Modélisation et exécution flexible de processus collaboratifs

Mamadou Lakhassane Cisse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 2 - Jean Jaurès

Cotutelle internationale : Université Cheikh Anta Diop de Dakar

---

Présentée et soutenue par

Mamadou CISSE

Le 30 octobre 2020

Modélisation et exécution flexible de processus collaboratifs

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Bernard COULETTE

Jury

M. Alassane BAH, Directeur de thèse
M. Antoine BEUGNARD, Rapporteur
M. François CHAROY, Rapporteur
M. Bernard COULETTE, Co-directeur de thèse
M. Samba DIAW, Examinateur
M. Ousmane SALL, Rapporteur
Mme Hanh-Nhi TRAN, Examinatrice

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse - Jean Jaurès*
Cotutelle internationale *avec l'Université Cheikh Anta Diop de Dakar*

Présentée et soutenue le *15/07/2020* **par :**

Mamadou Lakhassane Cissé

Modélisation et Exécution flexible de processus collaboratifs

---

JURY

Alassane BAH	Professeur des Universités	Univ. C. A. Diop de Dakar
Antoine BEUGNARD	Professeur des Universités	Télécom Brest
François CHAROY	Professeur des Universités	Université de Nancy
Bernard COULETTE	Professeur des Universités	Univ. Toulouse Jean Jaurès
Samba DIAW	Maître de Conférences HDR	Univ. C. A. Diop de Dakar
Ousmane SALL	Professeur des Universités	Université de Thiès
Hanh-Nhi TRAN	Maître de Conférences	Université Paul Sabatier

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (IRIT – UMR 5505)
Directeur(s) de Thèse :
Bernard COULETTE et *Alassane BAH*
Rapporteurs :
Ousmane SALL, *François CHAROY* et *Antoine BEUGNARD*

Table des matières

Liste des figures

Liste des tableaux

Abstract

Nowadays, collaboration and teamwork are becoming a necessity in most processes, especially in software engineering to develop complex software products. Collaboration can consist of coordination between various tasks to synchronize their progress (weak collaboration). It can also occur in the form of a cooperation of actors within a given task, called a collaborative task, to achieve a common goal (strong collaboration). The challenge is then to organize this strong collaboration to optimize quality and development time.

For several years, a lot of research has been carried out in different communities on various aspects of collaborative work. Some approaches provide constructs to model the activities that must be coordinated during the execution of the process, but they do not allow to finely and dynamically control this execution according to the evolution of the context of the project. The latter represents the set of elements that can affect how to collaborate within a task. It can be the number of actors available, the importance given to the task, the structure of the inputs / outputs (composite or not), dependencies between the components of a composite artefact, etc.

The objective of this thesis is to propose an approach allowing the flexible execution of a collaborative process. The idea is to let the process actors dynamically define the collaboration strategy that meets their needs during execution.

For this, we first propose a **language allowing to model a collaborative process and in particular the notion of collaborative task** then to represent it at execution as a multi-instance task. Each instance is executed by a separate actor playing the role carrying out the task. The scheduling of instances must respond to the most appropriate collaboration strategy

depending on the context of the project.

To guide the project manager in choosing a strategy for the execution of a collaborative task, we have defined a set of **collaboration patterns** which correspond to the most usual strategies depending on the context (for example : sequential execution of task instances to do refinement, parallel execution of task instances to do co-work).

The **semantic of execution of collaborative tasks** is formalized by means of UML state machines describing the life cycle of the elements of the process : tasks, products, actors. The transitions between the states are carried out by actions which are configured by the collaboration pattern to be applied.

To support our approach, we have developed a prototype execution engine in the form of a web application allowing to store on one hand models of collaborative processes, on the other hand collaboration patterns, and to assist the project manager and stakeholders involved in the execution of a given process. An experimental validation was carried out on the basis of a representative case study of software design processes.

Résumé

De nos jours, la collaboration et le travail d'équipe deviennent une nécessité dans la plupart des processus, en particulier en génie logiciel pour développer des produits logiciels complexes. La collaboration peut consister en une coordination entre diverses tâches pour synchroniser leur progression (collaboration faible). Elle peut également se produire sous la forme d'une coopération d'acteurs au sein d'une tâche donnée, appelée tâche collaborative, pour atteindre un objectif commun (collaboration forte). Le défi est alors d'organiser cette collaboration forte pour optimiser la qualité et le temps de développement.

Depuis plusieurs années, de nombreuses recherches ont été menées dans différentes communautés sur divers aspects du travail collaboratif. Certaines approches fournissent des constructions pour modéliser les activités qui doivent être coordonnées pendant l'exécution du processus, mais elles ne permettent pas de contrôler finement et dynamiquement cette exécution en fonction de l'évolution du contexte du projet. Ce dernier représente l'ensemble des éléments qui peuvent affecter la façon de collaborer au sein d'une tâche. Il peut s'agir du nombre d'acteurs disponibles, de l'importance accordée à la tâche, de la structuration des artéfacts (composite ou non) en entrée/sortie, des dépendances entre les composants d'un artéfact composite, etc.

L'objectif de cette thèse est de proposer une approche permettant l'exécution flexible d'un processus collaboratif. L'idée est de laisser les acteurs du processus définir dynamiquement la stratégie de collaboration qui répond à leurs besoins lors de l'exécution.

Pour cela, nous proposons tout d'abord un **langage permettant de modéliser un processus collaboratif et notamment la notion de tâche collaborative** puis de la représenter à l'exécution comme une tâche multi-instances. Chaque instance est exécutée par un acteur séparé

jouant le rôle réalisant la tâche. L'ordonnancement des instances doit répondre à la stratégie de collaboration la plus appropriée en fonction du contexte du projet.

Pour guider le chef de projet dans le choix d'une stratégie pour l'exécution d'une tâche collaborative, nous avons défini un ensemble de **patrons de collaboration** qui correspondent aux stratégies les plus usuelles en fonction du contexte (par exemple : enchainement séquentiel des instances de tâche pour faire du raffinement, exécution en parallèle des instances de tâches pour faire du co-travail).

La **sémantique d'exécution des tâches collaboratives** est formalisée par l'intermédiaire de machines à états UML décrivant le cycle de vie des éléments du processus : tâches, produits, acteurs. Les transitions entre les états sont réalisées par des actions qui sont paramétrées par le patron de collaboration à appliquer.

Pour supporter notre approche, nous avons développé un prototype de moteur d'exécution sous la forme d'une application web permettant de stocker d'une part des modèles de processus collaboratifs, d'autre part des patrons de collaboration, puis d'assister le chef de projet et les acteurs concernés dans l'exécution d'un processus donné. Une validation expérimentale a été menée sur la base d'une étude de cas représentative des processus de conception logicielle.

Chapitre 1

Introduction

Sommaire

1.1 Contexte

De nos jours, il est largement admis que la collaboration est indispensable pour la réalisation d'un système complexe (Le Moigne, 1999) (développement logiciel, fabrication d'un avion, etc). Dès lors, la description et la formalisation de cette collaboration deviennent une nécessité pour contrôler et supporter son exécution. Dans la littérature, il existe plusieurs définitions de la collaboration (Ellis et al., 1991; Briggs et al., 2006). Elle est, en général, définie comme la coordination de l'ensemble des activités, la coopération dans un espace de travail partagé et la communication au sein d'un projet.

Dans les dernières décennies, de nombreuses recherches ont été menées dans différentes communautés, notamment *"Computer-Supported Cooperative Work (CSCW)"* et *"Process Management"*, sur divers aspects des travaux collaboratifs (Robillard and Robillard, 2000; Mistrík et al., 2010). La communauté *CSCW* s'intéresse spécialement à la définition des processus collaboratifs et au développement des outils facilitant la collaboration tandis que la communauté

Process Management se concentre plus davantage la modélisation et la mise en oeuvre de la collaboration dans le but de gérer les processus collaboratifs à l'exécution. Ces travaux couvrent la façon de penser et de travailler, la manière de modéliser, contrôler et supporter la collaboration.

Nos travaux sont en continuité de plusieurs études qui ont été menées par notre équipe de recherche sur l'ingénierie des processus logiciels. Ces études ont porté aussi sur le développement de langages pour modéliser les processus et les environnements pour contrôler leur exécution. Concernant la modélisation de procédés, un premier travail a été effectué sous la forme d'un langage de modélisation de procédés exécutable inspiré du langage Eiffel (Crégut and Coulette, 1996; Crégut and Coulette, 1998). La notion de patron de processus paramétrable a ensuite été définie par Tran et al. (Tran et al., 2007; Diaw et al., 2011; Tran et al., 2011). Plusieurs extensions de SPEM ont été développées pour modéliser d'une part les processus à base de modèles (Diaw et al., 2011), d'autre part les processus collaboratifs (Kedji et al., 2012). Concernant l'exécution des processus, un moteur d'exécution a été développé dans le cadre de l'environnement support RHODES (Crégut and Coulette, 1996; Crégut and Coulette, 1998). (Kabbaj et al., 2008) ont proposé un environnement pour gérer les déviations lors de l'exécution des processus. (Kedji et al., 2012) ont développé un outil pour supporter l'exécution des processus collaboratifs ad'hoc.

Nos travaux de recherche sur la gestion des processus collaboratifs s'inscrivent dans la continuité des travaux présentés ci-dessus. Cette thèse commencée au sein de *l'Université Cheikh Anta Diop de Dakar* a été poursuivie dans le cadre d'une cotutelle avec *l'Université de Toulouse Jean Jaurès* au sein de l'équipe *SMART* de l'*IRIT*.

1.2 Problématique

L'un des objectifs de la gestion de processus est de contrôler l'exécution des processus, c'est-à-dire coordonner les différentes parties prenantes pour leur permettre de collaborer efficacement afin d'atteindre un but commun. Considérant qu'une tâche est la plus petite unité de travail d'un processus, nous distinguons deux types de collaboration : (1) la collaboration "faible" entre les acteurs qui exécutent différentes tâches dans le cadre d'une activité commune ; (2) la collaboration "forte" entre les acteurs au sein d'une même tâche. Si la collaboration faible est généralement bien gérée dans les systèmes de gestion de processus (Heinl et al., 1999; Charoy

et al., 2006; Rumpe, 2014), la collaboration forte n'a été abordée que par très peu d'auteurs (Kedji et al., 2014). En conséquence, contrôler finement l'exécution d'une tâche collaborative, avec plusieurs participants, est souvent une gageure (Ellner et al., 2010; Liptchinsky et al., 2014).

Nous nous intéressons à la mise en œuvre de la collaboration forte au sein d'une tâche collaborative exécutée par plusieurs acteurs. Dans cette thèse, nous nous concentrons sur la notion de tâche ***multi-instance***, une forme particulière de tâche collaborative faisant intervenir plusieurs acteurs jouant le même rôle dans sa réalisation. De ce fait, chaque acteur intervient sur une instance spécifique de la tâche durant l'exécution mais tous les acteurs partagent le même objectif, généralement la création ou la maintenance d'un produit commun. Ainsi, l'ensemble des instances de tâche contribuent à l'accomplissement de la tâche collaborative. Les relations entre les instances d'une tâche multi-instance sont définies selon la stratégie de collaboration choisie qui détermine l'ordonnancement et l'interaction de ces instances durant l'exécution de la tâche. Dans la suite, nous employons le terme de "tâche collaborative" dans le sens de tâche multi-instance.

Nous identifions deux besoins pour une gestion optimale d'une tâche collaborative. Le premier besoin est de pouvoir gérer les relations entre ses instances pour fournir une vision fine de la collaboration au sein de la tâche. Le deuxième besoin est de permettre à la tâche collaborative d'adapter dynamiquement sa stratégie de collaboration aux changements de contexte dans un projet donné. A notre connaissance, il n'y a pas encore de solution satisfaisante pour répondre à ces deux besoins en même temps. Pour disposer des informations sur les relations entre les instances d'une tâche collaborative, une solution possible est de définir explicitement ces relations lors de la modélisation. En figeant les instances d'une tâche collaborative dans le modèle, cette solution ne permet pas d'adapter l'exécution de la tâche collaborative à une autre stratégie de collaboration lorsque le contexte d'exécution change.

Le défi pour cette thèse est de proposer une gestion à la fois fine et flexible des tâches collaboratives. Plus précisément, nous cherchons à répondre aux deux questions suivantes :

QR 1. Comment représenter une stratégie de collaboration qui définisse, à l'exécution, les relations entre les instances d'une tâche collaborative ?

QR 2. Comment permettre à une tâche collaborative de choisir dynamiquement sa stratégie de collaboration durant l'exécution pour s'adapter aux changements de contexte ?

1.3 Objectifs de la thèse

Pour répondre aux questions de recherche formulées ci-dessus, nous visons une approche permettant d'une part la modélisation des stratégies de collaboration et d'autre part l'application dynamique d'une stratégie de collaboration à une tâche collaborative durant son exécution. L'idée est de ne pas définir statiquement, au moment de la modélisation, une seule stratégie pour exécuter la tâche collaborative dans toutes les circonstances, mais de laisser les acteurs du processus choisir, au moment de l'exécution, une stratégie de collaboration correspondant à leurs exigences dans un contexte spécifique. Nous pouvons ainsi offrir la flexibilité à l'exécution en adoptant le principe d'une modélisation partielle ("looseness" en anglais) lors de la conception du processus.

Pour cela, nous introduisons d'abord le concept de *patron de collaboration* pour représenter une stratégie de collaboration. Un patron de collaboration fournit un modèle d'exécution décrivant les relations entre les instances d'une tâche collaborative. Appliquer un patron à une tâche collaborative à l'exécution revient à définir la façon de mettre en oeuvre la collaboration au sein de ses instances.

Pour formaliser le concept de patron de collaboration et implémenter son utilisation, nous proposons un langage de modélisation de processus appelé *ECPML (Executable Collaborative Process Modeling Language)*. Ce langage permet de modéliser non seulement le processus statiquement mais aussi son instance à l'exécution. Ainsi, ECPML est utilisé pour décrire les processus et aussi les patrons de collaboration. ECPML est défini avec une *sémantique opérationnelle* pour permettre d'instancier et d'exécuter les éléments de processus. De cette façon, nous pouvons appliquer dynamiquement des patrons de collaboration pour permettre une exécution flexible des tâches collaboratives.

Pour satisfaire ces objectifs, notre travail a donné lieu aux contributions suivantes :

Un langage de représentation de processus collaboratifs. Il s'agit de proposer une représentation des processus collaboratifs avec ECPML de manière à disposer à la modélisation et à l'exécution de tous les concepts permettant de contrôler l'exécution d'une tâche durant son cycle de vie.

Un ensemble de patrons de collaboration. Les patrons décrivent les stratégies typiques de collaboration. L'approche permet, lorsqu'on dispose d'un modèle de processus, de

choisir en fonction du contexte d'exécution, le patron de collaboration le plus approprié. En effet, le contexte d'exécution concerne les éléments qui peuvent influer la manière de collaborer. Il s'agit du nombre d'acteurs disponibles, de la composition éventuelle des artefacts en entrée/sortie de la tâche, des dépendances le cas échéant entre les sous-éléments d'un artefact, etc. Le modèle d'exécution d'un patron est décrit avec le langage ECPML.

Une sémantique opérationnelle. La sémantique d'exécution des tâches collaboratives permet un suivi de l'exécution tout en permettant son évolution dans le cas où un changement de contexte aurait lieu. La sémantique des éléments du processus est fondée sur la notion de machines à états d'UML. Elle définit les états, les événements et les actions associés à chaque élément. Cette formalisation permet de contrôler l'exécution du processus en assurant les changements d'états de chaque élément. Cette sémantique à base de patrons utilise le mécanisme de late-binding pour l'instanciation d'une tâche collaborative.

Un moteur d'exécution de processus collaboratifs. Notre moteur d'exécution, CPE ("Collaborative Process Engine"), est une application web chargée de l'instanciation et de l'exécution d'un processus. Il supporte les différentes fonctionnalités permettant l'instanciation du processus et le suivi de l'exécution de chaque tâche (collaborative ou non). Dans le prototype de CPE développé, le gestionnaire du processus est chargé du choix des patrons de collaboration pour l'exécution.

1.4 Structure du mémoire

La suite du document est organisée comme suit :

Chapitre 2 : Ce chapitre est consacré à l'état de l'art de notre travail de thèse. La revue de la littérature a été organisée en trois grandes sections. Nous présentons, dans un premier temps, les fondements des processus, puis les deux exigences (collaboration et flexibilité) sur lesquelles se base notre étude bibliographique. Nous présentons ensuite une synthèse des travaux de la littérature sur la définition de la collaboration, la modélisation et l'exécution flexible au regard des exigences mentionnées ci-dessus. Ce chapitre est conclu par un rappel des objectifs et une justification de nos contributions.

Chapitre 3 : Dans ce chapitre, nous présentons le méta-modèle ECPML conforme à MOF. Nous décrivons le volet structurel qui permet de définir les modèles d'ECPML et le volet comportemental qui permet de définir les cycle de vie des éléments d'un modèle d'ECPML sous forme de machines à états d'UML. Un exemple fil conducteur permet d'illustrer l'approche.

Chapitre 4 : Ce chapitre propose un ensemble de patrons de collaboration formalisés avec notre langage ECPML et permettant d'instancier un processus. Ces patrons sont présentés suivant différents contextes d'exécution en prenant en compte à chaque fois la situation concernant les ressources.

Chapitre 5 : Le chapitre 5 présente la sémantique d'exécution d'un processus collaboratif. Nous y décrivons les comportements des différents éléments d'un processus (tâche, acteur, produit). Les comportements de ces éléments sont présentés sous forme de machines à états d'UML. Pour chaque action, nous présentons un algorithme qui est implémenté dans le prototype.

Chapitre 6 : Dans ce chapitre, nous décrivons le prototype de l'outil CPE ("Collaborative Process Engine") permettant d'instancier et d'exécuter un processus collaboratif. Ce prototype implémente les différents algorithmes des machines à état des éléments du processus. Nous présentons ensuite une étude de cas dans le domaine de la conception logicielle permettant de valider notre approche. Cette étude de cas est mise en oeuvre avec CPE.

Chapitre 7 : Ce chapitre conclut ce manuscrit et présente un ensemble de perspectives de ce travail de thèse.

Chapitre 2

Etat de l'art

Sommaire

2.1 Introduction

Comme présenté dans le chapitre précédent, notre travail de recherche porte sur la problématique de la gestion (définition, modélisation et exécution) de la collaboration dans les processus (logiciels, systèmes ou métiers). Le but de ce chapitre est de faire une étude bibliographique centrée sur les travaux relatifs aux processus collaboratifs.

De nos jours, il est largement admis que la collaboration est incontournable dans la mise en oeuvre du processus d'élaboration d'un système complexe (développement logiciel, fabrication d'un avion, etc). Dès lors, sa description et sa formalisation sont une nécessité pour mieux contrôler et supporter son exécution. Dans la littérature, il existe plusieurs définitions de la collaboration (Ellis et al., 1991; Raposo et al., 2001; Polancic et al., 2015). Elle est, en général, définie comme une activité ou une tâche pouvant être exécutée par plusieurs individus dans le but d'atteindre un objectif commun.

Dans ce chapitre, nous nous intéressons à la place de la collaboration dans l'ingénierie des processus. De ce fait, nous illustrons les approches traitant de la définition, de la modélisation et de l'exécution de la collaboration aussi bien dans le domaine du *"Process Management"* que du *"Workflow Management"*. Les travaux sont classifiés et analysés sur la base de deux caractéristiques que nous considérons comme des exigences que sont la *collaboration* et la *flexibilité*.

Nous présentons dans un premier temps (section 2.2) les fondements des processus. L'objectif n'est pas de refaire l'historique des processus mais de présenter une synthèse des différents concepts du domaine que nous utilisons tout au long de cette thèse. Nous introduisons de ce fait la collaboration ainsi que ses différentes formes d'utilisation dans la littérature.

Dans un second temps (section 2.3), nous discutons des processus collaboratifs en mettant l'accent sur la notion de collaboration. Nous donnons la définition d'un processus collaboratif et présentons les défis de la gestion de tels processus.

Dans la section 2.4, nous présentons les deux exigences (*collaboration* et *flexibilité*) ciblées par notre étude bibliographique. La première exigence est décomposée en trois aspects (*coordina-*

tion, *communication* et *coopération*) alors que la deuxième est déclinée en quatre (*flexibilité par conception*, *flexibilité par spécification incomplète*, *flexibilité par déviation* et *flexibilité par changement*).

Dans la section 2.5, nous présentons les approches existantes au regard de ces exigences. Un bilan des approches étudiées pour chaque exigence est présenté sous forme de tableau comparatif pour montrer les limites et/ou manques de chaque approche par rapport à nos objectifs.

Nous concluons ce chapitre d'état de l'art en rappelant nos objectifs et les limites des travaux existants, et en justifiant les contributions qui seront présentées dans les chapitres suivants.

2.2 Fondement des processus

Un processus est un ensemble d'activités qui doivent être réalisées dans des conditions déterminant leur ordre (van der Aalst and van Hee, 2002), en vue d'atteindre un but commun. Toutes les organisations utilisent un ou des processus dans le développement de leurs activités, qui peuvent différer bien sûr par leur degré de sophistication. Pour son analyse et son automatisation, il est nécessaire d'utiliser un modèle permettant de décrire les différentes étapes du processus et de disposer d'outils pour son contrôle. Les modèles de processus sont décrits par le biais de *SGP* (*Systèmes de Gestion de Processus*) connus en anglais sous le terme de *PMS* (*Process Management System*).

Les deux phases que l'on distingue classiquement dans la gestion de processus sont la *modélisation* (*"Process modeling"*) et l'*exécution ou mise en oeuvre* ("*Process enactment"*). Compte tenu de cette catégorisation, la gestion de processus peut être définie comme une discipline permettant la représentation d'un processus dans le but de sa formalisation et du contrôle de son exécution (Curtis et al., 1992; Roschelle, 1996).

Dans les sections suivantes, nous présentons d'abord les concepts de base d'un processus, puis la modélisation et la mise en oeuvre de processus.

2.2.1 Concepts de base des processus

Les concepts présentés dans cette section sont inspirés de SPEM (Object Management Group (OMG), 2008), l'un des standards pour la représentation des processus.

2.2.1.1 Tâche

Une tâche est une unité de travail de base contrôlable dans un processus. Les tâches sont exécutées par des acteurs jouant des *rôles*. Une tâche manipule des produits (artefacts) pour faire évoluer leur état.

Pour coordonner les tâches dans un projet, il faut définir des flux de contrôle entre elles. Un *flux de contrôle* (*"control-flow"*) décrit une contrainte temporelle sur l'exécution de deux tâches. Par exemple, une tâche B peut être exécutée successivement (*séquentiel*) à une tâche A ou simultanément (*parallèle*).

Au sens de la collaboration forte que nous ciblons dans cette thèse, une tâche collaborative est une tâche exécutée par plusieurs individus jouant le même rôle (Kedji et al., 2012).

Plusieurs tâches peuvent être regroupées sous forme d'*activité*. Dans SPEM par exemple, une activité est définie comme un élément qui décrit une unité de travail dans un processus. Elle représente, selon le méta-modèle SPEM, un bloc (contenant des tâches, acteurs et produits) assignable à un ou plusieurs rôles. Dans notre travail, nous visons principalement le contrôle de l'exécution des processus. De ce fait, nous nous intéressons aux tâches, qui sont les éléments assignables à un rôle.

2.2.1.2 Rôle

Un rôle décrit la qualification d'une personne devant exécuter une tâche. Il regroupe les compétences nécessaires pour réaliser une tâche dans un projet. Un rôle peut être joué durant l'exécution par une entité (par exemple un acteur humain) qui exécute une tâche en utilisant éventuellement une ressource non humaine (par exemple un outil).

2.2.1.3 Produit

Un produit représente un artefact manipulé par une tâche. La relation entre un produit et une tâche s'exprime sous forme de *flux de données* ("*data-flow*"). Le flux de données indique la façon dont les produits sont utilisés dans une tâche. Un produit peut ainsi être en *entrée* d'une tâche, en *sortie* ou bien en combinaison des deux (*entrée-sortie*)

2.2.2 Modélisation de processus

La modélisation d'un processus a pour objectifs : (1) de produire une représentation du processus permettant une communication entre les acteurs du processus ; (2) de formaliser le processus en vue d'avoir un support pour contrôler son exécution.

La modélisation consiste en l'élaboration d'une description d'un processus sous la forme d'un *modèle de processus* (*process model*) (Feiler and Humphrey, 1993; Zamli and Lee, 2001; Lonchamp, 1993). Cette description, faite au moyen d'un *langage de modélisation de processus* (*Process Modeling Language (PML)*), permet de définir les différents éléments composant un processus (Acuña and Ferré, 2001; Bendraou et al., 2010). Ces éléments (Tâche, Produit, Rôle) sont liés pour donner naissance à un modèle de processus.

La Figure 2.1 montre, sous forme simplifiée, les principales relations entre les concepts de base d'un processus. Un rôle joué par un acteur du processus est chargé de l'exécution d'une tâche. Durant son exécution, la tâche manipule un produit. La sémantique de cette manipulation n'est pas précisée sur cette figure. Cependant, il peut s'agir d'une création ou d'une modification.

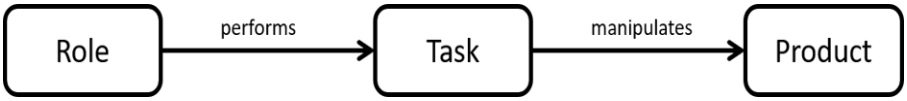


FIGURE 2.1: Modèle conceptuel de base d'un processus (extrait).

2.2.3 Mise en oeuvre de processus ("Process enactment")

Pour mettre en œuvre un processus, il faut avoir recours à un langage de modélisation de processus ayant une sémantique opérationnelle. Celle-ci permet d'implémenter le moteur de

processus pour contrôler son exécution. La mise en œuvre comprend le *déploiement* du processus (*instanciation des différents éléments du modèle de processus* et *affectation des instances de tâches aux acteurs*) et le contrôle du processus durant l'*exécution*.

— L'étape de *déploiement* est composée de deux sous-étapes : *instanciation* et *assignation*.
  - Dans l'*instanciation*, chaque élément du modèle de processus est instancié. Un processus peut être mis en oeuvre dans différents projets. L'instanciation d'un processus crée les éléments de processus dans le contexte d'un projet spécifique. Pour les tâches, produits et rôles du modèle, nous obtenons respectivement des *instances de tâches*, des *instances de produits* et des *acteurs*.
    - Une instance de tâche (par exemple "revue d'un document par Bob") représente une version exécutable d'une tâche exécutée par un seul acteur associé à un rôle dans le modèle de processus.
    - Une instance de produit (par exemple "version d'un document révisé") représente un artefact consommé, modifié ou créé par une instance de tâche.
    - Une instance de rôle représente une entité ayant les qualifications nécessaires pour exécuter une tâche. Un acteur est une entité externe ayant les compétences pour jouer un rôle donné. Ce peut être un acteur humain dans le cas d'une tâche manuelle ou interactive, ou un programme dans la cas d'une tâche automatique.
  - La phase d'instanciation est suivie de l'*assignation* des instances aux acteurs réels du projet. Par exemple l'instance de tâche "revue d'un document" pourra être assignée à l'acteur Bob.
— L'*exécution* est l'étape durant laquelle les acteurs exécutent les instances de tâches en manipulant les instances de produit suivant les flux de données définis lors de la modélisation. Un moteur de processus contrôle la coordination des tâches assignées aux acteurs du processus.

Les relations entre éléments du modèle à la modélisation seront les mêmes entre les instances de ces éléments à l'exécution. De ce fait, la relation entre une tâche A et une tâche B sera la même entre l'instance de tâche de A et l'instance de tâche de B. Dans ce contexte, nous pouvons obtenir un modèle de processus exécutable. La Figure 2.2 montre les éléments d'une instance de processus et les relations entre eux. Un acteur jouant un rôle est chargé de l'exécution d'une instance de tâche. Durant son exécution, l'instance manipule (création ou modification) une

instance de produit.

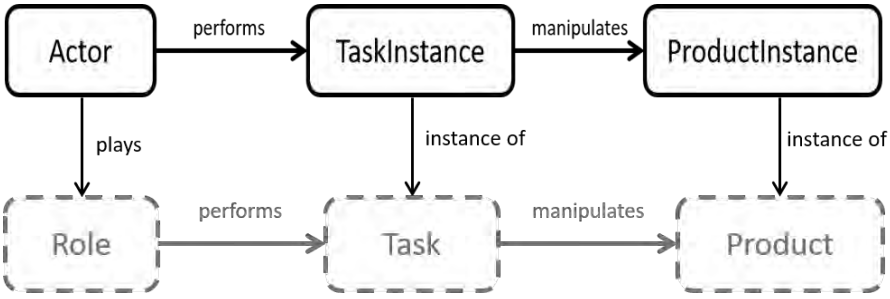


FIGURE 2.2: Modèle conceptuel de base d'une instance de processus (extrait).

2.2.4 Flexibilité du processus

L'exécution consiste à dérouler le cycle de vie du projet ou de l'activité considérant que le processus est une activité. Elle concerne également l'assistance aux acteurs humains en leur proposant les opérations à exécuter et les outils d'exécution (Kabbaj et al., 2007; Diaw, 2011; Ariouat et al., 2016). Durant l'exécution d'un processus, divers besoins de changement peuvent apparaître (Fisichella and Matera, 2011). Cependant, pour une exécution controlée, il est admis que les processus doivent être en mesure de s'adapter aux changements d'environnements ou de stratégies (Reichert and Weber, 2012; Schonenberg et al., 2008b). En d'autres termes, l'exécution d'un processus doit permettre une certaine flexibilité pour pouvoir s'adapter aux changements sans perdre le contrôle de l'avancement du processus.

La flexibilité est la capacité du processus, à l'exécution, à réagir aux changements prévus ou imprévus. L'exécution flexible d'un processus permet aux acteurs de s'adapter aux situations imprévues en gardant le contrôle sur l'état du processus ou de les éviter en gérant les déviations (Kabbaj et al., 2007; da Silva et al., 2010).

Il existe plusieurs travaux traitant de la flexibilité (Reichert and Dadam, 1998; Heinl et al., 1999; van der Aalst et al., 2009; Reichert and Weber, 2012; Mundbrod et al., 2015). Dans la suite, nous nous inspirons de la taxonomie de (Schonenberg et al., 2008a), qui propose quatre types de flexibilité :

— la *flexibilité par conception* (*"Flexibility by design"*) : c'est la possibilité de prévoir des chemins d'exécution alternatifs (*variantes*) durant la modélisation du processus de

sorte que le chemin d'exécution le plus approprié puisse être sélectionné au moment de l'exécution pour chaque instance du processus,

— la *flexibilité par spécification incomplète* (*"Flexibility by underspecification"*) : avec ce mécanisme, le modèle de processus est exécuté tout en étant partiel au moment de l'exécution. Les parties du processus non précisées sont complétées durant l'exécution,

— la *flexibilité par déviation* (*"Flexibility by deviation"*) : elle permet aux utilisateurs d'ignorer une partie du processus spécifié afin d'exécuter d'autres tâches. La déviation est définie comme une incohérence entre le modèle du processus logiciel et l'exécution observée par l'environnement (Kabbaj et al., 2007; da Silva et al., 2011). Nous proposons trois étapes pour gérer les déviations quand elles sont détectées :

- *Une prise en main tardive.* L'idée est d'ignorer la déviation sur le moment et de décaler sa correction.
- *Une évaluation des risques.* Les utilisateurs doivent avoir une vision des impacts de la déviation sur les objectifs du processus.
- *Un support de correction.* Avoir un plan de correction est nécessaire pour les utilisateurs, compte tenu du risque d'augmentation du nombre d'incohérences (Egyed et al., 2008).

— la *flexibilité par changement* (*"Flexibility by change"*) : c'est la possibilité de changer la définition du processus pendant l'exécution de sorte que le modèle d'une instances de processus en cours d'exécution migre vers le nouveau modèle obtenu.

2.3 Processus collaboratifs

Dans cette section, nous présentons les notions de collaboration et de processus collaboratifs. La clarification de ces deux notions, qui sont au cœur de notre problématique de thèse, permet de présenter les défis dans la gestion des processus collaboratifs.

2.3.1 Notion de collaboration

La collaboration est l'association de plusieurs acteurs pour atteindre un but commun (Briggs et al., 2006). Une collaboration peut être menée pour répartir le travail entre plusieurs individus

afin d'améliorer la productivité ou pour combiner les compétences individuelles et profiter ainsi de l'intelligence collective pour que la réalisation de la tâche soit plus efficace (Weiss, 2005). En effet, la collaboration peut s'affranchir des contraintes spatio-temporelles c'est-à-dire ne nécessite pas une présence physique des acteurs au même endroit ni une disponibilité au même moment. Elle nécessite donc des moyens de gestion des travaux individuels pour surmonter et respecter ces contraintes.

Pour gérer la collaboration, plusieurs aspects sont à considérer. Inspiré par (Ellis and Wainer, 1994), Fuks (Fuks et al., 2005) a défini le *modèle 3C* (Coordination, Communication, Coopération) constitué des aspects suivants :

— La *coordination* : elle fait référence à la gestion des individus, la synchronisation des activités qui sont accomplies par chacun, et les ressources utilisées. C'est l'acte de gérer les interdépendances entre les activités ou tâches exécutées dans l'atteinte d'un objectif (Malone and Crowston, 1990). Selon ces auteurs, les éléments qui influencent la coordination dans un processus collaboratif sont les suivants :

- Le *"work sequence"* : il représente une relation temporelle et une contrainte sur l'ordre d'exécution entre deux tâches,
- Le *"data flow"* : ce critère porte sur le passage des données entre les différentes tâches. Ce passage peut porter sur deux tâches de natures différentes ou bien sur les instances d'une même tâche,
- Les *"shared resources"* : différentes ressources peuvent être partagées par les tâches. Une ressource exécute une tâche (par exemple un acteur) ou bien est utilisée pour l'exécution d'une tâche (par exemple un outil).

— La *communication* : La communication est l'échange d'informations entre les membres d'une équipe (Lanubile, 2009). La coordination a souvent un impact dans la communication. En effet, plus le besoin de coordination est fort, plus la communication doit être rigoureuse pour éviter des erreurs qui pourraient subvenir dans l'exécution du processus. Les éléments sur lesquels la communication peut porter sont ceux qui ont un impact sur la réalisation des objectifs du processus. Nous pouvons ainsi considérer :

- *l'état global du système* (*"Awareness system state"*). Avec ce critère, nous pouvons décrire l'*état des tâches*, l'*état des artéfacts*, l'*assignation des ressources (acteurs et outils)*. La communication par rapport à ces éléments peut être *formelle* ou *informelle*.

- les *discussions* : ce critère concerne les décisions formelles autour du processus (de la modélisation et de l'exécution).

— La *coopération* : La coopération (production d'artefacts dans un espace partagé) porte sur la création des produits utilisés au cours de l'exécution du processus ou sur les produits modifiés par les utilisateurs durant l'accomplissement de leurs tâches (Laurillau and Nigay, 2002). La production des artefacts peut donner lieu à des éléments concrets ou tangibles (documents) ou à des éléments intangibles (décisions, votes) suivant la nature du processus étudié. Pour cet aspect, nous nous intéressons à la manière dont les produits sont manipulés dans les différentes tâches collaboratives du processus. Nous aurons alors les critères suivants :

- *"Data Distribution"* : Cet élément décrit comment les produits sont utilisés par une tâche donnée. Un produit peut être en *entrée* seule, en *sortie* ou en *entrée-sortie*.
- *"Data Transfer"* : Cet élément décrit le mode d'accès au produit. Il représente la façon selon laquelle le produit manipulé ou créé est obtenu de son dépôt. On note ici deux modes d'accès au produit : par *référence* ou par *copie*. Dans l'accès par référence, seule l'adresse du produit est utilisée. Toute modification de ce dernier impacte alors les tâches qui manipulent le même produit. Dans l'accès par copie, une version du produit à un instant est utilisée. Toute modification de ce produit n'impacte que cette version, les autres copies gardant leur valeur.
- *"Data Structure"* : Cet élément décrit la structuration du produit manipulé ou créé. La structure du produit définit sa composition. Un produit peut être *composite* (il contient plusieurs composants), ou *élémentaire*.

Le modèle décrit dans la Figure 2.3 représente une instanciation du modèle 3C pour les groupes de travail par (Fuks et al., 2005). Dans cette instanciation, les trois aspects sont combinés pour favoriser la *conscience partagée (awareness)* du processus. La coopération nécessite une communication accrue dans l'objectif des prises de décision à propos des situations qui pourraient se passer durant la collaboration. Cette communication à son tour pourra générer un besoin de coordination pour réorganiser par exemple les tâches durant la coopération.

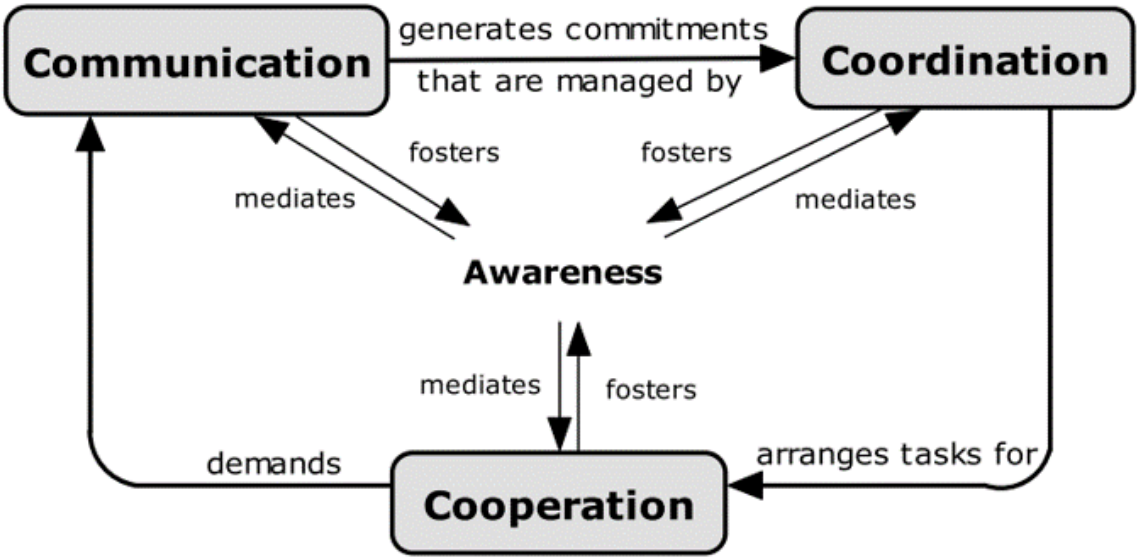


FIGURE 2.3: Modèle 3C instancié pour les groupes de travail (Fuks et al., 2005).

2.3.2 Notion de Processus Collaboratif

La notion de processus collaboratif a été introduite pour formaliser et gérer la collaboration au sein d'un processus. Un modèle de processus collaboratif décrit la répartition du travail en son sein ainsi que les contraintes imposées. Une instance de processus collaboratif représente la collaboration entre acteurs. Dans la suite de cette thèse, nous adoptons la terminologie suivante :

Processus collaboratif : c'est un processus qui est réalisé par différents acteurs. Un processus est décomposé en différentes tâches. Les acteurs peuvent réaliser des tâches seuls ou peuvent participer à la réalisation d'une même tâche. La figure 2.4 montre un exemple de processus modélisé en SPEM 2.0 sous forme de trois tâches : *"Design Modules"* réalisée par un concepteur (*Designer*), *"Develop Module 1"* et *"Develop Module 2"* réalisées chacune par un développeur (*Developer*). Les tâches *"Develop Module 1"* et *"Develop Module 2"* utilisent respectivement les produits *"Module 1"* et *"Module 2"*.

Comme mentionné dans l'introduction (section 2.1), nous distinguons deux formes de collaboration :

— *Collaboration faible* : Cette collaboration consiste en une répartition du travail entre plusieurs acteurs pour accomplir différentes tâches. Ces acteurs peuvent jouer le même rôle ou des rôles différents. C'est le type de collaboration le plus courant.
Dans la figure 2.5, la collaboration porte sur la coordination entre la tâche *Design Mo-*

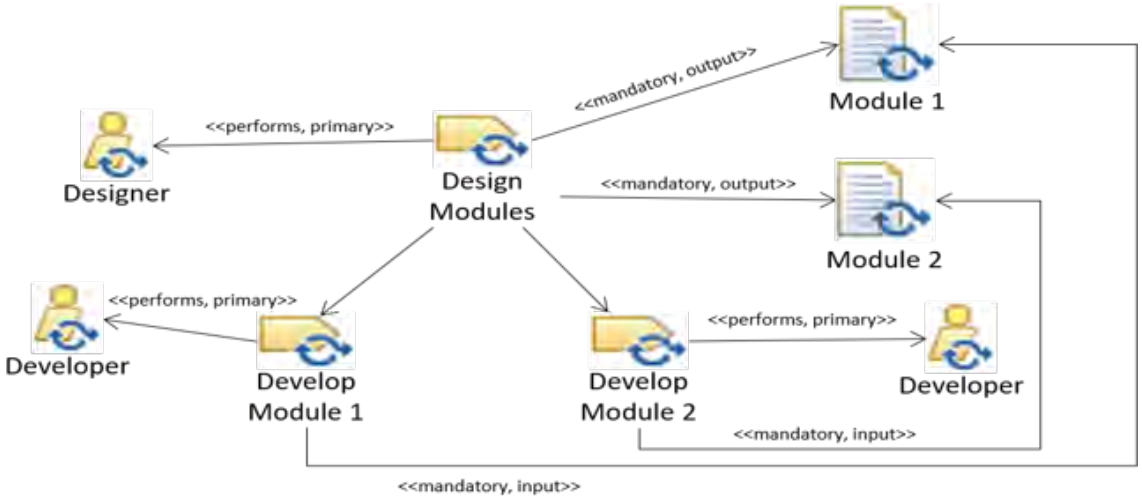


FIGURE 2.4: Exemple de processus simple modélisé avec SPEM 2.0.

dules et les deux tâches de développement (*Develop Module 1* et *Develop Module 2*). Cette collaboration permet de spécifier la relation entre les différentes tâches du processus (*"inter-task relation"*). Les tâches dans cet exemple sont réalisées par les rôles *designer* et *developer* (voir figure 2.5).

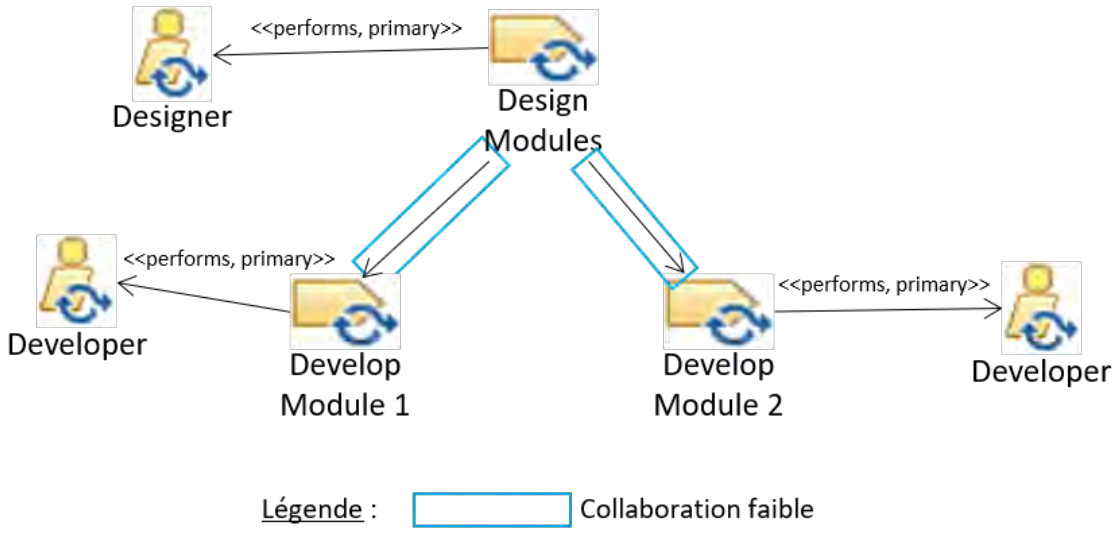


FIGURE 2.5: Exemple de collaboration faible.

— *Collaboration forte* : cette collaboration consiste en une répartition du travail entre plusieurs acteurs pour accomplir la même tâche. Ces acteurs jouent le même rôle.
Dans notre exemple, il peut arriver que la tâche *"Design Modules"* soit collaborative. La collaboration dans cette tâche va porter sur la relation entre ses différentes instances et leur coordination. De ce fait sur la figure 2.6, nous avons trois acteurs, *Alice, Bob, Eve*, jouant le rôle de *Designer* pour la réalisation de trois instances de la tâche "Design

Module". Cette notion de collaboration forte nous amène à considérer une tâche sous forme de plusieurs instances.

