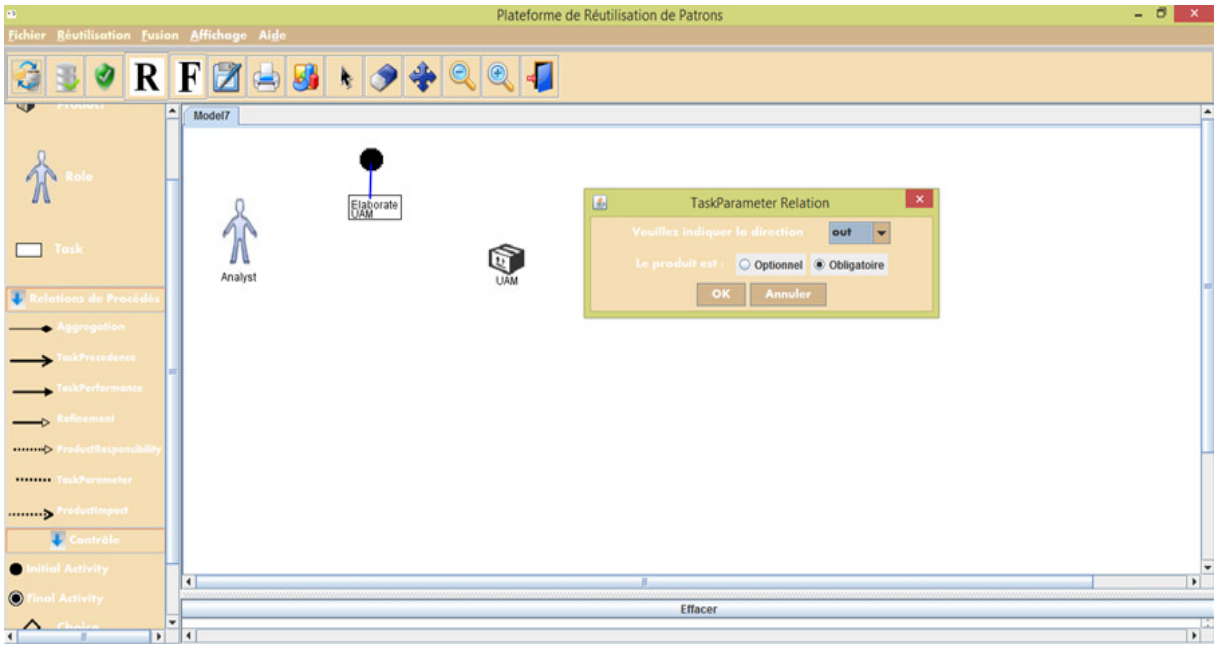


Figure IV.14. Interface d'édition de modèles de procédés

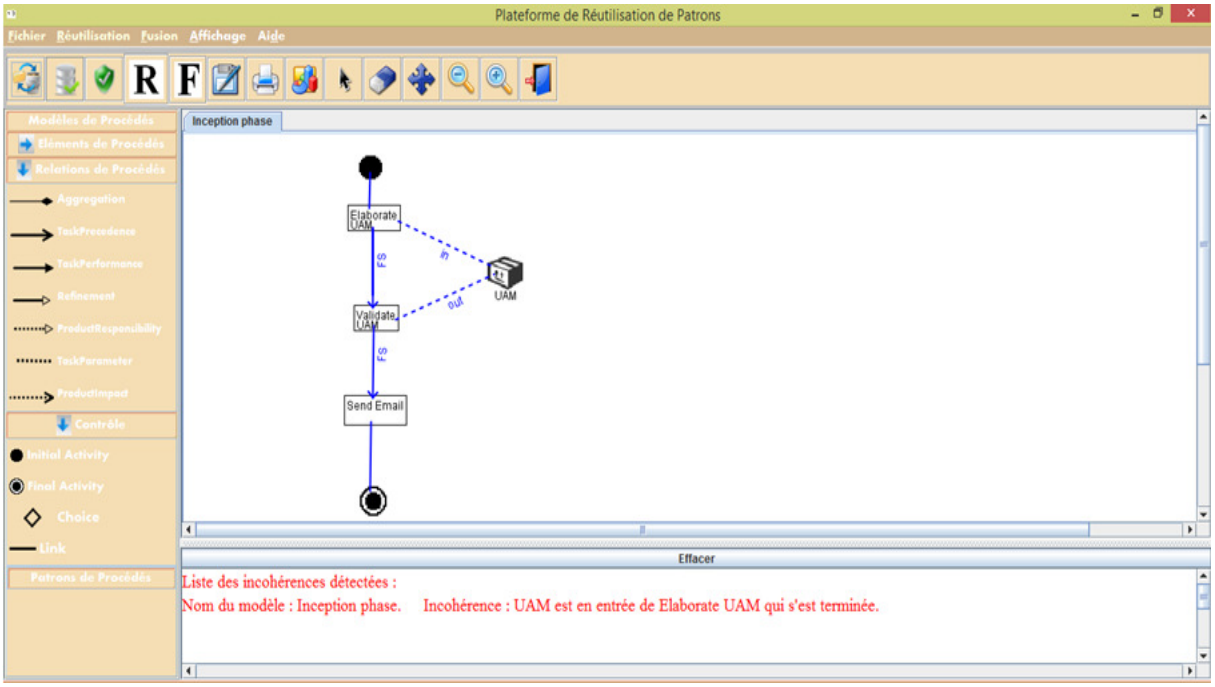


Figure IV.15. Détection des incohérences

Le modèle *Inception phase* (comme tout autre modèle de procédés dans notre plateforme) peut à tout moment être sauvegardé au niveau de la base de données. Il pourra également être restauré à tout moment de cette base, pour être exploité ou pour subir des modifications. La figure IV.16 montre les outils de sauvegarde et de restauration de modèles de procédés.

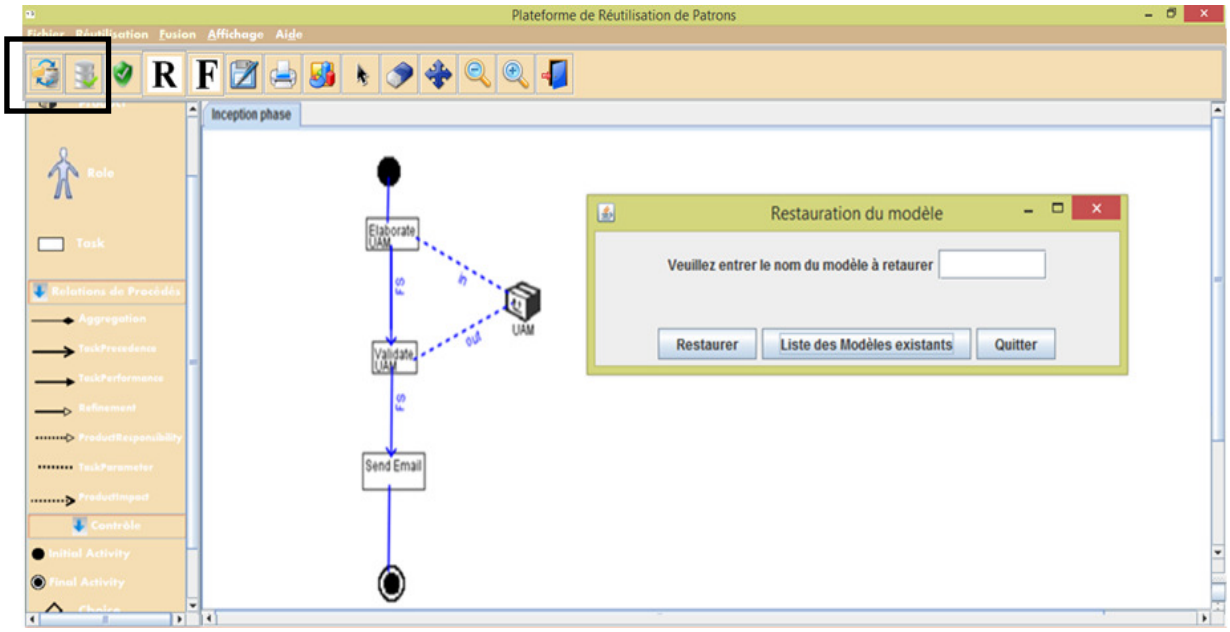


Figure IV.16. Outils de sauvegarde et de restauration de modèles

La création du patron *FeedbackDevelopment* dans notre plateforme (comme tout autre patron) se base principalement sur la création d'un modèle de procédés, qui représentera la solution du patron. Une fois la solution du patron créée et sauvegardée dans la base, elle sera juste sélectionnée dans le champ *Solution* lors de la création du patron, en plus du nom du patron qui sera renseigné, comme le montre la figure IV.17. Le champ Solution est en effet une liste déroulante des modèles de procédés disponibles dans la base.

Notons que la sauvegarde et la restauration des patrons dans la plateforme est semblable à celle des modèles de procédés.