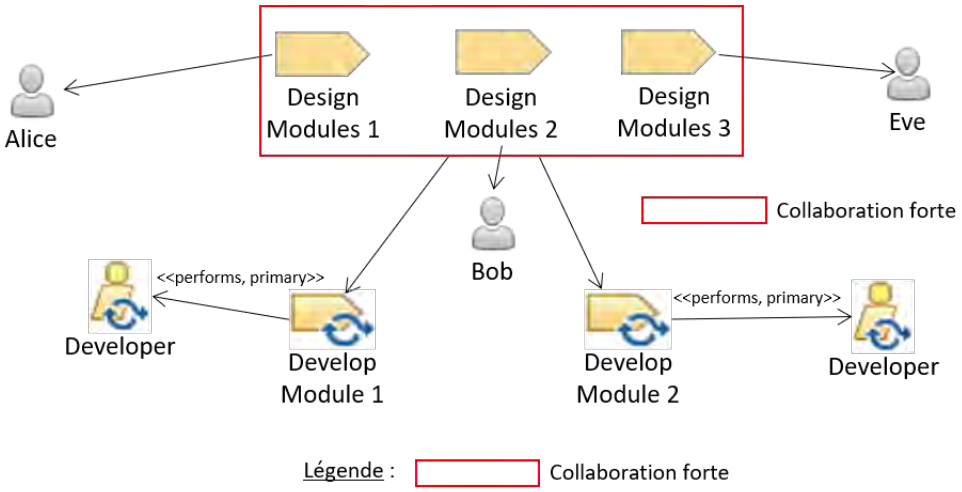


FIGURE 2.6: Exemple de collaboration forte.

Tâche collaborative : une tâche collaborative est une tâche réalisée par une collaboration forte. A l'exécution, la tâche collaborative donne lieu à plus d'une instance de la même tâche. Nous parlons dans ce cas de *tâche multi-instance*.

Dans la suite de cette thèse, nous focalisons notre étude sur la notion de collaboration forte. Un processus collaboratif est un processus contenant une ou plusieurs tâches collaboratives.

La Figure 2.7 représente la décomposition d'une instance de tâche collaborative en plusieurs instances de tâches élémentaires. De ce fait, l'instance d'une tâche collaborative est désignée par le terme de *tâche multi-instance*. Le nombre d'instances élémentaires dépend de la complexité de la tâche collaborative, et du contexte du projet.

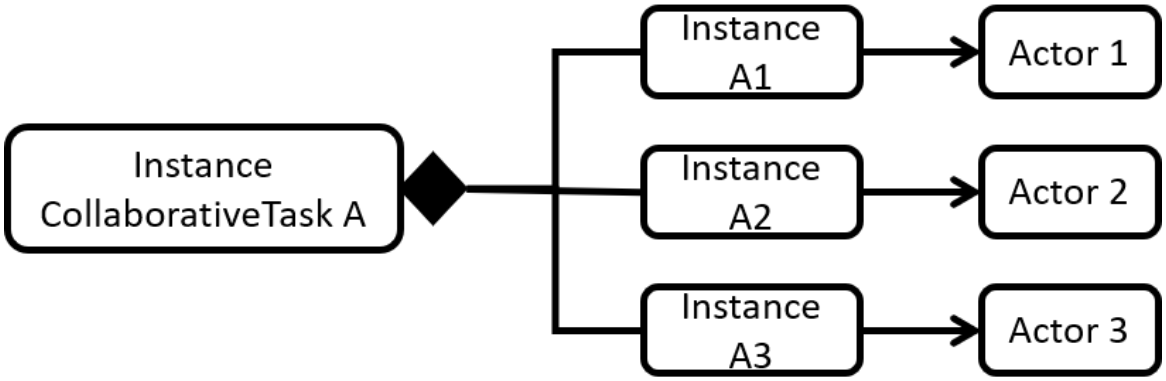


FIGURE 2.7: Décomposition d'une instance de tâche collaborative.

Dans cette figure, *instance A1*, *instance A2* et *instance A3* sont des instances de la tâche collaborative A. Ces instances sont gérées à l'exécution et sont assignées chacune à un acteur spécifique.

Pour permettre un contrôle fin des interactions entre les acteurs du processus, une bonne coordination est essentielle. (Malone and Crowston, 1990) ont proposé une *théorie de la coordination*. Cette dernière est importante dans la collaboration parce que permettant une bonne utilisation des ressources utiles à l'accomplissement des tâches. La coordination permet de savoir à quel moment chaque participant du processus doit intervenir. Elle permet aussi d'influer sur l'utilisation des produits en les rendant disponibles pour l'activité adéquate et ce au bon moment dans le cycle de vie d'un processus.

Avec l'avènement du développement agile qui permet une planification souple et adaptative au changement (évolution fonctionnelle et/ou évolution technique), les exigences d'un projet ne sont plus fixées dès le début. Il est alors difficile d'avoir un modèle de processus complet dès le début sans changement au cours de l'exécution. L'impact sur la collaboration est que les activités et les rôles de chaque participant ne peuvent pas forcément être définis et exhaustifs au début de l'exécution. Ajouté à cela, il peut se produire des changements de contexte amenant une évolution et fréquente du processus. Pour supporter ces changements de contexte, les processus collaboratifs doivent être agiles d'où la notion de flexibilité à l'exécution.

2.4 Exigences pour la Gestion de Processus Collaboratifs

Une gestion efficace de processus collaboratifs devrait fournir un contrôle à la fois fin et flexible de ces processus. Pour atteindre cet objectif, il faut définir les exigences qu'un SGP (Système de Gestion de Processus) doit satisfaire pour supporter un processus collaboratif.

Le manque d'informations fournies par les SGP sur les relations entre artefacts et/ou le statut des membres de l'équipe est souvent constaté (Dustdar, 2004). Ce manque impacte l'un des aspects les plus importants des systèmes qui est la *communication* (Ellis et al., 1991; Ignat et al., 2015). En effet, cet aspect, appliqué tout au long du processus favorise la *coordination* et

la *coopération* dans le processus. Ces trois aspects permettent, ensemble, une gestion efficace de la collaboration.

Un ensemble d'exigences auxquelles devrait répondre un système de gestion de workflow supportant un travail collaboratif a été décrit dans (Charoy et al., 2006). La possibilité de *changer le type* ou *l'instance du processus* en cours d'exécution revêt un caractère important tout comme la flexibilité à l'exécution. Cela est justifié par le fait que les processus collaboratifs sont pour la plupart étendus dans le temps (Charoy et al., 2006).

Nous déduisons de cela deux exigences importantes pour un processus : la *collaboration* et la *flexibilité*. Ces exigences, détaillées dans les sous-sections ci-dessous, permettent de faire ressortir des critères sur lesquels nous avons fondé notre étude bibliographique. Chaque *exigence* peut être définie selon différents aspects à couvrir pour la respecter. Les *aspects* à leur tour sont décomposés en *critères* pour lesquels nous définissons les différentes *solutions* utilisées par les approches pour leur réalisation. Nous obtenons ainsi le tableau générique 2.1 ci-dessous qui sera utilisé comme cadre pour l'analyse des différentes approches choisies.

Exigence	**Aspects**	**Critères**	**Solutions**
Exigence 1	Aspect 1	Critère 1	Solution 1, Solution 2
		Critère 2	Solution 1
	Aspect 2	Critère 1	Solution 1, Solution 2
		Critère 2	Solution 1, Solution 2
		Critère 3	Solution 1, Solution 2, Solution 3

TABLE 2.1: Tableau générique de décomposition d'une exigence.

2.4.1 Exigence de collaboration

Le concept de collaboration renvoie ici à la définition donnée à la dimension collaborative des systèmes de gestion de processus par (Ariouat et al., 2016). Cette définition correspond aussi au modèle 3C défini par Fuks (Fuks et al., 2005) qui est structuré selon les trois aspects suivants :

— *la coordination*,
— *la communication*,
— *la coopération*

Ces trois aspects serviront de cadre de référence pour l'évaluation et la comparaison des approches qui supportent l'exigence liée à la collaboration.

2.4.1.1 La coordination

Les critères proposés dans la littérature pour cet aspect sont : le *flux de travail ("work sequence")*, le *flux de données ("data flow")* et les *ressources partagées ("shared resources")* (Malone and Crowston, 1990). Il s'agit d'analyser les solutions mises en place par les travaux cités pour satisfaire ces critères de coordination.

Ces critères permettent une bonne coordination des tâches à l'exécution, c'est-à-dire des instances de tâches. Ils peuvent être précisés au moment de la modélisation du processus (*"modeling time"*) ou au moment de l'exécution (*"execution time"*).

Dans le cas du "modeling time", seules les relations inter-tâches (*"inter-task"*) sont décrites dans le modèle. Instancier ces relations ne permet de synchroniser que les instances de tâches différentes.

Dans le cas de "l'execution time", les relations entre les instances sont spécifiées au moment de l'exécution. Ces relations englobent à la fois celles entre les instances de différentes tâches (relations inter-tâches) et celles entre différentes instances d'une même tâche collaborative (*"intra-task"*).

2.4.1.2 La communication

L'aspect communication caractérise la manière dont les approches assurent la conscience partagée des éléments du processus. Les différents critères considérés ici sont la communication sur l'*état du système* (ou de l'exécution) et les *discussions*. Ces critères permettent d'assurer une communication en vue d'une bonne exécution du processus.

Lorsque la communication ne se fait pas à travers un outil, elle est souvent *informelle* : *discussion de couloirs*, *échanges spontanés*, *non officiels*, etc. La *communication formelle* d'un autre côté est organisée et maîtrisée. Elle peut se faire à travers les outils de la communauté CSCW (*"Computer-Supported Cooperative Work"*).

2.4.1.3 La coopération

L'aspect coopération est caractérisé par les critères définissant la manière dont les produits sont manipulés dans le processus. Il s'agit du *mode d'accès* (*"data distribution"*), du *mode de passage* (*"data transfer"*) et de la *structuration* (*"data structure"*). Les solutions pour ces aspects sont décrites au niveau de la section 2.3.1.

Le tableau 2.2 résume les aspects, les critères et les solutions concernant l'exigence de collaboration.

Exigence	**Aspects**	**Critères**	**Solutions**
Collaboration	Coordination	Flux de travail	Inter-tâche, Intra-tâche
		Flux de données	Inter-tâche, Intra-tâche
		Ressources partagées	Acteur, Outil
	Communication	Conscience sur l'état du système	Formelle, Semi-Formelle
		Discussions	Formelle, Semi-Formelle
	Coopération	Mode d'accès	In, Out, Inout
		Mode de passage	Référence, copie
		Structuration	Composite, Non-Composite

TABLE 2.2: Tableau récapitulatif des aspects et critères concernant la collaboration.

2.4.2 Exigence de flexibilité

Pour rappel, la flexibilité est la capacité du processus, à l'exécution, à réagir face aux changements prévus ou imprévus.

Les changements prévus peuvent être intégrés complètement lors de la modélisation via des structures exprimant des choix, des boucles, des abandons etc : il s'agit de la *flexibilité par conception* ("flexibility by design"). Si à la modélisation, il est prévu certains changements mais que les solutions pour les traiter ne soient pas encore définies, il s'agit de la *flexibilité par spécification incomplète* ("flexibility by underspecification"). Pour s'adapter aux changements imprévus, il est possible à l'exécution d'accepter les *déviations* ou "flexibility by deviation" ou de modifier le modèle de processus ("flexibility by change").

2.4.2.1 Flexibilité par conception

Comme décrit dans la section 2.2.4, il s'agit de prévoir des chemins alternatifs d'exécution dès la modélisation. Pour cet aspect, nous considérons le critère de *variabilité*.

Ce critère décrit la capacité à prévoir des exécutions alternatives dans le modèle de processus (Schonenberg et al., 2008b). Le choix d'un chemin d'exécution dépendra du contexte. Les chemins peuvent être décrits de différentes manières :

— Des *variantes de processus* (Rastrepkina, 2010). Les variantes sont fondées sur un processus originel appelé *modèle de processus de base*. Ce modèle de base est obtenu suivant certains principes. Il peut être la *variante la plus utilisée* pour éviter les configurations multiples. Il peut aussi être *l'intersection de toutes les variantes*. Cela fait que le choix d'une variante dans un contexte changeant ne nécessite pas de suppression mais juste des ajouts de fragments. Ce mécanisme est encore appelé *personnalisation par extension* dans (Rosa et al., 2017).

— des *"PML Constructs"* c'est-à-dire des constructions de langage de modélisation de processus qui permettent de modéliser les variantes possibles directement dans le processus. Dans ce cas de figure, le modèle de base est l'union de toutes les variantes. Dans (Rosa et al., 2017), il est encore appelé *personnalisation par restriction*. La restriction ici fait référence au fait qu'il faille enlever des fragments pour adapter le modèle à une exécution précise.

2.4.2.2 Flexibilité par spécification incomplète

Cet aspect illustre les solutions mises en place pour permettre une modélisation flexible du processus. Cette flexibilité permet aux processus d'anticiper sur les choix de contextes possibles avant l'exécution. Suivant la taxonomie de (Schonenberg et al., 2008a), nous associons à cet aspect le critère de *"looseness"* ou *modélisation partielle*.

Ce critère décrit le caractère permettant à l'aspect comportemental d'un processus d'être incomplet durant la modélisation. En effet, il est presque impossible pour certains types de processus (systèmes complexes, ...) de connaître tout le cheminement d'exécution dès la modélisation. Il est alors difficile de spécifier dès la modélisation l'exact flux de travail de ces processus. Les

auteurs (Schonenberg et al., 2008a) le définissent comme la capacité à exécuter un processus avec une spécification partielle de l'exécution.

Nous distinguons deux mécanismes pour satisfaire ce critère :

— Le *"late-modeling"* : permettant de choisir un fragment de processus parmi un ensemble déjà défini mais aussi d'en modéliser de nouveaux pour compléter la modélisation. Le choix des fragments dans le late-modeling se fait avant l'exécution.
— Le *"late-binding"* : permettant de choisir un fragment d'exécution parmi un ensemble déjà défini.

2.4.2.3 Flexibilité par déviation

Cet aspect illustre les solutions permettant de faire face aux imprévus qui se produisent lors de l'exécution du processus. Pour cet aspect, nous considérons le critère de *déviation*.

La déviation décrit un type d'adaptation pour faire face aux imprévus durant l'exécution d'un processus ou pour adapter l'exécution à un contexte précis (da Silva et al., 2011). Un processus peut faire face à des imprévus ou satisfaire des exigences évolutives (Kabbaj et al., 2007). Dans la littérature, les déviations sont définies comme les actions exécutées et qui ne sont pas décrites dans le modèle de processus initial (Smatti and Ahmed-Nacer, 2014). Avec ce mécanisme, il est possible de s'adapter au contexte changeant durant l'exécution d'un processus sans forcément changer le modèle. Les déviations ne sont pas des événements exceptionnels (Lanubile and Visaggio, 2000) et ont lieu souvent durant l'exécution d'un processus. Certaines approches (Yang et al., 2007) la définissent comme une inconhérence qui apparaît durant l'exécution.

Deux solutions peuvent être retenues pour faire face aux déviations :

— *Ignorer la tâche déviante*. Il est possible d'annuler son exécution. L'annulation de la tâche implique de passer à la tâche suivante dans le modèle de processus sans possibilité de la réexécuter à nouveau dans le PSEE ("Process-centered Software Engineering Environment")
— *"Model-relaxing"*. Le modèle de processus n'est pas modifié. Le PSEE permet l'exécution des actions pour satisfaire la nouvelle situation et donne la possibilité de dévier du processus initial.

2.4.2.4 Flexibilité par changement

Tout comme l'aspect déviation, la flexibilité par changement porte sur les changements durant l'exécution du processus. Pour cet aspect, nous considérons le critère d'*évolution*.

Ce critère prend en compte les évolutions potentielles d'un processus au cours de son exécution. Ces évolutions peuvent porter sur l'ensemble des ressources ou sur les tâches à exécuter. L'évolution consiste à produire une nouvelle version du processus à exécuter. On identifie deux sortes d'évolution :

— un *changement ad-hoc* consistant à changer une instance ponctuelle du processus
— un *changement global* consistant à faire migrer le modèle du processus dans sa globalité vers une nouvelle version.

Le tableau 2.3 résume les différents aspects et les critères associés pour l'exigence de flexibilité.

Exigence	**Aspects**	**Critères**	**Solutions**
Flexibilité	Flexibilité par conception	Variabilité	PML Constructs, Variantes
	Flexibilité par spécification incomplète	"Looseness"	Late-binding, Late-modeling
	Flexibilité par déviation	Déviation	Model-relaxing, Ignorer la tâche déviante
	Flexibilité par changement	Evolution (adaptation)	Ad-hoc, Evolutif

TABLE 2.3: Tableau récapitulatif des aspects, critères et solutions concernant la flexibilité.

2.5 Approches existantes pour la gestion des processus collaboratifs

Les sujets autour des processus collaboratifs sont traités principalement par deux communautés de chercheurs : *CSCW (Computer-Supported Cooperative Work)* et *PM (Process Management)*.

Les travaux du CSCW concernent la définition des modes de collaboration tant dans les aspects cognitifs que sociaux. L'accent est mis sur la définition de systèmes d'information et d'outils supportant le travail en groupe (Schmidt and Bannon, 1992). Les domaines de recherche dans

le CSCW concernent principalement le développement d'outils de support pour le partage et la communication. Ils sont encore connus sous le nom de *"Groupware"* (Ellis and Wainer, 1994).

Les travaux du PM reposent moins sur les aspects sociaux que ceux du CSCW, même s'ils constituent une importante direction de recherche (Ariouat et al., 2016). Dans cette communauté, l'accent est mis sur la définition des modes de collaboration et sur la coordination des tâches des processus. Le *"Process Management"* peut être vu comme une discipline dont l'objectif est l'identification et la formalisation des processus dans le but de communiquer et de contrôler leur exécution (Dumas et al., 2013; van der Aalst et al., 2003b).

Dans la suite, notre état de l'art est centré sur les travaux de la communauté du *"Process Management"*. L'analyse des approches existantes est fondée sur les critères caractérisant les deux exigences définies ci-dessus : collaboration et flexibilité.

Les travaux présentés ci-dessous sont structurés selon trois volets : la définition de la collaboration, la modélisation de la collaboration, et la flexibilité de l'exécution des processus. La flexibilité par conception, telle que présentée plus haut, sort du périmètre de notre étude. Pour chaque approche, nous faisons sa description puis proposons une discussion présentant les avantages et les limites de l'approche vis-à-vis des critères définis ci-dessus.

2.5.1 Travaux sur la définition de la collaboration

Les travaux de cette catégorie concernent les approches qui définissent la collaboration dans le cadre de l'ingénierie des processus. Le but est d'avoir une base permettant de représenter la collaboration ou le travail collaboratif.

Suivant les besoins de collaboration plusieurs modèles ont été développés. Comme vu dans la section 2.4.1 ci-dessus, plusieurs auteurs (Ellis et al., 1991; Fuks et al., 2005) présentent la *communication*, la *coordination* et la *coopération* comme les aspects qu'un système collaboratif doit satisfaire en vue d'assister les utilisateurs. La satisfaction de ces aspects donne lieu à des modèles récurrents dans la façon de collaborer. Dans le cadre des objectifs de notre travail, nous privilégions les modèles introduits sous forme de patrons, qui en mettant le focus sur la coordination des tâches (workflow), correspondent bien à notre problématique de recherche. Ainsi, dans cette section, nous présentons essentiellement les patrons de collaboration de Lonchamp

(Lonchamp, 1998) et les patrons de workflow de Van der Aalst (van der Aalst et al., 2003b), qui sont des références en matière de travaux de recherche dans le domaine.

2.5.1.1 Patrons de Lonchamp

Description de l'approche

Dans ses travaux, Lonchamp (Lonchamp, 1998) a défini des patrons de collaboration génériques pour la modélisation de situations récurrentes. La définition donnée au concept de *patron* se base sur les travaux de (Alexander et al., 1977). Lonchamp a défini un ensemble de treize patrons couvrant les situations de collaboration les plus représentatives. Il propose une notation spécifique pour illustrer visuellement les tâches, les flux de contrôle et les données partagées entre tâches. A titre d'exemple, nous illustrons les deux patrons suivants : le patron *"co-work"*, le patron *"multiprocessing"*.

— Patron *"co-work"* (figure 2.8). Ce modèle est utilisé pour la production collaborative d'un document. C'est le cas où plusieurs acteurs travaillent en coopération sur une tâche élémentaire (en mode synchrone ou asynchrone). La production du document final résulte d'une série de décisions individuelles fusionnées ou de décisions collectives. La notation utilisée ici indique qu'il s'agit d'une tâche atomique (symbole A sur la figure) et collective (icone en bas à droite).

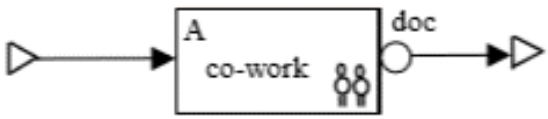


FIGURE 2.8: Patron de collaboration "co-work" (Lonchamp, 1998).

— Patron *"multiprocessing"* (figure 2.9). Ce modèle est utilisé lorsque plusieurs acteurs produisent simultanément leur propre version du document (ou de l'artefact) sans avoir accès aux contributions des autres. Il s'ensuit la nécessité d'intégrer les différentes versions produites. Ce modèle introduit une tâche, *rework*, qui est optionnelle et peut être raffinée en cours d'exécution (indiqué par le point d'interrogation).

Discussion

Lonchamp définit des patrons pour décrire des situations de collaboration. Il illustre la multi-

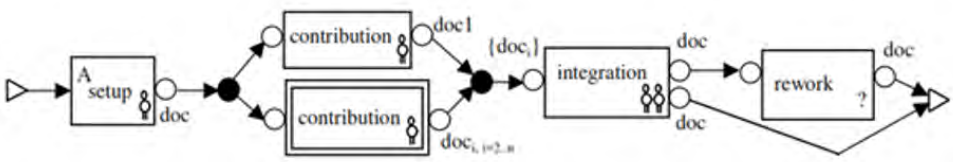


FIGURE 2.9: Patron de collaboration "multiprocessing" (Lonchamp, 1998).

instanciation avec le concept de *Multiple Task"*. Il gère le flux de contrôle des tâches à un niveau inter-tâche mais ne définit pas l'ordonnancement des instances d'une même tâche (intra-tâche). Lonchamp définit deux types de flux de données pour gérer le partage des produits : un *flux synchrone* et un *flux asynchrone*. Les concepts d'acteurs et d'outils sont pris en compte pour spécifier qui est en charge de l'exécution d'une tâche.

Pour la communication, Lonchamp se base sur les outils CSCW mais n'inclut pas directement dans son approche de moyens de mise en oeuvre de cette exigence.

Concernant la coopération, Lonchamp prévoit la distribution des données en entrée et en sortie mais pas la structuration composite d'un produit. Pour le mode de passage, Lonchamp a mis en place un mécanisme de copie ou de transfert des données. Les données peuvent être en lecture seule (flèche en pointillée) ou en lecture/écriture (flèche bidirecitonnelle en pointillée).

2.5.1.2 Patrons de Van der Aalst

Description de l'approche

Van der Aalst (van der Aalst et al., 2003a) propose des patrons de workflow décrivant des fonctionnalités récurrentes des systèmes de workflow. Ces patrons sont classés en différentes catégories : *"control"*, *"resource"*, *"data"*, *"exception handling"* , *"presentation"* et *"log imperfection"*. Etant donné le grand nombre de patrons décrits par Van der Aalst, nous présentons à titre illustratif quelques patrons représentatifs de la catégorie *"control"*.

— Patron *"sequence"* (figure 2.10). Une tâche est exécutée après l'achèvement de la tâche précédente. Ce modèle est utilisé lorsque l'instance d'une tâche ne peut être exécutée que si celle qui la précède a fini de s'exécuter. Cela peut être imposé, par exemple, par le manque de disponibilité de ressources (acteurs ou produits), ou une volonté de procéder par raffinement. Dans ce patron, $I1$ et $o1$ représentent respectivement l'entrée

et la sortie. Les tâches en séquentiel sont représentées par un rectangle (A et B). $p1$ représente le produit intermédiaire échangé.

FIGURE 2.10: Patron de workflow "sequence" (van der Aalst et al., 2003a).

— Patron *"parallel"* (figure 2.11). Plusieurs tâches sont effectuées en parallèle. Lorsque les ressources requises pour effectuer deux tâches sont disponibles, ce modèle peut être utilisé pour leur exécution. Cependant, cela nécessite qu'il n'y ait pas de liens entre eux ni de ressources à partager entre les deux.

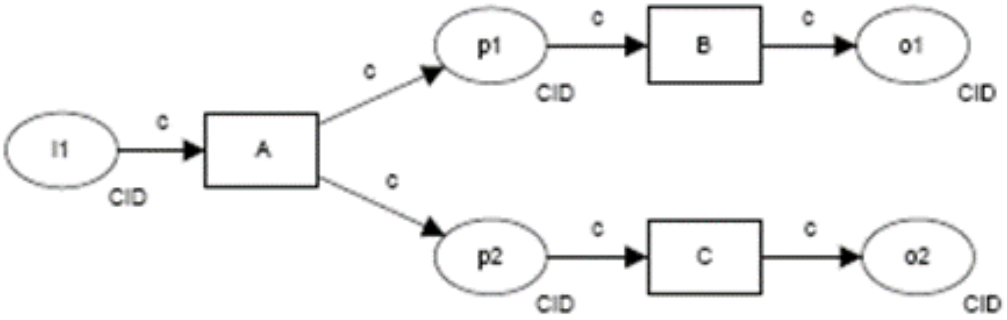


FIGURE 2.11: Patron de workflow "parallel"(van der Aalst et al., 2003a).

— Patron *"synchronization"* (figure 2.12). Il s'agit d'exprimer la convergence de plusieurs tâches exécutées simultanément vers une tâche après la fin de leur exécution. Le point particulier de ce modèle est que l'exécution d'une tâche suivante (la tâche C sur la figure) nécessite la fin de toutes les tâches à synchroniser (A et B).

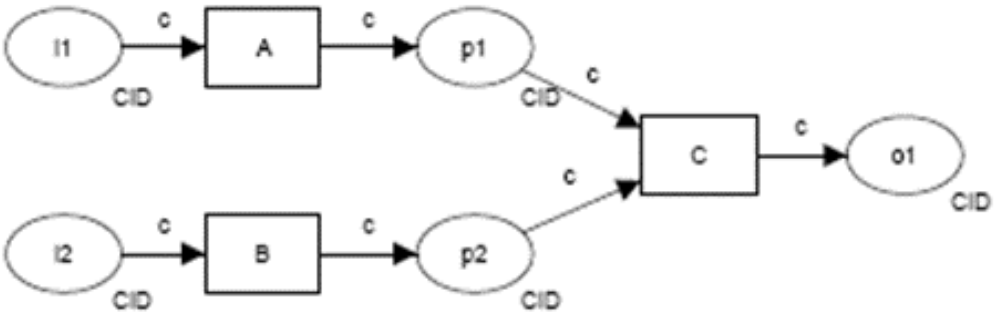


FIGURE 2.12: Patron de workflow "synchronization"(van der Aalst et al., 2003a).

— Patron *"simple merge"* (figure 2.13). Ce modèle est organisé de la même manière que le précédent à la différence que l'exécution d'une nouvelle tâche (la tâche C sur la figure) ne nécessite que la fin de l'exécution de l'une des tâches exécutées en parallèle (A ou B).

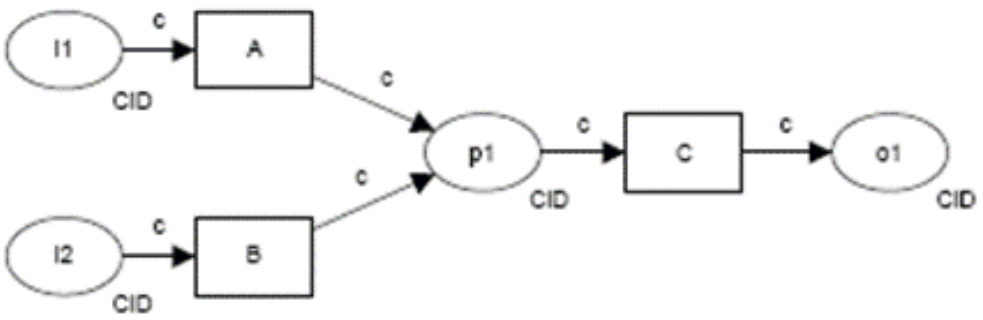


FIGURE 2.13: Patron de workflow "simple merge"(van der Aalst et al., 2003a).

Discussion

Les patrons de van der Aalst, définissent un ordonnancement entre différentes tâches. Cet ordonnancement se fait au niveau inter-tâche.

Tout comme ceux de Lonchamp, les patrons de van der Aalst ne couvrent pas l'aspect communication.

Avec les patrons sur les transferts de données, van der Aalst satisfait les exigences du mode de transfert avec un transfert par référence (*"pattern data transfer by reference"*) et un transfert par copie (*"pattern data transfer by value"*). Il définit un ensemble de données spécifiques à chaque instance d'une tâche multi-instance. Cependant, ces patrons n'indiquent pas si les éléments de l'ensemble sont de même nature, ce qui ne permet pas de définir le concept de produit composite.

2.5.1.3 Bilan des approches étudiées sur la définition de la collaboration

Cette section décrit notre analyse des travaux sur la définition de la collaboration. Le tableau 2.4 présente une comparaison des approches à base de patrons analysées par rapport aux critères de la collaboration.

Exigences	Approches / Critères	Lonchamp	van der Aalst
Coordination	**Flux de travail**	Inter-task	Inter-task
	Flux de données	Inter-task	Inter-task
	Ressources partagées	Actor, Tool	Non prévue
Communication	**Conscience sur l'état du système**	Non prévue	Non prévue
	Discussions	Non prévue	Non prévue
Coopération	**Mode d'accès**	In / Out	In / Out
	Mode de passage	Référence, Copie	Référence, Copie
	Structuration	Non prévue	Non prévue

TABLE 2.4: Tableau comparatif des approches traitant de la définition de la collaboration.

De ce tableau, nous pouvons déduire que les patrons ici représentés sont orientés coordination des tâches. Aucun des deux approches étudiées ici ne prévoit de mécanismes de communication concernant la collaboration. Les discussions ne sont pas formalisées. La structuration des produits n'est pas couverte ce qui rend difficile la manipulation d'un produit composite.

2.5.2 Travaux sur la modélisation de la collaboration dans les processus

Dans cette section, nous présentons les travaux qui couvrent la modélisation de la collaboration dans les processus en proposant explicitement un langage de modélisation formalisé. Nous retrouvons dans la littérature étudiée les *standards de modélisation* tels que *SPEM* (Object Management Group (OMG), 2008) *BPMN* (Object Management Group (OMG), 2011), les *langages mettant l'accent sur la collaboration* tels que *Collaboro* (Izquierdo and Cabot, 2016), *MMCollab* (Bennani et al., 2019), *CMSPEM* (Kedji et al., 2011), et le *méta-modèle de Hawryszkiewycz* (Hawryszkiewycz, 2005). Nous évaluons ces travaux sur les trois aspects de la collaboration présentés ci-avant : coordination, communication et coopération.

Comme le langage supportant notre approche (cf. chapitre 3) est inspiré de SPEM, nous mettons l'accent dans cette partie de l'état de l'art sur ce méta-modèle.

2.5.2.1 SPEM 2.0

Description de l'approche

SPEM (Object Management Group (OMG), 2008) est une spécification de l'OMG qui décrit les processus de production de logiciels ou systèmes. Le méta-modèle de la norme SPEM 2.0 est conforme au méta-modèle MOF. SPEM est aussi défini sous forme de profil UML, et comme tel, peut bénéficier de l'outillage d'UML.

Les exigences de qualité qui sont satisfaites dans la norme SPEM 2.0 sont :

- *Réutilisation de savoir-faire* : SPEM 2.0 permet de séparer le contenu réutilisable d'une méthode de développement de son application dans un processus spécifique. Une méthode de développement décrit les activités d'un développement, leurs produits en entrée et en sortie, leurs rôles sans spécifier leur ordre d'exécution alors qu'un processus réutilise ces activités et les lie selon un ordre donné.
- *Généricité* : SPEM 2.0 propose un mécanisme d'extension qui permet de décrire et de documenter divers types de processus (*"Kind"* pour typer les concepts de SPEM 2.0). Par exemple, une activité peut être typée comme une phase ou comme une itération.
- *Flexibilité* : Le mécanisme de plug-in de SPEM 2.0 introduit des concepts pour la gestion et la maintenance des librairies d'une méthode de développement ou d'un processus. Avec le mécanisme de *"plug-in"*, un concepteur de processus pourra enrichir ou ré-utiliser tout ou partie de processus définis.
- *Modularité* : SPEM 2.0 propose plusieurs mécanismes de modularité pour la réutilisation des processus (*"ProcessPatterrn"*, *"ProcessComponent"*, *"ActivityUseKind"*, etc.). Les composants de processus (*"ProcessComponent"*) sont des paquetages particuliers qui appliquent le principe d'encapsulation ; le concept de patron de processus (*"ProcessPattern"*) permet de modéliser un savoir-faire et de l'appliquer dans un contexte ; le concept *"ActivityUseKind"* permet de définir les différentes manières de réutiliser une activité dans un processus.
- *Exécutabilité* : SPEM 2.0 ne propose pas de formalisme particulier pour la description comportementale d'un processus. Cependant, il propose un mécanisme permettant de lier un modèle de processus à un ou plusieurs modèles comportementaux basés par exemple sur un diagramme d'activité UML 2.0, une machine à états d'UML, ou la notation BPMN (Business Process Modeling Notation) (Bendraou et al., 2007a; Bendraou et al.,

2007b).

SPEM est structuré selon sept paquetages. Chaque paquetage étend les fonctionnalités du paquetage dont il dépend. Les classes sont introduites dans une unité de bas niveau pour permettre une extension dans les unités supérieures en ajoutant des propriétés supplémentaires pour réaliser des exigences de modélisation plus complexes. Les sept paquetages sont les suivants : *"Core"*, *"ProcessStructure"*, *"ProcessBehavior"*, *"ManagedContent"*, *"MethodContent"*, *"ProcessWithMethods"* et *"MethodPlugin"*. La Figure 2.14 montre l'architecture fonctionnelle du méta-modèle SPEM 2.0.

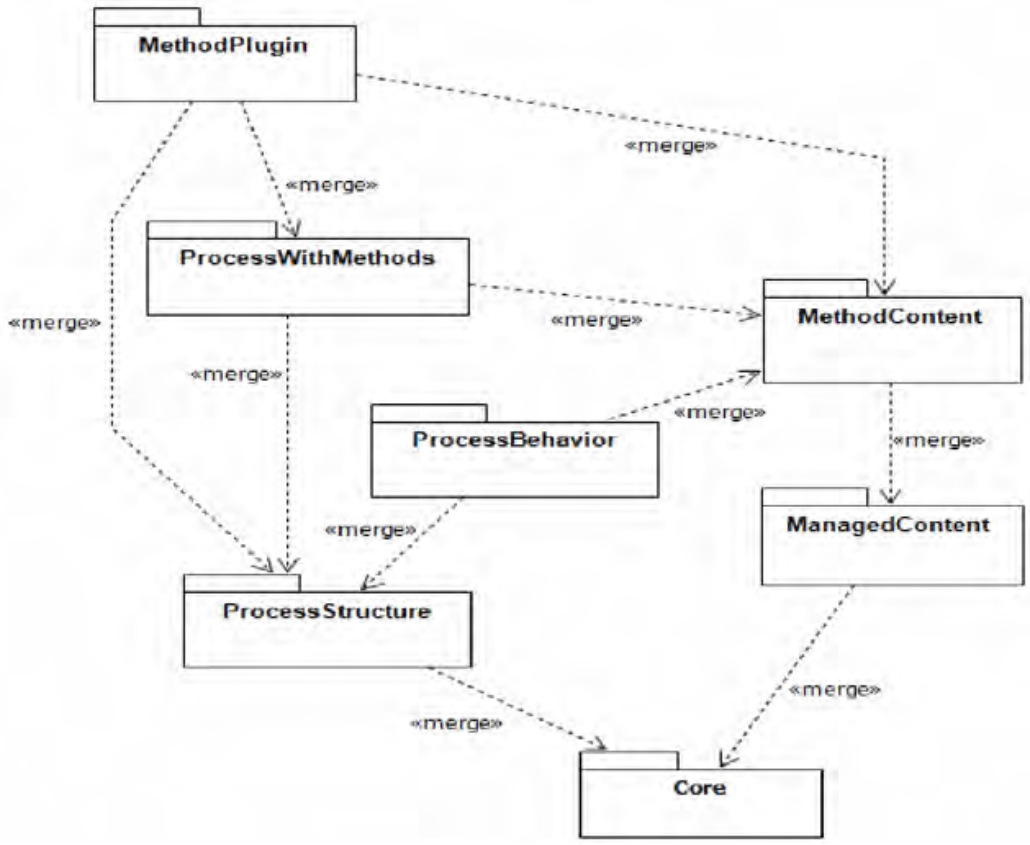


FIGURE 2.14: Structuration en paquetages du méta-modèle SPEM 2.0 (Object Management Group (OMG), 2008).

Discussion

SPEM permet de gérer la coordination des activités du processus étant entendu qu'une activité est décomposable en tâches. Cette coordination s'effectue entre les tâches du processus et ne supporte pas explicitement le concept de tâche collaborative. Il s'agit d'une collaboration faible ou encore d'une description *"inter-task"*du flot de contrôle (*Work Sequence*). SPEM gère la multi-instanciation avec les attributs *"hasMultipleOccurences"* et *"isRepeatable"*. Toutefois, il n'y a pas de lien entre les différentes instances. De ce fait, cette multi-instanciation permet une

répétition des tâches d'une activité mais il n'est pas possible de contrôler finement l'enchainement de ces tâche. Considérant que SPEM supporte uniquement la collaboration faible, le *"data-flow"* et les *"shared resource"* sont aussi gérés à un niveau *"inter-task"*, ce qui explique l'absence du concept d'acteur (*actor*).

Concernant l'aspect coopération, SPEM 2.0, à travers le concept de *"WorkDefinitionParameter"*, prévoit les différents types de distribution d'un produit, à savoir en entrée, en sortie ou en combinaison des deux (entrée-sortie). Cependant, dans SPEM, la structuration permettant de spécifier qu'un produit peut être composite ou non, n'est pas prise en compte ni le mode de passage.

Nous pouvons dire en conclusion que SPEM satisfait le critère de coordination et en partie celui de coopération mais pas celui de communication.

2.5.2.2 BPMN

Description de l'approche

Tout comme SPEM, BPMN (Object Management Group (OMG), 2011) est une spécification de l'OMG. C'est un standard pour les processus d'entreprise (*"business process"*). Il fournit une notation graphique pour la spécification des processus. La notation de BPMN permet d'avoir une nomenclature standardisée pour une utilisation non seulement technique mais aussi non technique. Cela permet d'avoir une même notation permettant de regrouper différents points de vue. Dans ce sens, BPMN facilite la communication au sein des processus pour l'ensemble des participants (développeurs, clients, analystes).

Un modèle BPMN propose différents types de sous-modèles :

- des *processus privés (private processes)* : ils désignent les processus internes à une entreprise. On les appelle encore *workflow*.
- des *processus publics (public processes)* : ils désignent les interactions entre un processus quelconque et un processus privé. Ils incluent les activités utilisées pour la communication entre les deux processus.
- des *chorégraphies (choreography)* : elles désignent le comportement attendu des participants à un processus.

— des *collaborations* : elles décrivent l'interaction entre entités (exemple : patient et docteur). Les interactions sont symbolisées par des messages échangés par les participants (les entités).

Discussion

La méta-classe *"SequenceFlow"* permet de définir le séquencement entre les différentes activités du processus. Tout comme SPEM, cette coordination est précisée à la modélisation (description *"inter-task"*). Cependant, avec le concept *"MultiInstanceLoopCharacteristics"*, BPMN permet de spécifier la multi-instanciation d'une activité avec un nombre défini d'instances. Ces instances peuvent être exécutées soit en séquentiel soit en parallèle (description *"intra-task"*). Toutefois, le nombre d'instances doit être connu à l'avance ce qui rend cette multi-instanciation assez rigide. La communication au niveau de BPMN est assuré par le concept *"Message"*.

Les produits manipulés dans BPMN sont représentés par le concept *"DataObject"*. *DataInput* et *DataOutput* représentent les spécialisations d'un *DataObject* et permettent de couvrir les modes de distribution d'un produit respectivement en entrée et en sortie. L'attribut *"isCollection"* permet de spécifier un produit composite sous forme de collection sans spécifier la nature de la collection. BPMN prévoit un attribut *isReference* permettant de déterminer si un *item* est une référence à un élément externe ou contenu dans un autre élément. Cela permet donc de définir différents modes de passage pour un *DataObject*.

2.5.2.3 CMSPEM

Description de l'approche

Une extension de SPEM dénommée CMSPEM (*Collaborative Model-Based Software & Systems Process Engineering Metamodel*) a été définie dans (Kedji et al., 2014). La Figure 2.15 donne les différents paquetages composant CMSPEM. Cette extension rajoute au méta-modèle SPEM des concepts nécessaires au support de la collaboration dans les processus logiciels. Dans son méta-modèle, Kedji représente sous forme de concepts les intervenants d'un projet de développement logiciel sous forme d'acteurs physiques, les artéfacts et les lots de travail manipulés par ces acteurs, ainsi que les relations entre ces éléments.

CMSPEM permet de réduire la distance entre la modélisation des processus et les aspects relevant de leur mise en œuvre. Cela se justifie par le fait que Kedji intègre des objectifs

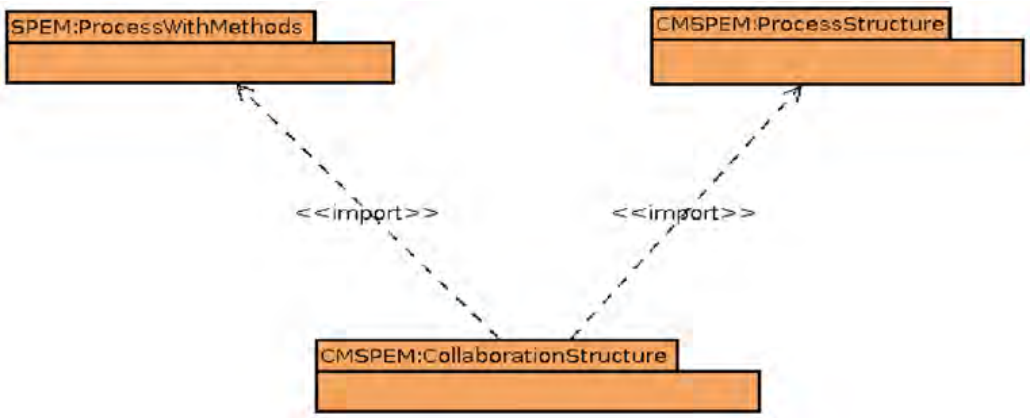


FIGURE 2.15: Structure des paquetages de CMSPEM (Kedji et al., 2014).

relatifs à l'implémentation des processus logiciels en se basant sur un modèle conceptuel du support au développement collaboratif. Les concepts de base sont représentés par les méta-classes suivantes :

— *Actor* qui désigne un participant humain ayant un rôle dans le projet,

— *ActorSpecificWork* qui représente un lot de travail affecté à un participant au projet,

— *ActorSpecificArtifact* qui représente une copie d'un artéfact destinée à un acteur.

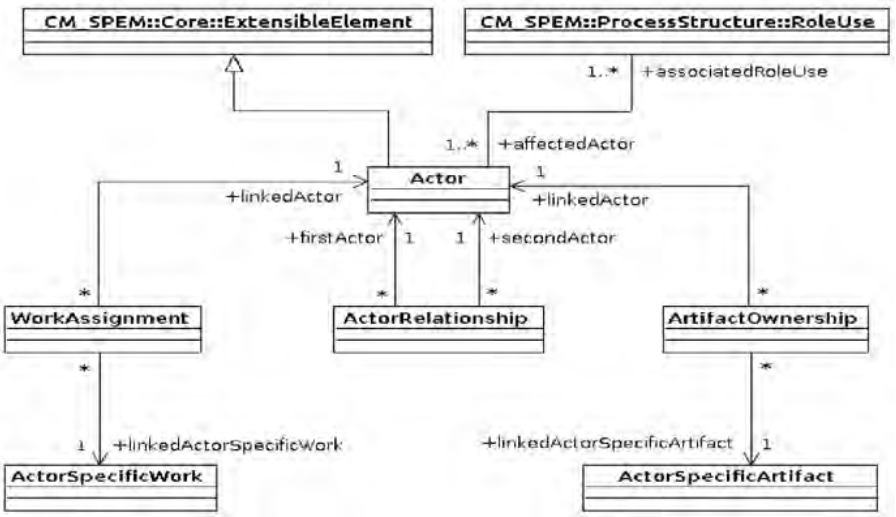


FIGURE 2.16: Concept Actor et ses relations dans CMSPEM.

A l'aide d'un mécanisme d'événements, Kedji spécifie le comportement d'un modèle de processus CMSPEM avec les méta-classes suivantes : *"Event"*, *"EventSource"*, *"EventHandler"* et *"EventSubscription"*. Un catalogue d'événements est proposé, parmi lesquels :

— L'événement *"ActorRoleAssignment"* signale qu'un rôle est assigné à un acteur,

— L'événement *"ActorSpecificArtifactChange"* signale les changements qui s'opèrent sur

un artéfact,

— Les événements *"ActorSpecificWorkStart"* et *"ActorSpecificWorkEnd"* signalent respectivement le début et la fin de l'exécution d'une tâche.