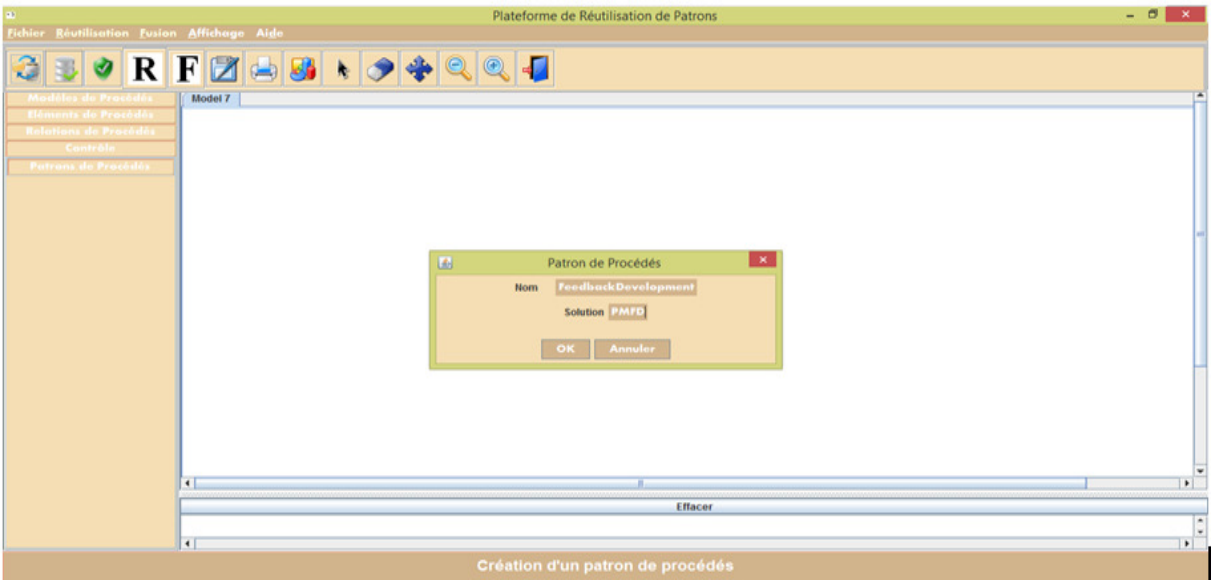


Figure IV.17. Création d'un patron de procédés

La réutilisation d'un patron pour modifier un modèle de procédés, montrée par la figure IV.18, s'effectue par la sélection: du modèle à modifier, du patron à réutiliser, et du mode de réutilisation. Une fois la sélection validée, la substitution de paramètres est lancée. Cette substitution permet de faire remplacer certains éléments du patron (ou éventuellement tous les éléments du patron) considérés comme paramètres formels, par des éléments du modèle qui leurs correspondent et considérés comme paramètres effectifs.

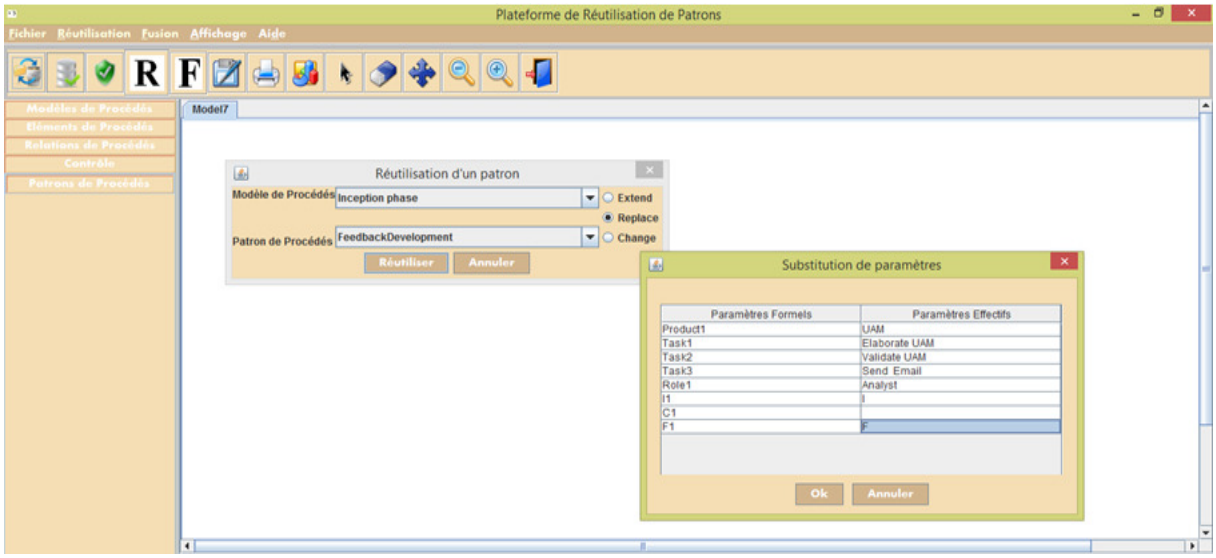


Figure IV.18. Réutilisation d'un patron

Dans notre cas, les paramètres du patron *FeedbackDevelopment* vont correspondre à ceux du modèle *Inception phase*, dans l'ordre décrit dans la table IV.4.

Eléments du patron *FeedbackDevelopment*	**Eléments du modèle *Inception phase***
Product1	UAM
Task1	Evaluate UAM
Task2	Validate UAM
Task3	Send Email
Role1	Analyst
I1	I
F1	F
C1	Aucun élément dans le modèle

Table IV.4. Correspondance de paramètres entre modèle et patron

Cette réutilisation va engendrer un conflit, qui sera pourtant intercepté et géré par notre opérateur. En effet, la substitution de paramètres dans le présent cas fait correspondre le rôle *Analyst* du modèle au rôle *Role1* du patron, et la tâche *Send Email* du modèle à la tâche *Task3* du patron. Cependant, *Analyst* est *Performer* pour *Send Email* dans le modèle à modifier, alors que *Role1* et *Assistant* pour *Task3* dans le patron à réutiliser. Cela crée le conflit où un rôle est en même temps *Performer* et *Assistant* pour la même tâche (cf. 2.2.4 du chapitre III); mais ce conflit sera intercepté et sera traité selon le mode de réutilisation sélectionné.

La figure IV.19 montre le résultat de la réutilisation du patron *FeedbackDevelopment* sur le modèle de procédés *Inception phase*, en mode *Extend*. Ce mode favorise le modèle à modifier en cas de conflit, donc la relation entre le rôle *Analyst* et la tâche *Send Email* sera *Performer*, et le résultat de la réutilisation sera ainsi exempt de conflits.

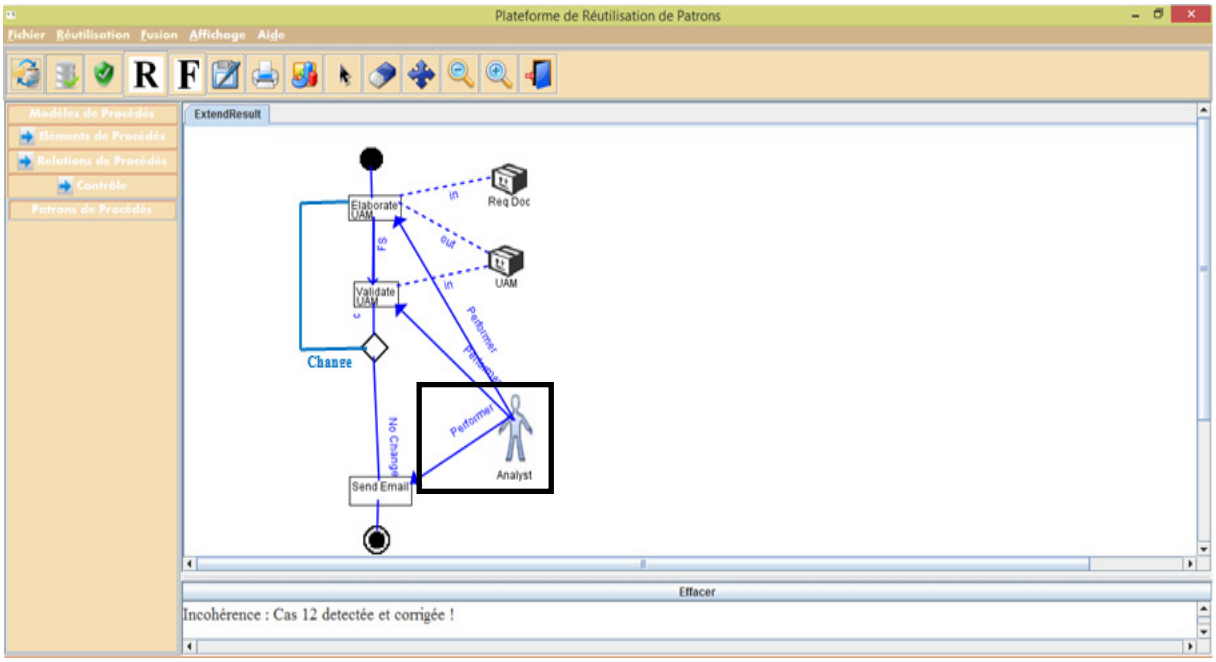


Figure IV.19. Résultat de la réutilisation en mode *Extend*

La figure IV.20 montre le résultat de la réutilisation en mode *Replace*. Ce mode favorise le patron en cas de conflit, donc la relation entre le rôle *Analyst* et la tâche *Send Email* sera *Assistant*, et le résultat de la réutilisation sera ainsi exempt de conflits.

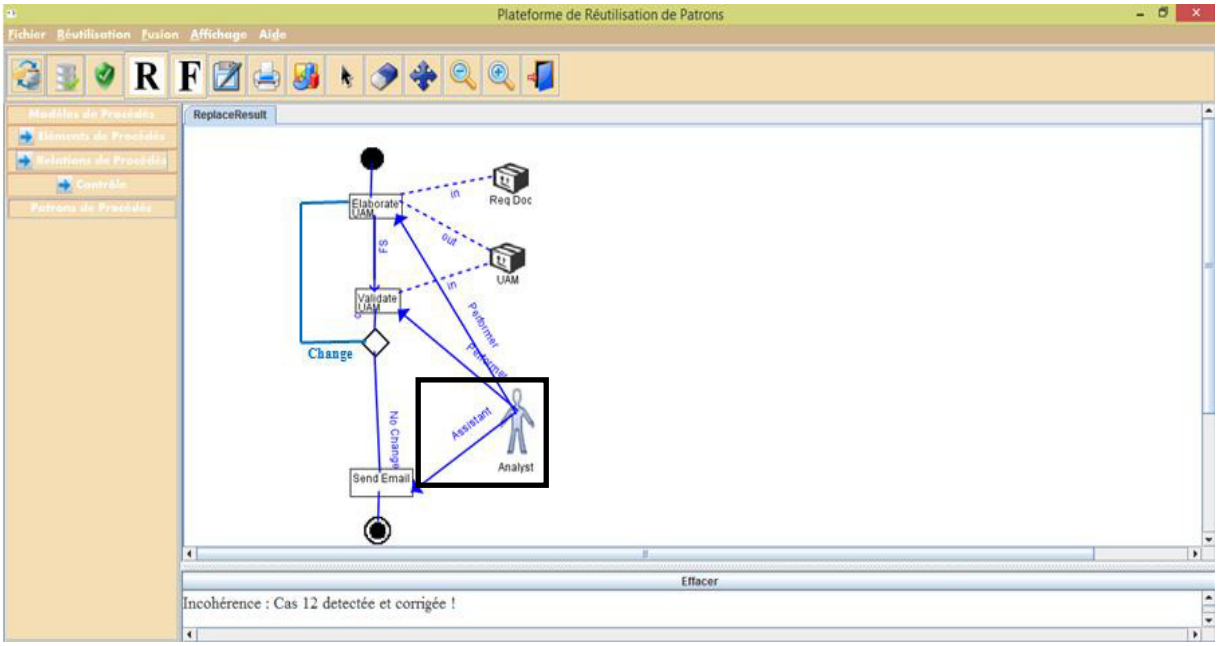


Figure IV.20. Résultat de la réutilisation en mode *Replace*

Le résultat de la réutilisation est ainsi un modèle de procédés cohérent, qui représente le modèle initial mais avec sa lacune palliée. En effet, le modèle résultant offre un retour arrière vers la tâche *Elaborate UAM*, dans le cas où des changements s'avèrent nécessaires sur le produit *UAM*.

5. Conclusion

Nous avons présenté dans ce chapitre la plateforme qui constitue une implémentation de notre approche. Nous avons d'abord présenté cette plateforme en termes de fonctionnalités offertes et de notation graphique utilisée, puis nous avons présenté son architecture. Cette architecture comporte six modules, collaborant pour assurer les fonctionnalités ciblées.

La plateforme a été réalisée par le biais d'un ensemble d'outils, et un langage orienté objet facilitant l'implémentation du méta modèle, lui même conçu selon l'approche orientée objet. Cette plateforme a été utilisée dans l'évaluation de notre approche, et dans l'exécutions de différents exemples, entre autres ceux présentés dans ce document, mais également pour illustrer l'étude de cas présentée dans ce chapitre. Cette étude de cas a porté sur la réutilisation du patron *FeedbackDevelopment* pour modifier la modèle de procédés *Inception phase*; une réutilisation susceptible de générer un conflit, qui a pu être intercepté et traité par notre opérateur de réutilisation.

CHAPITRE

CONCLUSION GENERALE

Dans ce chapitre final, nous dressons un bilan des matières abordées (cf. I), et donnons une analyse critique des principales contributions de la présente thèse (cf. II). Enfin, nous énumérons quelques perspectives qui pourront êtres poursuivies dans de futurs travaux de recherche (cf. II).

I. Synthèse des matières abordées

L'étude bibliographique effectuée dans le présent travail de thèse montre que les patrons offrent plusieurs avantages à la communauté des procédés logiciels. Diverses thématiques de recherche ont donc abordé ces patrons, telles que: la modélisation des patrons qui prend deux axes, les formalismes pour la représentation des patrons, et les langages de description de patrons logiciels. L'identification des patrons qui consiste à cibler les sources de connaissances contenant des problèmes récurrents, pour ensuite spécifier les solutions à offrir par ces patrons. L'organisation des patrons qui cible principalement les relations entre patrons. La réutilisation de patrons qui permet de tirer profit des connaissances offertes par les patrons.

L'étude des travaux de recherche sur la réutilisation de patrons montre que la problématique de réutilisation prend deux aspects différents: la gestion des patrons qui serviront à modéliser des procédés, et la gestion des modèles de procédés créés en réutilisant des patrons. Ce second aspect inclut la création d'un modèle complet en réutilisant un patron, la création d'une partie seulement d'un modèle en réutilisant un patron, et la modification d'un modèle ou d'une partie d'un modèle en réutilisant un patron. L'étude montre aussi que la réutilisation de patrons pour modifier des modèles de procédés est une forme de réutilisation encore négligée. En effet, cette forme se base principalement sur un mécanisme de fusion, qui agit sur le modèle de procédés à modifier aussi bien que sur le patron à réutiliser.

Avant de pouvoir découvrir de plus près cette forme de réutilisation avec son mécanisme de fusion, le choix du méta modèle pour représenter patrons et modèles s'est imposé. Le méta modèle adopté dans le présent travail a été choisi après plusieurs observations des méta modèles présentés dans la littérature. En effet, plusieurs travaux de recherche ont classifié les constituants des modèles de procédés, et ont montré que les plus importants sont les tâches, les produits et les rôles. Notre méta modèle s'est basé par conséquent sur ses constituants là, et sur les relations nécessaires pour les associer; étant basé sur des notions communément adoptées, ce méta modèle est ainsi facile pour la compréhension et pour l'exploitation.

La fusion impliquée par la réutilisation de patrons pour modifier un modèle de procédés est susceptible de créer des conflits dans le modèle résultant, qui peuvent survenir à plusieurs niveaux selon les relations impliquées. Pour mieux maitriser ces conflits, nous les avons classés en niveaux, à savoir: le premier niveau (conflits de premier ordre), le deuxième niveau (conflits de second ordre), le troisième niveau (conflits de troisième ordre) ainsi que le niveau des conflits de cycles indirects.

Nous nous sommes intéressés dans le présent travail aux conflits de premier ordre, causés par des relations contradictoires reliant de façon directe un même couple d'éléments de procédés. Nous avons exhaustivement identifié les conflits de premier ordre, en analysant toutes les combinaisons de relations possibles entre un couple d'éléments de procédés, et ceci pour tous les couples d'éléments. Notre étude nous a permis d'identifier la liste des dix-huit conflits de premier ordre, dus principalement aux problèmes suivants: les cycles directs, les conflits entre les relations Aggregation et Refinement, les conflits de la relation TaskParameter, les conflits

de la relation TaskPerformance, les conflits entre les relations Aggregation et TaskPrecedence, les conflits entre les relations Refinement et TaskPrecedence, et enfin les conflits entre les relations Refinement et ProductImpact. Tous ces conflits sont détectés et résolus lors de la réutilisation de patrons, grâce à notre opérateur de réutilisation.

L'algorithme donné dans le présent travail fournit la sémantique opérationnelle de cet opérateur de réutilisation. Il produit comme résultat un modèle de procédés provenant de la fusion du modèle de procédés à modifier avec la solution fournie par le patron réutilisé, tout en prenant en considération le mode de réutilisation choisi par le concepteur de procédés, et tout en résolvant les conflits de premier ordre.

Nous avons évalué notre approche, ensuite nous l'avons comparée par rapport à l'unique approche dans la littérature qui traite de la procédure de fusion dans la réutilisation de patrons. À cet effet, nous avons abordé une étude de cas en appliquant les deux approches, et en comparant les résultats obtenus par chacune.

Enfin, notre approche a été concrétisée à travers une plateforme mettant en œuvre la réutilisation de patrons, qui se focalise sur la réutilisation de patrons dans la modification de modèles de procédés existants. La plateforme se base sur la représentation graphique des différentes notions du méta modèle, et exploite une base de données pour la sauvegarde et la restauration des modèles de procédés ainsi que des patrons.