L'attribut *"isPartialCopy"* d'un *ActorSpecificArtificat* permet de définir qu'un produit est une copie partielle d'un produit plus général. Il assure une décomposition et une structuration du produit en plusieurs composants affectés à différents participants dès la modélisation. Le concept *"ArtifactUse"* de CMSPEM étend *"ProcessParameter"* de SPEM, il bénéficie des attributs permettant de gérer les directions (*"in"*, *"out"* et *"inout"*) du produit manipulé.

Discussion

CMSPEM étant une extension de SPEM, il bénéficie des mécanismes permettant de mettre en œuvre la coordination au sein de la collaboration. En effet, CMSPEM utilise la méta-classe *"ActorSpecificWorkRelationship"* qui est une spécialisation du concept de WorkSequence pour spécifier le séquencement entre deux tâches. Cependant, tout comme SPEM, ces critères (*WorkSequence*, *DataFlow*, ...) doivent être spécifiés lors de la modélisation du processus.

Les événements proposés par CMSPEM sont implémentés dans un outil. De ce fait, il assure l'aspect communication de la collaboration. Ces événements permettent de communiquer sur les choix d'exécution (assignation d'un acteur, changement d'un artefact, ...) et de rester informé sur l'état du système, à savoir quelle tâche est en cours ou a fini son exécution.

La coopération dans CMSPEM est satisfaite par la structuration d'un produit au travers de l'attribut *"isPartialCopy"*, ce qui permet de définir un produit composite. CMSPEM gère aussi les différentes directions d'un produit ce qui satisfait le critère de distribution. CMSPEM ne couvre pas les solutions pour le mode de passage.

2.5.2.4 Méta-modèle de Hawryszkiewycz

Description de l'approche

Le méta-modèle de Hawryszkiewycz (Hawryszkiewycz 2005) combine les concepts sémantiques provenant de diverses approches de modélisation de la collaboration. Les concepts de *groupe de travail*, *rôle*, *participant*, *activité*, *action* et *artefact* (voir figure 2.17) sont utilisés pour faire une description de la collaboration. Ces concepts sont liés entre eux par des relations. Dans son approche, Hawryszkiewycz déclare qu'une collaboration réussie nécessite trois dimensions

importantes à savoir :

— La *culture sociale (Social Culture)*,

— La *technologie* qui est le facilitateur (*Technology*),

— La *gestion des connaissances* (*Knowledge Management*).

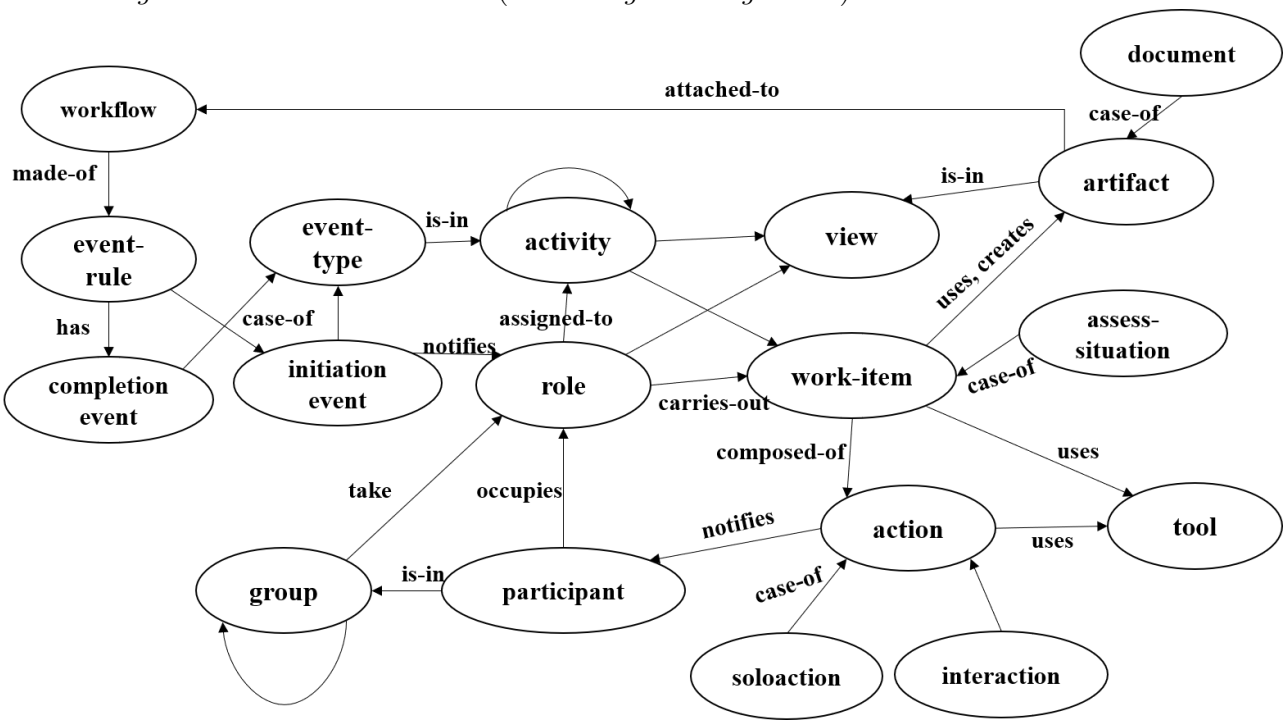


FIGURE 2.17: Méta-modèle décrivant la collaboration (Hawryszkiewycz, 2005).

Discussion

L'approche de Hawryszkiewycz vise à décrire les éléments nécessaires pour la définition d'un processus, mais ne fait pas explicitement de focus sur les critères de collaboration. Plus précisément, les mécanismes de séquencement entre les activités (*Work Sequence*) ne sont pas prévus. A travers le concept *"Interaction"*, Hawryszkiewycz décrit une communication semi-formelle entre les individus qui collaborent sur les activités. Cette communication s'effectue sous forme de discussions. Le méta-modèle prévoit des événements signalant le début ou la fin d'un *"work item"* qui représente un ensemble d'actions et d'interactions nécessaires pour produire des résultats.

2.5.2.5 Collaboro

Description de l'approche

Collaboro (Izquierdo and Cabot, 2016) est un méta-modèle qui décrit l'implication de plusieurs personnes dans la définition d'un langage de modélisation spécifique à un domaine (DSML -

Domain Specific Modeling Language). La participation des individus se fait par des propositions de changement dans la conception du DSML. Ces propositions sont par la suite discutées par la communauté en vue de leur amélioration. Un système de vote permet de sélectionner une solution parmi plusieurs proposées. La figure 2.18 montre les éléments décrivant le méta-modèle Collaboro.

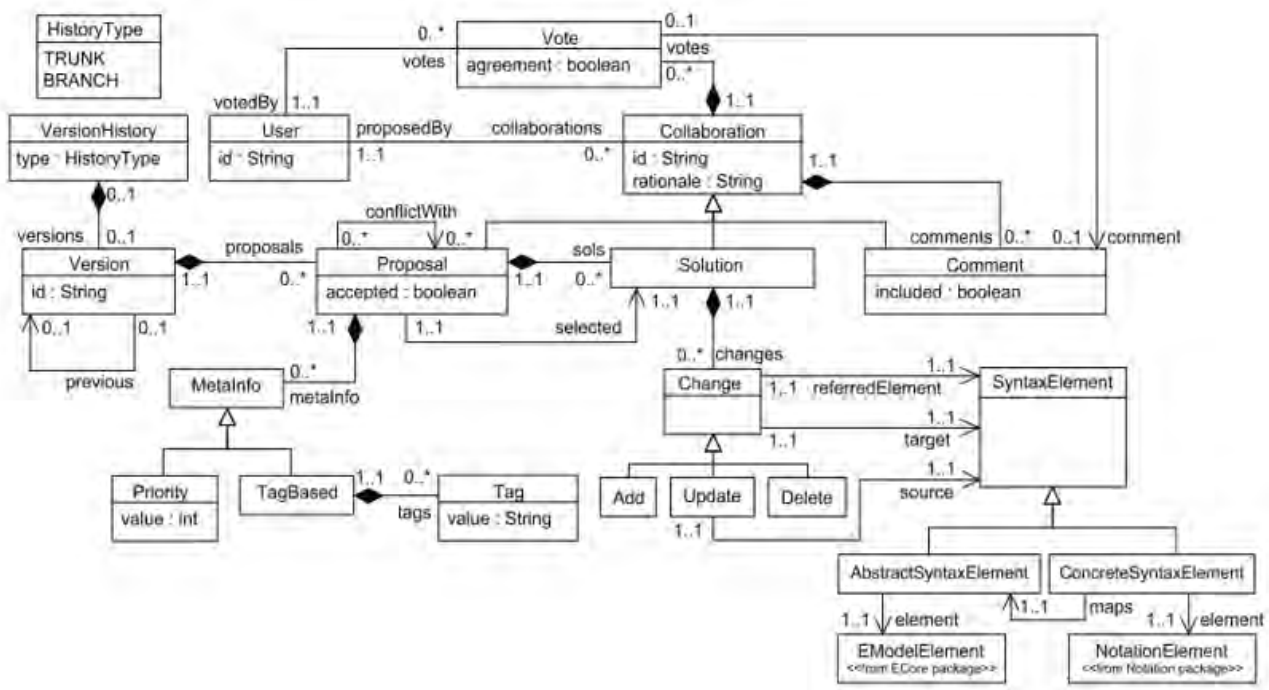


FIGURE 2.18: Méta-modèle Collaboro (Izquierdo and Cabot, 2016).

Collaboro permet la représentation statique (*proposition de changement*) et dynamique (*vote*) de la collaboration.

— *Aspects statiques.* La méta-classe *"VersionHistory"* représente les versions d'une collaboration. Cela permet d'avoir plusieurs versions manipulables chacune séparément. Au besoin, ces versions sont fusionnées pour constituer une branche unique.

Dans le méta-modèle, il existe trois types de collaboration : des propositions de changements (*"Proposal"*), des solutions (*"Solution"*) et des commentaires (*"Comment"*). La collaboration est représentée par la méta-classe *"Collaboration"*.

— *Aspects dynamiques.* L'aspect dynamique de la collaboration est mise en oeuvre à travers des votes, représentés par la méta-classe *"Vote"*. Un vote indique l'accord ou le désaccord d'un utilisateur au sein d'une collaboration. Il peut être justifié par un commentaire.

Discussion

Les propositions et votes dans Collaboro ne suivent pas un séquencement permettant d'avoir une coordination entre eux. La façon de collaborer repose sur les commentaires et solutions associées aux propositions et non sur la réalisation ou production de livrables pour un objectif commun. De ce fait, Collaboro n'inclut pas de mécanisme permettant de satisfaire les critères de coordination. Toutefois, le système de votes et de commentaires proposé permet de définir une communication semi-informelle entre les différents intervenants du processus.

2.5.2.6 MMCollab

Description de l'approche

MMCollab (Bennani et al., 2019) est un méta-modèle support à la formalisation de la collaboration en général et de la prise de décision en groupe en particulier, qui se situe dans la lignée du méta-modèle Collaboro (Izquierdo and Cabot, 2016). Il permet de décrire les concepts nécessaires lors d'une prise de décision collaborative ; ces concepts concernent les acteurs, le choix de la politique de prise de décision, la collecte des propositions sur lesquelles vont porter les évaluations individuelles et l'aboutissement aux décisions collectives relatives à ces propositions. *MMCollab* permet d'organiser les rôles respectifs des acteurs dans le processus de prise de décision. Les propositions sont possiblement organisées sous forme d'arborescence. MMCollab permet aussi d'agréger les évaluations individuelles associées aux différentes propositions afin d'aboutir aux décisions collectives.

MMCollab est conforme à *MOF (Meta Object Facility)* et est structuré en cinq paquetages : le premier paquetage *Actors* concerne les acteurs de la prise de décision, le deuxième *Proposals* concerne la collecte des propositions, le troisième l'évaluation des propositions (*Evaluation*), le quatrième la politique d'élaboration des décisions collectives (*CollectiveDecision*), tandis que le cinquième (*CoreConcepts*) contient les concepts fondamentaux de MMCollab et importe les quatre autres paquetages, comme décrit sur la Figure 2.19.

MMCollab importe aussi les paquetages Process Structure de SPEM et CMSPEM.

Discussion

MMCollab permet la coordination des intervenants (actors) prenant part à la prise de décision en groupe. Cette coordination s'appuie sur les concepts du paquetage *MMCollab : :Actors*. Concernant la communication, à travers son outil *HMCS-Collab*, la démarche associée à MMCol-

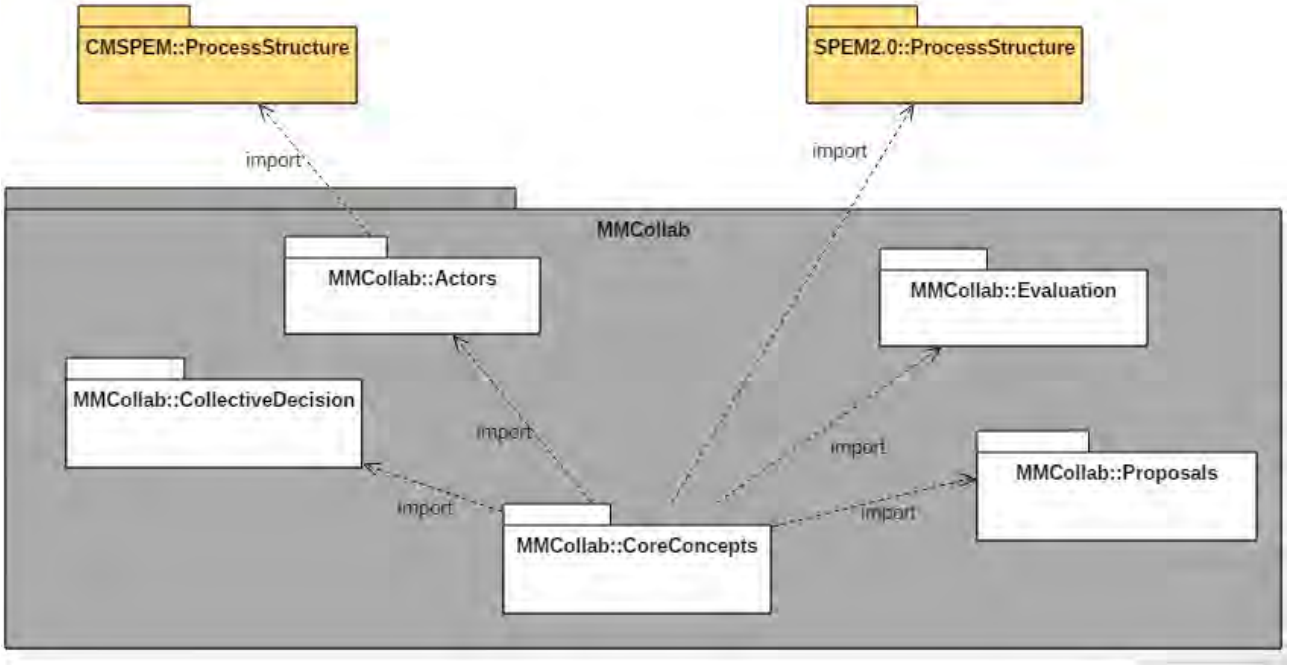


FIGURE 2.19: Organisation en paquetages du méta-modèle MMCollab (Bennani et al., 2019).

lab gère la communication pour la prise de décision sur la base de stratégies prédéfinies et semi-automatisées. Cependant, MMCollab ne satisfait que partiellement le critère de coopération. En effet, il gère des produits sous forme de *proposals* qui peuvent être composites mais ne gère pas la distribution de données.

2.5.2.7 Bilan des approches sur la modélisation de la collaboration

Cette section porte sur l'analyse des travaux traitant de la modélisation des processus collaboratifs en général et de la collaboration en particulier.

Le tableau 2.5 présente une classification des approches par rapport aux trois critères de la collaboration.

Le fait majeur observé est que la majorité des approches étudiées (SPEM, CMSPEM, BPMN) traitent de la coordination entre les tâches du processus (inter-task). Le séquencement des instances que pourrait contenir une tâche multi-instance est impossible étant donné que ces approches ne prennent pas en compte la multi-instanciation d'une tâche collaborative. D'un point de vue communication, CMSPEM propose, à travers un outil, une liste d'événements permettant de répondre aux exigences de ce critère. Il assure de ce fait la connaissance de l'état

du système durant le déroulement du processus. Étant une extension de SPEM, CMSPEM supporte au même titre les différentes directions qu'une donnée peut avoir (*in*, *out* et *inout*). Contrairement aux autres approches, CMSPEM et BPMN supportent la structure composite des produits. Les approches Collaboro et de Hawryszkiewycz ne traitent pas de la coordination dans la collaboration mais proposent une communication informelle ou semi-formelle entre les acteurs. MMCollab traite de la prise de décision tout comme Collaboro. Il propose un outil (HMCS) permettant une bonne communication mais ne couvre pas les autres aspects de la collaboration. CMSPEM autorise la décomposition d'un produit en composants, ce qui permet de pouvoir répartir la collaboration entre plusieurs participants.

Aspects	**Approches / Critères**	**SPEM 2.0**	**BPMN**	**CMSPEM**	**Hawryszkiewycz**	**Collaboro**	**MMCollab**
Coordination	**Flux de travail**	Inter-task	Inter-task, Intra-task	Inter-task	Non prévue	Non prévue	Inter-task
	Flux de données	Inter-task	Inter-task	Inter-task	Inter-task	Non prévue	Non prévue
	Ressources partagées	Tool / Role	Performer	Actor	Actor / Tool	Non prévue	Actor
Communication	**Conscience sur l'état du système**	Non prévue	Non prévue	Formelle	Semi-Formelle	Informelle	Formelle
	Discussions	Non prévue	Non prévue	Formelle	Semi-Formelle	Semi-Formelle	Formelle
Coopération	**Mode d'accès**	In / Out / Inout	In / Out	In / Out / In-out	Non prévue	Non prévue	Non prévue
	Mode de passage	Non prévue	Référence, Copie	Non prévue	Non prévue	Non prévue	Non prévue
	Structuration	Non prévue	Composite	Composite	Non prévue	Non prévue	Composite

TABLE 2.5: Tableau comparatif des approches liées au support de la collaboration.

2.5.3 Travaux sur la flexibilité de l'exécution des processus

Concernant la mise en oeuvre des processus, plusieurs approches ont été définies dans la littérature. Ces approches résultent pour la plupart en la définition de nouveaux méta-modèles ou d'extensions de méta-modèles existants intégrant des concepts d'exécution.

Dans cette section, nous présentons les approches traitant de l'exécution flexible des processus. La flexibilité par conception, mentionnée dans la section 2.4.2, est hors du périmètre de notre étude. Parmi ces travaux nous avons retenu les approches *DECLARE* (Pesic et al., 2007), *ADEPT* (Dadam and Reichert, 2009), *SPEM4MDE* (Diaw et al., 2011), *eSPEM* (Ellner et al., 2010) et *xSPEM* (Bendraou et al., 2007a). Ces travaux sont issus des standards dans le domaine du *"Process Management"*. Les travaux qui sont présentés dans la suite sont évalués par rapport aux critères traitant de la flexibilité de l'exécution des processus à savoir *l'imprécision* (*"looseness"*), *la variabilité*, *l'évolution* et *la déviation* (cf section 2.4.2 ci-dessus).

2.5.3.1 DECLARE

Description de l'approche

DECLARE (Pesic et al., 2007) est une approche déclarative, c'est-à-dire que toute activité dans le processus est possible tant qu'elle n'est pas explicitement interdite. DECLARE supporte la flexibilité par la mise en place de moyens permettant de différer certains choix à l'exécution (*"decide to decide later"*) (van der Aalst et al., 2009). A l'aide de contraintes, elle peut spécifier, pour les différents choix d'exécution, leurs conditions d'exécution. Cependant, l'introduction de contraintes n'est pas forcément exhaustive étant donné l'impossibilité de capturer toutes les exigences d'un système dès sa conception. En outre, il peut être difficile pour les utilisateurs de comprendre le modèle en entier lorsque celui-ci contient beaucoup de contraintes (Pesic et al., 2007).

Le système est composé de trois composants principaux :

— *Designer* : utilisé pour la création des contraintes et la modélisation du modèle de processus.

— *Framework* : utilisé pour le déroulement des instances du modèle de processus et la gestion des changements durant l'exécution.

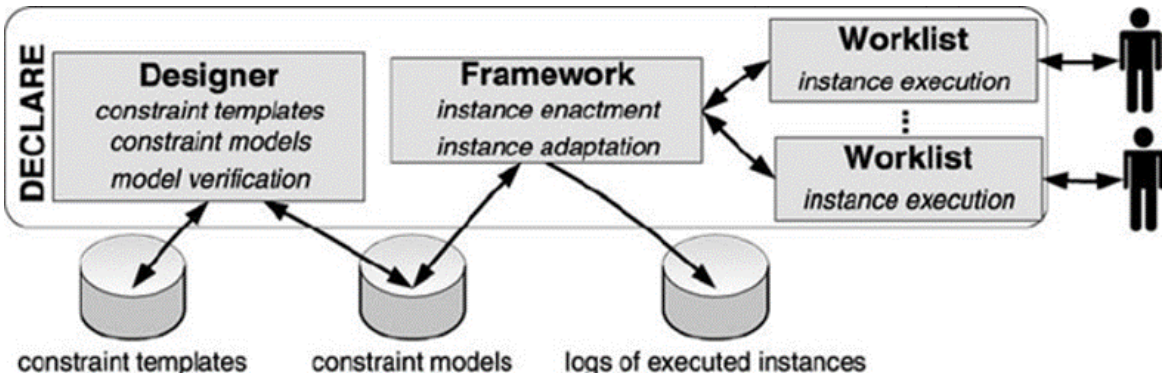


FIGURE 2.20: L'architecture de DECLARE (Pesic et al., 2007).

— *Worklist* : utilisé pour le stockage des instances du modèle de processus en cours d'exécution, propre à un utilisateur donné.

Discussion

En permettant de définir des contraintes, DECLARE permet de satisfaire le critère de modélisation partielle décrit dans la section 2.4.2. En effet, au lieu d'incorporer des chemins alternatifs, il permet de spécifier une exclusion mutuelle entre plusieurs alternatives d'exécution sans spécifier laquelle prévaut dès la modélisation. DECLARE prend en compte deux types d'évolution en permettant l'ajout et la suppression d'activités : *"ad-hoc change"* ou *"evolutionary change"*. Dans le premier type, l'évolution porte sur une instance du processus en cours d'exécution ou un lot d'instances choisies. Dans le second type, l'évolution porte sur l'ensemble des instances du processus. Il existe deux types de contraintes dans DECLARE : des contraintes obligatoires que l'exécution doit satisfaire et des contraintes optionnelles laissées au choix de l'utilisateur. Cela permet d'adapter l'exécution à un contexte changeant. Avec l'utilisation des contraintes, DECLARE permet d'avoir un modèle de référence et différentes variantes dans l'exécution selon la solvabilité d'une contrainte par rapport aux autres. Ces contraintes peuvent être définies pour proposer plusieurs chemins d'exécution. Cela permet de satisfaire le critère de variabilité. Concernant l'adaptation, DECLARE ne donne pas de moyen de dévier d'un processus et donc ne satisfait pas le critère de déviation. En effet, DECLARE considère une déviation comme un comportement à part entière et non comme une exception.

2.5.3.2 ADEPT

Description de l'approche

Le projet ADEPT (Dadam and Reichert, 2009) a été développé dans l'objectif de proposer des

systèmes d'informations sensibles au processus (PAIS - Process Aware Information System), flexibles et adaptés au monde médical. ADEPT permet une définition du flux de contrôle et des données ainsi que l'assignation des acteurs (Reichert et al., 2003). La présence de contraintes temporelles (exemple : *"X doit finir deux jours avant le début de Y"*) permet de contrôler le séquencement des différentes tâches du modèle de processus.

La Figure 2.21 ci-dessous présente l'interface d'édition de processus dans ADEPT.

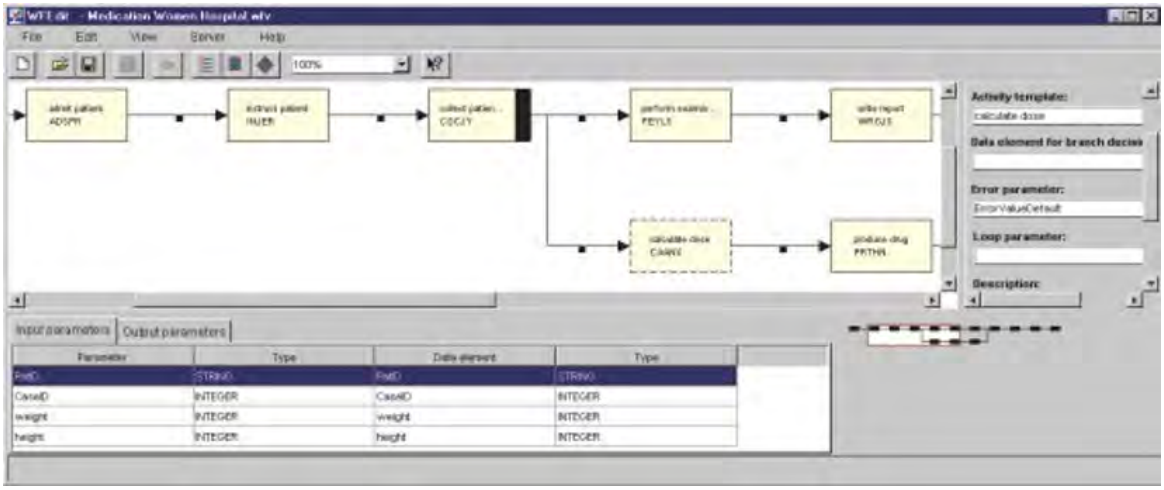


FIGURE 2.21: Éditeur de processus de ADEPT (Dadam and Reichert, 2009).

Discussion

ADEPT ne satisfait pas le critère d'imprécision. En effet, le schéma du processus à exécuter est modélisé en entier en amont de son exécution. Par rapport à l'évolution, ADEPT permet de modifier des instances ad-hoc de processus en cours d'exécution (*"on-the-fly"*). Cette évolution est faite dans l'objectif d'ajouter ou de supprimer des fragments de processus de façon dynamique (Reichert et al., 2003; Ariouat et al., 2016). Pour s'adapter aux connaissances médicales évolutives, ADEPT propose un mécanisme permettant de migrer les instances d'un processus vers une nouvelle version de schéma. Etant donné que les hôpitaux et les cliniques sont des environnements où les situations médicales peuvent être multiples et à traiter simultanément, ADEPT propose non seulement une exécution distribuée des processus suivant la localisation de la clinique mais aussi une possibilité de choix de processus suivant le cas à traiter. Par contre, cette modélisation ne se fait pas sur la base d'un même processus et dès lors on ne peut pas parler de variantes d'un même processus. La sémantique opérationnelle d'un wofkl-flow avec ADEPT permet d'ignorer l'exécution d'une tâche qui ne peut être activée ou d'ajouter dynamiquement une autre tâche. Cela permet de satisfaire l'un des critères de la déviation.

2.5.3.3 SPEM4MDE

Description de l'approche

Dans (Diaw et al., 2011), les auteurs proposent le méta-modèle SPEM4MDE, une extension dédiée au support des processus d'Ingénierie Dirigée par les Modèles (IDM). Cela permet aux concepteurs d'expliciter les aspects spécifiques au développement IDM, notamment les concepts de modèles, méta-modèles et transformations de modèles. SPEM4MDE propose également un langage dédié à la modélisation comportementale des processus IDM pour assister les développeurs en leur fournissant à tout moment l'état de leurs activités et les opérateurs de mise en oeuvre applicables. Pour cela, SPEM4MDE réutilise le paquetage *BehaviorStateMachines d'UML Superstructure* (Object Management Group (OMG), 2007) décrivant les machines à états d'UML2.2. L'exécutabilité des transformations est rendue possible en utilisant le standard QVT (Object Management Group (OMG), 2016). Le méta-modèle de SPEM4MDE est composé de plusieurs paquetages : *"MDE ProcessStructure"*, *"Model Relationship"* et *"MDE Process Behavior"*. Ce dernier, avec les paquetages *"BehaviorStateMachines"* d'UML 2.2 et les paquetages MOF 2.0 de QVT permettent de décrire le volet comportemental de SPEM4MDE. Ce volet comportemental concerne la description du comportement des éléments d'un processus par le biais de machines à états. Ces machines à états décrivent les comportements génériques des éléments d'un processus. Ces comportements peuvent donc être adaptés ou réutilisés pour un processus spécifique.

Discussion

Contrairement à SPEM, SPEM4MDE explicite dans son métamodéle les concepts centraux de l'approche IDM (modèle, méta-modéle, transformation). Un paquetage a été introduit dans SPEM4MDE pour décrire les aspects comportementaux d'un processus IDM. Cependant, du point de vue de la flexibilité de l'exécution, le processus d'exécution est spécifié dès la modélisation ce qui n'est pas compatible avec l'aspect de flexibilité par spécification incomplète. Par l'intermédiaire du standard QVT, SPEM4MDE introduit un mécanisme pour décrire la sémantique opérationnelle des transformations. Avec les concepts de SPEM4MDE, il est possible de modéliser l'ensemble des variantes possibles dans le processus. Ainsi, avec le concept *TransformationImpl*, SPEM4MDE permet de décrire plusieurs implémentations d'une même définition de transformation : règles (ATL, QVT), programme (Java), template.

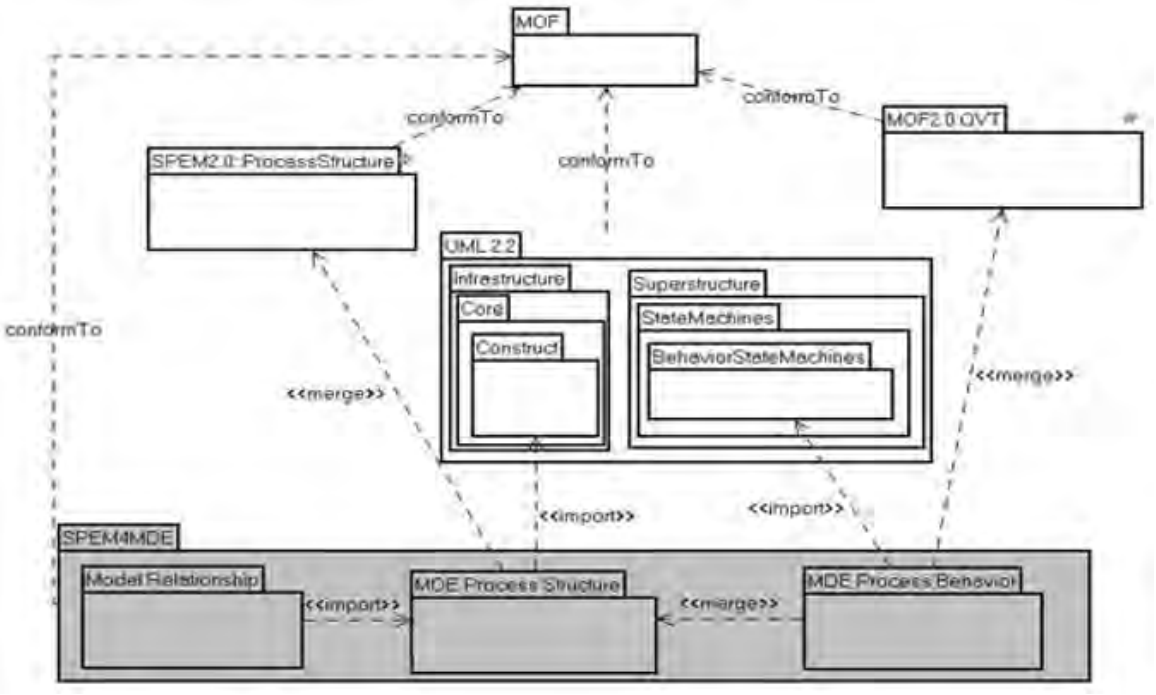


FIGURE 2.22: Organisation en paquetages du méta-modèle SPEM4MDE (Diaw et al., 2011).

2.5.3.4 eSPEM

Description de l'approche

eSPEM (Ellner et al., 2010) est une autre extension de SPEM, basée sur le méta-modèle UML, pour la modélisation fine du cycle de vie d'un processus. Il supporte l'exécution automatique des processus de développement. Le métamodéle d'eSPEM a été proposé pour pallier le fait que SPEM ne propose pas de modèle comportemental. Les auteurs proposent de combiner les avantages des fonctionnalités offertes par SPEM avec les concepts de modélisation du comportement du langage UML. Pour exprimer le comportement des concepts de SPEM, eSPEM propose un paquetage contenant des méta-classes et des associations permettant de lier des *"WorkDefinitions"* de SPEM à un comportement. La Figure 2.23 illustre le concept *Activité* d'eSPEM et le modèle comportemental associé. Pour le contrôle des états et des transitions des éléments durant l'exécution, eSPEM utilise les *machines à état* d'UML. Toutefois, ces machines à état ne sont utilisées que pour le contrôle du cycle de vie des produits (*"WorkProduct Lifecycle"*).

Discussion

Pour permettre de s'adapter aux changements se produisant durant l'exécution, eSPEM propose la méta-classe *"TaskScheduler"*. Un *TaskScheduler* est responsable de la planification des tâches qui peuvent être dynamiquement créées durant l'exécution du processus. Un *TaskScheduler* est

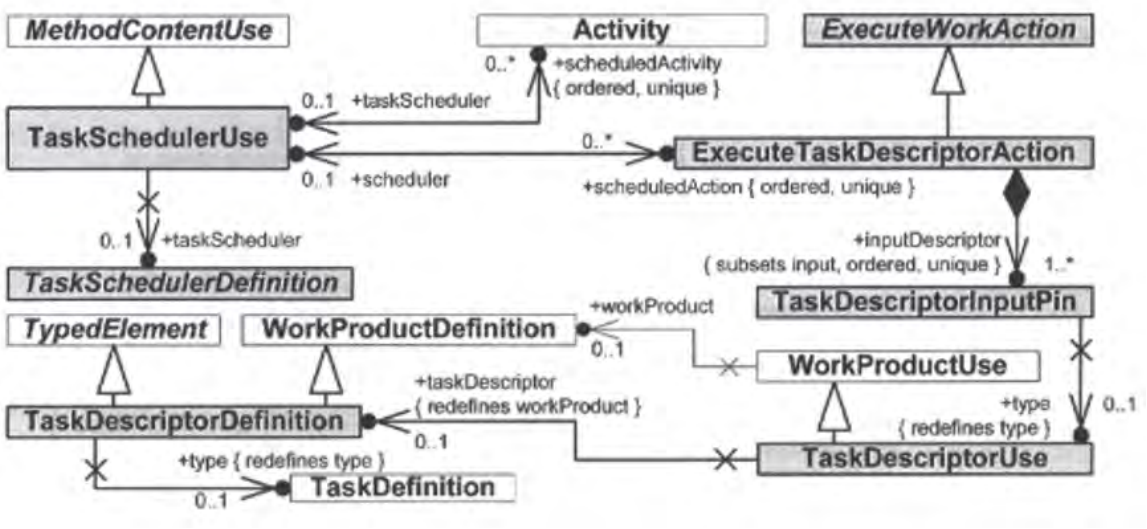


FIGURE 2.23: Extrait du méta-modèle d'eSPEM illutrant le concept Activity (Ellner et al., 2010).

lié à une activité ce qui donne lieu à une évolution ad-hoc. De même, pour permettre des choix durant l'exécution et réduire les efforts durant la modélisation, la méta-classe *"ProcessToMethodMapping"* a été introduite. Elle permet, dans le cas où plusieurs méthodes sont adaptées pour l'exécution d'une tâche, d'en choisir une. Ces différentes méthodes peuvent être considérées comme des constructions de modèles de processus étant donné que le chemin d'exécution choisi peut différer. Ces différentes solutions permettent à eSPEM de couvrir les critères de variabilité et d'évolution.

2.5.3.5 xSPEM

Description de l'approche

xSPEM pour *"eXecutable SPEM"* (Bendraou et al., 2007a) est une approche proposée pour étendre les concepts de SPEM 2.0 afin de prendre en charge l'exécution de processus logiciels. Le principe fondamental est de permettre aux modèles de processus d'être validés par un mapping avec les réseaux de Petri et d'être contrôlés par une transformation en BPEL. Le méta-modèle xSPEM est composé de cinq paquetages, comme décrit par la figure 2.24. Le paquetage *"xSPEM_Core"* contient les concepts de SPEM 2.0 liés à l'exécution d'un processus. Dans le paquetage *"xSPEM_ProjectCharacteristics"*, qui étend le paquetage *"xSPEM_Core"*, sont définis les éléments permettant d'adapter un processus à un projet défini. Ces éléments sont les propriétés spécifiques à l'allocation de ressources et à la planification des activités.

Dans (Bendraou et al., 2007a), les auteurs considèrent les concepts de *"RoleUse"* et de *"WorkProductUse"* comme des ressources nécessaires à la réalisation d'une activité. L'exécution d'un modèle de processus est géré dans le paquetage *"xSPEM_ProcessObservability"*. Pour définir la sémantique d'un langage spécifique à un domaine, xSPEM réutilise une approche basée sur les propriétés (Combemale et al., 2007). Pour l'exécution du modèle de processus, BPEL est utilisé via un mapping avec les concepts de xSPEM. Par exemple, *"WorkProductUse"* de xSPEM est mappé avec *"BPEL Variable"*.

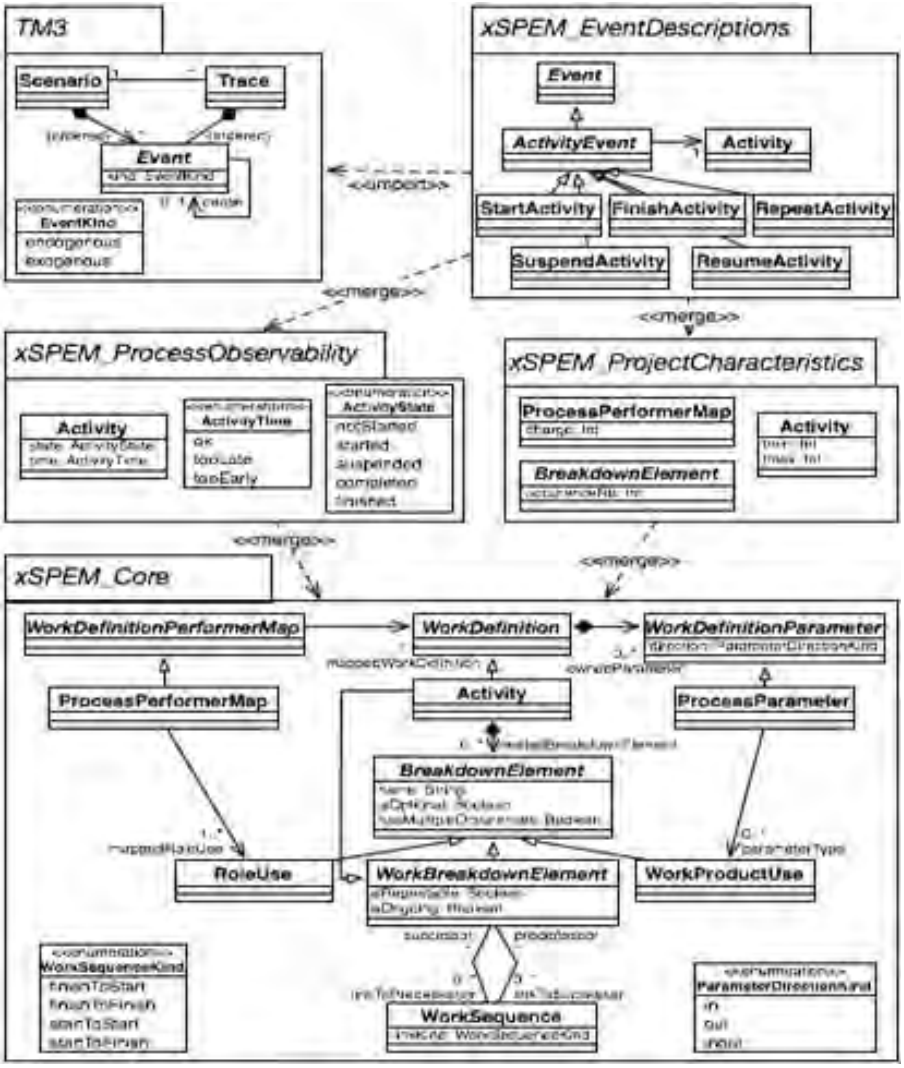


FIGURE 2.24: Méta-modèle de xSPEM (Bendraou et al., 2007a).

Discussion

Le modèle de processus d'exécution avec xSPEM est obtenu après la modélisation. Cela ne permet pas de satisfaire les différents critères de flexibilité par spécification incomplète définis dans la section 2.4.2. Avec l'utilisation de BPEL, le gestionnaire de processus peut toutefois ajouter des éléments ou variables d'exécution. Ces additions peuvent être apparentées à des

variantes étant donné que le modèle d'exécution obtenu avec BPEL va différer du modèle de processus xSPEM.

2.5.3.6 Bilan des approches sur la flexibilité de l'exécution des processus collaboratifs

Dans les sections précédentes, nous avons présenté les approches traitant de l'exécution flexible de processus. Plusieurs langages ont été proposés pour l'exécution des processus. Cependant, à travers l'étude bibliographique réalisée, nous constatons que peu d'entre eux satisfont totalement les différents critères de flexibilité ("looseness", variabilité, évolution et déviation).

Le tableau 2.6 synthétise la comparaison des approches présentées précédemment. Cette comparaison se fait selon les critères de flexibilité rappelés ci-dessus et décrits dans la section (2.4.2).

DECLARE propose une approche déclarative de la modélisation, fondée sur de nombreuses fonctionnalités nécessaires à un système de gestion de workflow. Toutefois, capturer toutes les contraintes qu'un système doit respecter est une tâche fastidieuse qui peut s'avérer très coûteuse en temps. Décrire ce qui devrait se passer est plus simple que décrire ce qui devrait être évité. Elle couvre l'ensemble des critères étudiés excepté la déviation. La solution adoptée par la plupart des approches pour faire face à l'évolution reste une évolution de l'instance en cours d'exécution. Cette solution n'est toutefois pas prise en charge par SPEM4MDE ni par xSPEM. Ces deux dernières approches, de même qu'eSPEM, présentent des limites dans l'exécution flexible des modèles de processus. Néanmoins, eSPEM utilise des variantes de processus et SPEM4MDE des PML Constructs pour satisfaire la variabilité et donc la flexibilité par conception. Dans les approches étudiées, seule ADEPT couvre la déviation en ignorant l'exécution d'une tâche ou en permettant l'ajout dynamique d'une autre tâche.

Aspects	**Approches / Critères**	**DECLARE**	**ADEPT**	**SPEM4MDE**	**eSPEM**	**xSPEM**
Flexibilité par conception	**Variabilité**	PML Constructs	Non prévue	PML Constructs	PML Constructs	Variantes
Flexibilité par spécification incomplète	**"looseness"**	Late-binding	Non prévue	Non prévue	Non prévue	Non prévue
Flexibilité par changement	**Evolution (adaptation)**	Ad-hoc / Evolutionary	Ad-hoc	Non prévue	Ad-hoc	Non prévue
Flexibilité par déviation	**Déviation**	Non prévue	Ignorer la tâche	Non prévue	Non prévue	Non prévue

TABLE 2.6: Tableau comparatif des approches traitant de la flexibilité.

2.6 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art dans le domaine des processus collaboratifs. Nous avons commencé par présenter les fondements des processus, ce qui nous a permis d'expliciter les concepts de base des processus et les notions de modélisation et de mise en oeuvre de processus. Nous avons ensuite défini la notion de collaboration qui est au centre de cette thèse mais aussi explicité la notion de processus collaboratif et présenté les défis dans la gestion des processus collaboratifs.

Nous avons ensuite présenté les deux exigences que doit satisfaire un système de gestion de processus, à savoir la collaboration et la flexibilité. La présentation détaillée de ces deux exigences nous a permis d'identifier les aspects puis les critères qui ont guidé l'étude bibliographique.

Ensuite, nous avons établi un panorama des travaux existants relatifs à la définition, la modélisation et l'exécution flexible des processus. Nous avons ainsi classé les travaux au regard des aspects de la collaboration (coordination, communication et coopération) et de la flexibilité (flexibilité par conception, flexilibité par spécification incomplète, flexibilité par déviation, flexibilité par changement). Cette étude a permis de faire ressortir les limites des approches actuelles, parmi lesquelles nous pouvons lister :

— **Absence d'approche supportant à la fois la flexibilité et la collaboration.**
L'étude des travaux de la littérature faite ci-dessus nous a permis de voir que généralement, les approches mettent l'accent soit sur les aspects de collaboration soit sur ceux de flexibilité mais pas les deux à la fois.