II. Analyse des contributions

Au cours de la modélisation de procédés, les concepteurs de procédés ont parfois besoin de restructurer ou d'enrichir un modèle existant, afin de le corriger, de satisfaire certaines contraintes ou d'augmenter son efficacité. L'opération de réutilisation de patrons pour modifier des modèles de procédés existants répond à ce besoin, en fournissant un mécanisme de fusion qui permet de restructurer ou d'enrichir un modèle de procédés conformément à un patron.

Notre travail s'inscrit dans ce contexte et propose une approche automatique, implémentant une procédure de fusion dans la réutilisation de patrons pour modifier des modèles de procédés. Le présent travail commence par établir un état de l'art sur les modèles de procédés, les patrons et la réutilisation de patrons, pour ensuite étudier la problématique de réutilisation de patrons dans la modification de modèles de procédés. Puis, un méta modèle est choisi pour régir les modèles de procédés et patrons. Après cela, les conflits dus à la réutilisation de patrons sont abordés et sont classés en niveaux, et les conflits de premier ordre sont particulièrement et exhaustivement étudiés. Un opérateur de réutilisation de patrons pour modifier des modèles de procédés avec la gestion des conflits de premier ordre est proposé, et l'approche est mise en œuvre à travers une plateforme. Enfin, une évaluation montrant la valeur ajoutée de notre approche est établie.

Le présent travail de thèse se classe parmi le peu de travaux qui traitent la réutilisation de patrons avec fusion de modèles et gestion de conflits. Nous avons procédé à l'évaluation de

notre approche et l'avons comparée par rapport à l'approche la plus complète du domaine. Dans cette dernière, les conflits engendrés par la fusion ne sont pas gérés, ce qui peut produire des modèles de procédés incohérents. En revanche, notre approche traite les conflits de premier ordre, afin d'aboutir à un résultat cohérent.

Arrivés au terme de ce travail, nous voulons discuter de certains points qui y sont liés. D'abord, nous avons considéré dans notre approche chaque patron à réutiliser comme étant cohérent, puisque "les patrons offrent par définition des solutions prouvées" [HAG04]. Pour cette raison, aucune vérification n'est appliquée sur un patron et il est réutilisé tel quel. Nous avons considéré également que le modèle de procédés à modifier est cohérent. Par conséquent, aucune vérification n'est effectuée sur le modèle avant la réutilisation. Toutefois, si un conflit existe dans ce modèle de procédés, il sera détecté au cours de la procédure de fusion et sera traité exactement comme un conflit causé par la fusion.

Nous avons choisi dans notre approche de considérer le cycle TaskPrecedence comme un conflit. Cependant, ce cycle peut en fait être un conflit, comme il peut aussi bien être un retour arrière effectué exprès; le concepteur de procédés est le seul capable de décider si un tel cycle est un conflit ou non. Il convient cependant de noter que si le cycle TaskPrecedence existe dans le patron à réutiliser, il ne sera pas détecté par notre opérateur de réutilisation, puisque le patron sera réutilisé tel quel (sans passer par une étape de vérification).

Nous précisons que nous nous sommes inspirés dans notre approche de l'opérateur PatternApplying [TRA07b] sur deux points. (a) Les trois modes de réutilisation: change, extend et replace; et (b) la vérification des conflits potentiels par le parcourt des relations sortantes. Cependant, cela n'affecte nullement la nouveauté de notre approche et sa valeur ajoutée. En effet, nous avons proposé un opérateur de réutilisation capable de détecter les conflits de premier ordre et de les résoudre automatiquement, ce qui n'est pas possible avec l'opérateur PatternApplying.

Dans le présent travail de thèse, nous avons parlé de patrons en général sans considérer un catalogue particulier ou une source spécifique de patrons. Cependant, nous ne prétendons pas du tout que notre approche est en mesure de réutiliser tout patron indépendamment de toute condition. En fait, notre objectif est de proposer une approche qui exploite les patrons dont la solution est régie par un méta modèle équivalent au notre. Ce dernier étant un méta modèle suffisamment standard pour couvrir un large éventail de patrons. Il en est de même pour les modèles de procédés à modifier par notre approche.

En ce qui concerne la procédure de choix du patron à réutiliser (il faut d'abord choisir le patron adéquat avant de le réutiliser), l'automatisation d'une telle procédure de choix dépasse le cadre du présent travail de thèse. Nous discuterons cependant de ce volet dans la section Perspectives de ce document.

III. Perspectives

Le travail présenté dans ce document peut se prolonger vers plusieurs perspectives; nous présentons quelques unes dans ce qui suit.

Une perspective pour enrichir ce travail consiste à distinguer automatiquement un retour arrière d'un conflit de cycle TaskPrecedence. Cela nous amènera à des questions plus approfondies sur la boucle de relations TaskPrecedence.

Il sera possible pour notre approche d'aborder des (solutions de) patrons ou des modèles de procédés gouvernés par des méta modèles différents du notre, grâce à des solutions d'appariement de méta modèles [KLE03]. Cet aspect n'est pas pris en compte dans le présent travail, mais peut faire l'objet de travaux futurs pour étendre l'éventail de modèles et patrons manipulés à travers notre approche.

En ce qui concerne le problème de choix du patron à réutiliser, il relève de la thématique de recherche de patrons (Patterns Retrieval). Sa résolution consistera tout d'abord à concevoir un opérateur de recherche de patrons permettant, à partir d'une collection de patrons, d'identifier et de sélectionner l'ensemble des patrons les plus adaptés au problème posé (le problème à résoudre par réutilisation de patrons). Un tel ensemble de patrons pourra être construit principalement en comparant les sections "Problem" des patrons explorés avec le problème posé, en se basant sur une mesure de similarité par exemple. L'ensemble sera ensuite enrichi en exploitant les relations entre patrons; là encore, émerge le besoin d'une méthode automatique pour analyser les relations entre patrons, lorsque ces derniers sont exprimés dans des formalismes différents ou appartiennent à divers catalogues. Le patron à réutiliser sera ensuite choisi dans cet ensemble en appliquant plus de critères de sélection, en analysant par exemple les sections "Context", "Forces" ou autres sections du patron.

Nous nous sommes focalisés dans le présent travail sur les conflits de premier ordre, afin de mieux maitriser le problème de réutilisation automatique. Toutefois, ces conflits ne sont pas les seuls susceptibles de se produire, il y a aussi des conflits de second ordre, de troisième ordre ainsi que les conflits des cycles indirects, sur lesquels nous travaillerons pour enrichir d'avantage notre approche.

Pour concrétiser cette approche, nous avons élaboré une plateforme pour réutiliser des patrons dans la modification de modèles de procédés. La prochaine étape sera l'amélioration de cette plateforme, pour en faire d'elle une plateforme de modélisation de procédés complète et efficace. À cet effet, elle doit être d'abord complétée avec une fonction d'affichage du résultat de la réutilisation, puisque actuellement le résultat est reproduit manuellement (pour éviter les chevauchements et estimer les meilleures positions des éléments et relations). Ensuite, une idée est d'enrichir la plateforme avec une fonction suggérant des patrons connexes. En effet, une telle fonction permettra aux concepteurs de procédés de connaître les patrons disponibles et liés à celui en cours de réutilisation. Ces patrons connexes pourront compléter, enrichir, affiner le patron en cours de réutilisation, ou complètement le remplacer. Cela donnera plus de chance aux concepteurs de procédés d'exploiter les connaissances offertes par ces patrons

connexes. Une telle fonction sera possible grâce à l'extraction automatique des relations potentielles entre patrons [HAC15], mais doit cependant être étudiée plus en profondeur.

Au terme de ce travail, il nous est important de mentionner les lacunes qui demeurent existantes dans le domaine des patrons, et que de futurs travaux de recherche pourront adresser.

La pleine réalisation de la réutilisation de patrons est entravée par l'absence de formalisation standard de la notion de patron [BOT10]. Les patrons et les langages de patrons, ainsi que les relations entre patrons, sont souvent présentés dans la littérature de manière informelle ou semi formelle [ROU16]. De plus, les formalismes existants ne sont pas consensuels. Ce manque de fondement formel rend difficile la mise au point d'outils de support, afin d'utiliser toute la puissance des patrons logiciels [ROU16].

D'autre part, étant donné que la majorité des travaux sur les patrons sont restés confinés dans les laboratoires de recherche, il y a eu par conséquent peu de retour d'expérience et de commentaires à partir du monde industriel, et les patrons demeurent ainsi privés de maintenance évolutive [TRA07]. En effet, de tels commentaires permettront d'enrichir les patrons et surtout de documenter des exemples de leur réutilisation, ce qui simplifiera et encouragera de futures opérations de réutilisation de patrons.

Il est également important de souligner le fait que les patrons logiciels ainsi que les collections de patrons ont tendance à être écrits pour résoudre des problèmes spécifiques, sans se soucier de la façon dont un patron pourrait ou devrait être réutilisé avec d'autres patrons connexes [HEN07]. Une réutilisation qui prend en charge les patrons connexes devra être basée sur les relations entre patrons. Cependant, ces relations sont difficiles à discerner lorsqu'elles ne sont pas explicitement mentionnées dans chaque patron, et leur extraction (si elles ne sont pas explicites) est évidemment une activité complexe. Ces faits mettent en désavantage la réutilisation de patrons [HAC15].

En fin, pour réutiliser efficacement des patrons dans des projets logiciels, il est important de pouvoir accéder facilement à différents patrons dans différents domaines d’application. Ainsi, une typologie des problèmes et des contextes sera nécessaire pour aider à organiser les patrons en *bases de patrons*. Il sera également nécessaire de développer des outils de recherche et de sélection de patrons à partir de ces *bases de patrons*.

Références bibliographiques

[ALA03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P.Barros ; Workflow Patterns ; Distributed and Parallel Databases, 14(3), pages 5-51 ; July 2003.

[ALE77] C. Alexander ; A pattern language : towns, buildings, constructions ; Oxford university press ; 1977.

[AMB98] S.W. Ambler ; Process Patterns: Building Large-Scale Systems Using Object Technology ; Cambridge University Press ; 1998.

[AMB98a] S.W. Ambler ; An Introduction to Process Patterns ; AmbySoft Inc. White Paper ; 1998.

[AMB99] S.W. Ambler ; More Process Patterns: Delivering Large-Scale Systems Using Object Technology ; New York: SIGS Books Cambridge University Press ; 1999.

[AMB16] S.W. Ambler ; What is the History of Process Patterns? ; The Process Patterns Resource Page ; Ambysoft Inc. ; http://www.ambysoft.com/processPatternsPage.html ; Last access 04/2016.

[AND11] M. Andreessen ; Why Software Is Eating The World ; in The wall street journal ; 2011; http://www.wsj.com/articles/SB10001424053111903480904576512250915629460 ; Last access 04/2016.

[ANN16] ANNIE : a Nearly-New Information Extraction System ; Last access 2016 ; http://gate.ac.uk/sale/tao/splitch6.html

[AOU12] F. Aoussat ; Réutilisation des procédés logiciels: Une approche à base d'architectures logicielles ; Thèse de doctorat ; Université de Nantes ; France ; 2012.

[BEC87] K. Beck, W. Cunningham ; Using Pattern Languages for Object-Oriented Programs; OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming ; 1987.

[BEN07] R. Bendraou ; UML4SPM : un langage de modélisation de procédés de développement logiciel exécutable et orienté modèle ; Thèse de doctorat ; Université Pierre & Marie Curie ; France ; 2007.

[BEN09] R. Bendraou, J.M. Jezequel, F. Fleurey ; Combining aspect and model driven engineering approches ; In Trustworthy Software Development Processes: Proceedings of International Conference on Software Process ; ICSP 2009 Vancouver, Canada, May 16-17, 2009 ; Volume 5543 of Lecture Notes in Computer Science Programming and Software Engineering ; Qing Wang, Vahid Garousi, Raymond Madachy, Dietmar Pfahl (eds).

[BEZ03] J. Bezivin ; La transformation de modèles ; INRIA-ATLAS et université de Nantes, Ecole d'été d'informatique, cours 6, CEA EDF INRIA ; 2003.

[BEZ04] J. Bezivin and E. Breton ; Applying the basic principles of model engineering to the field of process engineering ; Novatica N° 171, Software Process Technologies ; 2004. http://www.ati.es/novatica/infonovatica.html.

[BIG12] B. Biglari, R. Ramsin ; Generic process framework for developing high-integrity software ; in New Trends in Software Methodologies, Tools and Techniques ; H. Fujita and R. Revetria (eds.) ; IOS Press ; 2012.

[BLA05] X. Blanc ; MDA en action, Ingénierie logicielle guidée par les modèles ; 1ère édition ; Éditions Eyrolles ; ISBN : 2-212-11539-3 ; 2005.

[BOT10] P. Bottoni, E. Guerra, J. de Lara ; A language-independent and formal approach to pattern-based modelling with support for composition and analysis ; Information and Software Technology, 52 (8), pp. 821-844 ; Elsevier B.V. ; 2010.

[BRO12] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer ; An introduction to model versioning, in Formal Methods for Model-Driven Engineering ; Lecture Notes in Computer Science ; M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.) ; vol. 7320, pp. 336–398 ; Springer ; 2012.

[BRO12a] P. Brosch, U. Egly, S. Gabmeyer, G. Kappel, M. Seidl, H. Tompits, M. Widl, M. Wimmer ; Towards semantics-aware merge support in optimistic model versioning ; in Models in Software Engineering, volume 7167 of the series Lecture Notes in Computer Science, pp. 246–256 ; Springer ; 2012.

[BUN15] M. Bunke ; Software-security patterns: degree of maturity ; Proceedings of the 20th European Conference on Pattern Languages of Programs ; Kaufbeuren, Germany ; 2015.

[CAS00] Cass, A.G., Staudt Lerner, B., McCall, E.K., Osterweil, L. J.,Sutton, Jr., S.M., and Wise, A. ; Little-JIL/Juliette: A Process Definition Language and Interpreter ; In Proceedings of the 22nd Internantioanl Conference on Software Engineering ; June 2000.

[CHA09] Sylvain Chardigny ; Extraction d'une architecture logicielle à base de composants depuis un système orienté objet. Une approche par exploration ; Thèse de doctorat ; Université de Nantes ; 2009.

[CHE08] S. Cherry and P. Robillard ; The social side of software engineering–A real ad hoc collaboration network ; International journal of human-computer studies, Vol. 66, No. 7, pp. 495–505 ; 2008.

[COM08] B. Combemale ; Approche de métamodélisation pour la simulation et la vérification de modèle-Application à l'ingénierie des procédés ; Thèse de doctorat ; Institut National Polytechnique de Toulouse ; 2008.

[CON99] R. Conradi, M.J. Jaccheri. Process Modelling Languages ; In Software Process: Principles, Methodology and Technology. Lecture Notes in Computer Science ; Derniame, J.C., Kaba, B.A., Wastell, D. (eds.) ; Vol. 1500, 27-51 ; Springer-Verlag, Berlin Heidelberg New York ; 1999.

[COP94] J.O. Coplien ; A Development Process Generative Pattern Language ; In Proceedings of the Annual Conference on the Pattern Languages of Programs PLoP'94 ; 1994.

[COU01] Coulette B., Cregut X., Dong T., Tran D. ; Managing process through a base of reusable components ; ICEIS2001 ; 2001.

[COU02] Coulette B., Cregut X., Dong T., Tran D. ; A Metaprocess to define and reuse process components ; IDPT02 ; 2002.

[CVS16] Bibliography on Comparison and Versioning of Software Models ; http://pi.informatik.uni-siegen.de/CVSM/ ; Last access 08/2016.

[DAN01] Dan Thu Tran ; Formalisation et mise en oeuvre de la notion de composant de procédés logiciels ; Thèse doctorale N°1818 ; Institut National Polytechnique de Toulouse, France ; 2001.

[DAM14] H. K. Dam, A. Rede, A. Egyed. Inconsistency resolution in merging versions of architectural models, in Proceedings of the 11th IEEE/IFIP Conference on Software Architecture, IEEE, pp. 153–162, Washington, USA, 2014.

[DAM16] H. K. Dam, A. Egyedb, M. Winikoffc, A. Rederb, R. E. Lopez-Herrejon ; Consistent merging of model versions ; in The Journal of Systems and Software, volume 112, pp.137–155, Elsevier ; 2016.

[DEM03] S. Demeyer, S. Ducasse, O. Nierstrasz ; Object-Oriented Reengineering Patterns ; by Elsevier Science USA ; 2003.

[DER99] Derniame J. C., Kaba B.A., Wastell D (Editors) ; Software Process: Principles, Methodology and Technology ; Lecture Notes in Computer Science 1500, Springer, DOI : 10.1007/3-540-49205-4 ; 1999.

[DIA11] S. Diaw ; PEM4MDE : Un métamodèle et un environnement pour la modélisation et la mise en œuvre assistée de processus IDM ; Thèse de Doctorat ; Université de Toulouse ; 2011.

[DIC16] Dictionnaires de Français ; Larousse ; Last access 08/2016 ; http://www.larousse.fr/dictionnaires/francais/

[DIT02] T. Dittmann, V. Gruhn, M. Hagen ; Improved Support for the definition and usage of process patterns ; Position Paper ; Focus Group "What makes PatternLanguages work well?" ; EuroPlop'02 ; 2002.

[DIT16] Y. Dittrich ; What does it mean to use a method? Towards a practice theory for software engineering ; in Information and Software Technology, vol. 70, pp. 220-231 ; Elsevier ; 2016.

[ECL16] Eclipse ; http://www.eclipse.org/ ; Last access 09/2016.

[EGY06] Egyed, A. ; Instant consistency checking for the UML ; in Proceedings of the 28th International Conference on Software Engineering, pp. 381–390 ; 2006.

[EMF16] EMF Compare ; http://www.eclipse.org/emf/compare/ ; Last access 08/2016.