— **Formalisation insuffisante de la collaboration.**
Comme nous l'avons vu dans nos tableaux de synthèse des sections précédentes, les aspects tels que la coordination, la communication et la coopération, ne sont pas toujours couverts par les travaux de la littérature. Ces derniers se limitent à décrire les tâches d'un processus sous forme de boîtes noires et ne prennent pas en considération les efforts élémentaires fournis par chaque participant au sein d'une même tâche. Or nous pensons que la coordination des instances d'une tâche collaborative est un aspect primordial.

— **Rigidité dans la modélisation des tâches collaboratives et manque de flexibilité à l'exécution.**
Les travaux étudiés se limitent à spécifier dès la modélisation comment la collaboration

devrait se faire. Dès lors, toute modification dans la manière de collaborer au niveau de l'exécution oblige à modifier le modèle de processus de départ ou à dévier de ce processus.

Compte tenu de cette analyse, nous avons décidé de proposer une approche permettant de modéliser en premier lieu les concepts inhérents à la collaboration dans les processus. En effet, ces concepts doivent permettre de capter l'ensemble des exigences pour contrôler la contribution de divers individus collaborant avec un même objectif. Dans un second temps, les mécanismes d'exécution de ce processus étant clairement un manque identifié, nous proposons un moyen flexible permettant d'exécuter un processus collaboratif. Pour cela nous avons défini un ensemble de stratégies de collaboration sous forme de patrons, pouvant être appliquées dynamiquement à l'exécution. Les contributions de notre approche font l'objet des chapitres suivants.

Chapitre 3

Langage ECPML

Sommaire

L'objectif de ce chapitre est de fournir un langage formalisé pour décrire la collaboration dans le contexte de la mise en oeuvre de processus. Plus précisément, nous décrivons les éléments d'un processus collaboratif lors de la modélisation et leurs instances à l'exécution.

Ce chapitre est organisé comme suit. Dans la section 3.1, nous décrivons les motivations de notre approche. Nous présentons un exemple de processus que nous utilisons pour illustrer un modèle conforme à notre méta-modèle. La section 3.2 décrit notre langage ECPML avec son volet structurel et son volet comportemental, sous forme de deux paquetages. Nous présentons pour chaque paquetage les différents concepts qui le composent, leurs relations et leur notation graphique.

3.1 Motivation de notre approche

Dans le chapitre 2, nous avons fait une revue des différentes approches de la littérature autour des processus collaboratifs. Nous avons vu notamment les limites des travaux existants lors de la modélisation d'un processus collaboratif, limites qui portent principalement sur la formalisation des stratégies de collaboration. Notre objectif est de proposer une approche de modélisation qui permet l'exécution flexible des processus collaboratifs.

Pour répondre à cet objectif, nous proposons le langage de modélisation de processus ECPML (*"Executable Collaborative Process Modeling Language"*). Nous présentons ce langage sous la forme d'un méta-modèle conforme au MOF. L'intérêt de ce formalisme est qu'il permet de représenter les différents concepts de collaboration que nous introduisons ainsi que les instances des éléments de notre méta-modèle. Certains travaux couvrent la représentation de la collaboration (cf. chapitre 2), mais ils ne définissent pas suffisamment de moyens pour répondre efficacement aux aspects de la collaboration. Ce langage doit être suffisamment précis pour servir de base à la représentation d'un processus collaboratif.

Dans cette section, nous présentons une vue globale de notre approche. Le langage que nous proposons est exprimé sous forme d'un méta-modèle, appelé également ECPML, avec une sémantique opérationnelle permettant son exécution. Le méta-modèle permet la formalisation de la structure d'un processus collaboratif.

Pour ce faire, nous présentons le problème à traiter en nous servant d'un exemple de processus fil conducteur. Le probléme posé justifie la nécessité de notre approche et la solution que nous apportons. Cette solution est illustrée sur le même exemple fil conducteur qui sera utilisé tout au long de ce chapitre.

3.1.1 Exemple fil conducteur

Nous proposons un processus simple de *"rédaction et relecture d'un document"* appelé par la suite *"Writing and Reviewing a Document"* qui servira d'exemple dans le reste du manuscrit (Cisse et al., 2018). Ce processus est composé de trois tâches : *"Write Document"*, *"Review Document"* et *"Modify Document"*. La tâche *"Review Document"* est réalisée par le rôle *"Reviewer"* tandis que les tâches "Write Document" et *"Modify Document"* sont réalisées par le

rôle *"Author"*.

Durant le déroulement de ce processus, l'auteur (*author*) réalise un document qui est utilisé par le relecteur (*reviewer*). Ce dernier émet des annotations dans un autre document qui sera utilisé par l'auteur pour la correction. Les produits qui sont manipulés tout au long de ce processus sont le document *"manuscript"* et le document *"assessment"*.

Dans cet exemple, nous considérons le séquencement suivant entre les deux tâches : *Finish2Start (FS)* ; la première tâche doit être terminée pour que la seconde puisse démarrer.

La Figure 3.1 illustre le processus *"Writing and Reviewing a document"*. Nous pouvons considérer que la tâche *"Review Document"* peut être collaborative ou non, en fonction de l'importance accordée ou du contexte.

Puisque le séquencement entre les deux tâches "Review Document" et "Modify Document" est de type *Finish2Start*, la tâche *Modify Document* ne peut pas démarrer tant que la tâche *Review Document* n'est pas terminée (exemple : modification d'un document après réception des rapports de revue).

Si le séquencement entre ces deux tâches était de type *Start2Start*, la tâche *Modify Document* pourrait commencer dès le démarrage de la tâche *Review Document* (exemple : modification d'un document partagé en ligne dès que les reviewers démarrent la relecture).

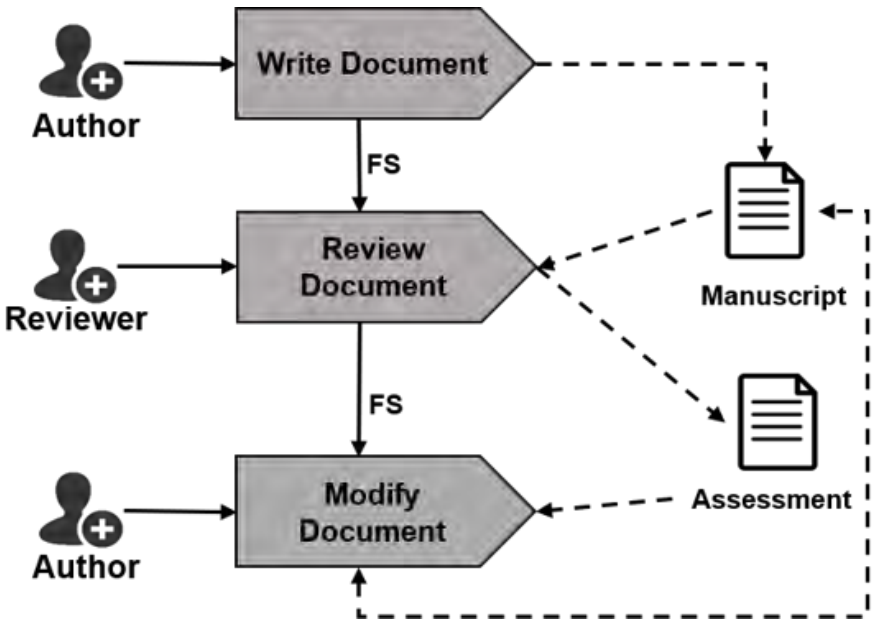


FIGURE 3.1: Exemple fil conducteur du processus "Writing and Reviewing a Document"

Considérant l'exemple de processus décrit ci-dessus, plusieurs difficultés peuvent survenir durant l'exécution. Nous détaillons ces problèmes dans la section suivante.

3.1.2 Problème posé

Pour un contrôle fin et flexible du processus, le moteur d'exécution (ou *process engine*) doit disposer de suffisamment d'informations sur les tâches exécutées. Cependant, comme observé dans le chapitre 2, la façon de collaborer lors de l'exécution d'une tâche collaborative n'est pas souvent décrite dans les approches traitées dans notre état de l'art. Par exemple, sur la figure 3.1, le modèle de processus ne donne aucune information sur la manière dont la revue du document doit être réalisée.

En général, il existe deux approches pour modéliser une tâche collaborative : (1) représenter la tâche collaborative uniquement dans le modèle de processus en ignorant les interactions entre les acteurs lors de son exécution (figure 3.1) ; (2) affiner la tâche et la remplacer par trois tâches définies statiquement dans le modèle de processus afin qu'elles soient gérables pendant l'exécution du processus (figure 3.2). La figure 3.2 montre un exemple de modèle de processus

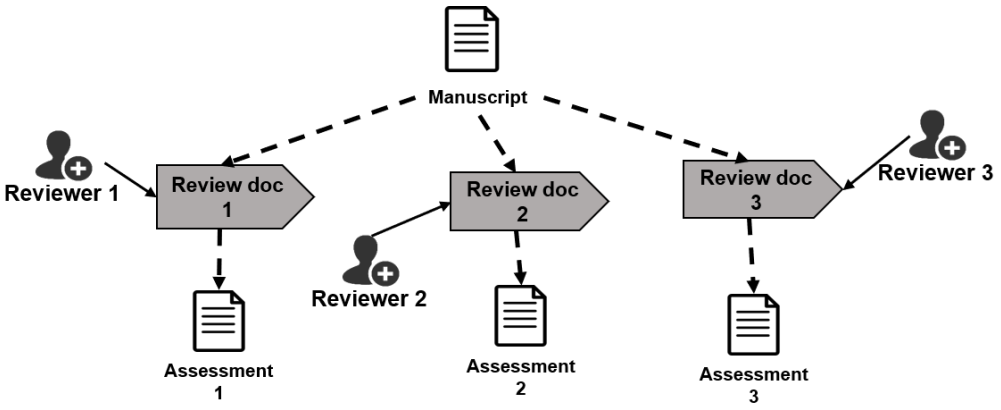


FIGURE 3.2: La tâche collaborative "Review Document" modélisée avec 3 tâches en parallèle.

pour ce second cas, avec des tâches en parallèle.

Dans le premier cas, la collaboration n'est ni décrite à la modélisation, ni contrôlée à l'exécution. Dans le deuxième cas, la collaboration est contrôlée mais est rigide et ne peut s'adapter aux changements de contexte. Les conséquences de cette rigidité sont que le process manager ne peut pas adapter la stratégie de collaboration pour répondre aux possibles évolutions (par exemple l'ajout ou la suppression d'un acteur).

3.1.3 Principe de la solution proposée

Pour faire face au problème posé ci-dessus, il est nécessaire de proposer une solution permettant une bonne flexibilité des modèles de processus. Cette flexibilité doit permettre d'avoir plusieurs modes d'exécution d'un processus donné. Par conséquent, si la tâche *Review Document* (cf figure 3.1) est collaborative, il doit être possible d'avoir plusieurs solutions pour établir les relations entre ses instances à l'exécution. Les figures 3.3 et 3.4 illustrent deux types d'exécution issus de ce modèle. Dans la première (figure 3.3), l'exécution s'effectue en parallèle et dans la deuxième (figure 3.4), elle est séquentielle.

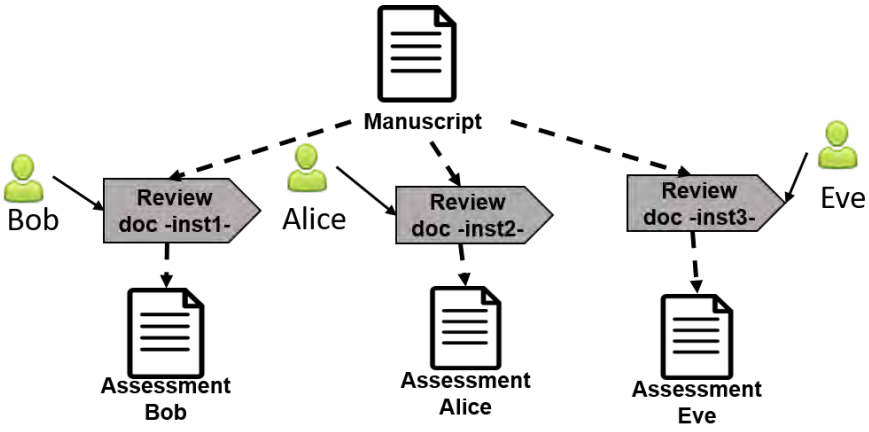


FIGURE 3.3: Exécution de la tâche "Review Document", où les acteurs travaillent en parallèle.

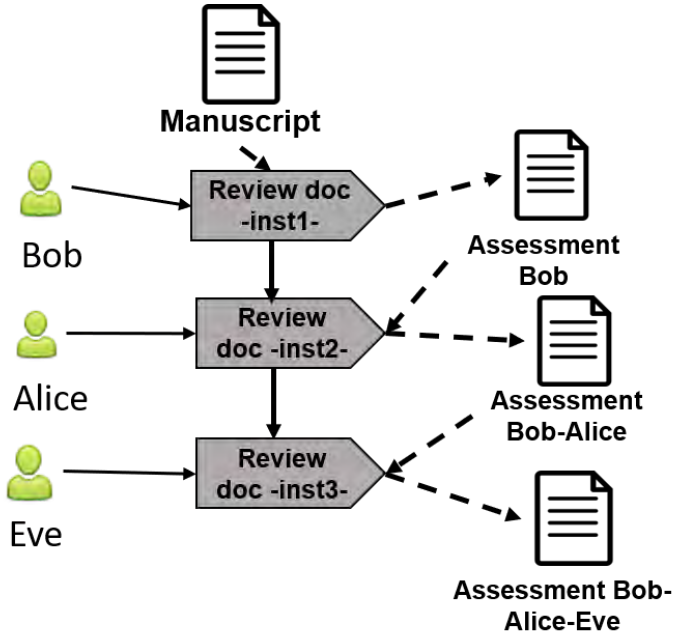


FIGURE 3.4: Exécution de la tâche "Review Document", où les acteurs travaillent en séquentiel.

Ainsi, nous proposons une solution flexible pour l'exécution des tâches collaboratives. Pour

permettre une exécution flexible, nous adoptons l'approche *"Flexibility by underspecification"*. Les différents éléments d'une tâche collaborative sont spécifiés en deux temps :

— au *moment de la modélisation*, seuls les éléments structurels de la tâche (son nom, le rôle qui la réalise, les produits utilisés) sont décrits,

— puis au *moment de l'exécution*, lorsque la tâche est instanciée en plusieurs instances affectées à différents acteurs jouant le rôle spécifié lors de la modélisation, les relations entre les instances de la tâche (séquencement, produits échangés, etc.) sont spécifiées.

Le défi dans l'approche adoptée est de savoir comment générer au moment de l'exécution le modèle comportemental d'une tâche collaborative sans modifier le modèle de processus de départ. Pour ce faire, nous définissons tout d'abord un *patron de collaboration* comme un modèle capturant une stratégie de collaboration qui détermine les relations typiques entre les instances d'une tâche multi-instance au moment de l'exécution. Ensuite, nous utilisons le mécanisme de liaison tardive (*late-binding*) pour permettre aux acteurs de sélectionner dynamiquement un patron de collaboration approprié et de l'utiliser comme modèle pour générer les relations inter-instances pour la tâche collaborative. Pour permettre l'adaptation de l'exécution d'une tâche collaborative à l'évolution de contexte du projet, le process manager peut modifier la stratégie de collaboration en sélectionnant un autre patron de collaboration plus approprié.

En outre, durant l'exécution, pour des besoins d'adaptation à l'évolution du contexte, une tâche peut évoluer pour devenir collaborative. Un exemple d'évolution pourrait être la décomposition d'une tâche jugée complexe pour la partager entre plusieurs participants. Pour ce faire, il est nécéssaire de prévoir un mécanisme dynamique permettant la transformation d'une tâche en tâche collaborative. Cette tâche collaborative a par la suite recours, comme toute tâche collaborative, à un patron de collaboration pour son instanciation et son exécution.

3.2 Le méta-modèle ECPML

Le méta-modèle ECPML est inspiré de SPEM pour les éléments de processus concernant la modélisation statique. Les concepts représentant les éléments d'un processus collaboratif à l'exécution ont été définis par nous-mêmes, étant donné que SPEM ne supporte pas l'exécution de processus ni la notion de tâche collaborative.

Notre langage de modélisation de processus permet de décrire des processus sur deux axes :

— *Un modèle structurel (du processus)* : Le modèle structurel décrit les éléments du processus, i.e. *tâche*, *produit*, *rôle*, et leurs *relations*. Ces éléments sont sans état et décrivent un processus statiquement sans considérer l'exécution.

— *Un modèle comportemental (du processus)* : Le modèle comportemental décrit les éléments du processus à l'exécution, i.e. *instance de tâche*, *instance de produit*, *acteur* et les relations entre eux. Les éléments dans ce modèle sont des instances des éléments du modèle structurel. Ils sont dynamiques et peuvent changer d'état selon une sémantique opérationnelle associée à chacun d'entre eux.

Pour profiter des fonctionnalités d'un standard de l'OMG reconnu, nous avons décidé de nous inspirer de SPEM. Pour réduire la complexité de notre méta-modèle (présenté ci-dessous), nous ne réutilisons pas le méta-modèle de SPEM dans sa totalité. Nous nous fondons sur le paquetage *Process Structure* pour la réutilisation de certains concepts de SPEM pour l'aspect structurel de notre méta-modèle.

La Figure 3.5 montre la décomposition en paquetages de notre méta-modèle. Il est composé de deux paquetages *ECPML_Core* et *ECPML_Execution*. Les éléments principaux du paquetage *ECPML_Execution* sont des instances d'éléments du paquetage *ECPML_Core*. Pour représenter cela, nous avons défini une nouvelle dépendance entre les paquetages, appelée *"instanceOf"*. La sémantique de cette dépendance est que certains éléments du paquetage *ECPML_Execution* instancient d'autres éléments du paquetage *ECPML_Core* pour l'exécution du processus. Le paquetage *ECPML_Execution* importe le paquetage *"BehaviorStateMachines"* d'UML pour définir le comportement des éléments exécutables. Dans *ECPML_Core*, nous réutilisons des méta-classes issues du paquetage *ProcessStructure* de SPEM.

3.2.1 Volet structurel du langage ECPML

L'aspect structurel définit les éléments statiques d'un processus. Dans cette section, nous introduisons le paquetage *ECPML_Core* permettant de représenter la structure d'un modèle de processus défini avec ECPML.

ECPML_Core (voir Figure 3.6) offre les éléments permettant de modéliser la structure d'un processus collaboratif, sous la forme de concepts et de relations entre eux. Ce paquetage est inspiré, pour certains éléments, du paquetage *ProcessStructure* de SPEM 2.0.

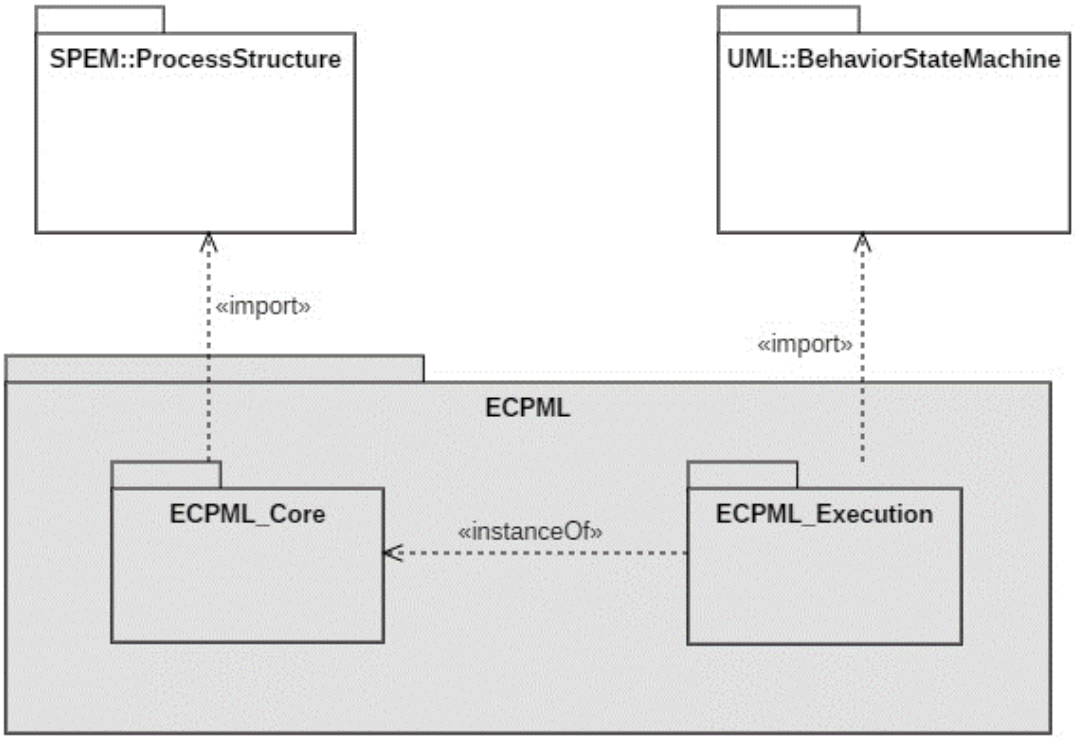


FIGURE 3.5: Organisation en paquetages du méta-modèle ECPML.

Nous décrivons le coeur d'un modèle de processus collaboratif à l'aide des quatre concepts suivants : *SingleTask*, *CollaborativeTask*, *WorkProduct*[1] et *Role*[2]. La Figure 3.6 montre ces éléments et les relations entre eux.

Dans la suite de cette section, nous décrivons les concepts de ce paquetage en proposant des éléments de syntaxe concrète graphique sous forme d'icônes. Pour chaque méta-classe, nous décrivons le *nom*, la *super classe* (néant s'il n'y en a pas), la *description* de la méta-classe, la liste de ses *attributs* s'il y en a, les *propriétés d'association*, les *contraintes* au niveau des propriétés s'il y en a, et la *notation* avec notre syntaxe concrète. Les propriétés d'association définissent les attributs issus des relations avec d'autress méta-classes. Donc pour une méta-classe donnée, nous décrivons les rôles que les méta-classes associées jouent dans ces relations.

3.2.1.1 Task

Super Classe

1. Ce concept est inspiré du concept WorkProductUse de SPEM 2.0.
2. Ce concept est inspiré du concept RoleUse de SPEM 2.0.

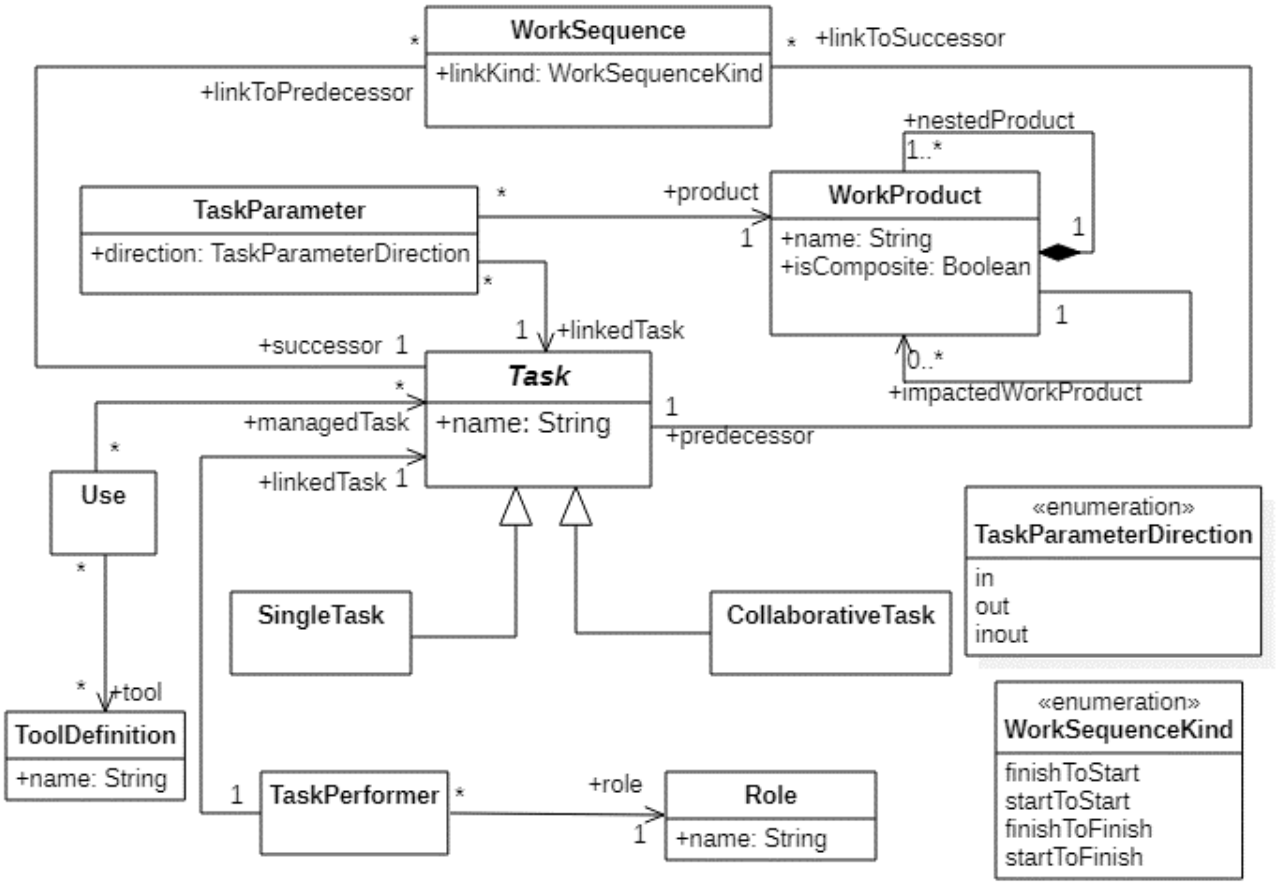


FIGURE 3.6: Paquetage ECPML_Core.

Néant

Description

Une tâche est un concept abstrait représentant une unité de travail dans ECPML. Il est lié à des produits via le concept *TaskParameter* et est assigné à un unique rôle via le concept *TaskPerformer*. Une tâche dans ECPML est définie à la modélisation et donne lieu à une ou plusieurs instances exécutables durant l'exécution.

Attributs

— *name : String*. Il décrit le nom de la tâche.

Propriétés d'associations

— *linkToSuccessor : WorkSequence[0..*]*. Elle spécifie la nature de la relation de succession d'une tâche avec une ou d'autres tâches.

— *linkToPredecessor : WorkSequence[0..*]*. Elle spécifie la nature de la relation de précédence d'une tâche avec une ou d'autres tâches.

3.2.1.2 Single Task

Super Classe

Task

Description

Le concept *SingleTask* désigne une tâche assignable à un acteur. Elle est non décomposable et non collaborative mais peut évoluer pour devenir collaborative. Elle est une spécialisation du concept abstrait *Task* et hérite des relations qu'il a avec les autres méta-classes (*TaskParameter* et *TaskPerformer*).

Notation

Stéréotype	Icône
SingleTask	

3.2.1.3 Collaborative Task

Super Classe

Task

Description

Le concept *CollaborativeTask* désigne une tâche collaborative. A l'exécution, la tâche donne lieu à plusieurs instances de tâches simples réalisées par différents acteurs.

Notation

Stéréotype	Icône
CollaborativeTask	

3.2.1.4 Work Product

Super Classe

Néant

Description

Le concept *WorkProduct* représente une entité tangible et concrète utilisée ou produite durant l'exécution d'une tâche. C'est un élément qui représente une *entrée* et/ou *sortie* d'une tâche. Il peut être assimilé au concept *WorkProductUse* de SPEM. Il est lié à la tâche grâce à la relation *TaskParameter*. Pour une tâche donnée, nous pouvons avoir plusieurs WorkProduct manipulés. Un WorkProduct peut être composite, ce qui n'est pas le cas de WorkProductUse dans SPEM.

Attributs

- *name : String* : Cet attribut décrit l'intitulé du produit.
- *isComposite : Boolean* : Cet attribut est utilisé pour définir si le produit manipulé est composite ou élémentaire. Un produit composite peut être décomposé récursivement en sous-produits.

Propriétés d'associations

- *nestedProduct : WorkProduct[1..*]*. Elle spécifie les composants d'un produit composite.
- *impactedProduct : WorkProduct[0..*]*. Elle spécifie les produits impactés par la modification d'un produit.

Notation

Stéréotype	**Icône**
WorkProduct	
WorkProductComposite	

3.2.1.5 Role

Super Classe

Néant

Description

L'élément *Role* est un concept représentant la qualification de l'acteur humain qui contribue à la réalisation d'une tâche. Il est lié à la tâche via la relation *TaskPerformer*. Un même rôle peut être amené à participer à plusieurs tâches.

Attributs

— *name : String* : Il décrit l'intitulé du rôle.

Notation

Stéréotype	*Icône*
Role	

3.2.1.6 Work Sequence

Super Classe

Néant

Description

Le concept *WorkSequence* représente la relation entre deux tâches dans ECPML_Core. C'est une contrainte d'ordonnancement qui porte sur le démarrage ou la fin d'une des tâches par rapport à l'autre. Ce concept provient de SPEM.

Attributs

— *linkKind : WorkSequenceKind*. Cet attribut définit le type de la relation de séquencement entre deux tâches. Sa valeur est donnée par une énumération définie par la méta-classe *WorkSequenceKind*.

Propriétés d'associations

— *successor : Task[1]*. Elle spécifie la tâche jouant le role successeur dans une association *WorkSequence*.

— *predecessor : Task[1].* Elle spécifie la tâche jouant le role prédécesseur dans une association *WorkSequence.*

Contraintes

[C1] Le successeur et le prédécesseur d'une tâche ne doivent pas référencer la même tâche.

context WorkSequence inv : self.successor <> self.predecessor

Notation

Stéréotype	**Icône**
n/A	

3.2.1.7 Work Sequence Kind

Description

L'énumération *WorkSequenceKind* décrit les différentes valeurs de la relation de séquencement (*WorkSequence*) entre deux tâches. Elle provient de SPEM. Considérant une tâche A suivie par une tâche B, les valeurs du séquencement possibles sont :

— *finishToStart* : La tâche B ne peut pas commencer tant que la tâche A n'est pas terminée.
— *startToStart* : La tâche B ne peut pas commencer tant que la tâche A n'a pas commencé. Une autre façon de décrire cette valeur est de dire que la tâche B peut démarrer dès que la tâche A commence son exécution.
— *startToFinish* : La tâche B ne peut pas finir son exécution tant que la tâche A n'a pas démarré.
— *finishToFinish* : La tâche B ne peut pas finir son exécution tant que la tâche A n'a pas fini la sienne.

3.2.1.8 Task Performer

Super Classe

Néant

Description

Le concept *TaskPerformer* est une méta-classe qui représente la relation entre le rôle chargé de l'exécution d'une tâche et la tâche considérée.

Propriétés d'associations

— *linkedTask : Task[1].* Elle spécifie la tâche exécutée par le rôle considéré.
— *role : Role[1].* Elle spécifie le rôle en charge de l'exécution d'une tâche. Un rôle peut être assigné à plusieurs tâches.

Notation

Stéréotype	**Icône**
n/A	

3.2.1.9 Task Parameter

Super Classe

Néant

Description

Le concept *TaskParameter* représente la relation entre une tâche (*Task*) et un produit (*WorkProduct*). Il permet de représenter quelles entrées sont utilisées/modifiées lors de l'exécution de la tâche, et quelles sorties sont produites à la fin de l'exécution de la tâche.

Attributs

— *direction : TaskParameterDirection.* Cet attribut représente le type du produit tel que spécifié par l'énumération *TaskParameterDirection*, à savoir en entrée (*in*), en sortie (*out*) ou en entrée-sortie (*inout*).

Propriétés d'associations

— *linkedTask : Task[1].* Elle spécifie la tâche manipulant un produit.
— *product : WorkProduct[1].* Elle spécifie le produit manipulé (création ou modification) par une tâche.

Notation

Stéréotype	Icône
n/A	

3.2.1.10 Task Parameter Direction

Description

Cette énumération définit les différentes valeurs que peut prendre un *TaskParameter* et permet de savoir si le paramètre représente une entrée, une sortie ou une entrée-sortie. Elle est tirée de SPEM. Les différentes valeurs possibles sont :

— *in* : Définit un paramètre d'entrée.
— *out* : Définit un paramètre de sortie.
— *inout* : Définit un paramètre d'entrée-sortie.

3.2.1.11 ToolDefinition

Super Classe

Néant

Description

Le concept *ToolDefinition* représente une ressource matérielle ou logicielle utilisée dans l'exécution d'une tâche. Il participe à l'élaboration de la tâche sans pour autant être la ressource principale.

Attributs

— *name : String*. Il décrit l'intitulé de l'outil.

Notation

Stéréotype	Icône
ToolDefinition	

3.2.1.12 Use

Super Classe

Néant

Description

La méta-classe *Use* représente la relation entre une ressource matérielle ou logicielle utilisée dans l'exécution d'une tâche et la tâche elle-même.

Propriétés d'associations

- *managedTask : Task[*].* Elle spécifie la ou les tâches dans lesquelles la ressource est utilisée. Un outil peut être utilisé dans la réalisation de plusieurs tâches.
- *tool : ToolDefinition[*].* Elle spécifie le ou les outils utilisés comme ressource par la tâche.

3.2.1.13 Application à l'exemple fil conducteur

Dans la figure 3.7, nous montrons un extrait du processus de l'exemple fil conducteur *"Writing and Reviewing a Document"* modélisé avec l'utilisation du paquetage *ECPML_Core*. Nous utilisons la syntaxe concrète du langage ECPML pour illustrer les différents concepts représentés. Dans ce processus la tâche *Review Document* est collaborative et est réalisée par un ensemble de *Reviewer*. Elle utilise le manuscrit pour produire un document d'évaluation (*Assessment*). Les *WorkProduct* présents dans ce processus sont ainsi le document *Manuscript* et le document *Assessment*. La *CollaborativeTask Review Document* est suivie par la *SingleTask Modify Document* avec le séquencement *Finish2Start*. Cette dernière tâche utilise le document d'évaluation ainsi que le manuscrit pour produire une version corrigée du manuscrit. Le document d'évaluation est en entrée (*in*) de la tâche *Modify Document* tandis que le document manuscrit est en entrée-sortie (*inout*).

3.2.2 Volet comportemental de ECPML

Dans cette section, nous introduisons le paquetage *ECPML_Execution* et les comportements permettant la mise en oeuvre des éléments du modèle de processus.

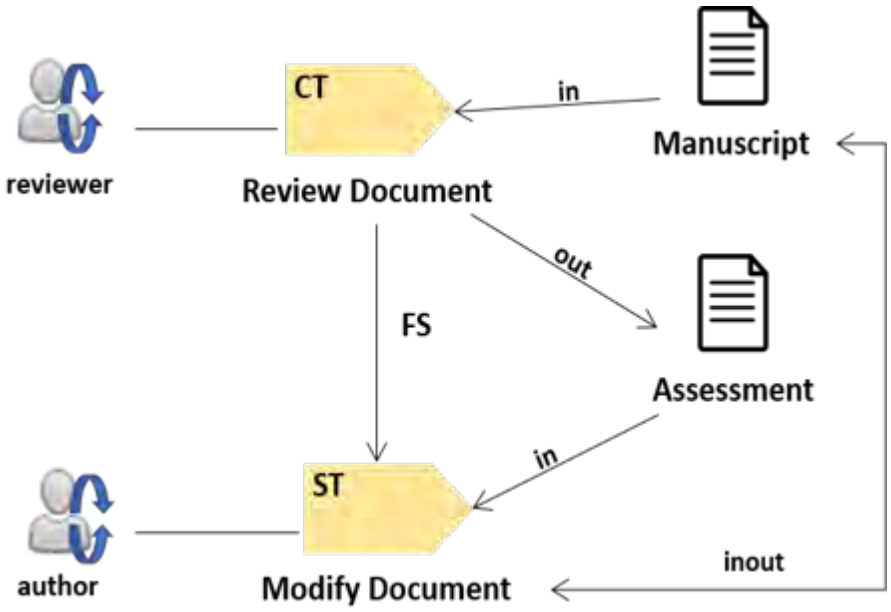


FIGURE 3.7: Illustration du volet statique à l'exemple de la relecture de document.

Le paquetage *ECPML_Execution* définit les concepts utilisés pour représenter l'exécution d'un processus. Pour ce faire, nous définissons d'une part les concepts représentant les instances créées à l'exécution à partir des éléments définis dans le modèle du processus, d'autre part le comportement de ces instances durant l'exécution du processus. Ce paquetage permet d'attribuer une sémantique opérationnelle aux concepts de la collaboration. Cela permet d'avoir le comportement des éléments du processus et ainsi, de pouvoir contrôler l'exécution du processus.

Les concepts principaux utilisés dans le paquetage *ECPML_Execution* sont les suivants : *SingleTaskInstance*, *CollaborativeTaskInstance*, *WorkProductInstance* et *Actor* (cf. figure 3.8). Ces concepts sont des instances des éléments structurels présentés sur la figure 3.6 comme montré plus bas sur la figure 3.9 à l'exception du concept *Actor* qui est une personne physique qui peut avoir les qualifications définies par un rôle.

La figure 3.9 présente les relations d'instanciation entre les éléments de ECPML_Execution et ceux du paquetage ECPML_Core. Les éléments *SingleTaskInstance*, *CollaborativeTaskInstance* et *WorkProductInstance* sont respectivement des instances des éléments *SingleTask*, *CollaborativeTask* et *WorkProduct*. Le concept *Actor* représente la ressource humaine affectée au Role. La relation permettant de définir cette instanciation est définie par *instanceOf* comme spécifié dans le diagramme de paquetage de la figure 3.5. Les relations entre les méta-classes de ce paquetage sont héritées des relations entre les méta-classes qui les définissent (du paquetage

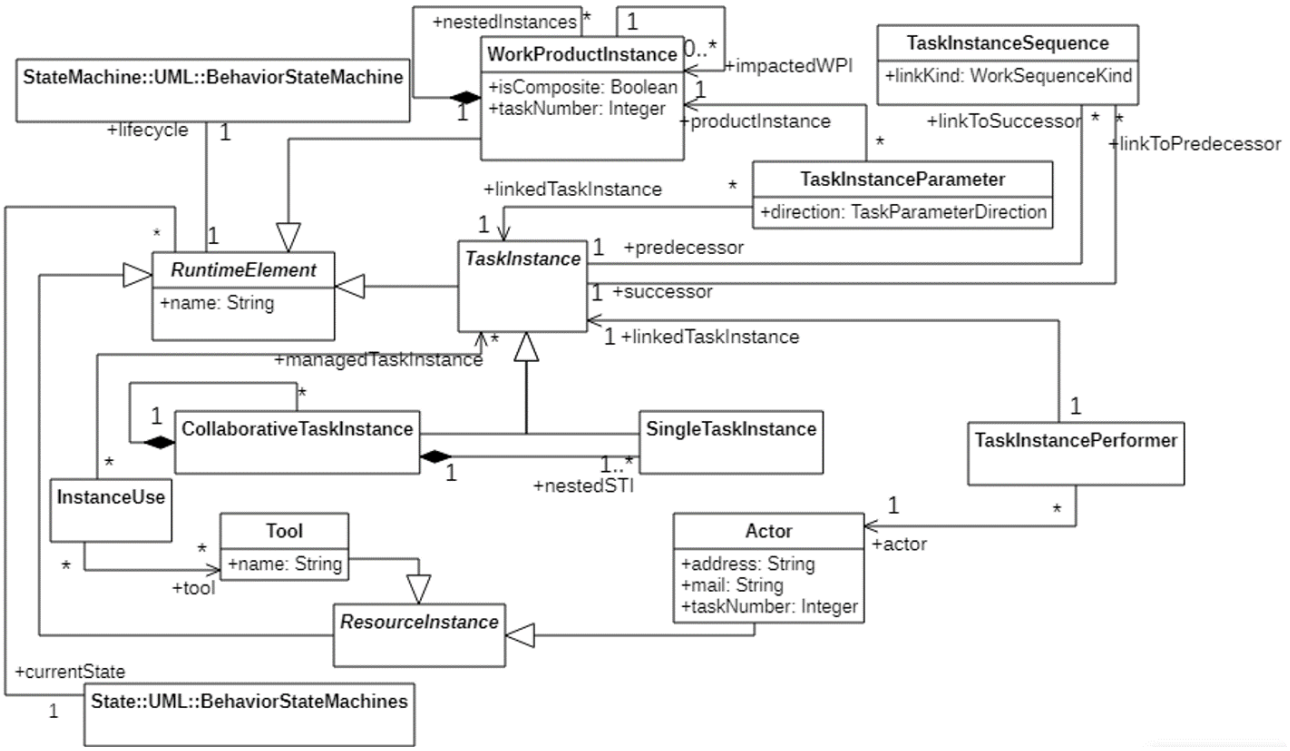


FIGURE 3.8: Paquetage ECPML_Execution.

ECPML_Core).

Pour associer un comportement aux éléments du processus, nous réutilisons le paquetage *BehaviorStateMachines* d'UML. Grâce à l'utilisation des machines à état, nous définissons les états des éléments du processus à l'exécution et les transitions marquant leurs évolutions.

Dans les sous-sections suivantes, nous décrivons les méta-classes du paquetage *ECPML_Execution* selon la même structure que celles du paquetage *ECPML_Core*. Cependant, pour les méta-classes spécialisant la méta-classe *RuntimeElement*, nous ajoutons une rubrique sur la machine à états associée afin d'introduire la description de leur comportement. La description des transitions est détaillée dans le chapitre 5.

3.2.2.1 Runtime Element

Super Classe

Néant

Description

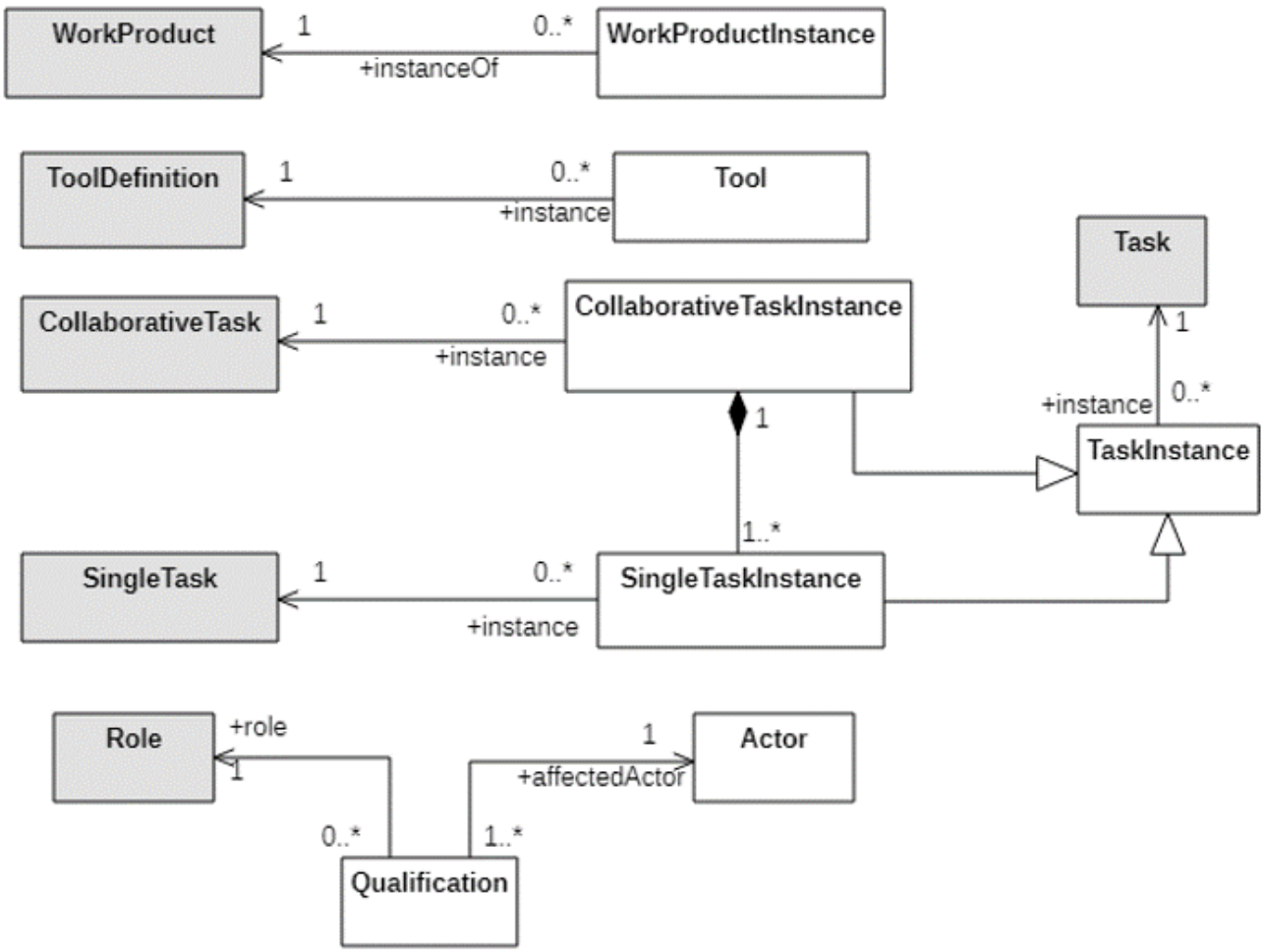


FIGURE 3.9: Relations d'instanciation entre éléments des deux paquetages de ECPML.