[ENG16] J. Engel, C. Martin, P. Forbrig ; A Unified Pattern Specification Formalism to Support User Interface Generation ; M. Kurosu (Ed.): HCI 2016, Part I, LNCS 9731, pp. 445–456 ; Springer International Publishing Switzerland ; 2016.

[EST05] J. Estublier ; Software are Processes Too ; Invited paper in Proceeding of Software Process Workshop ; China ; 2005.

[FER07] E. Fernandez, J. Pelaez, M. Larrondo-Petrie ; Attack Patterns: A New Forensic and Design Tool ; Advances in Digital Forensics III, International Federation for Information Processing, Volume 242/2007, pp. 345-357 ; (eds) P. Craiger and S. Shenoi ; Springer, Boston ; 2007.

[FIR01] Firesmith, D. and Henderson-Sellers, B. ; The OPEN Process Framework: An Introduction ; Published by Addison-Wesley ; 2001.

[FLE10] Fleurquin R. ; Des langages pour améliorer le développement et la maintenance des logiciels à base de composants ; Mémoire d'habilitation à diriger des recherches en informatique ; Université Européenne de Bretagne ; 2010.

[FOW02] M. Fowler ; Patterns of Enterprise Application Architecture ; Addison Wesley ; 2002.

[FUG00] A. Fuggetta ; Software Process: A Roadmap ; in Proceeding of 22nd International Conference on Software Engineering, Future of Software Engineering Track ; June 4–11, Limerick (Irlanda) ; ACM ; 2000.

[GAM95] E. Gamma, R. Helm, R. Johnson, J. Vlissides ; Design Patterns: Elements of Reusable Object-Oriented Software ; Addison-Wesley ; 1995.

[GAR06] S. Garcia-Camargo ; Ingénierie Concurrente en Génie Logiciel : Céline ; Thèse de Doctorat ; Université Joseph Fourier de Grenoble ; 2006.

[GAR07] M. GRARI ; Principes et états de l'art de l'approche MDA et applications pour des plateformes PHP orientées 3-tier ; Mémoire de DESA à l'UFR faculté des science, Oujda ; 2007.

[GAR16] G.A. Garcia-Mireles ; Addressing Product Quality Characteristics Using the ISO/IEC 29110 ; in Trends and Applications in Software Engineering, Advances in Intelligent Systems and Computing, volume 405, pp 25-34 ; J. Mejia et al. (eds.), Springer Switzerland ; 2016.

[GAT16] GATE general architecture for text engineering ; Last access 2016 ; http://gate.ac.uk/

[GER13] C. Gerth, J. M. Küster, M. Luckey, G.Engels ; Detection and resolution of conflicting change operations in version management of process models ; in Software and System Modeling, Volume 12, Issue 3, pp 517-535, Springer ; 2013.

[GHO10] M. F. Gholami, P. Jamshidi, F. Shams ; A procedure for extracting software development process patterns ; In proceedings of the 5th UKSim European Symposium on Computer Modeling and Simulation (EMS), pp.75-83 ; 2010.

[GNA03] M. Gnatz, F. Marschall, G. Popp , A. Rausch, W. Schwerin ; The Living Software Development Process ; Journal Software Quality Professional, Volume 5, Issue 3 ; 2003.

[GZA00] L. Gzara ; Les patterns pour l'ingénierie des Systèmes d'Information Produit ; These de doctorat de l'INPG, spécialité Génie Industriel ; 2000.

[HAC11] Asma Hachemi ; Composition de patrons logiciels - Contributions à la documentation Agile et à l'analyse automatique des relations entre patrons ; Mémoire de Magistère en informatique, USTHB. Algérie ; 2011.

[HAC12] A. Hachemi, M. Ahmed-Nacer ; Primary inter-patterns relationships analysis ; 11th African Conference on Research in Computer Science and Applied Mathematics CARI'2012 ; Algiers, Algeria ; 2012.

[HAC12a] A. Hachemi, M. Ahmed-Nacer ; Relations extraction on patterns lacking of Resulting Context ; Proceedings of the 4th International Conference on Web and Information Technologies ; Sidi Bel Abbes, Algeria ; April 29-30, 2012 ; M. Malki, S. Benbernou, S. Benslimane, A. Lehireche (Eds.) ; Ceur-ws.org, Vol-867, pp 282-287 ; 2012.

[HAC15] Hachemi A., Ahmed-Nacer M ; Automatic Extraction of Relationships Among Software Patterns ; In: Sobh T., Elleithy K. (eds) ; Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering ; Lecture Notes in Electrical Engineering, vol 313. Springer, Cham ; 2015.

[HAC17] A. Hachemi, M. Ahmed-Nacer ; Software process patterns: a roadmap ; Proceedingof the 14th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA) ; Tunisia, 2017 ; Pages: 887 – 894, IEEEXplore ; ISBN: 978-1-5386-3581-0 ; ISSN: 2161-5330 ; DOI: 10.1109/AICCSA.2017.197.
http://ieeexplore.ieee.org/document/8308383/

[HAC18] A. Hachemi, M. Ahmed-Nacer ; Reusing process patterns in software process models modification ; Journal of Software : evolution and process ; Gerardo Canfora, Darren Dalcher and David Raffo (eds. ; e1938 ; John Wiley & Sons Ltd ; DOI: 10.1002/smr.1938 ; 2018. https://doi.org/10.1002/smr.1938

[HAG02] M. Hagen ; Support for the definition and usage of process patterns ; Position Paper, What makes Pattern Languages work well? ; EuroPlop02 ; 2002.

[HAG04] M. Hagen and V. Gruhn ; Process Patterns - a Means to Describe Processes in a Flexible Way ; in Proceeding of the International Workshop on Software Process Simulation and Modeling ; Edinburgh, UK ; 2004.

[HAG04a] M. Hagen, V. Gruhn ; Towards flexible Software Processes by using Process Patterns ; in Proceeding of 8th International Conference on Software Engineering and Applications ; Cambridge, USA ; 2004.

[HEE09] Xiao-yang Hee, Ya-sha Wang, Jin-gang Guo, Wu Zhou and Jia-kuan Ma ; Weaving Process Patterns into Software Process Models ; In Proceeding of the 21st International Conference on Software Engineering & Knowledge Engineering ; Boston, USA ; 2009.

[HEN99] K. Henney ; Patterns Inside Out ; Talk presented at Application Development, London ; 1999.

[HEN07] S. Henninger, V. Correa ; Software Pattern Communities : Current Practices and Challenges ; Computer Science and Engineering Technical Report ; University of Nebraska Lincoln ; 2007.

[HUM88] W. S. Humphrey ; Characterizing the software process : a maturity framework ; IEEE Software, V.5, pp 73-79 ; 1988.

[IID99] Iida H. ; Pattern-Oriented Approach to Software Process Evolution ; In Proceedings of the International Workshop on the Principles of Software Evolution IWPSE99 ; Fuoka, Japan ; 1999.

[IID02] Iida, H., Tanaka, Y ; A Compositional Process Pattern Framework for Component based Process Modeling Assistance ; The 1st Workshop on Software Development Process Patterns ; 2002.

[ISO06] ISO Software Engineering - Metamodel for Development Methodologies ; ISO document, ISO/JTC 1/SC 7 ICS 35.080 ; 2006.

[IST08] Modelplex ; IST European Project contract IST-3408.

[JAV16] What is java? https://www.java.com ; Last access 09/2016.

[JLA11] Jlaiel, N., and Ben Ahmed, M. ; MetaProPOS : a meta-process patterns ontology for software development communities ; In Proceedings of the KES Conference on Knowledge

Based and Intelligent Information & Engineering Systems. Part I, LNCS 6881 Springer, 516-527 ; 2011.

[JLA12a] Nahla Jlaiel, Khouloud Madhbouh, Mohamed Ben Ahmed ; A Semantic Approach for Automatic Structuring and Analysis of Software Process Patterns ; in International Journal of Computer Applications, Volume 54– No.15 ; September 2012.

[JLA12b] Jlaiel, N., and Ben Ahmed, M. ; Towards a novel semantic approach for process patterns capitalization and reuse ; In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2012.

[KAB09] Mohammed I. KABBAJ ; Etat de l'art sur l'adaptation dynamique des procédés de développement de logiciels ; Rapport de recherche, IRIT/RR--2009-1--FR, IRIT ; 2009.

[KES13] M. Kessentini, W. Werda, P. Langer, M. Wimmer ; Search-based model merging ; In Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, pp. 1453–1460, ACM, New York, USA ; 2013.

[KLE03] A.G. Kleppe, J. Warmer, W. Bast ; MDA Explained. The Model Driven Architecture: Practice and Promise ; Addison-Wesley Longman Publishing Co., Inc., Boston, USA ; 2003.

[KNE02] R. Kneuper ; Supporting software processes using knowledge management ; in Handbook of Software Engineering and Knowledge Engineering ; vol. II: Emerging Technologies. SK Chang (eds.) ; ISBN: 978-981-02-4974-8. World Scientific Publishing Co. Pte. Ltd. ; 2002.

[KRA88] G.E. Krasner, S.T. Pope ; A Description of the Model-View-Controller User Interface Paradigm in The Smalltalk-80 System ; Journal of Object Oriented Programming ; vol. 1, no. 3, pp. 26-49 ; 1988.

[KUB05] A. Kubo, H. Washizaki, A. Takasu, Y. Fukazawa ; Analyzing Relations among Software Patterns based on Document Similarity ; Proceeding of IEEE Internationale Conference on Information Technology : Coding and Computing ; 2005.

[MAR16] P.V. Martins, A.R. da Silva ; ProPAM/Static: A Static View of a Methodology for Process and Project Alignment ; Book chapter in Trends and Applications in Software Engineering, Advances in Intelligent Systems and Computing ; volume 405, pp.47-57, J. Mejia et al. (eds.) ; DOI 10.1007/978-3-319-26285-7_5, Springer International Publishing Switzerland ; 2016.

[MOF11] MOF core v 2.4.1 ; Object Managment Group. http://www.omg.org/cgi-bin/doc?formal/2011-08-07 ; 2011.

[MON99] C. Montangero, J. Derniame, B.A. Kaba, B. Warboys ; The software process modelling and technolog ; in LNCS GmbH. Vol. 1500 ; 1999.

[OMG08] OMG ; Software & Systems Process Engineering Meta-Model Specification Version 2.0 ; available at http://www.omg.org/spec/SPEM/2.0/PDF ; 2008.

[OMG16] OMG ; About OMG ; http://www.omg.org/gettingstarted/gettingstartedindex.htm ; last access 09/2016.

[OSM12] Addy Osmani; Learning JavaScript Design Patterns ; O'Reilly; ISBN: 978-1-449-33181-8 ; 2012.

[OST87] L.J. Osterweil. Software Processes Are Software Too ; in Proceedings of the Ninth International Conference of Software Engineering ; pages 2-13, IEEE Computer Society Press ; 1987.

[PRA10] T.V Prabhakar, K. Kumar ; Design Decision Topology Model for Pattern Relationship Analysis; Asian PLOP 2010 Tokyo ; 16-17 Mars 2010.

[PRI87] Priéto-Diaz, R., Freeman, P ; Classifying Software for Reusability ; IEEE Software, vol. 4, n°1; January 1987.

[RAM08] R. Ramsin, R. F. Paige, Process-centered review of object oriented software development methodologies, ACM Computing Surveys, vol. 40, no. 1, Feb. 2008, pp. 1-89.

[RED12] A. Reder, A. Egyed. Computing repair trees for resolving inconsistencies in design models, in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 220–229, ACM, New York, USA, 2012.

[RIB00] Ribo J.M, Franch X. PROMENADE, a PML intended to enhance standardization, expressiveness and modularity in SPM. Research Report LSI-00-34-R, Dept. LSI, Politechnical University of Catalonia. 2000.

[RIB02] Ribo J.M, Franch X. Supporting Process Reuse in PROMENADE. Research Report LSI-02-14-R, Dept. LSI, Politechnical University of Catalonia. 2002.

[ROU16] A. Rouhi, B. Zamani, Towards a formal model of patterns and pattern languages, Information and Software Technology, Vol. 79, ISSN 0950-5849, Published by Elsevier B.V., pp. 1-16, 2016.

[ROY09] Gilles Roy. Conception de bases de données avec UML. Presses de l'université du Quebec. Canada. 2009.

[RUB13] J. Rubin, M. Chechik. N-way model merging, in Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, pp. 301–311, ACM, New York, USA, 2013.

[SAB08] M. Sabetzadeh, S. Nejati, S. Easterbrook, M. Chechik. Global consistency checking of distributed models with TReMer, in Proceedings of the 30th International Conference on Software Engineering, pp. 815–818, ACM, New York, USA, 2008.

[SEI03] Ed Seidewitz, What models mean, IEEE Software, Volume: 20, Issue: 5, pp. 26–32, 2003.

[STA10] B. Staudt Lerner, S. Christov, L.J. Osterweil, R. Bendraou, U. Kannengiesser, et A. Wise; Exception Handling Patterns for Process Modeling; IEEE transactions on software engineering, vol. 36; 2010.

[STO01] H. Storrle; Describing Process Patterns with UML; European Workshop in Software Process Technology, Germany; 2001.

[TAS07] S. Tasharofi, R. Ramsin, Process Patterns for Agile Methodologies, in Situational Method Engineering: Fundamentals and Experiences. Ralyte J. et al. (eds.), The International Federation for Information Processing, vol 244. Springer, Boston, 2007.

[TRA07] H. Tran. Modélisation de procédés logiciels à base de patrons réutilisables. Thèse de doctorat, Université Toulouse II. 2007.

[TRA07a] H.N. Tran, B. Coulette, B.T. Dong. Modeling Process Patterns and Their Application, in Proceeding of the 2nd International Conference on Software Engineering Advances, French Riviera, France. 2007.

[TRA07b] H.N. Tran, B. Coulette, B.T. Dong, Broadening the Use of Process Patterns for Modeling Processes, in Proceeding of the 9th International Conference on Software Engineering and Knowledge Engineering, Boston, USA, 2007.

[TRA11] Hanh Nhi Tran, Bernard Coulette, Dan Thu Tran, My Hang VU, Automatic Reuse of Process Patterns in Process Modeling, proceeding of SAC11, March 21-25, 2011, TaiChung, Taiwan.

[UML15] OMG, UML 2.5 Superstructure Specification, http://www.omg.org/spec/UML/2.5/, 2015

[VOR09] P. Vora ; Web application design patterns ; Morgan Kaufmann, Elsevier ; 2009.

[WAN10] Ya-sha Wang, Xiao-yang He, Jin-gang Guo and Jia-rui Jiang, Software Process Reuse by Pattern Weaving, in Proceeding of the 22nd International Conference on Software Engineering & Knowledge Engineering, San Francisco, USA. 2010.

[WIE12] K. Wieland, P. Langer, M. Seidl, M. Wimmer, G. Kappel. Turning conflicts into collaboration, in Computer Supported Cooperative Work, DOI 10.1007/s10606-012-9172-4, Springer, 2012.

ANNEXE A

Algorithme de réutilisation

Cet annexe détaille l'algorithme véhiculant la sémantique opérationnelle de notre opérateur de réutilisation de patrons.