La méta-classe *RuntimeElement* est une généralisation abstraite des éléments d'une instance de processus. Un RuntimeElement a un état et est associé à un machine à états définissant ses possibles transitions d'état.

Attributs

— *name : String*. Il décrit le nom de l'élément.

Propriétés d'associations

— *lifecycle : StateMachine*. Elle spécifie le cycle de vie d'un RuntimeElement. Le cycle de vie définit les différents états par lesquels l'élément passe durant son exécution, et les transitions entre ces états.

— *currentState : State*. Elle spécifie l'état courant dans lequel le RuntimeElement se trouve à l'exécution. Le concept *State* provient du paquetage *UML : :BehaviorStateMachine*.

3.2.2.2 State Machine : :UML : :BehaviorStateMachine

Super Classe

Néant

Description

Le concept *StateMachine* représente une machine à états utilisée pour décrire le comportement des éléments exécutables du processus. C'est une réutilisation du concept de même nom du paquetage *BehaviorStateMachine* du méta-modèle d'UML (Object Management Group (OMG), 2007). Une machine à états décrit le cycle de vie d'un élément exécutable. Elle comprend un état initial, un ensemble d'états et de transitions entre eux, et un état final.

3.2.2.3 Task Instance

Super Classe

RuntimeElement

Description

Le concept *TaskInstance* est un élément abstrait qui représente une instance de tâche, soit *SingleTaskInstance* ou *CollaborativeTaskInstance*. Il peut avoir des paramètres qui sont des *WorkProductInstance*. Le séquencement entre deux instances de tâches est défini dans *TaskInstanceSequence*.

Propriétés d'associations

- *linkToSuccessor : TaskInstanceSequence[*]*. Elle lie une tâche à son ou ses successeurs.
- *linkToPredecessor : TaskInstanceSequence[*]*. Elle lie une tâche à son ou ses prédécesseurs.

3.2.2.4 Single Task Instance

Super Classe

TaskInstance

Description

Le concept *SingleTaskInstance (STI)* est un élément de base du modèle d'exécution du processus. Il représente une instance de *SingleTask*. C'est l'élément assignable à un acteur via *TaskInstancePerformer*. Etant une instance de *SingleTask*, il manipule des produits durant l'exécution via le concept *TaskInstanceParameter*.

Notation

Stéréotype	**Icône**
SingleTaskInstance	

Comportement d'une STI

Le comportement d'une instance de tâche (*SingleTaskInstance*) est défini par la machine à états de la figure 3.10. Dans son cycle de vie, nous avons défini quatre états : *Instantiated*, *Assigned*, *InProgress* et *Finished*. La figure 3.10 montre la machine réduite à l'enchaînement de ses états. Ses transitions sont détaillées dans le chapitre 5.

L'état *Instantiated* spécifie que l'instance de tâche a été créée. Par la suite, lorsqu'un acteur est assigné à la STI, elle passe à l'état *Assigned*.

Suivant le séquencement ou la stratégie de collaboration, diverses conditions doivent être satisfaites avant le début de la STI. Lorsque ces conditions sont remplies, la STI passe à l'état *InProgress* pour son exécution. De même, durant l'exécution, il est possible de faire évoluer la STI pour la rendre collaborative. Cette évolution provoque une transition vers l'état final de la STI étant donné que la STI devient une CTI (et donc son exécution est prise en charge par la machine à états d'une CTI).

L'état *Finished* marque la fin de l'exécution de la STI lorsque les conditions de fin de tâche sont remplies.

3.2.2.5 Collaborative Task Instance

Super Classe

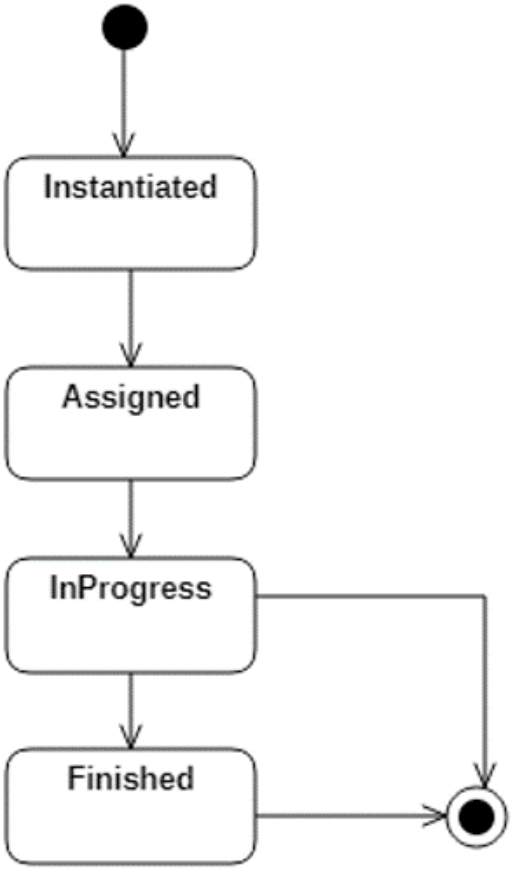


FIGURE 3.10: Cycle de vie d'une instance de tâche simple (réduit à l'enchaînement des états).

Task Instance

Description

Le concept *CollaborativeTaskInstance (CTI)* désigne une instance du concept *CollaborativeTask*. Il représente une tâche multi-instance qui peut être exécutée par plusieurs acteurs jouant le même rôle. Il contient au moins une *SingleTaskInstance* et a son propre cycle de vie. Chaque STI est exécutée par un acteur différent.

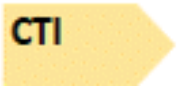
Propriétés d'associations

— *nestedSTI : SingleTaskInstance[1..*]*. Elle spécifie les *SingleTaskInstance* qui sont englobées dans l'instance d'une tâche collaborative.

Contraintes

[C1] Chaque STI de la liste nestedSTI est exécutée par un acteur différent.

Notation

Stéréotype	Icône
CollaborativeTaskInstance	

Comportement d'une CTI

Le comportement d'une instance de tâche collaborative (*CollaborativeTaskInstance*) est défini par la machine à états de la figure 3.11. Nous avons défini quatre états : *Instantiated*, *Assigned*, *InProgress* et *Finished*.

A partir de l'état *initial*, la transition vers l'état *Instantiated* est déclenchée dès l'instanciation du processus. A ce stade, la tâche (*Task*) dans le modèle est utilisée pour créer une instance de tâche collaborative (*CollaborativeTaskInstance*). Durant cette transition, les relations entre les différentes STI sont créées conformément à la stratégie de collaboration choisie et à la description faite dans la section 3.1.

A partir de l'état *Instantiated*, des acteurs (*Actor*) sont assignés aux différentes instances de tâche (*SingleTaskInstance*) composant l'instance de la tâche collaborative. La tâche passe de l'état *Instantiated* à l'état *Assigned*.

A partir de l'état *Assigned*, si les conditions permettant le démarrage de la tâche sont vérifiées, la tâche passe à l'état *InProgress*. Les acteurs assignés commencent l'exécution de leurs STIs conformément à la stratégie de collaboration définie.

Durant le déroulement de la tâche (*InProgress*), plusieurs choix s'offrent à l'acteur en fonction du contexte auquel il est soumis : soit terminer l'exécution de la tâche, soit changer la stratégie de collaboration en cours d'exécution. Lorsqu'un changement de contexte survient, cela peut impacter la stratégie de collaboration. Le changement de stratégie redéfinit les relations qui existaient entre les différentes instances de tâche (*STI*) de l'instance de tâche collaborative (*CTI*).

Lorsque la tâche collaborative a fini d'être exécutée, elle passe à l'état *Finished*. Cette transition s'effectue lorsque les conditions de fin sont respectées (par exemple, toutes les instances de tâches (*STI*) doivent être terminées, sans être supprimées).

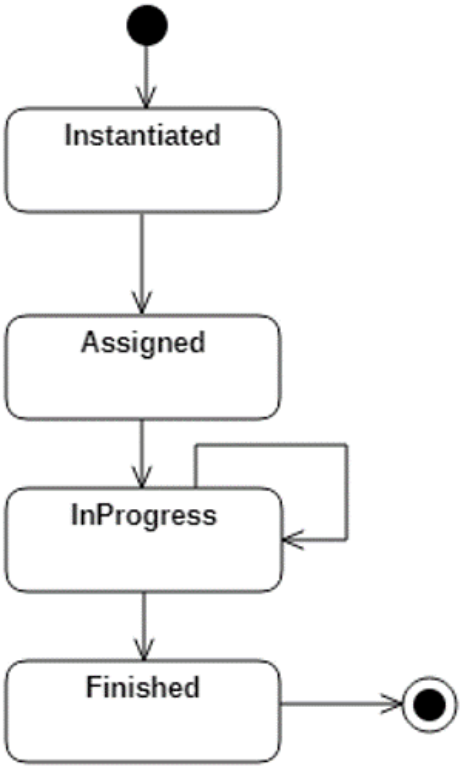


FIGURE 3.11: Cycle de vie d'une instance de tâche collaborative (réduit à l'enchaînement des états).

3.2.2.6 Resource Instance

Super Classe

RuntimeElement

Description

ResourceInstance est un concept abstrait définissant un élément chargé de la réalisation d'une instance de tâche. Il peut s'agir d'une ressource humaine (*actor*) ou non humaine (programme, *tool*).

3.2.2.7 Actor

Super Classe

ResourceInstance

Description

Actor est une ressource humaine utilisable pour exécuter une instance de tâche. Un acteur spécifique peut travailler simultanément sur plusieurs tâches distinctes. Cependant cette simultanéïté n'est pas contrôlée. Il a un rôle spécifique dans une tâche. En tant que *RuntimeElement*, il dispose d'une machine à états (*state machine*) décrivant les différents états par lesquels il passe lors de l'exécution. Il est lié à un rôle par la relation *Qualification* qui spécifie qu'un acteur peut jouer plusieurs rôles (cf. figure 3.9).

Attributs

— *address : String*. Il définit l'adresse de l'acteur.
— *mail : String*. Il définit l'adresse électronique de l'acteur.
— *taskNumber : Integer*. Il définit le nombre de STIs dont l'acteur est responsable.

Notation

Stéréotype	**Icône**
Actor	

Comportement d'un acteur

Dans la figure 3.12, nous montrons le comportement d'un acteur (*Actor*). Le concept *Actor* est utilisé pour exécuter une instance de tâche (*SingleTaskInstance*) et manipuler (création ou modification) des livrables (*WorkProductInstance*). Dans son cycle de vie, nous avons défini trois états : *Free*, *Assigned* et *Performing*.

L'état *Free* spécifie que l'acteur est dans le système mais n'est pas en train d'exécuter une instance de tâche, ou a fini l'exécution de ses instances de tâche.

Par la suite, il peut passer à l'état *Assigned*. Cela signifie que l'acteur est assigné à une instance de tâche pour l'exécution. La transition réflexive au niveau de l'état *Assigned* est activée lorsqu'une autre instance de tâche est assignée à l'acteur.

Dès lors, ce dernier peut commencer à exécuter l'instance lorsque les conditions sont remplies. L'état *Performing* permet de spécifier que l'acteur a démarré l'exécution de l'une des instances dont il est responsable. En cours d'exécution d'une instance, l'acteur peut recevoir d'autres

tâches à exécuter. Cette nouvelle assignation provoque une transition réfléxive sur l'état *Performing*.

A partir de l'état *Performing*, l'acteur peut finir l'exécution de ses instances de tâche et retourner à l'état *Free*, ce qui signifie que l'acteur redevient libre pour être éventuellement assigné à une autre tâche

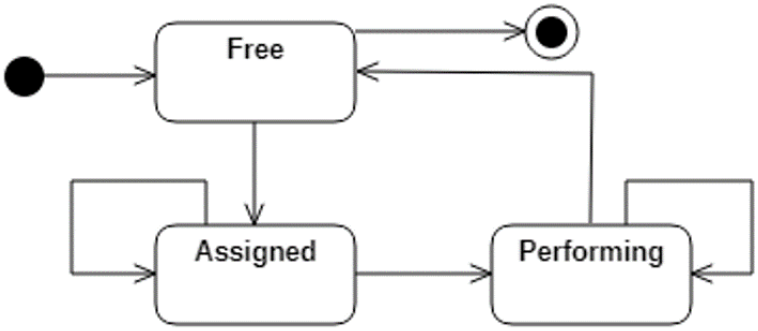


FIGURE 3.12: Cycle de vie d'un acteur (réduit aux états et à leur enchaînement).

3.2.2.8 Tool

Super Classe

ResourceInstance

Description

L'élément *Tool* est une ressource non-humaine utilisable pour exécuter une instance de tâche. Il est considéré comme une ressource matérielle ou logicielle.

Notation

Stéréotype	Icône
Tool	

3.2.2.9 Work Product Instance

Super Classe

RuntimeElement

Description

Le concept *WorkProductInstance* représente une instance d'un produit (*Work Product*) manipulé (création ou modification) par une instance de tâche. Il peut s'agir d'un produit *d'entrée* ou de *sortie* pour l'instance de tâche. Il est lié à l'instance de tâche via la méta-classe *TaskInstanceParameter*. Un *WPI* peut être composite i.e décomposé en sous-produits à l'exécution.

Attributs

— *isComposite : Boolean* : Cet attribut est utilisé pour définir si le produit manipulé est composite ou élémentaire.

— *taskNumber : Integer*. Cet attribut définit le nombre de STIs qui utilise la WPI.

Propriétés d'associations

— *nestedInstances : WorkProductInstance[0..*]*. Elle spécifie les instances de produit encapsulées.

— *impactedWPI : WorkProductInstance[0..*]*. Elle spécifie la dépendance entre deux instances de produit. La modification de l'une des instances de produit pourra impacter l'autre instance de produit.

Notation

Stéréotype	**Icône**
WorkProductInstance	
WorkProductInstanceComposite	

Comportement d'une WPI

La figure 3.13 détaille les différents états par lesquels l'instance de produit (*WorkProductInstance*) peut passer durant l'exécution de l'instance de tâche. Nous avons défini trois états : *Defined*, *InUse* et *Validated*.

L'état *Defined* signifie que la tâche est instanciée et que les produits définis sur le modèle sont aussi instanciés. Lorsque le produit passe à cet état, cela signifie qu'il attend d'être modifié (dans le cas d'un produit en sortie) ou d'être consommé (dans le cas d'un produit en entrée).

L'état *InUse* spécifie que le produit subit des modifications ou est utilisé pour la création du produit de sortie. Cela implique le démarrage de l'instance de tâche.

Quand l'instance de tâche termine son exécution, le produit passe à l'état *Validated*.

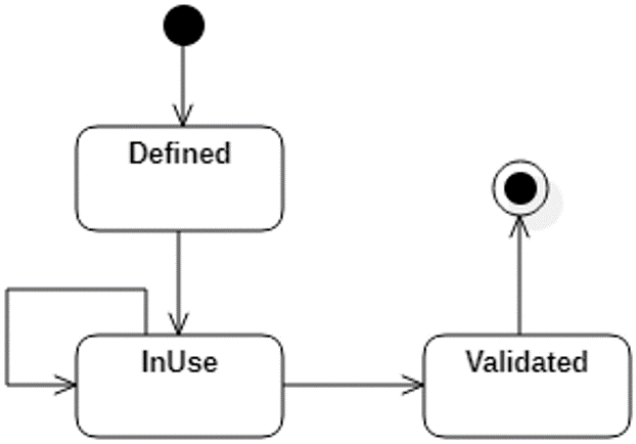


FIGURE 3.13: Cycle de vie d'une instance de produit (réduit aux états et à leur enchaînement).

3.2.2.10 Task Instance Sequence

Super Classe

Néant

Description

Le concept *TaskInstanceSequence* est une instance d'une contrainte d'ordonnancement (*WorkSequence*) du package *ECPML_Core*. Elle représente la relation entre deux instances de tâche. La dépendance entre les deux tâches est exprimée par l'intermédiaire d'un attribut définissant le séquencement. *TaskInstanceSequence* possède les mêmes attributs et propriétés d'association que *WorkSequence* du paquetage ECPML_Core.

Attributs

— *linkKind : WorkSequenceKind* (Les valeurs possibles de l'énumération *WorkSequenceKind* ont été données dans la section 3.2.1.7 ci-dessus).

Propriétés d'associations

— *successor : TaskInstance[1].* Instance de tâche qui succède.

— *predecessor : TaskInstance[1].* Instance de tâche qui précède.

Contraintes

C1 La relation de séquencement ne doit pas lier une instance de tâche avec elle-même.
context TaskInstanceSequence inv : self.successor <> self.predecessor

Notation

Stéréotype	**Icône**
n/a	

3.2.2.11 Task Instance Performer

Super Classe

Néant

Description

Le concept *TaskInstancePerformer* est semblable au concept *TaskPerformer* du paquetage *ECPML_Core*. Il représente la relation entre l'acteur chargé de l'exécution d'une instance de tâche et l'instance de tâche considérée.

Propriétés d'associations

— *linkedTaskInstance : Task[1].* Spécifie l'instance de tâche assignée à l'acteur considéré.

— *actor : Actor[1].* Spécifie l'acteur associé à l'instance de tâche.

Contraintes

C1 L'association *linkedTaskInstance* ne peut lier un Actor qu'à un SingleTaskInstance.
Context TasKInstancePerformer inv :self.linkedTaskInstance.oclsTypeOf (SingleTaskInstance)

Notation

Stéréotype	Icône
n/a	

3.2.2.12 Task Instance Parameter

Super Classe

Néant

Description

Le concept *TaskInstanceParameter* est semblable au concept *TaskParameter* du paquetage *ECPML_Core*. C'est une méta-classe qui représente la relation entre une instance de tâche (*TaskInstance*) et une instance de produit (*Work Product Instance*).

Attributs

— *direction : TaskParameterDirection* : Cet attribut représente le type de l'instance de produit comme spécifié par l'énumération *TaskParameterDirection* du paquetage *ECPML_Core*. Le produit peut être en entrée (*in*) en sortie (*out*) ou en entrée-sortie (*inout*).

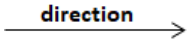
Propriétés d'associations

— *linkedTaskInstance : TaskInstance[1].* Elle spécifie l'instance de tâche manipulant un produit.
— *productInstance : WorkProductInstance[1].* Elle spécifie l'instance de produit manipulé (création ou modification) par une instance de tâche.

Contraintes

[C1] L'association *linkedTaskInstance* ne peut lier un WorkProductInstance qu'à un SingleTaskInstance.
Context TasKInstanceParameter inv :self.linkedTaskInstance.oclsTypeOf (SingleTaskInstance)

Notation

Stéréotype	**Icône**
n/a	

3.2.2.13 Qualification

Super Classe

Néant

Description

Le concept *Qualification* représente la relation qui lie un rôle (*Role*) à l'acteur (*Actor*) qui le joue. Un même rôle peut être joué par plusieurs acteurs.

Propriétés d'associations

- *affectedActor : Actor*. Il représente l'acteur spécifique qui joue le rôle.
- *role : Role*. Il représente le rôle joué par l'acteur.

3.2.2.14 InstanceUse

Super Classe

Néant

Description

La méta-classe *InstanceUse* représente la relation entre un outil (*Tool*) et une instance de tâche (*TaskInstance*).

Propriétés d'associations

- *managedTaskInstance : TaskInstance[*]*. Elle représente la ou les instances de tâche qui utilisent la ressource.
- *tool : Tool[*]*. Elle spécifie un ou plusieurs outils utilisés comme ressource par l'instance de tâche.

Contraintes

[C1] L'association *managedTaskInstance* ne peut lier un Tool qu'à un SingleTaskInstance.
Context InstanceUse inv :self.managedTaskInstance.oclsTypeOf (SingleTaskInstance)

3.2.2.15 Application à l'exemple fil conducteur

Nous présentons le résultat obtenu après instanciation du processus "Writing and Reviewing a Document" de notre exemple fil conducteur, avec l'utilisation du paquetage *ECPML_Execution*. Nous utilisons notre syntaxe concrète pour illustrer les différents concepts représentés. Etant donné que seule la tâche *Review Document* est collaborative, nous choisissons d'omettre les autres tâches du processus dans la figure 3.14. Pour alléger la figure, nous omettons la CTI englobante des trois STIs illustrées.

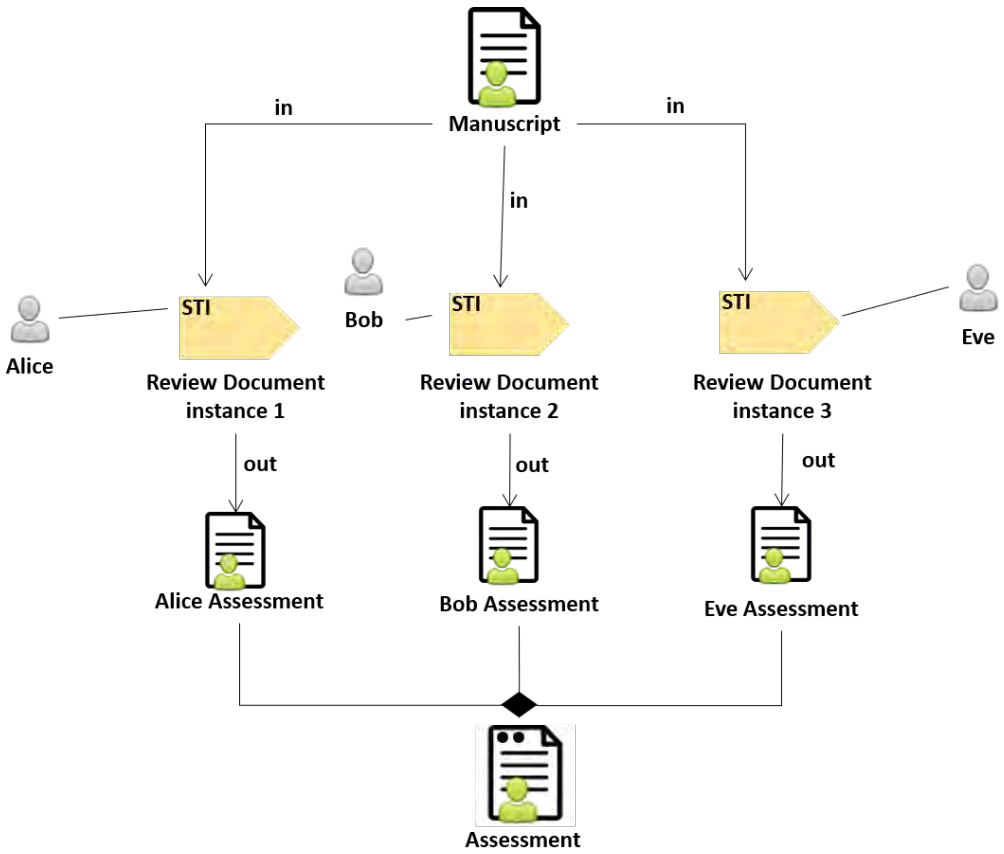


FIGURE 3.14: Modèle d'exécution de la tâche collaborative "ReviewDocument" selon une stratégie parallèle.

Pour illustrer l'exécution, nous avons opté pour trois instances de tâche, assignées aux acteurs *Alice*, *Bob* et *Eve*, et exécutées en parallèle. Chaque acteur réalise sa tâche et produit un livrable

qui est son document d'évaluation (*Assessment*). L'ensemble des trois livrables constitue le livrable final de la tâche collaborative.

La figure 3.15 présente l'exécution du même processus en séquentiel, comme spécifié par le séquencement FS (FinishToStart). Les acteurs *Alice*, *Bob* et *Eve* exécutent l'un à la suite de l'autre les tâches *Review Document instance 1*, *Review Document instance 2* et *Review Document instance 3*. De la même façon que pour l'exécution parallèle, le document *Manuscript* est utilisé en entrée des trois instances de tâches. Avec cette stratégie, Bob peut tenir compte de l'évaluation d'Alice pour faire la sienne ; de même Eve bénéficie de l'évaluation de Bob pour effectuer la sienne. Pour alléger la figure, nous omettons la CTI englobante des trois STIs illustrées.

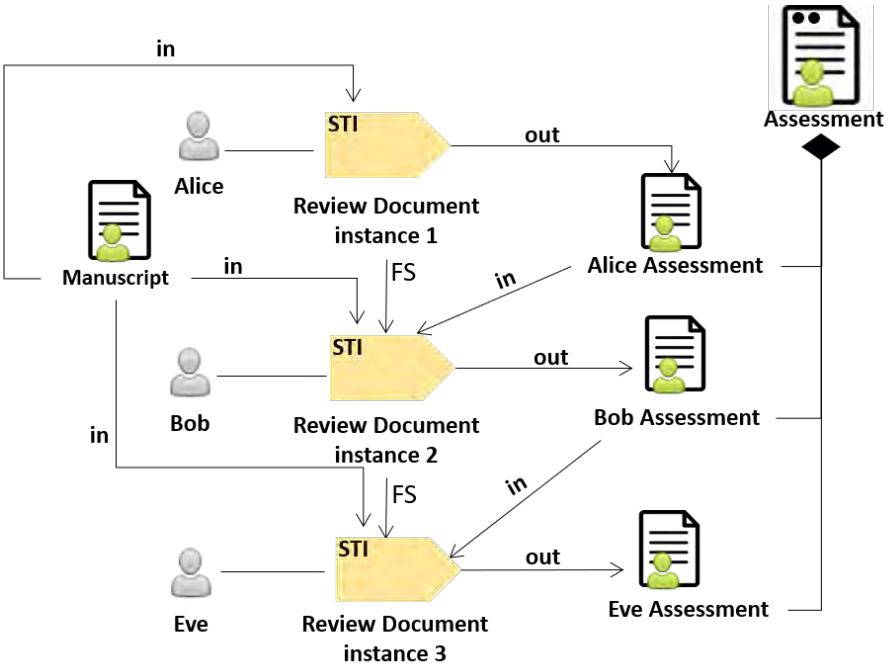


FIGURE 3.15: Modèle d'exécution de la tâche collaborative "ReviewDocument" selon une stratégie séquentielle.

3.3 Conclusion

Dans ce chapitre, nous avons présenté le langage *ECPML* permettant la modélisation et l'exécution d'un processus collaboratif. Rappelons que nous visons une exécution flexible basée sur la génération du modèle comportemental d'une tâche collaborative au moment de l'exécution par l'utilisation du mécanisme du *late-binding*.

Nous avons ensuite présenté les concepts permettant de représenter un processus collaboratif et de contrôler son exécution. Le méta-modèle proposé est inspiré de SPEM et est construit autour d'un volet structurel pour les concepts statiques et d'un volet comportemental pour la gestion du cycle de vie des éléments d'une instance de processus. Pour représenter les liens d'instanciation entre nos paquetages, nous avons défini la relation *instanceOf* qui représente une relation d'instanciation entre certains concepts du paquetage *ECPML_Core* et d'autres du paquetage *ECPML_Execution*. L'aspect comportemental de notre langage est exprimé par le biais de machines à états UML. La sémantique des actions de ces machines à états est détaillée dans le chapitre 5 qui est dédié à la sémantique.

Chapitre 4

Patrons d'exécution pour les tâches collaboratives

Sommaire

Dans le chapitre précédent, nous avons proposé un langage, *ECPML*, qui permet de décrire à la fois l'aspect structurel (*modélisation*) et l'aspect comportemental (*exécution*) d'un processus. Dans ce chapitre, nous présentons le concept de patron de collaboration et la manière de l'utiliser pour rendre l'exécution d'un processus collaboratif flexible. Nous introduisons tout d'abord la définition de patron de collaboration (section 4.1) et expliquons les principes d'organisation des stratégies de collaboration en patrons (Section 4.2). La description d'un patron de collaboration utilise le langage *ECPML*. Ensuite, nous présentons une liste de patrons de collaboration représentant des scénarios typiques qui se produisent durant l'exécution d'une tâche collaborative (Section 4.3). La section 4.4 présente l'utilisation des patrons pour l'exécution d'un processus. Nous concluons ce chapitre en faisant un bilan sur leur utilisation.

4.1 Notion de patron de collaboration

Le concept de patron de processus a été utilisé dans plusieurs travaux (Lonchamp, 1998; van der Aalst et al., 2003a; Tran et al., 2007). Dans ces travaux, un patron de processus fournit une solution à un problème récurrent de la modélisation de processus. Dans le cadre de cette thèse, nous utilisons les patrons de processus pour fournir les solutions aux problèmes d'instanciation et d'exécution d'une tâche collaborative.

Pour permettre l'exécution flexible d'une tâche collaborative, l'idée est de ne pas définir de manière statique le mode d'exécution de la tâche collaborative au moment de la modélisation,

mais de permettre à un acteur (concepteur du processus / gestionnaire de processus, etc.) de déterminer de manière dynamique les stratégies qui répondent à ses besoins au moment de l'exécution. Pour ce faire, nous avons défini un ensemble de patrons de collaboration représentant différentes stratégies pouvant être utilisées au moment de l'exécution d'une tâche collaborative en fonction du contexte du projet. Ainsi, le gestionnaire du processus peut choisir, parmi différents patrons de collaboration, celui qui lui paraît le plus approprié au contexte courant.

4.1.1 Stratégie de collaboration

Une tâche collaborative peut être mise en œuvre selon différentes stratégies. Une stratégie de collaboration définit la façon selon laquelle les acteurs collaborent pour réaliser la tâche (*"the way of working"*). Comme nous l'avons vu dans le chapitre précédent, chaque acteur exécute une instance STI de l'instance CTI de la tâche collaborative. Une stratégie de collaboration a pour but de définir les relations entre les STIs d'une CTI.

En général, il y a deux grandes stratégies de collaboration correspondant à deux types *d'ordonnancement des instances de tâche* : la *stratégie séquentielle* (*"sequential instances"*) et la *stratégie parallèle* (*"parallel instances"*). Les stratégies de collaboration avec un ordonnancement séquentiel impliquent que les instances de tâche soient exécutées les unes à la suite des autres. Dans l'ordonnancement parallèle, les instances de tâche ne sont pas soumises à une contrainte d'exécution et peuvent donc s'exécuter dans un ordre quelconque (exemple : *"les trois instances d'une même tâche peuvent s'exécuter au même moment."*).

Plusieurs facteurs peuvent impacter une stratégie de collaboration. Dans ce travail, nous en avons identifié trois : la *structure des produits manipulés*, *l'intention de la collaboration* et la *disponibilité des ressources*.

— *structure du produit* : un produit manipulé par une tâche peut être *composite* ou non. Lorsque le produit est composite, il peut y avoir ou non des *dépendances* entre les différents composants (voir figure 4.1).

— *intention de la collaboration* : la volonté derrière l'exécution de la tâche. L'intention définit l'objectif visé dans la manipulation ou la création du produit en sortie. Il peut s'agir d'un raffinement de produit, d'une division de l'effort, etc.

— *disponibilité des ressources*. Les différentes ressources traitées ici sont celles nécessaires

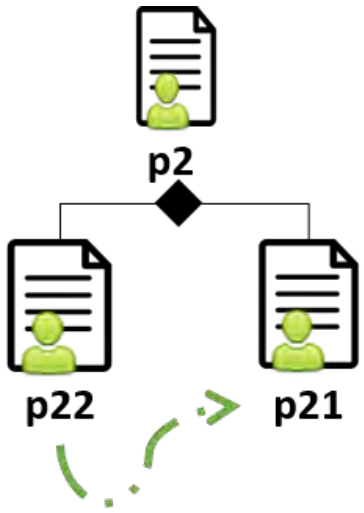


FIGURE 4.1: Dépendances entre les deux composants d'un produit. Ici p22 dépend de p21.

pour l'exécution de la tâche, à savoir des ressources humaines (acteurs), des ressources matérielles et des produits en entrée. Ces éléments constituent ce que nous appelons le contexte d'application. Ce contexte est évolutif et est fixé dynamiquement par le manager en fonction des ressources. A l'exécution, on peut avoir le cas idéal où toutes les ressources demandées sont disponibles (nombre de ressources = nombre de STIs à l'exécution) ou le cas limité où certaines ressources ne sont pas disponibles.

Autrement dit, dans le contexte d'un projet spécifique, le choix d'une stratégie de collaboration reflète l'intention des acteurs mais doit prendre en compte les contraintes imposées par la structure du produit et la disponibilité des ressources.

4.1.2 Définition d'un patron de collaboration

Nous définissons un patron de collaboration comme un patron de processus (Tran et al., 2007) dont l'objectif est de fournir une stratégie pour mettre en oeuvre une tâche collaborative dans un contexte spécifique. La solution d'un patron de collaboration est un modèle d'exécution qui décrit les relations typiques entre les instances d'une tâche collaborative au moment de l'exécution sous deux perspectives principales : le *"control-flow"* et le *"data-flow"*. La perspective *control-flow* fournit l'ordre d'exécution de différentes instances de tâches, représentées par les relations de *work sequence* entre les instances. La perspective *data-flow* spécifie, via la notion de *paramètre de tâche*, les données manipulées, échangées ou partagées par les instances de tâche.

Les patrons de collaboration présentés ici sont appliqués dynamiquement à l'exécution pour générer le modèle détaillé des instances de tâches à exécuter. Dans notre approche, au moment de la modélisation, un processus est partiellement défini : le modèle de processus ne contient que des éléments structurels. Les éléments définissant les relations entre les instances d'une tâche collaborative du processus, sont complétés à l'exécution par *l'application du patron de collaboration choisi*.

Considérant une tâche collaborative T dans un processus, son modèle peut être vu comme une fonction paramétrée *TM (c : CollaborationPattern)* où c est un paramètre formel qui sera lié à un patron concret "*cp*" sélectionné par le gestionnaire de processus. L'exécution de la tâche T consiste à exécuter *(TM (cp))* pour créer l'instance de tâche T_I. T_I est définie avec deux sources d'informations : les éléments structurels de T_I qui sont instanciés à partir du modèle de tâches T_M, les éléments dynamiques de T_I qui sont créés en appliquant le patron *cp*. Les éléments structurels représentent les relations de l'instance T_I avec l'acteur et les produits manipulés. Les éléments dynamiques représentent le séquencement des différentes $STIs$ de T_I.

4.2 Impact de la nature des produits sur le choix de la stratégie de collaboration

Nous avons vu ci-dessus que les produits en entrée ou en sortie peuvent avoir une structure composite ou non. Or cette structuration, de même que l'intention et la disponibilité des ressources ont un impact important sur l'ordonnancement des tâches et le nombre d'instances de la tâche collaborative.

Ci-dessous, nous présentons des situations illustrant la combinaison des éléments impactant la collaboration. Ces éléments sont présentés dans une situation *idéale*. Une situation est considérée comme idéale lorsque le nombre de ressources (acteurs, outils) nécessaire est égal au nombre d'instances voulu pour la tâche collaborative.

Nous considérons aussi, sans être exhaustifs, des variantes ou alternatives qui correspondent à ce que nous appelons *"une situation limitée"*. Une situation est dite limitée lorsque les acteurs ou les outils nécessaires pour l'exécution des instances de la tâche collaborative ne sont pas tous disponibles.

Considérons pour faciliter le raisonnement, une tâche collaborative avec deux produits P_1 et P_2, P_1 étant le produit en entrée et P_2 le produit en sortie. P_1 et P_2 peuvent être composites ou non. Si un produit est composite, ses composants peuvent avoir des dépendances entre eux ou non.

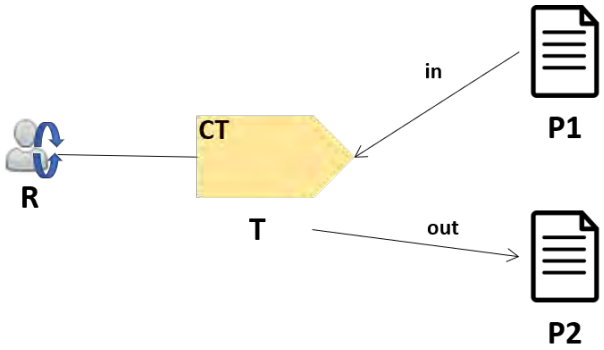


FIGURE 4.2: Modèle de processus illustrant les produits en entrée et sortie d'une tâche collaborative T.

Nous présentons dans la suite de cette section dix combinaisons de structure des produits d'entrée et de sortie P_1 et P_2, et leurs impacts sur la stratégie de collaboration.

4.2.1 Cas 1 : P_1 et P_2 non-composites

Plaçons-nous dans le cas d'un produit P_1 en entrée non-composite. Lorsque le produit P_2 en sortie est non-composite, il n'est pas aisé d'avoir un contrôle fin des différentes contributions de chaque acteur sur le produit en sortie. En effet, il n'est pas forcément possible de connaître la contribution exacte de chaque acteur dans le produit final.

Dans une situation idéale, il est possible de recourir :

- à une exécution en parallèle avec l'intention de faire du *co-travail libre* (*"free co-work"*) pendant laquelle chaque acteur agit sur le produit en sortie indépendamment de la contribution des autres (par exemple rédiger de manière collaborative un document sur Google Doc),
- ou à une exécution en séquence pour faire du *raffinement de produit* (*"refinement"*).

Dans une situation limitée, le raffinement n'est pas impacté. En effet le raffinement se faisant en mode séquentiel, il ne nécessite pas que les ressources soient disponibles en totalité pour démarrer l'exécution. Pour le *"free co-work"*, il sera par contre impossible d'exécuter toutes les

instances en parallèle. Une alternative serait d'exécuter en parallèle le nombre d'instances possibles et d'avoir un ordonnancement séquentiel pour les tâches restantes. Cet ordonnancement permettrait d'exécuter la ou les tâches restantes dès que des ressources se libèrent.

4.2.2 Cas 2 : P_1 non-composite et P_2 composite sans dépendances

Plaçons-nous de nouveau dans le cas d'un produit en entrée non composite. Lorsque le produit en sortie est composite, il est souhaitable pour optimiser l'exécution d'avoir autant d'instances de tâche que de composants. De ce fait, chaque instance permet d'élaborer un composant du produit de sortie.

Dans une situation idéale, avec le nombre de composants égal au nombre d'instances et au nombre de ressources, il s'agit de diviser l'effort pour augmenter par exemple la productivité (*"effort division"*). Chaque acteur exécute une instance de la tâche collaborative pour produire un composant du produit en sortie. Ce travail peut se faire en parallèle étant donné qu'il n'y a pas de dépendances entre les composants.

Dans une situation limitée, il est impossible de faire une exécution en parallèle complète. Comme dans le cas 1, si les ressources sont insuffisantes, il faut exécuter une partie avec la stratégie *"effort division"* et le reste en séquentiel.

4.2.3 Cas 3 : P_1 non-composite et P_2 composite avec dépendances totales

Plaçons-nous encore dans le cas d'un produit P_1 en entrée non composite. Considérons que le produit de sortie P_2 est composite avec des dépendances portant sur tous les composants (*"full dependency"*). Dans ce cas de dépendance, il ne doit pas y avoir de circularité. La figure 4.3 illustre cette situation. Dans cet exemple, nous avons trois composants. p_{23} dépend de p_{22} et p_{22} dépend de p_{21}.

Dans ce cas, les dépendances imposent l'ordre de production des composants du produit. Dans une situation de dépendance totale, l'exécution peut se faire en mode séquentiel pour avoir une réalisation par étapes du produit de sortie. Il s'agit d'un *co-travail complètement ordonné*

FIGURE 4.3: Illustration de la dépendance totale au niveau d'un produit composite.

ou *"full ordered co-work"*. La stratégie "full-ordered co work", imposée par la configuration du produit de sortie P_2, peut être appliquée dans une situation idéale ou limitée.

4.2.4 Cas 4 : P_1 non-composite et P_2 composite avec dépendances partielles

Considérons un produit en entrée non-composite et un produit en sortie composite. Nous supposons que les dépendances au niveau du produit de sortie portent uniquement sur certains composants (*"partial dependency"*).

Dans une situation de dépendance partielle (comme illustré par la figure 4.4), les instances de tâche manipulant les composants dépendants sont exécutées en mode séquentiel. Les instances restantes sont exécutées sans ordonnancement imposé. De ce fait, il est possible d'exécuter ces instances avant celles qui le sont en séquentiel. Il s'agit d'un *co-travail partiel* ou *"partial ordered co-work"*.

Dans une situation limitée, les instances de tâche manipulant les produits sans dépendances ne s'exécuteront pas en parallèle mais les unes à la suite des autres.

FIGURE 4.4: Illustration de la dépendance partielle entre les composants d'un produit composite.

4.2.5 Cas 5 : P_1 composite sans dépendances et P_2 non-composite

Considérons maintenant que le produit P_1 est composite avec des composants non dépendants, et que le produit en sortie P_2 est non composite. Les acteurs peuvent travailler de manière simultanée sans contrôle sur l'accès au produit en sortie. Chaque composant en entrée est utilisé par une instance de tâche indépendamment des autres instances. Il s'agit d'un *"free co-work"*.

Dans une situation limitée, l'exécution se déroule de la même façon que lorsque les deux produits P_1 et P_2 sont non composites. C'est-à-dire qu'on peut exécuter en parallèle le nombre d'instances possible et exécuter le reste dès qu'une ressource additionnelle devient disponible.

4.2.6 Cas 6 : P_1 composite avec des dépendances et P_2 non-composite

Lorsque le produit P_1 en entrée est composite avec des dépendances et le produit P_2 en sortie est non-composite, l'ordre d'exécution est imposé par P_1. Chaque composant du produit en entrée est utilisé dans un ordre précis par les différentes instances de tâche pour produire un produit de sortie non composite. Par conséquent, la réalisation du produit en sortie dépend successivement des contributions issues de chaque composant du produit en entrée. Il s'agit

d'un cas de raffinement (*"refinement"*).

En cas de ressources humaines insuffisantes, donc en situation limitée, l'ordre d'exécution séquentielle n'est pas impacté. En effet, le raffinement ne nécessite pas la présence de toutes les ressources dès le début.

4.2.7 Cas 7 : P_1 et P_2 composites sans dépendances

Lorsque les produits de la tâche collaborative (entrée et sortie) sont composites sans dépendances entre leurs composants, l'ordre d'exécution des instances de tâches n'est pas imposé. Un composant du produit en entrée peut être utilisé par une instance de tâche pour la production d'un composant du produit en sortie, ainsi de suite, sans impact sur les autres instances de tâche. En effet, c'est le même principe qui est suivi lors de la division de l'effort (cas 2) pour augmenter la productivité. L'exécution peut se faire alors en parallèle.

En situation limitée, chaque instance de tâche s'exécute toujours indépendamment des autres lorsque ses besoins en ressources sont satisfaits. Les autres instances de tâches s'exécutent dès qu'une ressource devient disponible. Le même principe que le *"effort division"* est suivi mais avec des ressources limitées. La différence est que les deux produits P_1 et P_2 sont composites.

4.2.8 Cas 8 : P_1 composite sans dépendances et P_2 composite avec dépendances

Une autre combinaison possible est quand le produit en entrée est composite sans dépendances et le produit en sortie composite avec dépendances. Dans ce cas, la production des différents composants du produit en sortie se fait dans un ordre défini par les dépendances entre eux. Le nombre d'instances est imposé par le nombre de composants du produit en sortie. Il s'agit du cas de *"full ordered co-work"* lorsque les dépendances sont totales. Dans une situation de dépendance partielle, nous retombons dans le cas du *"partial ordered co-work"*. Les instances manipulant les produits sans dépendances sont exécutées sans ordonnancement imposé, dans un mode de *"partial ordered co-work"*.

En situation limitée, par exemple lorsque le nombre d'acteurs nécessaires est insuffisant, l'exécution

n'est pas impactée étant donné qu'il s'agit d'une exécution séquentielle.

4.2.9 Cas 9 : P_1 et P_2 composites avec dépendances totales

Lorsque les deux produits P_1 et P_2 sont composites avec des dépendances totales, l'ordre d'exécution des différentes instances est dicté par l'ordre des dépendances entre les composants du produit en sortie. Comme pour le cas précédent, les dépendances obligent à opérer une exécution séquentielle des instances de la tâche. Il s'agit alors d'un *"full ordered co-work"*. Comme pour le cas précédent, le nombre d'instances est également imposé par le nombre de composants du produit en sortie.

En cas de situation limitée, avec des ressources limitées (acteurs), l'exécution n'est pas impactée donc on peut garder le *"full ordered co-work"*.

4.2.10 Cas 10 : P_1 composite avec dépendances et P_2 composite sans dépendances

Lorsque P_1 est composite avec des dépendances, et que P_2 est également composite mais sans dépendances, les instances de tâche s'exécutent indépendamment les unes des autres. Les dépendances au niveau du produit en entrée n'impactent pas l'ordonnancement étant donné que les entrées sont supposées être disponibles avant le début de la tâche collaborative. Par contre le nombre d'instances est imposé par le nombre de composants du produit en sortie. L'exécution des différentes instances de tâche peut être faite en parallèle. De ce fait, l'effort sera divisé au niveau de chaque instance de tâche pour la production des différents composants du produit en sortie. Il s'agit du *"effort division"*.

En situation limitée, lorsque le nombre d'acteurs nécessaires à l'exécution en parallèle des instances de la tâche collaborative est insuffisant, le même principe que le *"effort division"* est suivi. Les acteurs disponibles exécutent en parallèle les instances possibles et le reste est exécuté dès qu'une autre ressource devient disponible.

Dans la figure 4.5, nous synthétisons les dix cas de combinaison présentés ci-dessus, en supposant les situations idéales pour la disponibilité des ressources.

SORTIE / ENTREE		COMPOSITE			NON-COMPOSITE
		DEPENDANCES TOTALES	DEPENDANCES PARTIELLES	SANS DEPENDANCES	
COMPOSITE	DEPENDANCES	***Cas 9*** - *Full ordered co-work*	***Cas 9*** - *Partial ordered co-work*	***Cas 10*** - *Effort division*	***Cas 6*** - *Refinement*
	SANS DEPENDANCES	***Cas 8*** - *Full ordered co-work*	***Cas 8*** - *Partial ordered co-work*	***Cas 7*** - *Effort division*	***Cas 5*** - *Free co-work*
NON-COMPOSITE		***Cas 3*** - *Full ordered co-work*	***Cas 4*** - *Partial ordered co-work*	***Cas2*** - *Effort division*	***Cas 1*** - *Refinement* - *Free co-work*

FIGURE 4.5: Tableau de synthèse des cas avec les stratégies de collaboration associées.

Ces différentes combinaisons concernant les produits en entrée/sortie d'une tâche collaborative décrivent des situations conduisant à des stratégies de collaboration récurrentes. De ces stratégies, nous pouvons inférer des patrons que nous présentons dans la suite. Aussi, nous observons que l'ensemble des cas décrits permettent d'identifier six stratégies formalisées sous forme de patrons. Ces patrons n'ont pas pour but d'être exhaustifs, mais ils couvrent les situations idéales au sens utilisé ci-avant. En fonction du contexte, en prenant en compte des situations dites "limitées", il est possible de définir d'autres patrons que nous considérons comme des variantes.

4.3 Patrons de collaboration

Nous présentons ci-dessous un ensemble de patrons représentatifs des scénarios les plus fréquemment rencontrés durant l'exécution d'une tâche collaborative. La figure 4.6 montre la classification des patrons que nous présentons dans la suite de ce chapitre. Les six patrons présentés sont applicables dans les cas présentés précédemment dans la section 4.2. De ce fait, un même patron peut être appliqué dans plusieurs cas, dépendant du contexte présenté. Dans la présentation de la solution de chaque patron ci-dessous, nous privilégions le premier cas pour lequel le patron est applicable. Les autres cas de ce même patron sont présentés dans l'annexe C. Chacun des six patrons présentés correspond à une feuille de l'arbre (couleur rouge).

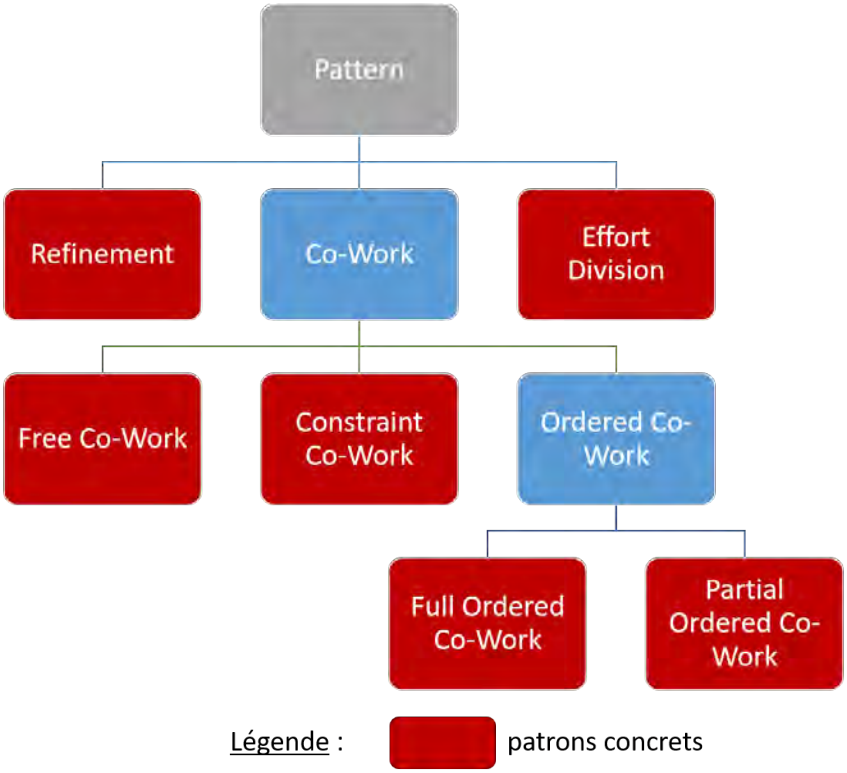


FIGURE 4.6: Classification des patrons de collaboration présentés.

4.3.1 Formalisme de description de patrons de collaboration

La figure 4.7 propose un méta-modèle définissant la manière de décrire un patron de collaboration dans notre approche.

Un patron peut être composé des éléments suivants :

— *motivation* : *type texte*. Elle décrit l'objectif derrière l'utilisation du patron c'est-à-dire la raison pour laquelle le patron peut être utilisé.

— *contexte* : *type texte*. Le contexte décrit les éléments influencant le choix d'utilisation du patron.

— *solution*. Elle est composée de :

- *description* : *type texte*. La description permet de représenter textuellement le modèle d'exécution du patron avec les éléments qui le composent.
- *modèle d'exécution* : *type modèle ECPML*. Il illustre les éléments du processus composant le patron. Dans ce modèle, nous montrons un scénario représentatif avec des éléments génériques.

Chaque patron de collaboration est présenté par un *nom*, la *motivation* avec un *exemple ty-*

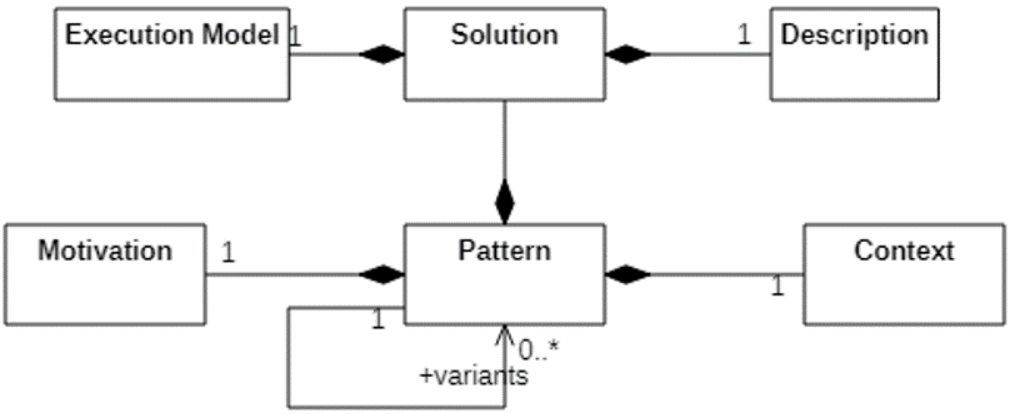


FIGURE 4.7: Méta-modèle pour la description d'un patron de collaboration.

pique d'utilisation, le *contexte d'utilisation*, la *solution* proposée comprenant la description des éléments du patron et d'éventuelles variantes *correspondant à des situations limitées*. La solution de chaque patron est décrite avec un *modèle d'exécution de la tâche collaborative*[1] composé de deux parties représentant la tâche collaborative lors de la modélisation et les instances de tâches associées durant l'exécution. Les variantes ne sont représentées qu'en phase d'exécution.

4.3.2 Patron "Refinement"

Nom : *Refinement*

Motivation : Ce patron est utilisé lorsque les acteurs de la tâche collaborative veulent créer un nouveau produit ou modifier un produit existant par affinage successif. Le raffinement est souvent adopté parce que la création du produit, dans sa totalité, demande des expertises, des compétences de différents acteurs. Le raffinement suppose que les manipulations (créations, améliorations, modifications, ...) d'un acteur lors de l'exécution d'une instance de tâche donnée impactent l'exécution de l'instance de tâche suivante par un autre acteur. Lors d'un *développement logiciel* par exemple, nous pouvons considérer la tâche collaborative *refactoring* réalisée par deux *développeurs*. Une instance de cette tâche est exécutée tout d'abord par un premier développeur qui modifie les parties non-fonctionnelles du *code* en se basant sur le *document des exigences*. Ce code source est ensuite réutilisé par le deuxième développeur exécutant une seconde instance de *refactoring* pour produire une version améliorée du code toujours avec le même document d'entrée.

Contexte : Ce patron sert à produire un produit *non-composite* en tenant compte de l'ordre

1. Les modèles sont présentés avec la syntaxe concrète de notre langage ECPML.

dans lequel les contributions se font par les différents acteurs. Il peut être utilisé lorsque le produit en entrée est non-composite (cas 1) ou quand il est composite avec des dépendances au niveau de ses composants (cas 6). Lorsque les deux produits sont non-composites, l'ordre des contributions est fixé par l'intention des acteurs. Lorsque le produit en entrée est composite avec des dépendances, les dépendances imposent l'ordre d'exécution. Ce patron ne tient pas compte de la disponibilité en totalité des ressources au même moment.

Solution : Considérons une tâche collaborative T, exécutée par un rôle R, avec un paramètre d'entrée P_1 et un paramètre de sortie P_2, représentant tous les deux des produits non-composites (cas 1). Ce patron est utilisé pour exécuter consécutivement une série de n Single Task Instances t_i, $i \in [1; n]$ à l'intérieur de la Collaborative Task Instance T. La valeur de n est donnée lors de l'instanciation de la Collaborative Task Instance T. Le paramètre d'entrée P_1 est utilisé par chaque t_i. Le paramètre de sortie P_2 est produit d'abord par la première STI (t_{11}) avant d'être utilisé comme entrée et sortie par la STI suivante (t_{12}) pour son raffinement. On applique le même principe pour les STI qui pourraient succéder à t_{12}.

La figure 4.8 montre les modèles du patron Refinement respectivement dans les phases de modélisation puis d'exécution avec deux instances de tâches simples de la tâche collaborative T. Les acteurs A_1 et A_2 exécutent les instances de tâche t_{11} et t_{12}, l'une à la suite de l'autre, d'où le séquencement *FinishToStart (FS)*. L'instance de produit p_1 est utilisée en entrée des deux instances de tâche. L'instance de produit p_2 est créée durant l'exécution de t_{11} et est utilisée en entrée de t_{12} pour être raffinée.

Variantes : Aucune. Dans une situation limitée, l'utilisation de ce patron n'est pas impactée. En effet si les acteurs ne sont pas tous disponibles en même temps, l'exécution séquentielle s'impose. De même si le produit en entrée est composite avec des dépendances, l'ordre d'exécution des STIs devient contraint mais l'exécution reste toujours séquentielle.

4.3.3 Patron "Free Co-work"

Nom : *Free Co-work*

Motivation : Ce patron est utilisé lorsque les acteurs veulent créer un produit à plusieurs sans être contraints par l'ordre d'exécution des instances de tâche. Il s'agit d'une contribution de plusieurs individus sur un même produit. Par exemple, lorsqu'il faut évaluer une proposi-

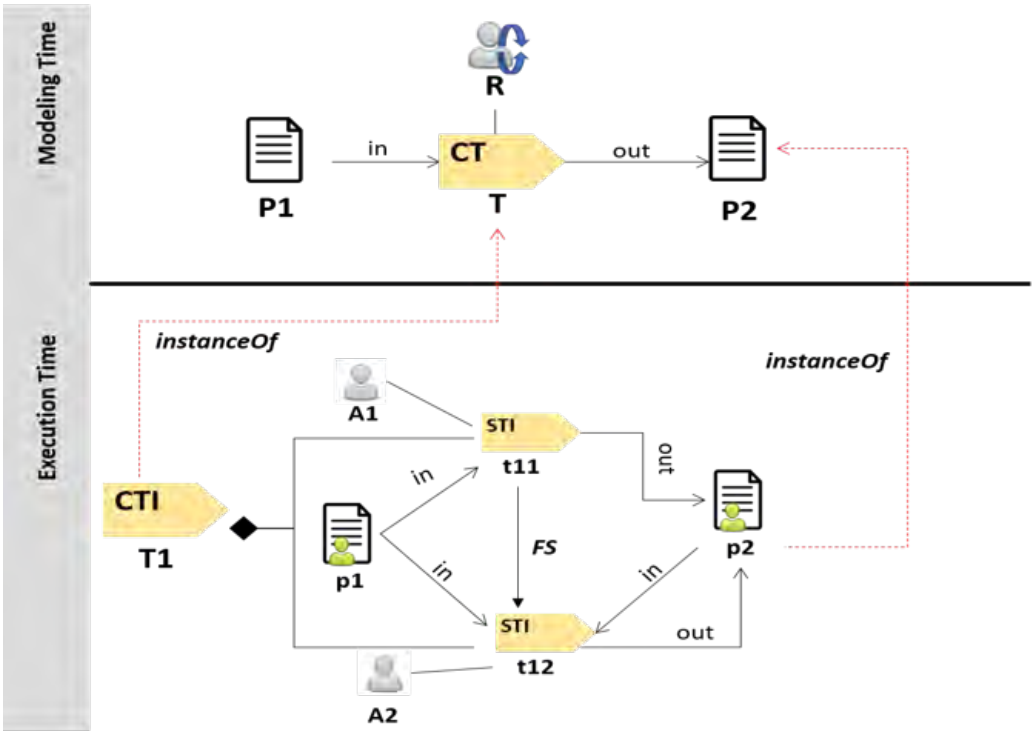


FIGURE 4.8: Patron de collaboration "refinement" pour une tâche collaborative T avec deux instances.

tion à plusieurs en créant un fichier d'annotations partagé, chaque participant peut rédiger son évaluation en même temps que les autres.

Contexte : Ce patron sert à élaborer un produit *non-composite* sans tenir compte de l'ordre dans lequel les contributions sont faites par les différents acteurs. Il est utilisé généralement lorsque le produit en entrée est non-composite (cas 1) ou lorsqu'il est composite sans dépendances au niveau de ses composants (cas 5). Les acteurs doivent être disponibles en totalité.

Solution : Considérons une tâche collaborative T, exécutée par un rôle R, avec un paramètre d'entrée P_1 et un paramètre de sortie P_2, représentant tous les deux des produits non-composites (cas 1). Ce patron est utilisé pour exécuter en parallèle une série de n *Single Task Instances* t_i, $i \in [1; n]$ à l'intérieur de la *Collaborative Task Instance* T. La valeur de n est donnée lors de l'instanciation de la Collaborative Task Instance T. Le paramètre d'entrée P_1 est utilisé par chaque t_i. Le paramètre de sortie P_2 est produit simultanément par l'exécution des différentes STI.

La figure 4.9 montre les modèles du patron *Free Co-work* respectivement dans les phases de modélisation puis d'exécution avec deux instances de tâches simples de la tâche collaborative T. Les acteurs A_1 et A_2 exécutent les instances de tâche t_{11} et t_{12} sans contrainte d'ordre.

L'instance de produit p_1 est utilisée en entrée des deux instances de tâche.

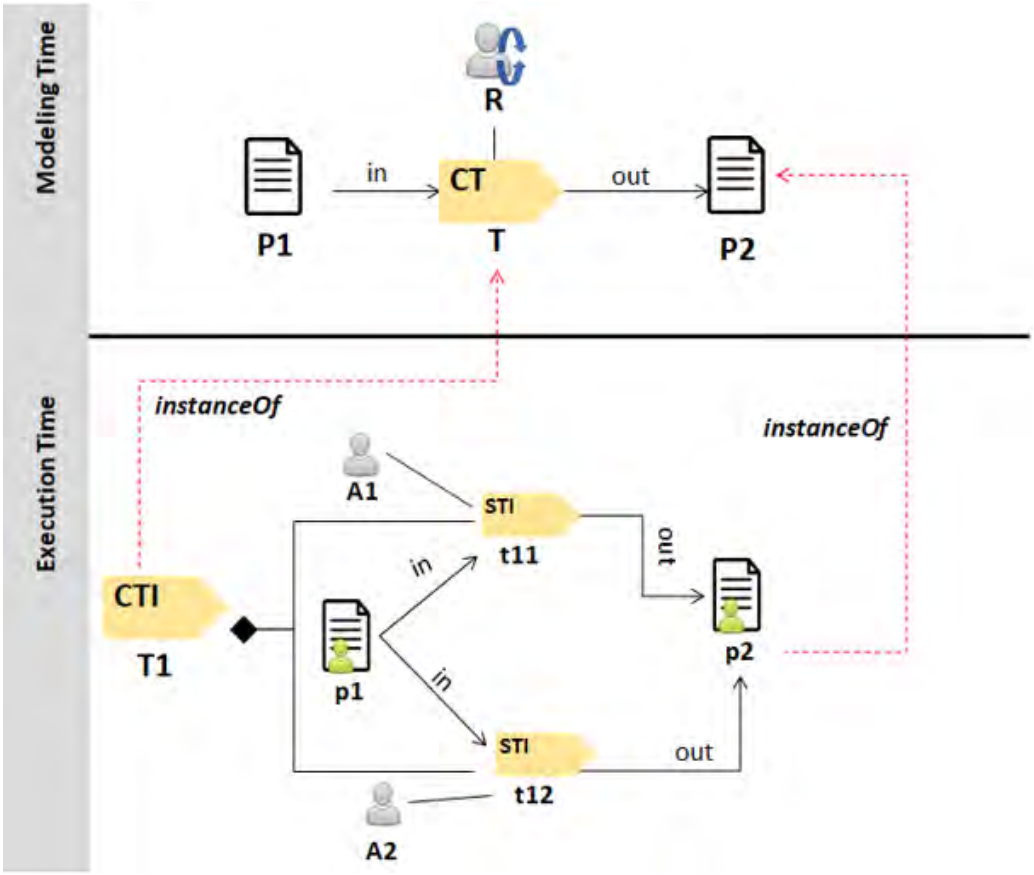


FIGURE 4.9: Patron de collaboration "free co-work" pour une tâche collaborative T avec deux instances.

Variantes : **"Co-work with limited resource"** : Dans une situation limitée, par exemple en l'absence de certains acteurs au démarrage de la tâche, on peut exécuter dès le début un certain nombre d'instances possibles, puis exécuter le reste des instances dès qu'une ressource devient disponible. La figure 4.10 décrit cette variante. Dans cette figure, nous avons deux tâches, t_{11} et t_{12} qui s'exécutent en parallèle. Dès que l'une d'entre elles finit son exécution, la tâche t_{13} peut commencer son exécution sans attendre l'autre tâche. Cette exclusion est symbolisée par le *"XOR"*.

4.3.4 Patron "Constraint Co-work"

Nom : *Constraint Co-work*

Motivation : Tout comme le patron "free co-work", ce patron est utilisé lorsque les acteurs veulent créer un produit à plusieurs mais avec une contrainte sur le début des tâches ou leur fin.

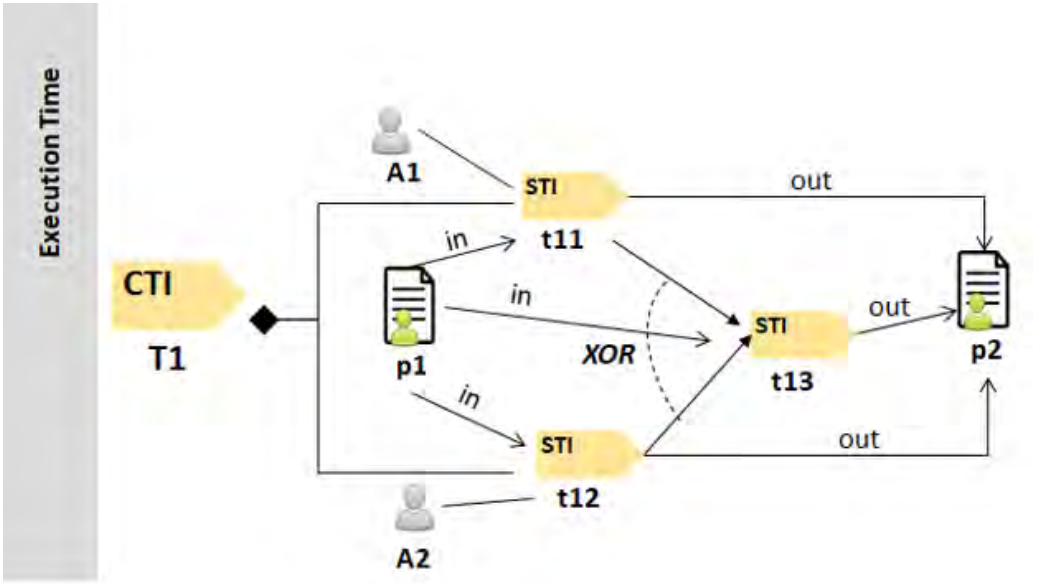


FIGURE 4.10: Variante du free co-work en situation limitée.

Par exemple, lors d'une validation d'un document à plusieurs, il peut arriver que des acteurs doivent attendre qu'une des entités participantes finisse sa validation pour finir la leur.

Contexte : Ce patron sert à élaborer un produit *non-composite* avec un ordonnancement sur les contributions faites par les différents acteurs. Il est utilisé généralement lorsque le produit en entrée est également non-composite (cas 1). Il peut aussi être utilisé lorsque le produit en entrée est composite sans dépendances au niveau des composants (cas 5).

Solution : Considérons une tâche collaborative T, exécutée par un rôle R, avec un paramètre d'entrée P_1 et un paramètre de sortie P_2, représentant tous les deux des produits non-composites (cas 1). Ce patron est utilisé pour exécuter une série de n *Single Task Instances* t_i, $i \in [1; n]$ à l'intérieur de la *Collaborative Task Instance* T. Le séquencement entre les instances est soit SS ("StartToStart"), soit FF ("FinishToFinish"), soit l'union des deux $SS\&FF$. SS indique que la seconde tâche ne peut démarrer son exécution que lorsque la première a démarré. FF indique que la seconde tâche ne peut finir son exécution que lorsque la première a fini la sienne. La valeur de n est donnée lors de l'instanciation de T. Le paramètre d'entrée P_1 est utilisé par chaque t_i. Le paramètre de sortie P_2 est produit simultanément par l'exécution des différentes t_i.

La figure 4.11 montre les modèles du patron *Constraint Co-work* respectivement dans les phases de modélisation puis d'exécution avec deux instances de tâches simples de la tâche collaborative T. Les acteurs A_1 et A_2 exécutent les instances de tâche t_{11} et t_{12} avec la contrainte de séquencement FF entre t_{11} et t_{12}. L'instance de produit p_1 est utilisée en entrée des deux

instances de tâche.

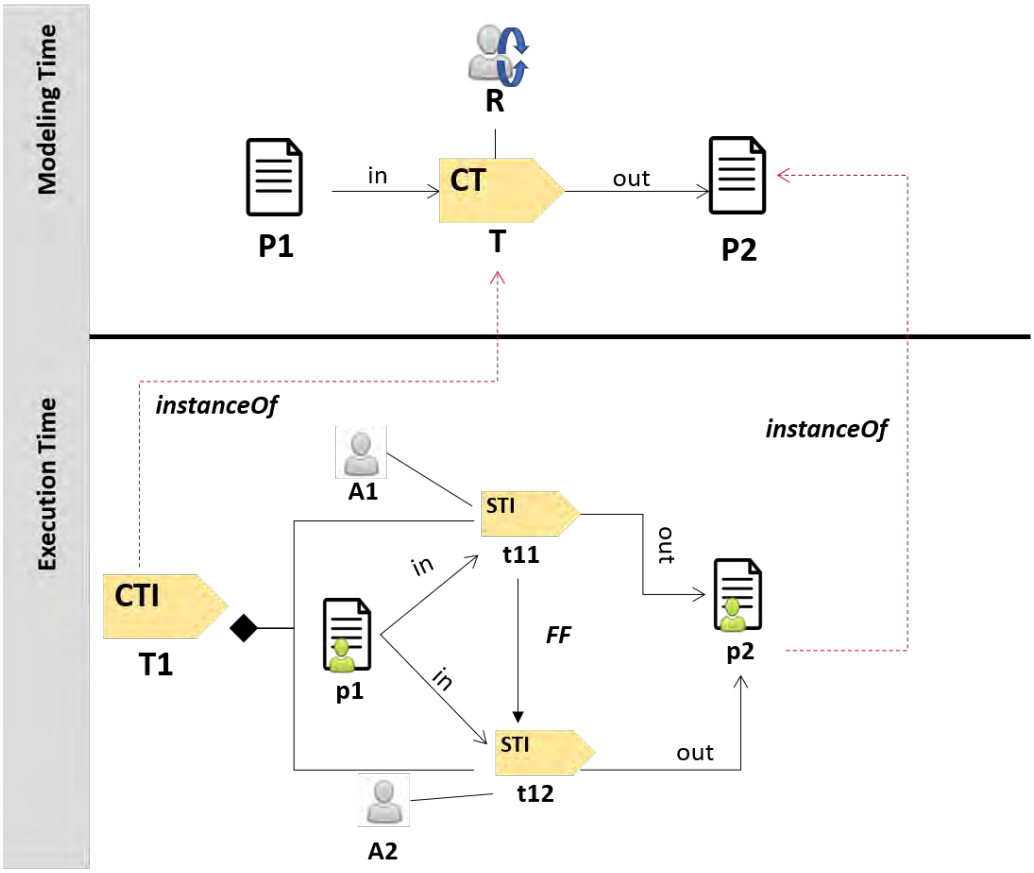


FIGURE 4.11: Patron de collaboration "constraint co-work" pour une tâche collaborative T avec deux instances.

4.3.5 Patron "Effort Division"

Nom : *Effort Division*

Motivation : Ce patron est utilisé pour diviser l'effort dans la production des différents composants d'un produit de sortie composite. Il est recommandé en l'absence de dépendance entre les composants. En prenant l'exemple de la tâche collaborative *"Review Document"*, les acteurs jouant le rôle *"Reviewer"* font leur relecture du document soumis par l'auteur dans un ordre quelconque pour fournir une *relecture globale* qui sera soumise à l'auteur.

Contexte : Pour pouvoir appliquer ce patron, le produit en sortie doit être composite sans dépendances au niveau des composants. Il peut être utilisé lorsque les ressources nécessaires pour l'exécution de la tâche collaborative sont disponibles en même temps. Le produit en entrée peut être composite avec dépendances (cas 10) ou sans (cas 7), ou non-composite (cas 2).

Solution : Etant donné une tâche collaborative T ayant un paramère d'entrée P_1 non composite (cas 2) et un paramètre de sortie P_2 composé de n composants indépendants $P2_i$, ce patron est utilisé pour exécuter un ensemble de n *Single Task Instances* t_i à l'intérieur de la Collaborative Task Instance de T de manière indépendante. Chaque instance de tâche t_i produit séparément une instance p_{2i} du composant P_{2i}. Les différentes instances de tâche peuvent être exécutées sans contraintes d'ordonnancement étant donné qu'il n'y a pas de séquencement entre elles. Elles doivent être exécutées en parallèle si l'on cherche à minimiser le temps d'exécution.

La Figure 4.12 montre le modèle du patron *"Effort division"* respectivement aux phases de modélisation et d'exécution avec deux instances de tâches de la tâche collaborative T. Dans cette figure, les acteurs A_1 et A_2 exécutent les instances de tâches t_{11} et t_{12} en parallèle. Chaque instance de tâche manipule un composant du produit composite p_2.

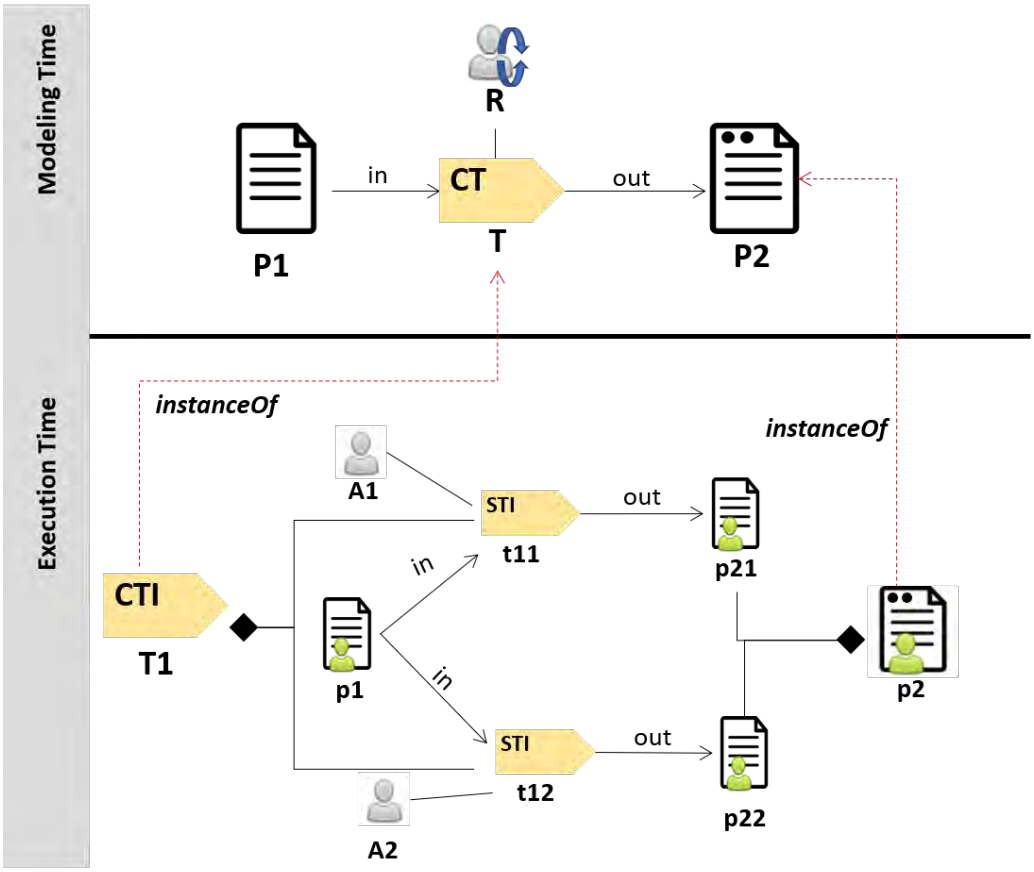


FIGURE 4.12: Patron de collaboration "effort division" pour une tâche collaborative T avec deux instances.

Variantes : **"Effort division with limited resource"** : Dans une situation limitée, par exemple en l'absence de certains acteurs au démarrage, on exécute le nombre d'instances pos-

sible et le reste des instances est exécuté dès qu'une ressource devient disponible. La figure 4.13 décrit une variante sur la base de trois tâches t_{11}, t_{12} et t_{13}. La tâche t_{13} s'exécute dès qu'une des deux autres instances finit de s'exécuter (via un "ou" exclusif) et produit la composante p_{23} du produit p_2.

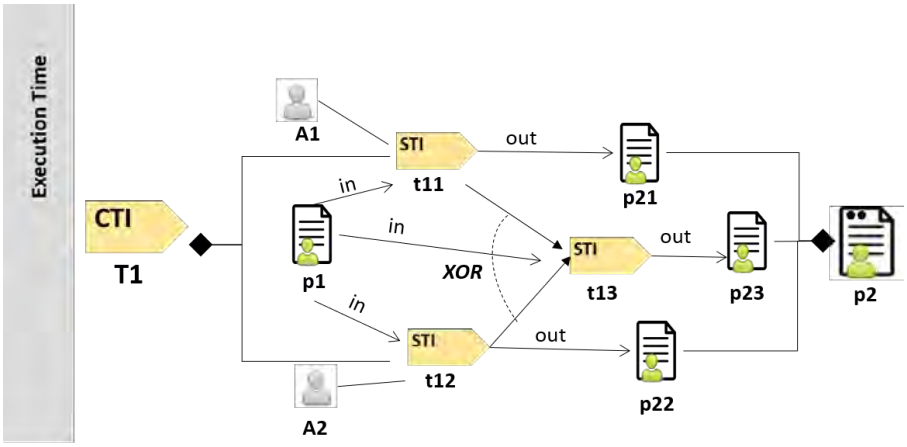


FIGURE 4.13: Variante du patron effort division en situation limitée.

4.3.6 Patron "Full Ordered Co-work"

Nom : *Full Ordered Co-work*

Motivation : Ce patron sert à la production de différents composants d'un produit en sortie, dans un ordre séquentiel imposé par les dépendances entre ses composants. Cela peut se produire lorsqu'on doit répartir le travail de production de l'artefact composite entre plusieurs acteurs en fonction de leur disponibilité. Par exemple, lors de la réalisation d'un document de conception, deux analystes pourront travailler en séquentiel pour produire l'un le diagramme de cas d'utilisation et l'autre le diagramme de séquence étant donné la dépendance entre ces deux diagrammes.

Contexte : Pour l'application de ce patron, le produit en sortie doit être composite et constitué de parties dépendantes. La structure du produit en entrée n'intervient pas dans le contexte d'exécution de ce patron. Cela veut dire qu'il peut être non composite (cas 3) ou composite avec dépendances (cas 9) ou sans dépendances (cas 8).

Solution : Etant donné une tâche collaborative T ayant un paramètre d'entrée P_1 non-composite et un paramètre de sortie P_2 composé de n composants dépendants P_{2i}, $i \in [1; n]$ (cas 3), ce patron est utilisé pour exécuter consécutivement n *Single Task Instances* t_i, $i \in [1; n]$

à l'intérieur d'une *Collaborative Task Instance* T. Chaque instance de tâche t_i manipule une instance p_{2i} du composant P_{2i} et est exécutée par un acteur différent jouant le rôle R. L'ordre d'exécution $FS(FinishToStart)$ parmi les instances de tâche est imposé par les dépendances entre les composants de $P2$: la création de P_{2i+1} a besoin de P_{2i} ; ainsi t_{i+1} (qui manipule P_{2i+1}) doit succéder à t_i (qui produit P_{2i}). La valeur de n est donnée par le chef de projet quand la tâche collaborative est déployée.

La figure 4.14 montre le modèle du patron *"Full Ordered Co-work"* respectivement aux phases de modélisation et d'exécution avec deux instances de la tâche collaborative T. Dans cette figure, les acteurs A_1 et A_2 exécutent les tâches t_{11} et t_{12} en séquentiel. Chaque instance manipule un composant du produit de sortie, avec une dépendance entre les deux composants.

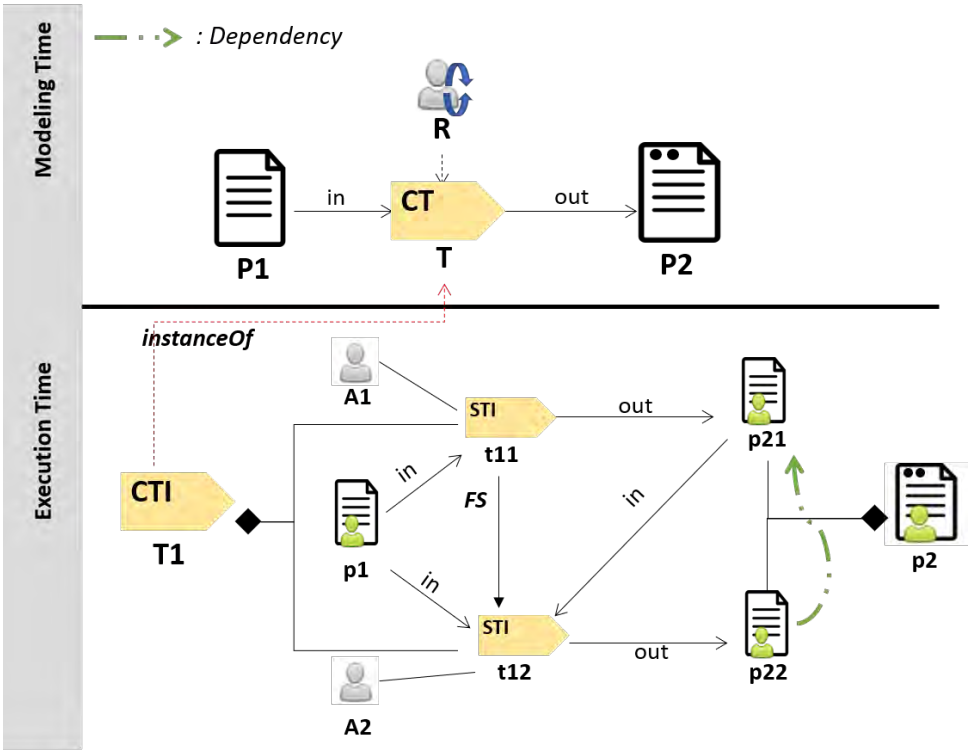


FIGURE 4.14: Patron de collaboration "full ordered co-work" pour une tâche collaborative T avec deux instances.

Variantes : Aucune. Etant donné l'exécution séquentielle, le manque de ressources n'impacte pas la stratégie d'exécution.

4.3.7 Patron "Partial Ordered Co-work"

Nom : *Partial Ordered Co-work*

Motivation : Ce patron sert à la production des différents composants d'un produit de sortie composite dans un ordre séquentiel déterminé par des dépendances partielles entre certains composants de ce produit. C'est un patron hybride qui prend en compte le mode "Effort Division" pour les composants sans dépendances et le mode "Full Ordered Co-work" pour les composants avec dépendances. Pour la réalisation du même document de conception que précédemment, un troisième acteur peut travailler indépendamment sur un diagramme de déploiement qui ne dépend pas des deux autres diagrammes.

Contexte : L'application de ce patron nécessite que le produit en sortie soit composite et constitué de parties partiellement dépendantes. Le produit en entrée peut être non composite (cas 4) ou composite sans dépendances (cas 8).

Solution : Etant donné une tâche collaborative T ayant un paramètre d'entrée P_1 non-composite et un paramètre de sortie P_{2i} composé de n composants partiellement dépendants P_i, $i \in [1; n]$ (cas 4), ce patron est utilisé pour exécuter consécutivement n *Single Task Instances* t_i, $i \in [1; n]$ à l'intérieur d'une *Collaborative Task Instance* T. Chaque instance de tâche t_i manipule une instance p_{2i} du composant P_{2i} et est exécutée par un acteur différent jouant le même rôle. L'ordre d'exécution FS parmi les instances de tâches n'est imposé qu'entre les composants de $P2$ dépendants. Les instances de tâche manipulant les composants sans dépendances (t_{13} dans cet exemple) sont exécutées indépendamment du reste (t_{11} et t_{12} dans cet exemple).

La figure 4.15 montre le modèle du patron respectivement aux phases de modélisation et d'exécution avec trois instances de tâches de la tâche collaborative T. Dans cette figure, les acteurs A_1 et A_2 exécutent les instances de tâche t_{11} et t_{12} en séquentiel en raison des dépendances entre les produits manipulés. L'instance de tâche t_{13} est exécutée par l'acteur A_3 soit avant les deux autres tâches soit après.

Variantes : **"Partial ordered co-work with limited resource"** : Lorsque le nombre de ressources est limité, les instances de tâche manipulant les composants sans dépendances (t_{13} dans cet exemple) pourront être exécutées après ou avant les instances de tâche manipulant les composants avec dépendances (t_{11} et t_{12} dans cet exemple). La figure 4.16 illustre cette situation. Sur cette figure, nous montrons trois instances de tâches exécutées en séquentiel. Ici, t_{13} est exécutée après les autres instances. Une autre stratégie d'exécution serait l'exécution de

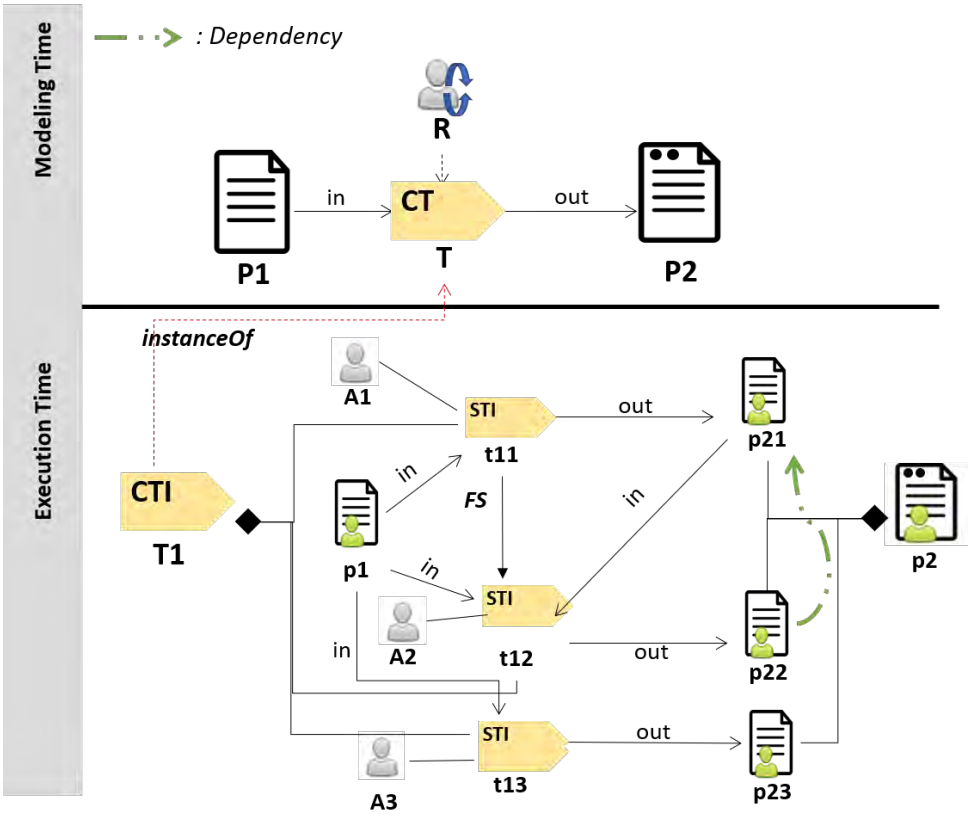


FIGURE 4.15: Patron de collaboration "partial ordered co-work" pour une tâche collaborative T avec deux instances.

t_{13} et t_{11} en parallèle, puis l'exécution de t_{12} lorsque t_{11} se termine étant donné la dépendance entre les produits qu'elles manipulent.

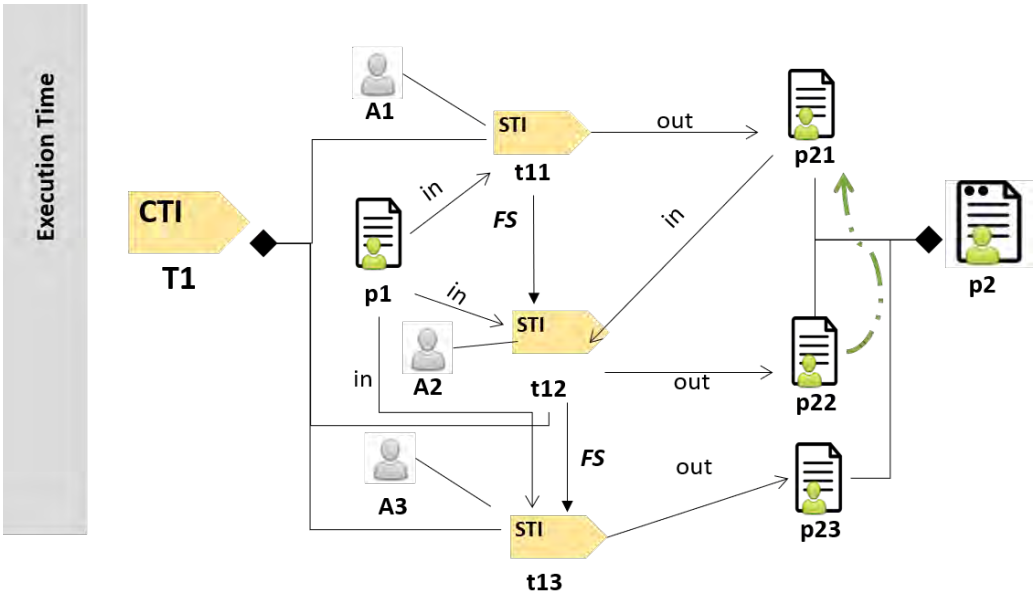


FIGURE 4.16: Variante du patron partial ordered co-work en situation limitée.

En guise de récapitulatif de la définition des patrons de collaboration, nous présentons ci-dessous le tableau 4.17, qui reprend les stratégies présentées à la section 4.2 sous forme de "cas", avec les patrons de collaboration applicables.

Patrons / Cas	Refinement	Free Co-work	Constraint Co-Work	Effort Division	Full Ordered Co-Work	Partial Ordered Co-Work
Cas 1	+	+	+			
Cas 2				+		
Cas 3					+	
Cas 4						+
Cas 5		+				
Cas 6	+					
Cas 7				+		
Cas 8					+	
Cas 9					+	
Cas 10				+		

FIGURE 4.17: Matrice de représentation des cas avec les patrons applicables.

4.4 Utilisation des patrons de collaboration

Dans cette section, nous présentons et illustrons le principe d'application des patrons de collaboration pour rendre l'exécution flexible.

4.4.1 Principe

Les patrons proposés permettent de décrire dynamiquement les relations relations entre les STIs au sein d'une tâche collaborative. Ces relations dépendent de la stratégie adoptée lors du choix d'un patron de collaboration. L'application d'un patron de collaboration à une *"Collaborative Task Instance"* peut être effectuée :

— *lors de la création de l'instance de la tâche collaborative*,

— *lors de son exécution* pour modifier la stratégie de collaboration utilisée pour la réalisation de la tâche,
— *lors de l'évolution* d'une "Single Task Instance" afin de la rendre collaborative.