BEGIN

- Result:= Sol; */*copy the pattern solution into Result*/*
- Result:=Substitute fp per ap; */*substitute in Result the pattern formal parameters fp by the actual parameters ap from the process model PM*/*
- if mode==replace or mode==extend then */*if the mode is change then go to the end of the algorithm, else (the mode is replace or extend) proceed to the merging and the conflicts management*/*
  - Result:= Add remaining elements of PM; */*Add the remaining elements of PM to Result*/*
  - For each element e of PM do : */*now, we should add the remaining relationships of PM to Result*/*
    - Find eres in Result that matches to e;
    - For each relationship r in e.Relations do : */*browse all relationships that have e as source, and for each one of them do :*/*
      - Find the element f in PM that is the target of r;
      - Find fres in Result that matches to f;
      - if r doesn't exist between eres and fres then */*in this case, we should add r while avoiding inconsistencies that may occur when adding*/*

```
if eres is Task then

/* if I'm at this level of the algo, it means that r exists between e and f
but not between eres and fres, then I have to add it IF NO PROBLEM. So I
start to test the potential conflicts */

  if r is TaskPrecedence then {

    boolean B = false;/* initially to false. As soon as an
    inconsistency is encountered, B will be set to true */

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    {

    if (ri==Aggregation and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}/*I
    test if not exist to avoid duplication, because of the loop for each*/
    /*else nothing*/}
```

```
  if (ri==Refinement and ri.dest==fres) then {B= true; if
  mode== extend then {Delete ri; Add(r) if Not Exist;}}

  }

  if NotEmpty (fres.Relations) then for each rj in
  fres.Relations do:

  {

  if (rj==Refinement and rj.dest==eres) then {B= true; if
  mode== extend then {Delete rj; Add(r) if Not Exist;}}

  if (rj==TaskPrecedence and rj.dest==eres) then {B= true;
  if mode== extend then {Delete rj; Add(r) if Not Exist;}}

  if (rj==Aggregation and rj.dest==eres) then {B= true; if
  mode== extend then {Delete rj; Add(r) if Not Exist;}}

  }

  If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
  OR [B==false] then Add(r) if Not Exist; /* if both vectors
  are empty then there is no conflict. If B is still False then there is
  no conflict too */

}

Else {

if r is Aggregation then {

  boolean B = false;

  if NotEmpty (eres.Relations) then for each ri in
  eres.Relations do:

  {

  if (ri==TaskPrecedence and ri.dest==fres) then {B= true;
  if mode== extend then {Delete ri; Add(r) if Not Exist;}}

  if (ri==Refinement and ri.dest==fres) then {B= true; if
  mode== extend then {Delete ri; Add(r) if Not Exist;}}

  }

  if NotEmpty (fres.Relations) then for each rj in
  fres.Relations do:

  {

  if (rj==Refinement and rj.dest==eres) then {B= true; if
  mode== extend then {Delete rj; Add(r) if Not Exist;}}

  if (rj==Aggregation and rj.dest==eres) then {B= true; if
  mode== extend then {Delete rj; Add(r) if Not Exist;}}

  if (rj==TaskPrecedence and rj.dest==eres) then {B= true;
  if mode== extend then {Delete rj; Add(r) if Not Exist;}}
```

```
    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;

  }

  Else {

  if r is Refinement then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    {

    if (ri==Aggregation and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}}

    if (ri==TaskPrecedence and ri.dest==fres) then {B= true;
    if mode== extend then {Delete ri; Add(r) if Not Exist;}}

    }

    if NotEmpty (fres.Relations) then for each rj in
    fres.Relations do:

    {

    if (rj==Aggregation and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    if (rj==Refinement and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    if (rj==TaskPrecedence and rj.dest==eres) then {B= true;
    if mode== extend then {Delete rj; Add(r) if Not Exist;}}

    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;

  }

  Else {

  if (r is TaskParameter of kind OUT) then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    {

    if (ri==TaskParameter and ri.dest==fres and ri.kind==IN)
    then { B= true; ri.kind:= IN-OUT ; }
```

```
  if (ri==TaskParameter and ri.dest==fres and ri.kind==IN-
  OUT) then { B= true;}

  }

  If [IsEmpty(eres.Relations)] OR [B==false] then Add(r)
  if Not Exist;

}

Else {

if (r isTaskParameter of kind IN-OUT) then {

  boolean B = false;

  if NotEmpty (eres.Relations) then for each ri in
  eres.Relations do:

  {

  if (ri==TaskParameter and ri.dest==fres and
  ri.kind==OUT) then {B= true; if mode== extend then
  {Delete ri; Add(r) if Not Exist;}}

  if (ri==TaskParameter and ri.dest==fres and ri.kind==in)
  then { B= true; ri.kind:= IN-OUT ; }

  }

  /* There is no problem in the Replace mode even if there are several
  ri, because we suppose that initially the models to be merged are
  coherent so it is impossible to have in and out at the same time in
  the vector eres.Relations*/

  If [IsEmpty(eres.Relations)] OR [B==false] then Add(r)
  if Not Exist;

}

Else {

if (r is TaskParameter of kind in) then {

  boolean B = false;

  if NotEmpty (eres.Relations) then for each ri in
  eres.Relations do:

  {

  if (ri==TaskParameter and ri.dest==fres and
  ri.kind==OUT) then {B= true; if mode== extend then
  {Delete ri; Add(r) if Not Exist;}}

  if (ri==TaskParameter and ri.dest==fres and ri.kind==IN-
  OUT) then { B= true;}

  }
```

```
    If [IsEmpty(eres.Relations)] OR [B==false] then Add(r)
    if Not Exist;

  }


  else Add(r); /* if I'm in this else, it means that r exists between e
  and f but not between eres and fres, so I have to add it and I can do it
  because there is no conflict */

  } } } } }

end_if

if eres is Role then

  if r is Refinement then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    if (ri==Aggregation and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}}

    if NotEmpty (fres.Relations) then for each rj in
    fres.Relations do:

    {

    if (rj==Aggregation and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    if (rj==Refinement and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;

  }

  Else {

  if r is Aggregation then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    if (ri==Refinement and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}}

    if NotEmpty (fres.Relations) then for each rj in
    fres.Relations do:

    {
```

```
    if (rj==Refinement and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    if (rj==Aggregation and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;

  }

  Else {

  if (r is TaskPerformance of kind Assistant) then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    if (ri==TaskPerformance and ri.dest==fres and
    ri.kind==Performer) then {B= true; if mode== extend then
    {Delete ri; Add(r) if Not Exist;}}

    If [IsEmpty(eres.Relations)] OR [B==false] then Add(r)
    if Not Exist;

  }

  Else {

  if (r is TaskPerformance of kind Performer) then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    if (ri==TaskPerformance and ri.dest==fres and
    ri.kind==Assistant) then {B= true; if mode== extend then
    {Delete ri; Add(r) if Not Exist;}}

    If [IsEmpty(eres.Relations)] OR [B==false] then Add(r)
    if Not Exist;

  }

  else Add(r);

  } } }

end_if

if eres is Product then

  if r is ProductImpact then {

    boolean B = false;
```

```
    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    {

    if (ri==Refinement and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}}

    }

    if NotEmpty (fres.Relations) then for each rj in
    fres.Relations do:

    {

    if (rj==ProductImpact and rj.dest==eres) then {B= true;
    if mode== extend then {Delete rj; Add(r) if Not Exist;}}

    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;

}

Else {

if r is Refinement then {

    boolean B = false;

    if NotEmpty (eres.Relations) then for each ri in
    eres.Relations do:

    {

    if (ri==Aggregation and ri.dest==fres) then {B= true; if
    mode== extend then {Delete ri; Add(r) if Not Exist;}}

    if (ri==ProductImpact and ri.dest==fres) then {B= true;
    if mode== extend then {Delete ri; Add(r) if Not Exist;}}

    }

    if NotEmpty (fres.Relations) then for each rj in
    fres.Relations do:

    {

    if (rj==Aggregation and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    if (rj==Refinement and rj.dest==eres) then {B= true; if
    mode== extend then {Delete rj; Add(r) if Not Exist;}}

    }

    If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
    OR [B==false] then Add(r) if Not Exist;
```

```
    }
    Else {
    if r is Aggregation then {
      boolean B = false;
      if NotEmpty (eres.Relations) then for each ri in
      eres.Relations do:
      if (ri==Refinement and ri.dest==fres) then {B= true; if
      mode== extend then {Delete ri; Add(r) if Not Exist;}}
      if NotEmpty (fres.Relations) then for each rj in
      fres.Relations do:
      {
      if (rj==Refinement and rj.dest==eres) then {B= true; if
      mode== extend then {Delete rj; Add(r) if Not Exist;}}
      if (rj==Aggregation and rj.dest==eres) then {B= true; if
      mode== extend then {Delete rj; Add(r) if Not Exist;}}
      }
      If [IsEmpty(eres.Relations) AND ISEmpty(fres.Relations)]
      OR [B==false] then Add(r) if Not Exist;
    }
    else Add(r);
    } }
  end_if
END
```

Résumé
La réutilisation est un moyen prometteur pour améliorer la qualité et la productivité des modèles de procédés, en particulier la réutilisation de patrons. Une telle réutilisation peut prendre les formes suivantes: la réutilisation de patrons pour créer un nouveau (fragment de) modèle de procédés, ou la réutilisation de patrons pour modifier un modèle de procédés existant. Cependant, cette dernière forme demeure délaissée en raison du manque d'outils automatiques qui la soutiennent, mais également parce qu'elle implique un mécanisme de fusion. En effet, deux modèles de procédés sont impliqués dans cette forme de réutilisation: le modèle à modifier et la solution du patron. Leur fusion peut produire un modèle cohérent, comme elle peut produire un résultat incohérent si des conflits apparaissent entre les modèles fusionnés.

L'objectif de cette thèse est de proposer une approche de réutilisation de patrons pour modifier les modèles de procédés. D'abord, pour régir ces modèles et patrons, nous avons opté pour un méta modèle simple, facile à comprendre et à utiliser. Après, l'étude des conflits a eu lieu, nous permettant d'observer que les conflits peuvent se produire à plusieurs niveaux. Nous nous sommes alors focalisés sur les conflits de premier ordre, et avons ainsi identifié exhaustivement les dix-huit conflits de ce niveau. La proposition d'un opérateur de réutilisation était l'étape suivante ; pour donner sa sémantique opérationnelle, nous avons proposé un algorithme de réutilisation avec gestion des conflits de premier ordre. Enfin, nous avons mise en place une plateforme de réutilisation pour concrétiser notre approche. La plateforme qui repose sur une représentation graphique, nous a permis d'évaluer notre approche et de la comparer pour mettre en avant sa valeur ajoutée.

Mots clés
Patron de procédés, modèle de procédés, réutilisation de patrons, fusion de modèles, conflit de premier ordre, méta modèle de procédés, modélisation de procédés, patron logiciels.

Summary
Reuse is a promising way to improve the quality and productivity of process models, especially the reuse of patterns. Such reuse can take the following forms: the reuse of patterns to create a new (fragment of) process model, or the reuse of patterns to modify an existing process model. However, this last form remains neglected because of the lack of automatic tools that support it, but also because it implies a merging mechanism. Indeed, two process models are involved in this form: the model to be modified and the pattern solution. Their merging can produce a consistent result, as it can produce an inconsistent result if conflicts occur between merged models.

The aim of this thesis is to propose a pattern reuse approach to modify process models. First, to govern these patterns and models, we opted for a simple meta model, easy to understand and use. After that, we study of the conflicts and observe that conflicts can occur on several levels. We then focused on first-order conflicts, and identified them exhaustively. The proposal of a reuse operator was the next step, with first-order conflict management. Finally, we proposed a reuse platform based on a graphical representation. The platform allowed us to evaluate our approach and to compare it to highlight its value added.

Keywords
Process pattern, process model, pattern reuse, models merging, first-order conflict, process meta model, process modeling, software pattern.