Lors de l'application d'un patron, les relations entre les instances de tâches définies dans le patron de collaboration choisi sont reproduites en instances de tâches réelles de la tâche collaborative concernée. Les différentes relations à appliquer sont celles définies par le *flux de données* puis celles définies par le *flux de contrôle*. L'implémentation de l'application d'un patron est détaillée au niveau du chapitre 5.

4.4.2 Application à l'exemple fil conducteur "Writing and Reviewing a Document"

Dans cette section, nous montrons comment les patrons de collaboration peuvent être appliqués pendant la mise en oeuvre de processus collaboratifs. Reprenant l'exemple du processus fil conducteur illustré au chapitre 3, nous montrons comment ce processus peut être instancié en utilisant un sous-ensemble des patrons de collaboration définis dans la section précédente.

Nous présentons aussi la transformation d'une instance de tâche (STI) en une instance de tâche collaborative (CTI). En effet, cette évolution peut être judicieuse dans certains contextes, comme détaillé dans la sous-section 4.4.2.2.

4.4.2.1 Application de patrons de collaboration à l'exemple de processus "Writing and Reviewing a document"

Pour rappel ce processus est composé des trois tâches : *"Write Document"*, *"Review Document"* et *"Modify Document"* (voir figure 4.18). La tâche *"Review Document"* est réalisée par le rôle *Reviewer* alors que les tâches *Write Document* et *Modify Document* sont réalisées par le rôle *Author*.

Durant ce processus, l'auteur réalise un document (*Manuscript*) qui est utilisé par les relecteurs. Un relecteur émet des annotations dans un autre document (Assessment) qui sera utilisé par l'auteur pour améliorer son document. Les produits manipulés au long de ce processus sont le document *manuscript* et le document *assessment* (qui contient les évaluations des relecteurs).

Pour l'application d'un patron nous allons nous concentrer sur la tâche collaborative *Review Document* qui est faite par un ensemble de *Reviewers*.

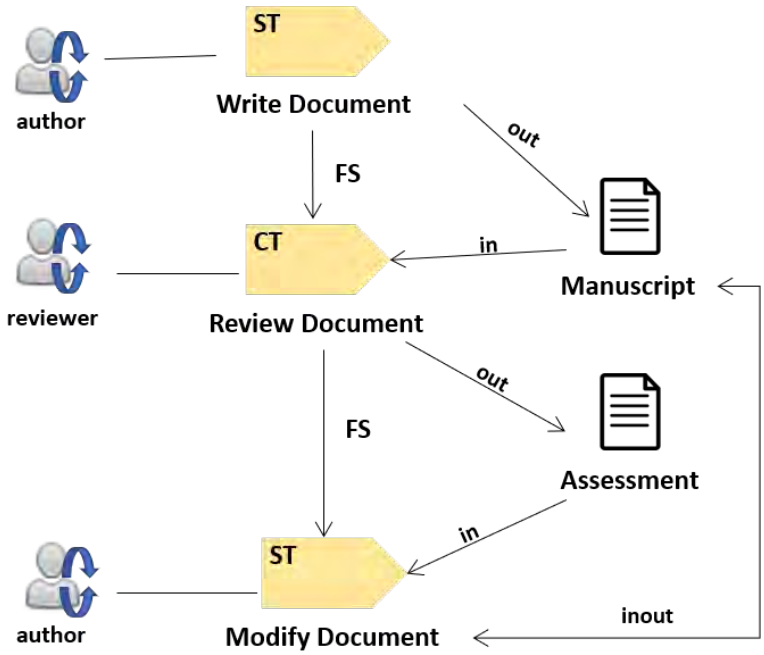


FIGURE 4.18: Processus *Writing and Reviewing a Document*.

Considérons un contexte où les trois acteurs *Alice*, *Bob*, *Eve* jouant le rôle de reviewer sont tous disponibles au début de l'exécution de la tâche collaborative Review Document. L'exécution pourrait se faire en utilisant un patron proposant une stratégie d'exécution en parallèle afin de privilégier la rapidité d'exécution.

Supposons maintenant que le produit en sortie, *Assessment*, soit composite sans dépendances. Avec ce nouvel élément de contexte, nous pouvons utiliser le patron *"effort division"* comme vu dans la section 4.3 plus haut. De ce fait chaque relecteur peut produire une partie du document assessment final sans impacter le travail des autres relecteurs. Ce patron "Effort division" implique que les différentes instances de la tâche *Review Document* soient exécutées en parallèle. La figure 4.19 montre le résultat obtenu après application du patron.

Dans la figure 4.19, les instances de tâches de Review Document peuvent s'exécuter selon le séquencement voulu par les acteurs ou choisi par le manager. Par exemple, *Review Document 2* de l'acteur *Bob* peut commencer à tout moment indépendamment des autres instances de tâche. Le produit final, *Assessment*, consiste en une composition des trois produits de sortie que sont *Alice Assessment*, *Bob Assessment* et *Eve Assessment*.

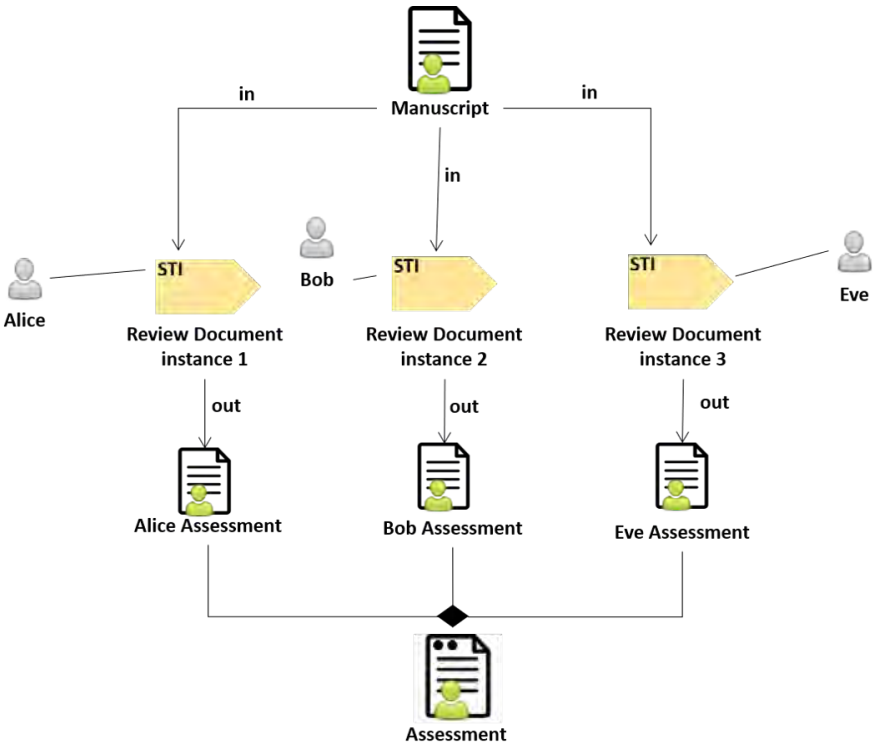


FIGURE 4.19: Modèle d'exécution en parallèle du processus avec un produit en sortie composite.

Différents cas imprévus peuvent survenir en cours d'exécution. Supposons par exemple que les trois *relecteurs* ne soient plus disponibles en même temps ou que l'on ait besoin du résultat de l'un avant de faire le reste (par exemple Bob attend le résultat du travail d'Alice, et Eve attend celui de Bob). Ce changement de contexte implique un changement de patron de collaboration. Dans ce cas précis, nous pourrons passer à une exécution séquentielle concernant les trois instances de tâche de *Review Document*. Le patron choisi dans ce contexte est le patron *"full ordered co-work"*, car le produit de sortie est composite avec des dépendances totales entre ses composants. Il implique que les instances de tâche soient exécutées de manière séquentielle.

La figure 4.20 illustre le résultat obtenu après le changement de patron.

Dans ce contexte d'exécution, *Review Document 3* démarre son exécution après la fin de l'exécution de *Review Document 2* et cette dernière démarre après la fin de *Review Document 1*. Chaque acteur fournit son évaluation (produit *Assessment*) qui sera utilisée au niveau de l'instance de tâche suivante ce qui justifie l'exécution séquentielle. Tout comme avec l'exécution parallèle, le produit final consiste en un regroupement des trois produits réalisés par les acteurs Alice, Bob et Eve.

Considérant toujours le fait que les trois acteurs ne sont pas disponibles au même moment,

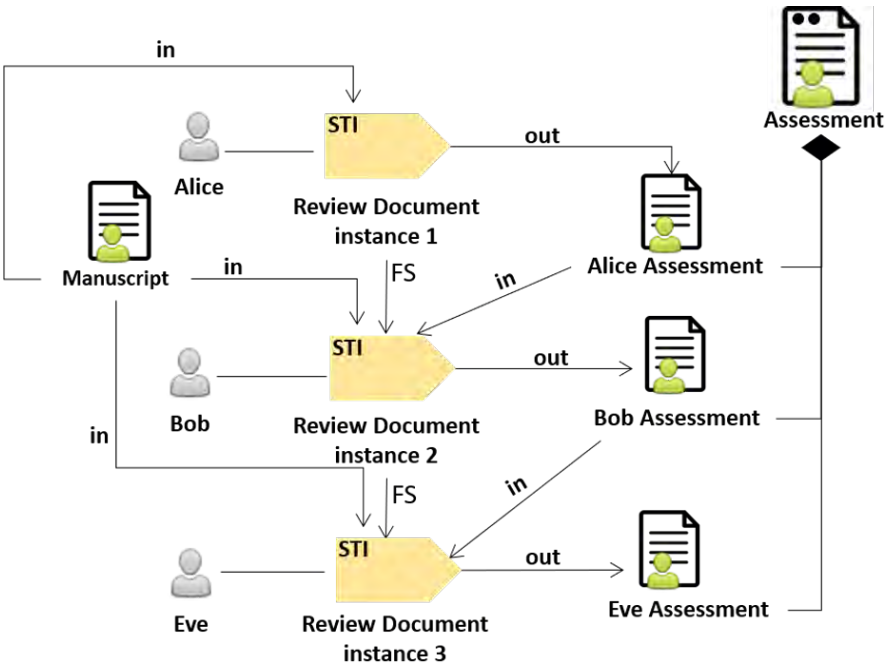


FIGURE 4.20: Modèle d'exécution séquentielle du processus avec un produit de sortie composite avec dépendances totales.

il serait intéressant de considérer le produit *Assessment* comme non composite. C'est possible dans le cas où il n'est pas indispensable de distinguer la contribution de chaque acteur dans le document d'évaluation final. Dans ce contexte, le patron *"free co-work"* pourrait être adapté à l'exécution de l'instance de tâche collaborative. Le résultat obtenu est illustré par la figure 4.21.

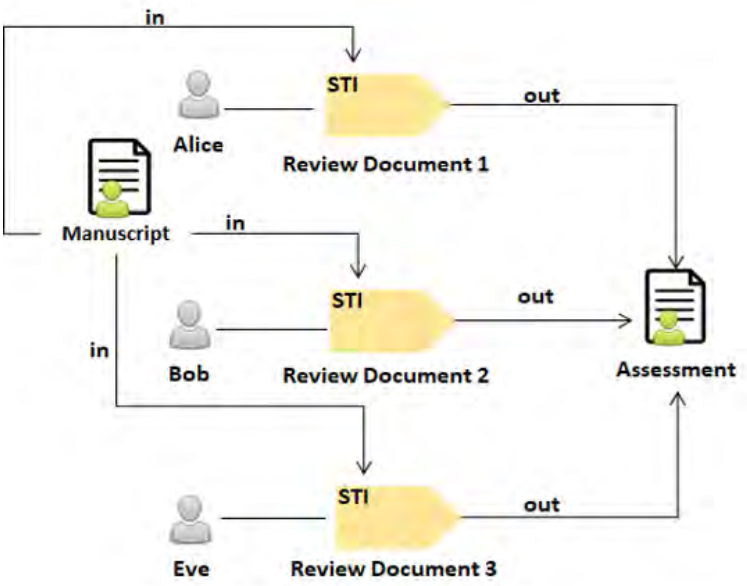


FIGURE 4.21: Application du patron "free co-work" pour le produit de sortie non-composite *Assessment*.

4.4.2.2 Evolution d'une instance de tâche (STI) en une instance de tâche collaborative (CTI)

Comme mentionné dans le chapitre précédent, il peut être intéressant qu'en cours d'exécution, une instance de tâche puisse devenir collaborative (par exemple pour intégrer la contribution d'un deuxième participant dans une tâche initialement prévue pour une personne). Cette fonctionnalité renforce la dynamicité de notre approche en permettant de s'adapter aux évolutions de contexte. Considérant l'exécution en séquence dans la figure 4.20, il peut arriver que si Eve manque de temps ou d'expertise sur certains aspects, on ait besoin de déléguer une partie de sa tâche à un quatrième acteur jouant le rôle de *reviewer* qui va, avec elle, analyser les évaluations des relecteurs précédents pour produire une évaluation finale. Dans ce cas l'instance de tâche *Review Document 3* pourrait devenir collaborative et donc nécessiter un patron de collaboration pour son déploiement et son exécution.

Après transformation de l'instance de tâche simple en instance de tâche collaborative, nous pouvons lui appliquer un patron qui devrait être *effort division* si le produit de sortie est composite sans dépendances. La figure 4.22 montre le résultat du déploiement. L'exécution des instances de tâche *Review Document 1* et *Review Document 2* se déroule toujours en séquentiel. Le produit *Eve Assessment* initialement prévu devient alors composite (*Eve_Charly Assessment*) avec une partie réalisée par l'acteur Eve (*Eve Assessment*) et une autre partie réalisée par l'acteur Charly (*Charly Assessment*). Les deux produits combinés forment le résultat de l'instance de la tâche collaborative *Review Document 3*, à savoir *Eve_Charly Assessment*.

Si le produit de sortie de cette nouvelle CTI est non composite, d'autres patrons pourraient être appliqués comme par exemple le patron *"Refinement"* avec une exécution séquentielle de *Review Document 3_1* par Charly suivie de *Review Document 3_2* par Eve.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un ensemble de patrons de collaboration utilisables lorsqu'il s'agit de déployer une tâche collaborative. L'intérêt de ces patrons est qu'ils permettent de définir à l'exécution la mise en oeuvre de la collaboration au sein des tâches définies statiquement lors de la conception du processus. Le choix du patron de collaboration à appliquer

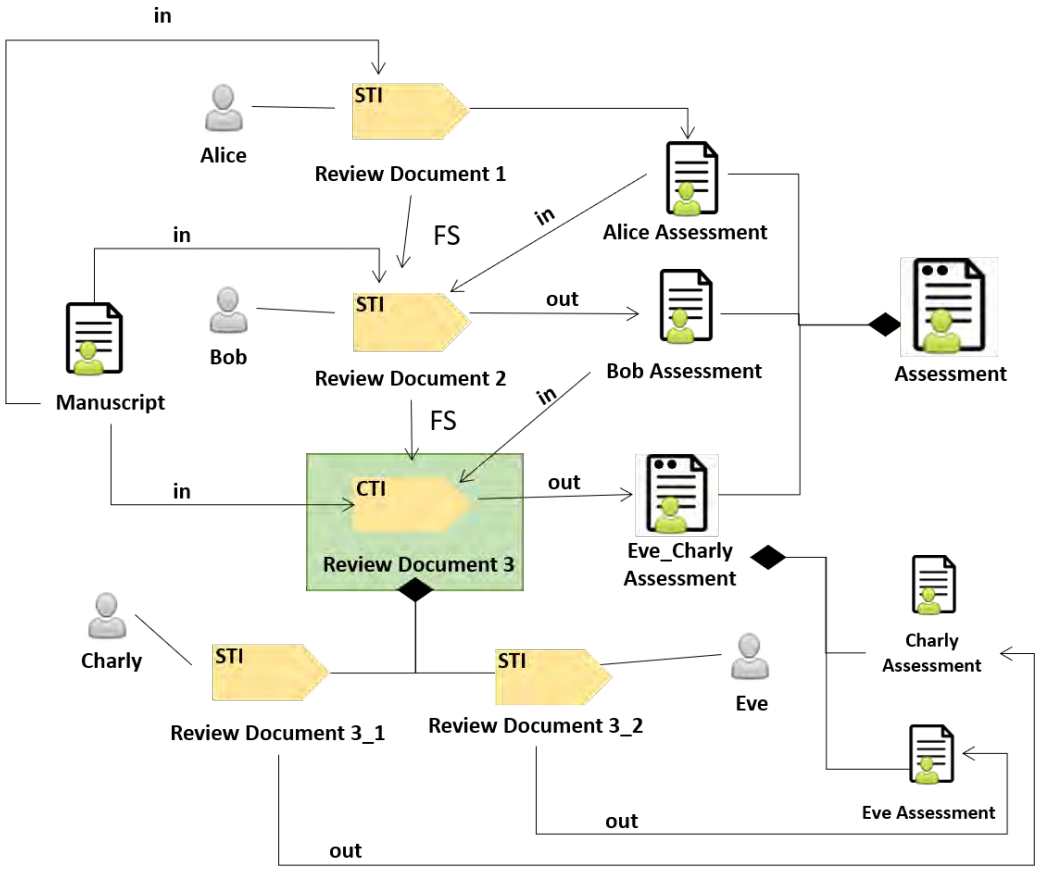


FIGURE 4.22: Illustration de l'évolution de la STI "Review Document 3" en CTI.

est adaptable en fonction du contexte du projet. Ce mécanisme rend ainsi flexible l'exécution des tâches collaboratives, ce qui satisfait un besoin clairement identifié par les gestionnaires de projets, notamment dans le développement du logiciel.

La définition de ces patrons n'a pas pour but d'être exhaustive. Cependant, les patrons présentés représentent les stratégies de collaboration les plus usuelles correspondant aux situations dites idéales (nominales). Selon le principe de classification des patrons vus plus haut, il serait aisé de définir d'autres patrons de collaboration en fonction du contexte (sous forme de variantes des patrons précédents).

Dans le chapitre suivant, nous présentons la sémantique du langage ECPML et en particulier des machines à états introduites dans le chapitre précédent pour décrire le comportement des éléments d'un processus (instance de tâche, instance de produit, et acteur). En outre, nous présentons l'algorithme d'application d'un patron en détaillant les différentes étapes jusqu'au déploiement de la tâche. L'application de cet algorithme est illustrée sur l'exemple du processus fil conducteur *"Writing and Reviewing a Document"*.

Chapitre 5

Sémantique opérationnelle d'un processus collaboratif

Sommaire

L'objectif de ce chapitre est de fournir une sémantique permettant de décrire les actions liées aux éléments du langage ECPML. La sémantique opérationnelle permet d'une part l'instanciation du processus, d'autre part de suivre et faire évoluer ce processus tout au long de son cycle de vie.

L'objectif majeur de la gestion de processus est de fournir aux utilisateurs un support et un contrôle de l'exécution du processus. Cela permet la coordination des tâches des acteurs en vue de suivre leur progression. Pour ce faire, à l'exécution, il faut que le modèle de processus

fournisse assez d'informations pour permettre de décider du démarrage ou de la fin d'une tâche. En d'autres termes, le langage de modélisation de processus doit définir, via une sémantique opérationnelle, le comportement à l'exécution de chaque élément de processus.

Dans la suite de ce chapitre, nous commençons d'abord par une description du principe de la sémantique opérationnelle (section 5.1). Dans la section 5.2, nous présentons la sémantique des éléments de base, c'est-à-dire instance de produit, acteur et instance de tâche. Pour chaque élément, nous reprenons la machine à états présentée au chapitre 3, et nous détaillons les transitions et les algorithmes associés. La section 5.3 présente la sémantique d'une instance de tâche collaborative. Dans la section 5.4, nous concluons ce chapitre avec un bilan sur l'utilisation des algorithmes présentés dans notre moteur de processus.

5.1 Principe de description de la sémantique opérationnelle des éléments du processus

Chaque élément du processus a une sémantique opérationnelle permettant de décrire son cycle de vie. La sémantique permet de définir le comportement de notre moteur de processus. Notre but est de fournir un moteur de processus flexible supportant l'exécution. Pour cela, il faut que la sémantique des éléments de processus soit aussi flexible. Cela nous a conduit à paramétriser la sémantique opérationnelle d'une tâche collaborative par un patron de collaboration (Cisse et al., 2019). En ce sens, nous assurons une flexibilité durant l'exécution d'une tâche collaborative. Ainsi, la sémantique de la tâche collaborative n'est pas figée mais intègre les relations issues d'un patron de collaboration.

Dans le chapitre 3, nous avons présenté une vue réduite des machines à état associées aux éléments de processus de notre langage ECPML, en focalisant la description sur les états du cycle de vie. La section suivante va décrire en détail ces machines à états. Pour rappel, nous considérons les éléments suivants : *"single task instance (STI)"*, *"collaborative task instance (CTI)"*, *"actor"* et *"work product instance (WPI)"* comme présenté sur la figure 5.1. Le passage entre deux états se fait par l'exécution des actions définies au niveau de la transition. Chaque transition est déclenchée par un événement. Cet événement survient suite à une action de l'utilisateur (mettre fin à une tâche par exemple) ou automatiquement après un signal venant

d'une action précédente.

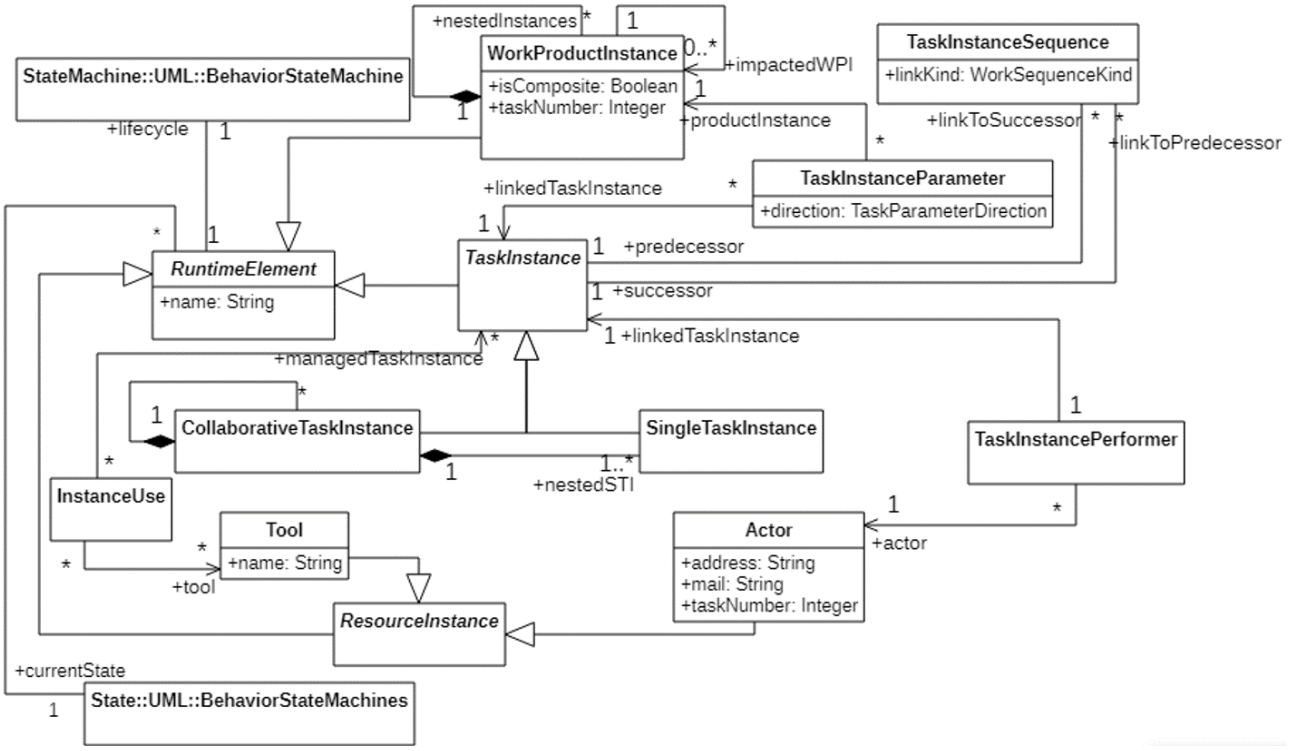


FIGURE 5.1: Paquetage *ECPML_Execution* du méta-modèle d'ECPML.

5.2 Sémantique opérationnelle des éléments de base d'un processus

Dans cette section, nous présentons le comportement des éléments de base du paquetage ECPML_Execution, à savoir : *l'instance de produit (WPI)*, *l'acteur (Actor)* et *l'instance de tâche simple (STI)*.

5.2.1 Comportement d'une instance de produit

Une instance de produit ("Work Product Instance" - WPI) passe par plusieurs états qui sont : *"Defined"*, *"InUse"* et *"Validated"*. Le comportement d'une WPI est relatif à la STI qui le manipule. Il évolue en même temps que le comportement de la STI.

5.2.1.1 Description des états

Les états d'une instance de produit sont définis comme présenté sur la figure 3.13 :

- *Defined* : un produit à l'état defined signifie qu'il y a une instance de produit dans le système et qu'il est référencé par une WPI.
- *InUse* : l'instance de produit est en train d'être utilisée par une ou plusieurs instances de tâche.
- *Validated* : un produit est à l'état validated quand son contenu est validé et stable.

5.2.1.2 Description des transitions

La figure 5.2 illustre la machine à états représentant le comportement d'une instance de produit.

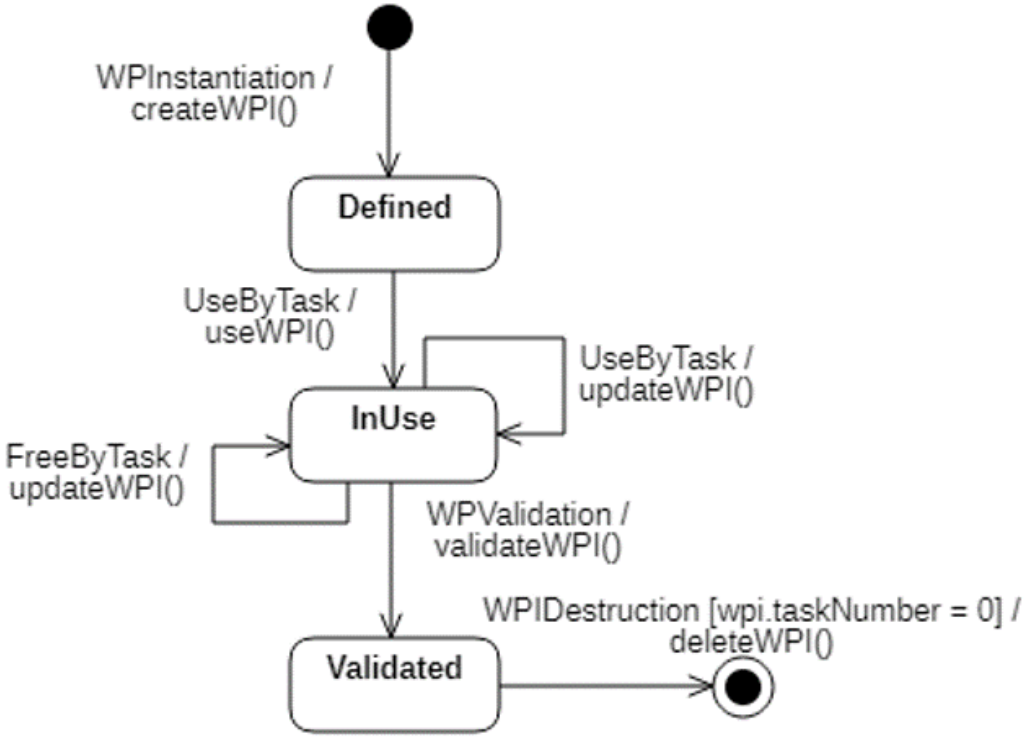


FIGURE 5.2: Comportement d'une instance de produit.

Chaque transition se fait après exécution d'une action déclenchée par un événement.

- Etat initial à l'état *"Defined"* : Lorsque l'événement *WPInstantiation* est déclenché, une WPI est créée dans la base des instances de notre système de gestion de processus. Cette création est faite par l'action *createWPI()* et référence un produit physique (de type WP donné) stocké dans le système de gestion de données sous forme de noeud. L'algorithme 1 décrit ce processus.

Algorithm 1: createWPI(). Création d'une Work Product Instance wpi.

```
Input: WorkProduct wp;
Output: WorkProductInstance wpi;
1 begin
2 |  wpi = createWPINode(wp);
3 |  wpi.currentState = "Defined";
4 end
```

— Etat *"Defined"* à l'état *"InUse"* : Lorsque l'événement *UseByTask* est déclenché par une des instances de tâche qui manipulent la WPI, elle passe à l'état *InUse*. L'action correspondante, *useWPI()*, (algorithme 2) permet un changement d'état de l'instance de produit et une incrémentation du nombre d'instances de tâche manipulant la WPI.

Algorithm 2: useWPI(). Utilisation d'une WPI.

```
Input: WorkProductInstance wpi;
1 begin
2 |  wpi.taskNumber++;
3 |  wpi.currentState = "InUse";
4 end
```

— Etat *"InUse"* à l'état *"InUse"* : Lorsque l'événement *UseByTask* ou *FreeByTask* est déclenché, l'état de la WPI ne change pas mais le nombre d'instances de tâche manipulant la wpi change. Cela permet à une WPI d'être utilisée par plusieurs instances de tâche en même temps. L'algorithme 3 décrit l'action *updateWPI()* qui se déroule pour ces deux événements.

Algorithm 3: updateWPI(). Mise à jour d'une WPI.

```
Input: WorkProductInstance wpi, Event e;
1  begin
2  |  switch e do
3  |  |  case UseByTask
4  |  |  |  wpi.taskNumber++;
5  |  |  end
6  |  |  case UseByTask
7  |  |  |  wpi.taskNumber--;
8  |  |  end
9  |  end
10 end
```

— Etat *"InUse"* à l'état *"Validated"* : Lorsque l'événement *WPIValidation* est déclenché par une des instances de tâche qui manipulent la WPI, elle passe à l'état *Validated* qui

marque sa stabilité. L'action *validateWPI()* (algorithme 4) permet le changement d'état de la WPI.

Algorithm 4: validateWPI(). Validation d'une WPI.

```
   Input: WorkProductInstance wpi ;
1  begin
2  |  wpi.currentState = "Validated" ;
3  end
```

— Etat *"Validated"* à l'état final : L'événement *WPIDestruction* déclenche la suppression d'un noeud WPI à condition qu'elle ne soit pas en train d'être utilisée par des instances de tâche. L'action *deleteWPI()* (algorithme 5) permet de supprimer le noeud avec la condition $wpi.taskNumber = 0$.

Algorithm 5: deleteWPI(). Suppression d'un noeud WPI.

```
   Input: WorkProductInstance wpi ;
1  begin
2  |  deleteWPINode(wpi) ;
3  end
```

5.2.2 Comportement d'un acteur

Le concept *Actor* représente la ressource humaine chargée de l'exécution d'une instance de tâche simple. L'acteur d'une tâche peut passer par plusieurs états : *"Free"*, *"Assigned"* et *"Performing"*. Nous détaillons dans la suite le comportement d'un acteur.

5.2.2.1 Description des états

Les états d'un acteur sont définis comme présenté sur la figure 3.12 :

— *Free* : un acteur, représenté par un noeud Actor, existe et est géré dans le système. Dans l'état *"Free"*, il n'a aucune instance de tâche à exécuter.

— *Assigned* : l'acteur est assigné à au moins une instance de tâche.

— *Performing* : l'acteur est en train d'exécuter au moins une instance de tâche.

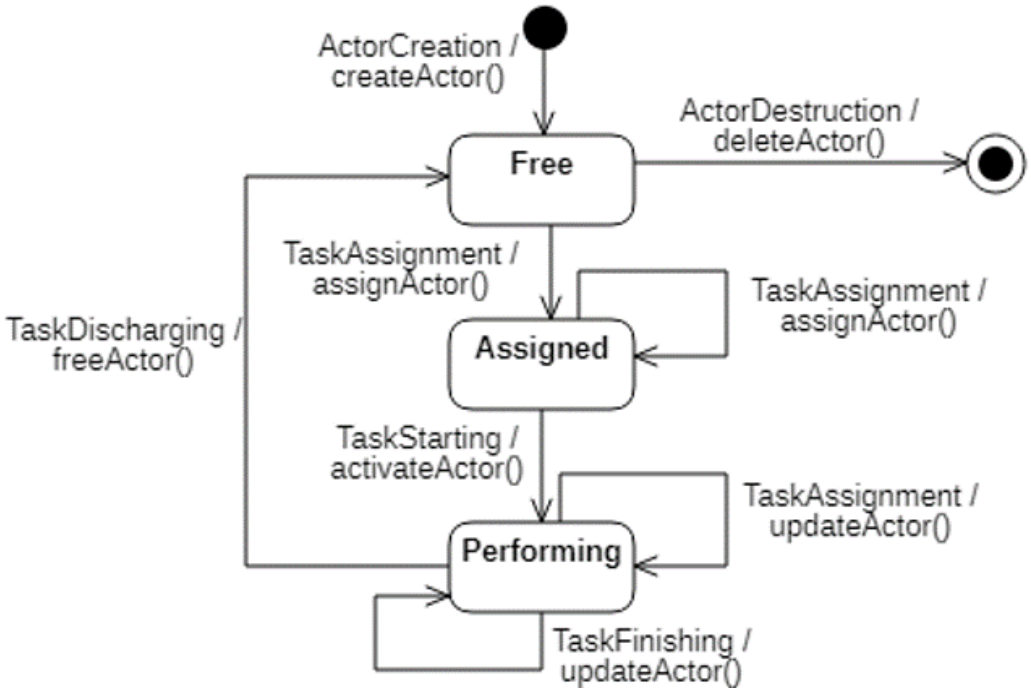


FIGURE 5.3: Comportement d'un acteur.

5.2.2.2 Description des transitions

Les transitions du comportement d'un acteur sont illustrées dans la figure 5.3. Dans la suite, nous détaillons les différentes transitions qui permettent de dérouler le cycle de vie d'un acteur.

— Etat initial à l'état *"Free"* : Lorsque l'événement *ActorCreation* est déclenché, un acteur géré dans le système de gestion des ressources est créé et représenté par un noeud a de type *Actor*. Ce noeud est créé dans la base des instances de notre système de gestion de processus. L'action *createActor()* doit associer les rôles de l'acteur au noeud a. La procédure *createQualification()* est décrite dans l'annexe D. L'algorithme 6 décrit ce processus.

Algorithm 6: createActor(). Création d'un noeud acteur.

```
   Input: Role[] rolelist;
1  begin
2  |  Actor a = createActorNode();
3  |  for i = 1 to rolelist.length() do
4  |  |  createQualification(a, rolelist[i);
5  |  end
6  |  a.currentState = "Free";
7  end
```

— Etat *"Free* à l'état *"Assigned"* : Lorsque l'événement *TaskAssignment* est déclenché

par une instance de tâche, l'acteur est mis à l'état *Assigned*. Le lien entre l'instance de tâche déclenchant l'événement et l'acteur est établi par l'instance. L'action *assignActor()* (algorithme 7) change l'état de l'acteur et incrémente le nombre d'instances de tâche dont il est responsable.

Algorithm 7: assignActor(). Assignation d'un acteur.

```
   Input: Actor a ;
1  begin
2  |  a.taskNumber + + ;
3  |  a.currentState = "Assigned" ;
4  end
```

— Etat *"Assigned"* à *"Assigned"* : A l'état *Assigned*, l'acteur peut continuer à se faire assigner des tâches. Son état ne change pas mais le nombre de tâches associées à lui est incrémenté.

— Etat *"Assigned"* à l'état *"Performing"* : Lorsque l'événement *TaskStarting* est déclenché par une des instances de tâche assignées à l'acteur, ce dernier passe à l'état *Performing*. L'action exécutée *activateActor()* est décrite par l'algorithme 8.

Algorithm 8: activateActor(). Activation d'un acteur.

```
   Input: Actor a ;
1  begin
2  |  a.currentState = "Performing" ;
3  end
```

— Etat *"Performing"* à l'état *"Performing"* : A l'état *Performing* l'acteur peut recevoir deux types d'événement : *TaskAssignment* (pour accepter de nouvelles instances de tâche) ou *TaskFinishing* (lorsqu'une instance de tâche dont il est responsable est finie). Dans les deux cas, l'acteur ne change pas d'état mais le nombre d'instances de tâche associées est soit incrémenté soit décrémenté. L'algorithme 9 décrit l'action *updateActor()* exécutée.

— Etat *"Performing"* à l'état *"Free"* : Lorsque l'événement *TasksDischarging* est déclenché, soit par le project manager soit lorsque le nombre d'instances de tâche associé à l'acteur est égale à zéro (a.taskNumber = 0), l'acteur passe à l'état *Free* et est de nouveau disponible pour une assignation d'instances de tâche. L'action *freeActor()* est décrite par l'algorithme 10.

— Etat *"Free"* à l'état final : l'événement ActorDestruction déclenche la suppression d'un

Algorithm 9: updateActor(). Mise à jour d'un acteur.

```
   Input: Actor a, Event e ;
 1 begin
 2 |   switch e do
 3 |   |   case TaskAssignment
 4 |   |   |   a.taskNumber + + ;
 5 |   |   end
 6 |   |   case TaskFinishing
 7 |   |   |   a.taskNumber − − ;
 8 |   |   end
 9 |   end
10 end
```

Algorithm 10: freeActor(). Libération d'un acteur.

```
   Input: Actor a ;
 1 begin
 2 |   a.currentState = "Free" ;
 3 end
```

noeud actor du système. L'action *deleteActor()* (algorithme 11) décrit cette suppression. La procédure *deleteQualification()* est décrite dans l'annexe D de ce manuscrit.

Algorithm 11: deleteActor(). Suppression d'un noeud acteur.

```
   Input: Actor a ;
 1 begin
 2 |   for i = 1 to a.Qualification.length() do
 3 |   |   deleteQualification(a.Qualification[i]) ;
 4 |   end
 5 |   deleteActorNode(a) ;
 6 end
```

5.2.3 Comportement d'une instance de tâche simple

Le concept STI désigne une tâche non décomposable et assignable à un acteur. Elle utilise une WPI en entrée pour produire une WPI en sortie. Durant son exécution, une STI passe par différents états : *"Instantiated"*, *"Assigned"*, *"InProgress"* et *"Finished"*.

Nous détaillons ci-dessous le comportement d'une instance de tâche (STI).

5.2.3.1 Description des états

Les différents états par lesquels passe une instance de tâche sont les suivants comme présenté sur la figure 3.10 :

— *Instantiated* : Une instance de tâche simple à l'état *instantiated* signifie qu'il y a un noeud STI dans le système, associé à cette tâche.

— *Assigned* : L'instance de tâche simple a été assignée à un acteur.

— *InProgress* : L'instance de tâche simple est en train d'être exécutée par un acteur.

— *Finished* : L'instance de tâche a fini d'être exécutée.

5.2.3.2 Description des transitions

La figure 5.4 décrit les transitions qui permettent de dérouler le cycle de vie de la STI.

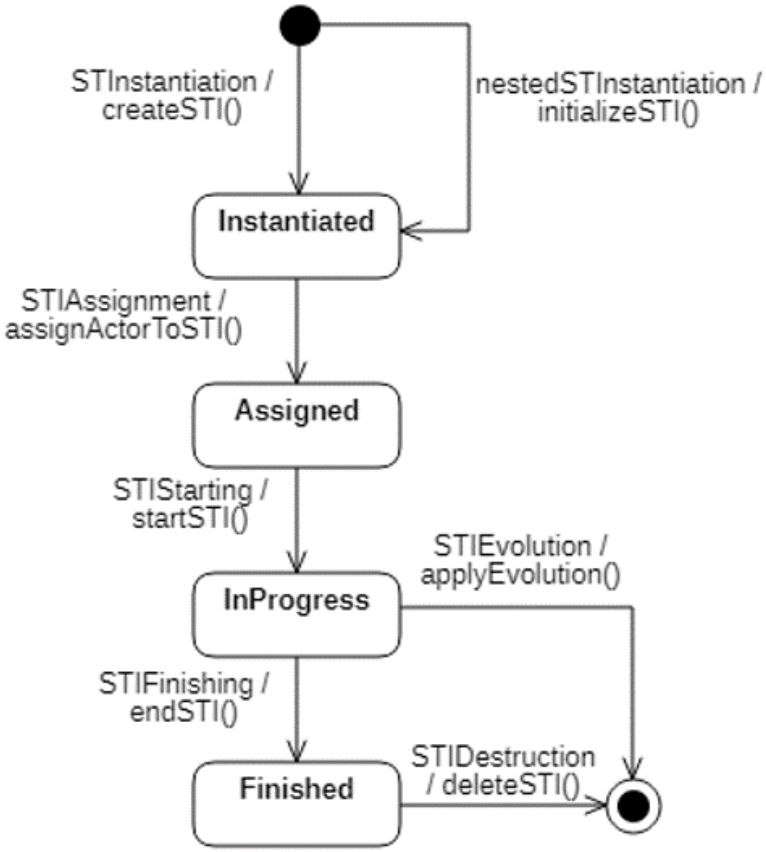


FIGURE 5.4: Comportement d'une instance de tâche simple.

Ci-dessous, nous détaillons les différentes transitions du cycle de vie d'une STI.

— Etat initial à l'état *"Instantiated"* : A l'état initial, la STI peut recevoir deux événements : *STInstantiation* est déclenché lorsqu'une STI indépendante est instanciée, *nestedSTInstantiation* est déclenché lorsqu'une STI au sein d'une CTI est créée. L'action *createSTI()* associée à l'événement *STInstantiation* crée un noeud STI représentant une instance de tâche à l'état *"Instantiated"* (*createSTINode(T)*, ligne 2). Ce noeud est par la suite lié aux instances de produit en entrée et en sortie (WPIs) nécessaires (ligne 3 à 9). Le "data flow" de ces WPIs est appliqué à la STI via *createTaskInstanceParameter()*. Après cela, la STI est liée aux instances de tâche qui précédent T dans le modèle de processus (ligne 10 à 15). L'algorithme 12 décrit les différentes instructions exécutées durant la création de la STI. L'action *initializeSTI()* suit le même déroulement que la création d'une STI. Le résultat de l'algorithme est illustré par la figure 5.5 qui montre la création d'une instance STI et ses paramètres à partir de la tâche simple *Modify Document*.

Algorithm 12: createSTI(). Création d'une STI à partir d'une tâche T.

```
   Input: Task T ;
   Output: SingleTaskInstance sti
 1 begin
 2     sti = createSTINode(T) ;
 3     TaskParameter[] tp = T.TaskParameter ;
 4     for i = 1 to tp.length() do
 5         WorkProductInstance wpi = getInstance (tp[i].product) ;
 6         TaskParameterDirection dir = tp[i].direction ;
 7         createTaskInstanceParameter (sti, wpi, dir) ;
 8         triggerEvent(UseByTask, wpi) ;
 9     end
10     WorkSequence[] wsq = T.WorkSequence ;
11     for i = 1 to wsq.length() do
12         TaskInstance ti = getInstance (wsq [i].predecessor) ;
13         WorkSequenceKind wsqk = wsq[i].linkKind ;
14         createTaskInstanceSequence (sti, ti, wsqk) ;
15     end
16     sti.currentState = "Instantiated" ;
17 end
```

— Etat *"Instantiated"* à l'état *"Assigned"* : Lorsque l'événement *STIAssignment* se produit, la liaison entre l'instance de tâche simple et l'acteur responsable est faite. L'action associée *assignActorToSTI()* est représentée par l'algorithme 13. Dans cet algorithme, nous avons les éléments suivants :

- *sti.SingleTask.TaskPerformer.role* est le rôle en charge de l'exécution de la SingleTask dont est issue la STI en cours.

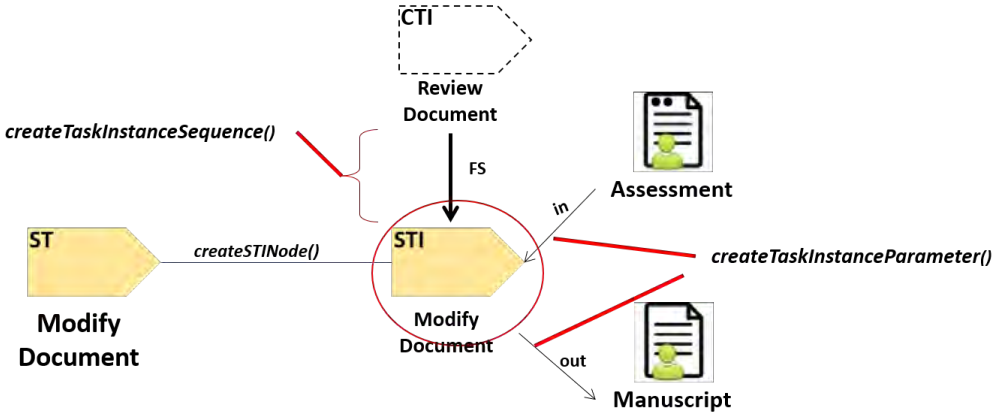


FIGURE 5.5: Résultat de l'instanciation de la tâche simple Modify Document.

Algorithm 13: assignActorToSTI(). Affectation d'une STI à un acteur.

Input: SingleTaskInstance *sti*, Actor *a* ;

```
1 begin
2 |   Role r = sti.SingleTask.TaskPerformer.role ;
3 |   if isIncluded(r, a.Qualification.role) then
4 |   |   createTaskInstancePerformer(sti, a) ;
5 |   |   sti.currentState = "Assigned" ;
6 |   |   triggerEvent(TaskAssignment, a) ;
7 end
```

- *createTaskInstancePerformer(sti,a)* crée un arc de type *TaskInstancePerformer* entre la STI et l'acteur.
- *triggerEvent(TaskAssignment, a)* déclenche l'événement au niveau de la machine à états de Actor pour changer son état.

— Etat *"Assigned"* à l'état *"InProgress"* : Lorsque l'événement *STIStarting* est déclenché, l'exécution de la *STI* démarre si toutes les conditions sont respectées. Il y a trois conditions à prendre en compte pour le démarrage de la STI : (1) Vérifier le type de séquencement entre la STI et ses prédécesseurs, (2) la disponibilité de l'acteur devant exécuter la STI et (3) la disponibilité des produits en entrée de la STI. L'action *startSTI()* est décrite par l'algorithme 14.

Algorithm 14: startSTI(). Démarrage de l'exécution d'une STI.

Input: SingleTaskInstance *sti* ;

```
1 begin
2 |   sti.currentState = "InProgress" ;
3 |   triggerEvent(TaskStarting, sti.TaskInstancePerformer.actor) ;
4 end
```

— Etat *"InProgress"* à l'état *"Finished"* : A la fin de l'exécution de la tâche, l'événement déclenché est *STIFinishing*. L'action correspondante, *endSTI()* est décrite par l'algorithme 15.

Algorithm 15: endSTI(). Fin de l'exécution d'une STI.

```
   Input: SingleTaskInstance sti ;
1  begin
2  |  sti.currentState = "Finished" ;
3  |  triggerEvent(TaskFinishing, sti.TaskInstancePerformer.actor) ;
4  |  for i = 1 to sti.TaskInstanceParameter.length() do
5  |  |  WorkProductInstance wpi = sti.TaskInstanceParameter[i].product ;
6  |  |  triggerEvent(FreeByTask, wpi) ;
7  |  end
8  end
```

— Etat *"Finished"* à l'état final : l'événement *STIDestruction* déclenche la suppression d'un noeud STI et les liaisons avec les autres éléments du processus. L'action correspondante *deleteSTI()*, est décrite par l'algorithme 16.

Algorithm 16: deleteSTI(). Suppression d'un noeud STI.

```
   Input: SingleTaskInstance sti ;
1  begin
2  |  //deleteDataFlow(sti.TaskInstanceParameter)
3  |  for i = 1 to sti.TaskInstanceParameter.length() do
4  |  |  WorkProductInstance wpi = sti.TaskInstanceParameter[i].product ;
5  |  |  deleteTaskInstanceParameter(sti, wpi) ;
6  |  end
7  |  //deleteControlFlow(sti.TaskInstanceSequence)
8  |  for i = 1 to sti.TaskInstanceSequence.length() do
9  |  |  TaskInstance ti = sti.TaskInstanceSequence[i].predecessor ;
10 |  |  deleteTaskInstanceSequence(sti, ti) ;
11 |  end
12 |  deletePerformer(sti.TaskInstancePerformer) ;
13 |  deleteSTINode(sti) ;
14 end
```

— Etat *"InProgress"* à l'état final : L'événement *STIEvolution* déclenche l'évolution de la STI vers une CTI. L'action associée, *applyEvolution()*, déclenche un événement associé à la machine à états de CTI pour créer la nouvelle CTI. Ce processus est décrit par l'algorithme 17. La figure 5.6 montre le résultat de l'évolution d'une STI.

Algorithm 17: applyEvolution(). Evolution d'une STI en CTI.

Input: SingleTaskInstance *sti*, CollaborationPattern *cp* ;
Output: CollaborativeTaskInstance *cti* ;

```
1 begin
2 |   CollaborativeTaskInstance cti = triggerEvent(STItoCTI, sti, cp) ;
3 |   deleteSTI(sti) ;
4 end
```

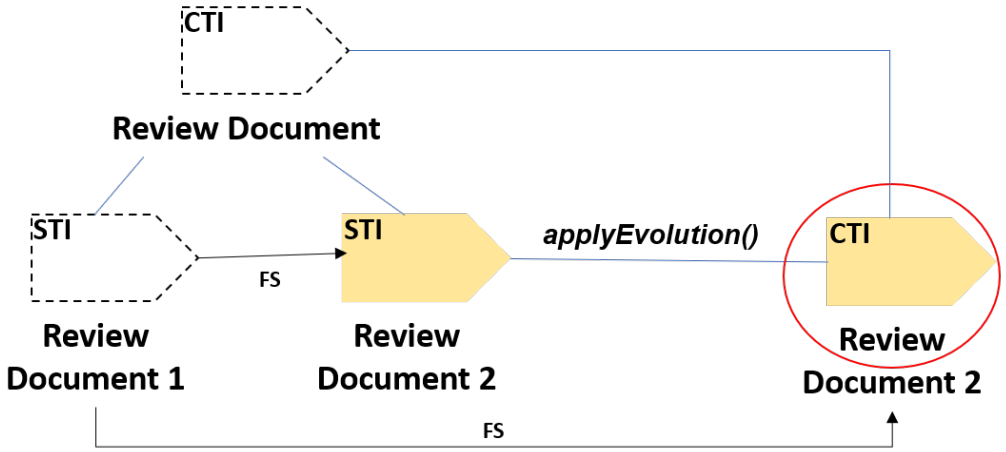


FIGURE 5.6: Résultat de l'évolution de la STI Review Document 2 en CTI.

5.3 Sémantique opérationnelle d'une instance de tâche collaborative

Une tâche collaborative est une tâche qui est réalisée par plusieurs acteurs jouant le même rôle à l'exécution. L'instance d'une tâche collaborative (*"collaborative task instance - CTI"*) est composée de plusieurs STIs, chaque acteur réalisant l'une d'elles. Lors du déploiement, un patron de collaboration est appliqué pour définir les relations entre les STIs de la CTI. Une instance de tâche collaborative passe par différents états durant son cycle de vie : *"Instantiated"*, *"Assigned"*, *"InProgress"* et *"Finished"*.

5.3.1 Description des états

Ci-dessous, nous détaillons les différents états par lesquels passe la CTI comme présenté sur la figure 3.11. Ces états sont les mêmes que pour la STI :

— *Instantiated* : la CTI est dans cet état lorsqu'elle est déployée avec un patron de collaboration. Cet état signifie que l'ensemble des instances de la tâche (STIs) qui la composent

ont été créées.

— *Assigned* : Lorsqu'au moins une des STIs a été affectée à un acteur, la CTI composite englobante est considérée comme assignée.

— *InProgress* : tout comme l'état précédent, la CTI est à l'état *InProgress* lorsqu'une des instances de la tâche a démarré son exécution (est à l'état InProgress).

— *Finished* : Cet état représente la fin de l'exécution de la CTI. Lorsque toutes les STIs ont fini leur exécution, la CTI composite englobante passe à l'état *finished*.

5.3.2 Description des transitions

Sur la figure 5.7 nous illustrons le cycle de vie de la CTI.

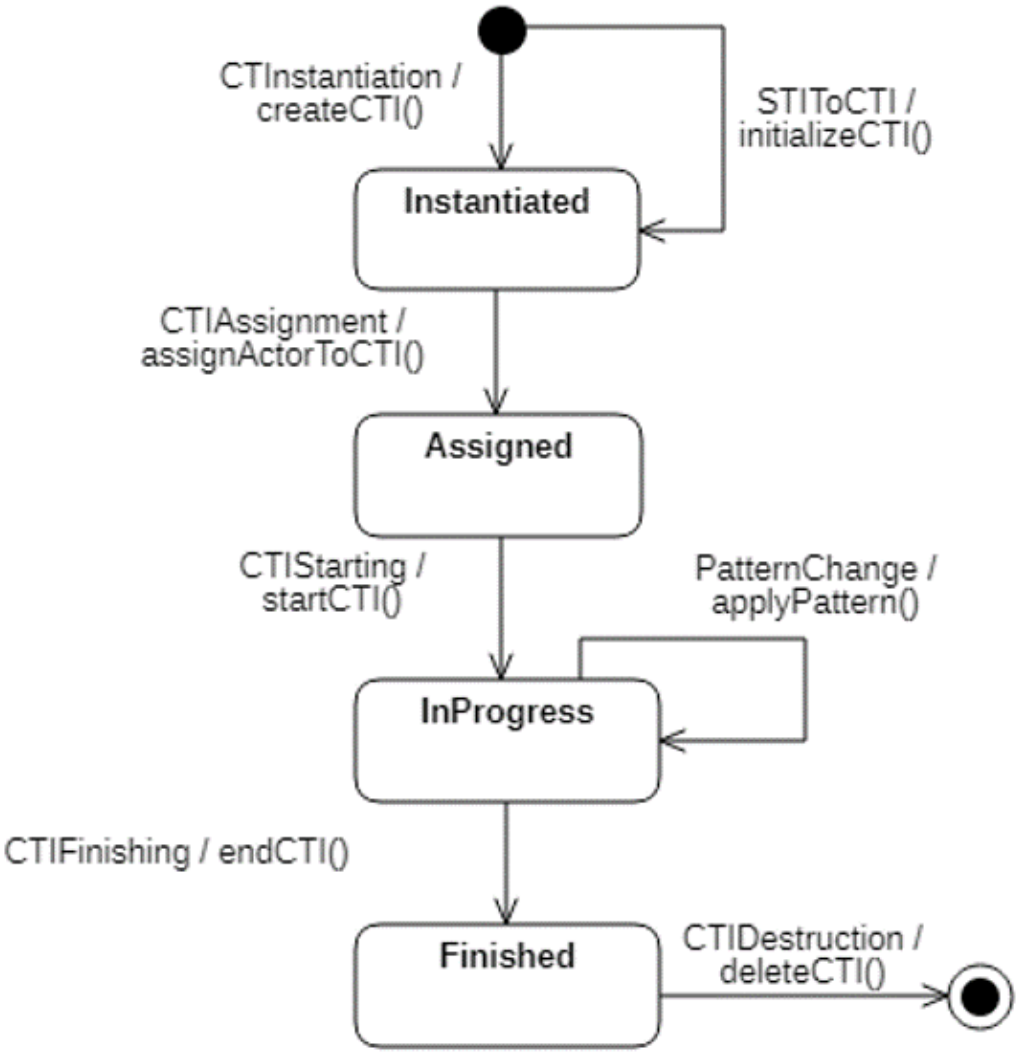


FIGURE 5.7: Comportement d'une instance de tâche collaborative.

Dans la suite, nous décrivons les transitions qui permettent de passer d'un état à l'autre au niveau d'une CTI.

— Etat initial à l'état *"Instantiated"* : Cette transition peut être déclenchée par deux événements : le premier, *CTInstanciation* est déclenché lorsqu'une CTI est instanciée à partir d'une tâche collaborative, le deuxième, *STItoCTI* se produit lorsque la CTI résulte d'une évolution d'une STI. C'est à ce moment-là qu'un patron de collaboration est appliqué à la tâche collaborative. Comme indiqué dans les chapitres précédents, nous utilisons le mécanisme de liaison tardive ("late-binding") pour appliquer un patron de collaboration à une CTI au moment de l'exécution afin d'obtenir de manière dynamique l'ordonnancement des différentes STIs composant la CTI. Pour permettre la liaison tardive, nous définissons les actions associées aux transitions de la machine à états de la CTI comme des fonctions paramétrées prenant les patrons de collaboration en paramètres effectifs pour définir l'ordonnancement des instances de tâche de la CTI. L'action associée au premier événement, *createCTI(cti, numberOfInstances)* est décrite par l'algorithme 18. Elle est exécutée pour créer un noeud CTI représentant une instance de la tâche collaborative T à l'état *Instantiated*. Cette action permet la création d'une instance CTI de la tâche T (cf. figure 5.8 ci-dessous). Nous montrons dans cet algorithme (18) les étapes essentielles de l'action *createCTI()* pour la création d'une CTI. Cet algorithme est illustré sur notre exemple de fil conducteur présenté au chapitre 3.
Cet algorithme contient des appels à des procédures, dont les descriptions sont ci-dessous. Ces procédures sont implémentées dans CPE. Il s'agit en premier lieu d'instancier une tâche collaborative. Après l'instanciation, un patron de collaboration est appliqué à la CTI obtenue pour décider de l'ordonnancement d'exécution des STIs.

- *createCTINode(Task T)* crée un objet CTI qui se réfère à l'instance de tâche collaborative nouvellement créée à partir de la tâche T.
- Lignes 3 à 9 : Nous récupérons la liste des paramètres de la tâche T source de la CTI. Pour chaque paramètre, nous récupérons les instances de produit ainsi que les directions de ces WPIs (in, out, inout). La liaison entre la CTI et les WPIs est effectuée par la procédure *createTaskInstanceParameter(). triggerEvent(UseByTask, wpi)* déclenche un événement au niveau de la machine à états de la WPI pour ses transitions.
- Lignes 10 à 15 : Nous récupérons le séquencement que la tâche source avait avec ses

Algorithm 18: createCTI(). Création d'une instance de tâche collaborative *cti* qui contient *numberOfInstances* STIs avec un patron de collaboration *cp*.

```
   Input: Task T, CollaborationPattern cp, Integer numberOfInstances;
   Output: CollaborativeTaskInstance cti, SingleTaskInstance[] sti;
 1 begin
 2     cti = createCTINode(T);
 3     TaskParameter[] tp = T.TaskParameter;
 4     for i = 1 to tp.length() do
 5         WorkProductInstance wpi = getInstance (tp[i].product);
 6         TaskParameterDirection dir = tp[i].direction;
 7         createTaskInstanceParameter(cti, wpi, dir);
 8         triggerEvent(UseByTask, wpi);
 9     end
10     WorkSequence[] wsq = T.WorkSequence;
11     for i = 1 to wsq.length() do
12         TaskInstance ti = getInstance (wsq[i].predecessor);
13         WorkSequenceKind wsqk = wsq[i].linkKind;
14         createTaskInstanceSequence(cti, ti, wsqk);
15     end
16     STI[] compsti; //liste des sti dans la cti
17     for i = 1 to numberOfInstances do
18         // créer une STI par l'action initializeSTI() dans la machine STI.
19         // Un nœud STI est créé dans l'état Instantiated mais n'est pas lié aux WPI.
20         compsti[i] = triggerEvent(nestedSTInstantiation, T);
21         cti.nestedSTI[i] = compsti[i];
22     end
23     // appliquer le patron cp à la cti pour défnir la répartition des (composants de) wpi
24     //de la cti aux sti, aussi pour défnir les work sequence entre sti de la cti
25     applyPattern(cti, cp);
26     cti.currentState = "Instantiated";
27 end
```

prédécesseurs. Ce séquencement est ensuite appliqué à la CTI via *createTaskInstanceSequence()*.

- Ligne 16 à 22 : Nous créons les STIs composant la CTI en nous basant sur le nombre d'instances ($numberOfInstances$). Ensuite, chaque STI est liée à la CTI. La valeur de $numberOfInstances$ peut être définie lors du déploiement de la tâche par le project manager ou déduite de la structure du produit de sortie de la tâche T conformément à ce qui a été présenté dans le chapitre 4. En effet, le nombre de composants de l'instance du produit en sortie implique un nombre d'instances de tâche équivalent pour les manipuler. À titre d'exemple, la figure 5.8 illustre l'instance de la tâche collaborative *ReviewDocument* de l'exemple fil conducteur du chapitre 3 instanciée avec trois relecteurs.

- La procédure *applyPattern (CollaborativeTaskInstance cti, CollaborationPattern cp)* permet d'appliquer le patron de collaboration cp choisi à la CTI afin d'établir les relations entre les différentes STIs à l'intérieur de la CTI (voir algorithme 21).

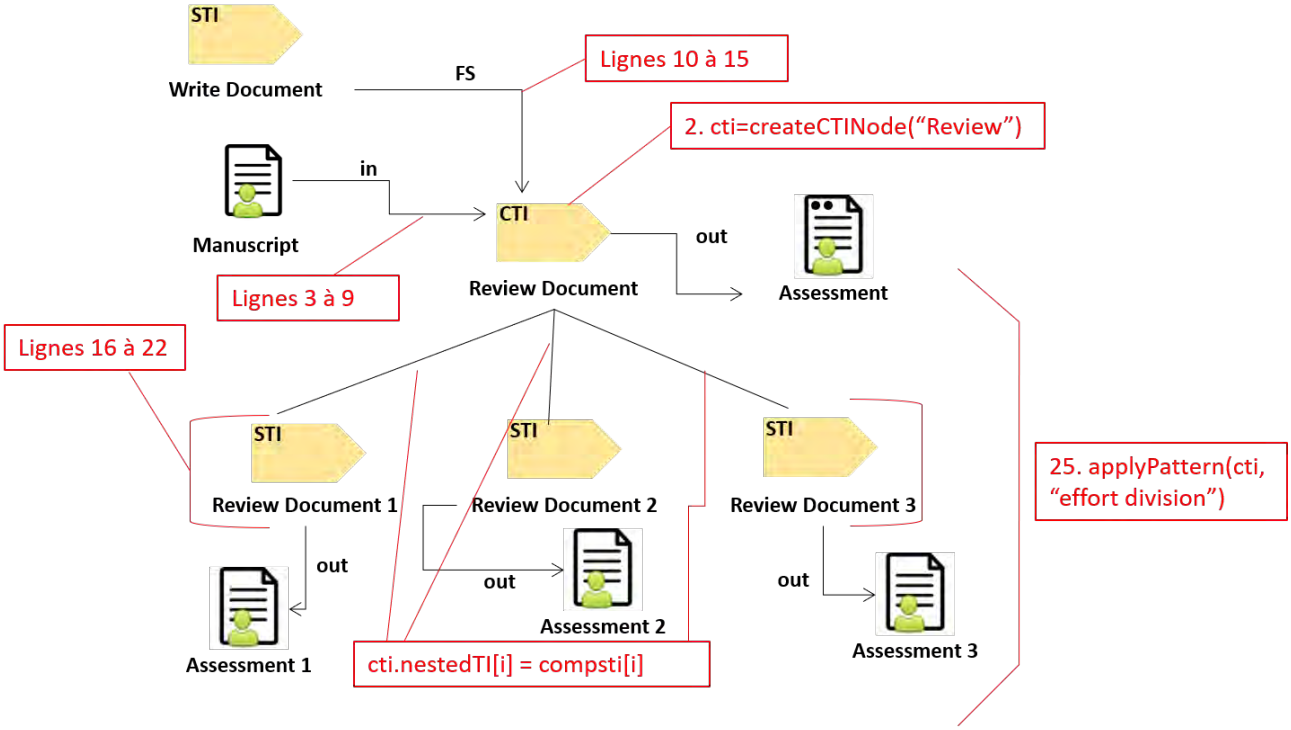


FIGURE 5.8: Résultat de la création de l'instance de tâche collaborative Review Document.

La figure 5.8 montre l'instance de processus obtenue après l'application du patron *effort division* à l'instance CTI de la tâche collaborative *Review a Document*. Dans cette figure, nous remarquons l'application des points décrits ci-dessus concernant l'algorithme. Le patron choisi représentant une stratégie parallèle pour exécuter la tâche collaborative, il n'existe pas de séquencement entre les trois STIs *Review Document 1*, *Review Document 2* et *Review Document 3* à l'intérieur de la CTI. L'action *initializeSTI()* associée à l'événement *STItoCTI* suit le même déroulement que la création d'une CTI.

— Etat *"Instantiated"* à l'état *"Assigned"* : L'événement *CTIAssigment* est déclenché lorsque l'assignation des acteurs à une CTI s'effectue. L'action *assignActorsToCTI(CTI cti, Actor[] a)* assigne chaque acteur à une STI dans la CTI. La description de cette action est faite par l'algorithme 19. La figure 5.9 montre le résultat de l'assignation des acteurs.

— Etat *"Assigned"* à l'état *"InProgress"* : cette transition se déroule lorsqu'une des STI de la CTI démarre son exécution. L'événement qui se produit est *CTIStarting*. Si la condition de démarrage de la tâche est vérifiée, l'action associée *startCTI(cti)* (algorithme 20)

Algorithm 19: assignActorsToCTI(). Assignation d'un ensemble d'acteurs à l'instance de tâche collaborative.

```
Input: CollaborativeTaskInstance cti, Actor[] a ;
1 begin
2 |  STI[] compsti = cti.nestedSTI ;
3 |  for i = 1 to cti.length do
4 |  |  triggerEvent(STIAssignment,compsti[i], a[i]) ;
5 |  end
6 |  cti.currentState = "Assigned" ;
7 end
```

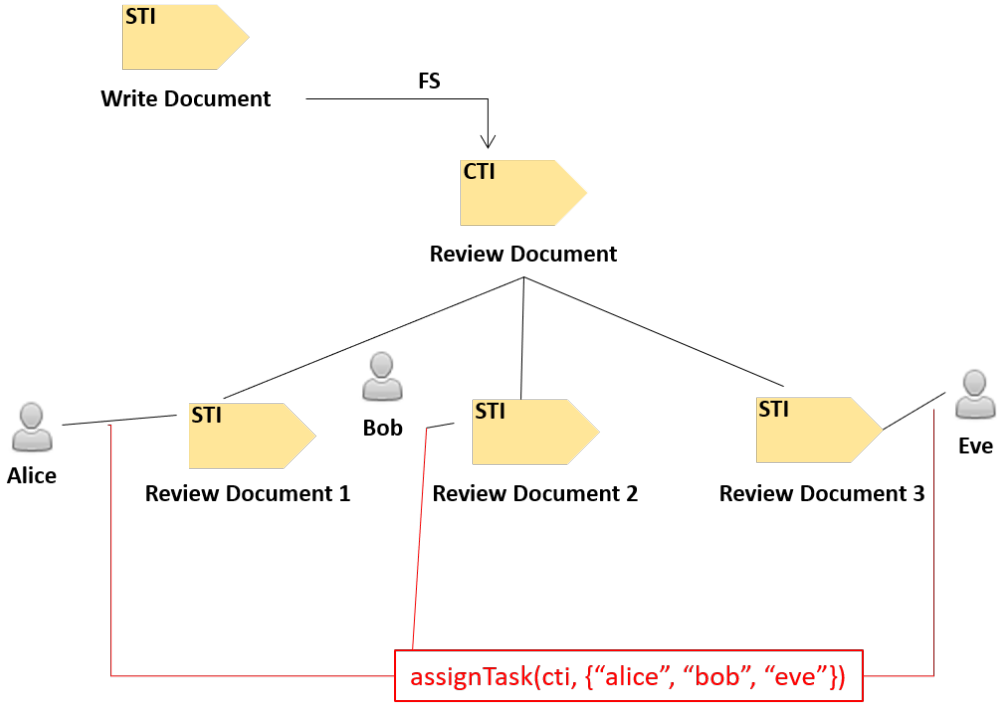


FIGURE 5.9: Résultat de l'assignation de la CTI Review Document à un groupe d'acteurs.

met l'instance de la tâche à l'état "InProgress". La condition à vérifier est que l'une des STIs contenues dans la CTI est à l'état *"InProgress"*.

— Etat *"InProgress"* à l'état *"InProgress"* : lors de l'exécution d'une CTI, les acteurs du processus peuvent vouloir changer la stratégie de collaboration de la tâche, c'est-à-dire choisir un autre patron de collaboration pour réaliser la CTI. Cette transition déclenche l'événement *PatternChange* qui permet d'appliquer un nouveau patron. L'action correspondante, *applyPattern (cti, cp)*, illustre les instructions permettant la mise en oeuvre du nouveau patron (voir algorithme 21).

Dans cet algorithme, *identifyPatternWorkSequence (CollaborationPattern cp)* et *identifyPatternDataFlow (CollaborationPattern cp)* permettent de reconnaître respectivement

Algorithm 20: startCTI(). Démarrage d'une CTI.

```
   Input: CollaborativeTaskInstance cti ;
1  begin
2  |  cti.currentState = "InProgress" ;
3  end
```

Algorithm 21: applyPattern(). Application du patron de collaboration *cp* à l'instance de tâche collaborative *cti*.

```
   Input: CollaborativeTaskInstance cti, CollaborationPattern cp ;
1  begin
2  |  wsType = identifyPatternWorkSequenceType(cp) ;
3  |  dfType = identifyPatternDataFlowType(cp) ;
4  |  sti = cti.nestedSTI ;
5  |  wpi = cti.TaskInstanceParameter.product ;
6  |  for i = 1 to sti.length − 1 do
7  |  |  createTaskInstanceSequence(sti[i], sti[i+1], wsType) ;
8  |  end
9  |  for i = 1 to wpi.length − 1 do
10 |  |  for j = 1 to sti.length − 1 do
11 |  |  |  createTaskInstanceParameter(sti[j], wpi[i], dfType) ;
12 |  |  end
13 |  end
14 end
```

le type de *work sequence* et le type de *data flow* des relations inter-instances définies à l'intérieur du patron de collaboration cp passé en paramètre. Ensuite, l'instruction *cti.TaskInstanceParameter.product* permet de récupérer l'ensemble des WPIs en entrée en en sortie de la CTI. La boucle de la ligne 6 à la ligne 8 permet d'établir les types de relations identifiés entre deux STIs à l'intérieur de la CTI. Cette boucle fait un parcours jusqu'à l'avant dernier élément du tableau de STIs étant donné que chaque élément courant (i) est lié avec celui qui le suit dans le tableau ($i + 1$). La boucle de la ligne 9 à la ligne 13 établit les relations entre les STIs et les WPIs correspondantes avec la direction adéquante (dfType - in, out, inout).

Durant l'exécution de la tâche *Review Document* décrite précédemment, la stratégie de révision peut être modifiée. Considérons ainsi que les relecteurs doivent effectuer une révision séquentielle par raffinement : chaque relecteur doit consulter l'évaluation de son prédécesseur avant de rajouter ses propres remarques pour éviter les redondances. Dans ce cas, ils pourraient vouloir remplacer la stratégie parallèle par une stratégie séquentielle. Cette situation déclenche l'événement *PatternChange* et l'action *applyPattern (cti, "full ordered cowork")* est exécutée pour donner le résultat décrit dans la figure 5.10.

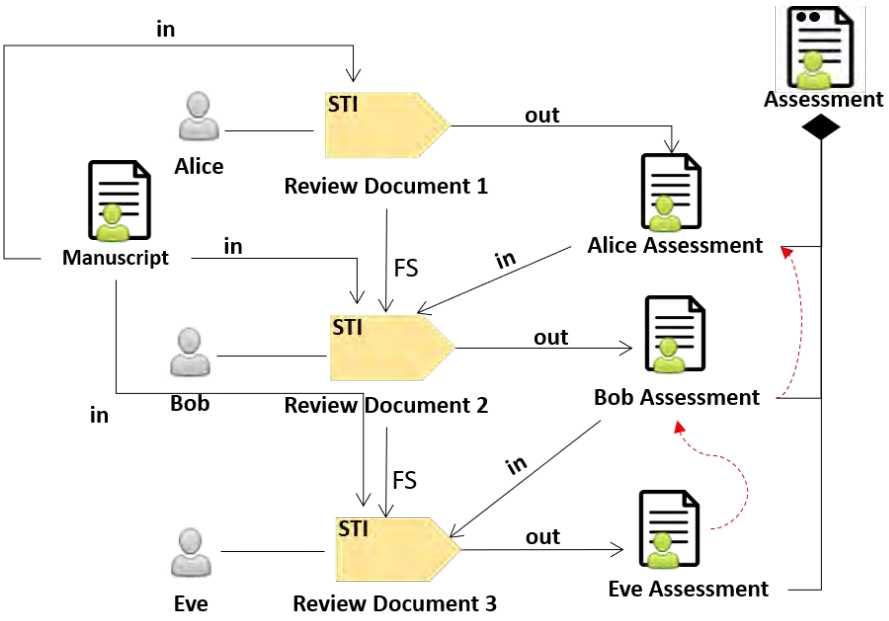


FIGURE 5.10: Résultat, sur la CTI, de l'application du patron "full ordered co-work".

La figure 5.10 illustre le changement de stratégie après application du patron *"Full Ordered Co-work"*.

— Etat *"InProgress"* à l'état *"Finished"* : cette transition se produit lorsque les acteurs du processus mettent fin à l'exécution de l'instance de tâche collaborative. La fin de la CTI intervient quand les acteurs ont fini d'exécuter leurs instances de tâche simple respectives. L'algorithme 22 décrit l'action endCTI().

Algorithm 22: endCTI(). Terminaison d'une CTI.

```
   Input: CollaborativeTaskInstance cti;
1  begin
2  |  cti.currentState = "Finished";
3  end
```

— Etat *"Finished"* à l'état final : L'événement *CTIDestruction* déclenche la suppression d'un noeud CTI et les liaisons avec les autres éléments du processus. Les STIs qui la composent sont également supprimées. L'algorithme 23 décrit l'action deleteCTI().

5.4 Conclusion

Dans ce chapitre, nous avons présenté la sémantique opérationnelle du langage ECPML à travers la description détaillée des machines à états correspondant aux cycles de vie des éléments

Algorithm 23: deleteCTI(). Suppression d'une CTI.

```
   Input: CollaborativeTaskInstance cti ;
 1 begin
 2 |   STI[] compsti = cti.nestedSTI ;
 3 |   for i = 1 to compsti.length() do
 4 |   |   triggerEvent(STIDestruction, compsti[i]) ;
 5 |   end
 6 |   // deleteDataFlow(sti.TaskInstanceParameter)
 7 |   for i = 1 to cti.TaskInstanceParameter.length() do
 8 |   |   WorkProductInstance wpi = sti.TaskInstanceParameter[i].product) ;
 9 |   |   triggerEvent(FreeByTask, wpi) ;
10 |   |   deleteTaskInstanceParameter(cti, wpi) ;
11 |   end
12 |   // deleteControlFlow(sti.TaskInstanceSequence)
13 |   for i = 1 to cti.TaskInstanceSequence.length() do
14 |   |   TaskInstance ti = cti.TaskInstanceSequence[i].predecessor- ;
15 |   |   deleteTaskInstanceSequence(cti, ti) ;
16 |   end
17 |   deleteCTINode(cti) ;
18 end
```

d'un processus. Nous avons à cet effet développé un ensemble d'algorithmes qui implémentent les actions des transitions entre états des éléments principaux du langage ECPML : *"WPI"*, *"Actor"*, *"STI"*, *"CTI"*. L'élément central de notre étude est l'instance de tâche collaborative sur laquelle nous appliquons un patron de collaboration pour obtenir les relations intra-instances à l'exécution.

Ces algorithmes ont été implémentés dans un moteur de processus (*"process engine"*) afin de faciliter le contrôle de l'instanciation et de l'exécution d'un processus collaboratif. Ce moteur de processus est le constituant principal de notre outil support CPE pour l'exécution d'un processus collaboratif.

Dans le chapitre suivant, nous décrivons d'une part l'outil CPE développé pour supporter notre approche, puis sa validation expérimentale à travers une étude de cas représentative.

Chapitre 6

Validation de l'approche

Sommaire

Pour valider notre approche, nous avons développé le prototype *CPE (Collaborative Process Engine)*, un système de gestion de processus supportant l'exécution flexible de processus collaboratifs permettant aux utilisateurs de choisir, au moment de la mise en œuvre, les stratégies collaboratives appropriées correspondant à leur modèle organisationnel.

Notre approche a été validée également de façon expérimentale par application de notre démarche à un exemple de processus représentatif dans le domaine de la conception logicielle.

Dans ce chapitre, nous commençons par présenter les différentes fonctionnalités de ce prototype (Section 6.1). La section 6.2 décrit l'architecture globale du sytème. La section 6.3 présente les différents composants utilisés dans l'implémentation du prototype CPE. La section 6.4 présente le cas d'étude sur lequel nous nous sommes appuyés pour illustrer l'exécution. Dans cette section, nous décrivons l'application d'un processus collaboratif, de sa modélisation jusqu'à son exécution. Ce cheminement est illustré avec le prototype CPE. La section 6.5 présente les limites de validité de notre prototype, et la section 6.6 permet de conclure ce chapitre.

6.1 Description des exigences fonctionnelles

Pour gérer l'exécution des processus collaboratifs, nous avons développé un moteur de processus, appelé *CPE (Collaborative Process Engine)*. Cette section présente les fonctionalités de CPE .

Le prototype réalisé permet principalement aux acteurs du processus de fournir un modèle de processus et d'instancier les tâches à instancier et les liens entre elles. Quand le processus est collaboratif le manager peut choisir, lors de l'instanciation, une stratégie de collaboration pour chacune des tâches collaboratives. Les stratégies correspondent aux patrons de collaboration présentés dans le chapitre 4.

Une fois l'instanciation effectuée, à chaque instance de tâche on alloue les ressources nécessaires à son exécution dès qu'elles sont disponibles. Il s'agit des produits en entrée, des outils et du ou des acteurs humains jouant le rôle associé à la tâche. L'affectation des ressources permet à chaque partie prenante de connaître ses propres tâches et l'ordonnancement entre elles. Chaque tâche pourra être contrôlée conformément à sa stratégie d'exécution. Par exemple pour une tâche collaborative instanciée avec le patron "refinement", l'exécution de la deuxième instance de tâche ne pourra se faire qu'après l'exécution de la première instance de tâche.

Grâce au prototype CPE, le project manager peut suivre l'exécution des tâches collaboratives et adapter la stratégie de collaboration pour l'exécution de celles-ci à tout moment en fonction des contraintes et des besoins du projet. CPE fournit aux acteurs du processus non seulement les fonctionnalités nécessaires pour accomplir leurs tâches (en vérifiant les conditions pour créer, démarrer, terminer ou attribuer des ressources à une instance de tâche), mais également une vue globale et en temps réel sur l'avancement de chaque tâche. Cela se fait en montrant les

informations sur la tâche collaborative à laquelle un acteur participe : l'état courant, les autres acteurs qui exécutent la tâche, les produits échangés.

La figure 6.1 présente un diagramme de cas d'utilisation illustrant les différentes fonctionnalités du prototype. Les acteurs du système sont le *project manager*, le *process designer* et le *développeur*. Ci-dessous, nous détaillons les cas d'utilisation de CPE :

— *Upload Process Model* décrit la soumission du modèle du processus dans CPE par le process designer.

— Après soumission, la génération d'une instance du processus (*Generate Process Instance*) est réalisée par le rôle *Project manager*.

— Le project manager peut continuer en appliquant un patron de collaboration (*Apply Collaboration Pattern*) à une tâche collaborative et en assignant les ressources aux tâches avec le cas *Assign Resources to a Task*.

— Le lancement de l'exécution est fait à travers le cas d'utilisation *Execute Task*. Ce cas peut être activé par le project manager ou le développeur. Durant l'exécution, ils peuvent aussi visualiser les états des différentes instances de tâche en cours d'exécution (*Visualize states*).

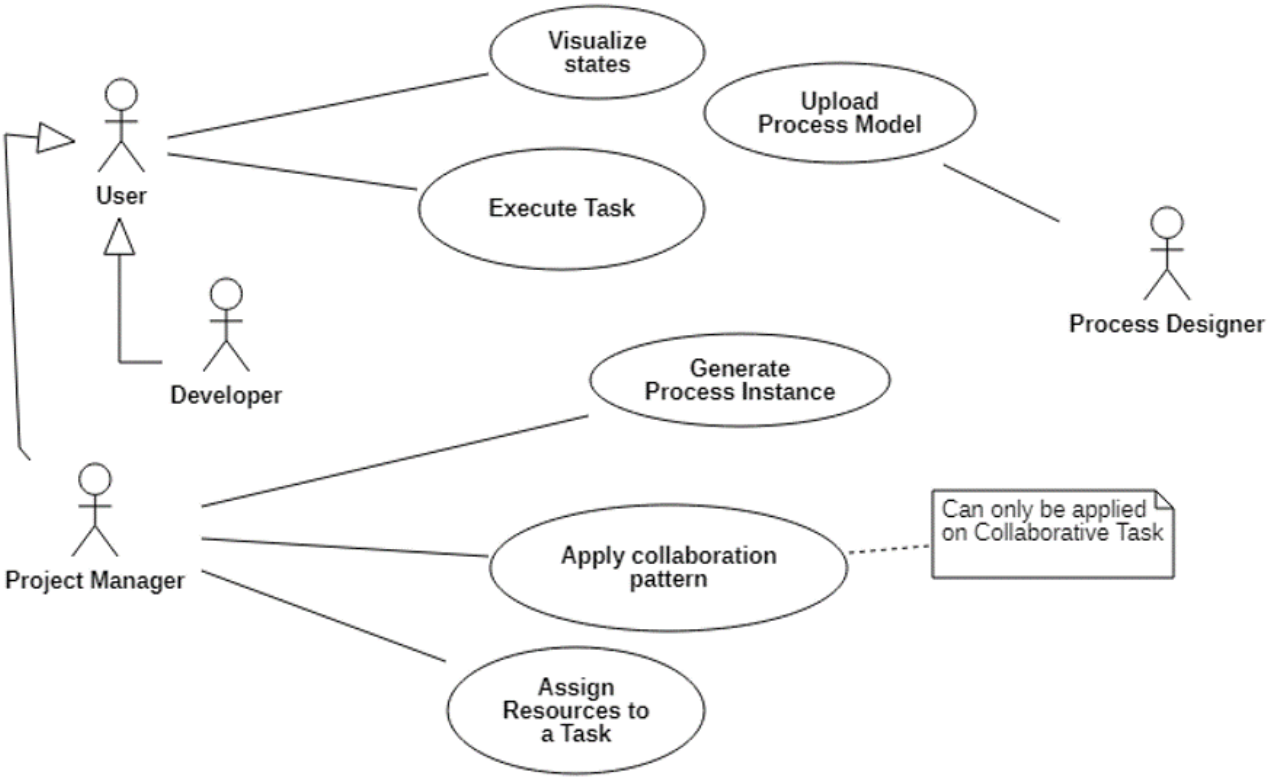


FIGURE 6.1: Diagramme de cas d'utilisation de CPE.

Les scénarios des différents cas d'utilisation sont décrits dans l'annexe C en utilisant le formalisme de description textuelle des diagrammes de séquence d'UML.

6.2 Architecture de CPE

CPE est constitué de plusieurs composants communiquant entre eux. Ces composants sont interconnectés sous la forme d'une architecture globale illustrée par la figure 6.2.

D'une part, un serveur de processus CPE a été développé. Ce serveur sert de moteur pour l'exploitation des modèles de processus décrits avec ECPML. L'aspect comportemental des éléments du processus est décrit dans le serveur (*State Machines*). Cela permet de dérouler l'exécution en tenant compte des changements d'états et des événements déclencheurs. Dans la suite nous détaillons les fonctionnalités du serveur.

D'autre part, un lien avec les patrons de collaboration et la lecture de ceux-ci a été mis en place. Cette interface de manipulation des patrons est connectée au serveur CPE. Ce lien permet l'instanciation des tâches collaboratives suivant la structure du patron choisi.

Le serveur CPE, que l'on peut aussi appeler le *process engine (moteur de processus)* assiste les acteurs dans la réalisation de leurs tâches, c'est-à-dire l'instanciation des tâches définies dans leur modèle de processus et la gestion de l'exécution de ces tâches. Le project manager choisit un modèle de processus dans le dépôt de processus (*Process Model Store*) puis procède à son instanciation en appliquant un patron de collaboration pour chaque tâche collaborative. Le patron choisi dépend du contexte du projet et provient du catalogue de patrons. Les instances créées à cette étape sont stockées dans le dépôt des instances (*Instances Strore*). Au moment de l'exécution, le moteur de processus met à jour les instances du processus, en utilisant la sémantique opérationnelle décrite dans les chapitres 4 et 5, pour faire évoluer le processus. Pour déployer une tâche collaborative, le nombre d'instances STI peut être fourni par le project manager ou imposé par la structure des produits en paramètres de la tâche.

Les produits physiques et les ressources humaines manipulés pendant l'exécution du processus sont gérés via une base de donnée externe : le *système de gestion des ressources (Resource Management System)*. Cette base de données est connectée à CPE qui gère uniquement les

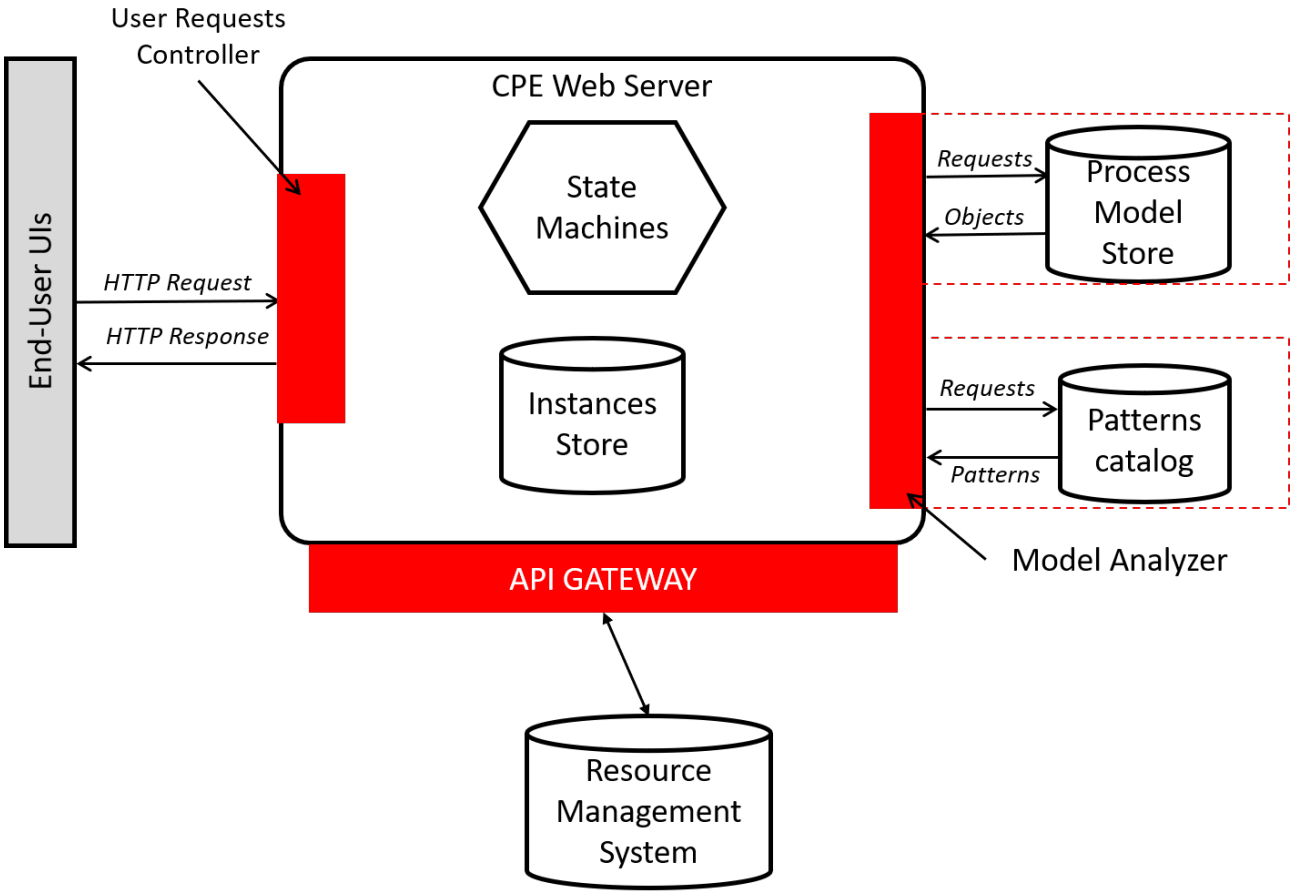


FIGURE 6.2: Architecture globale de CPE.

références vers les artefacts et les acteurs. *Instances Store* stocke les références vers les éléments contenus dans le *Resource Management System*

6.3 Description des composants de l'architecture

CPE est constitué principalement par les composants suivants : le *serveur CPE*, l'*analyseur de modèles (Model Analyzer)* et l'*interface d'exécution.*

6.3.1 Serveur CPE

La figure 6.3 décrit la structuration en couches du serveur CPE :

— Une couche accès : cette couche reçoit les requêtes HTTP au niveau du serveur en vue d'un traitement par les requêtes correspondantes. Elle est l'interface entre l'utilisateur

et le serveur. A travers cette couche, l'envoi d'événements pour le changement d'états des éléments du processus s'effectue.

— Une couche données : elle traite les événements reçus de la couche accès. Elle implémente les machines à états et exécute les actions selon les événements reçus. Elle est constituée des requêtes permettant la manipulation des éléments du processus. Ces requêtes permettent d'enregistrer les informations au niveau de la base de données après l'instanciation du processus. Parmi ces informations nous comptons les instances de tâche ainsi que les instances de produit.

— une couche stockage : elle est composée de la base de données des instances du processus. Elle reçoit les éléments à stocker dans la base de données provenant de l'intanciation au niveau de la couche données.

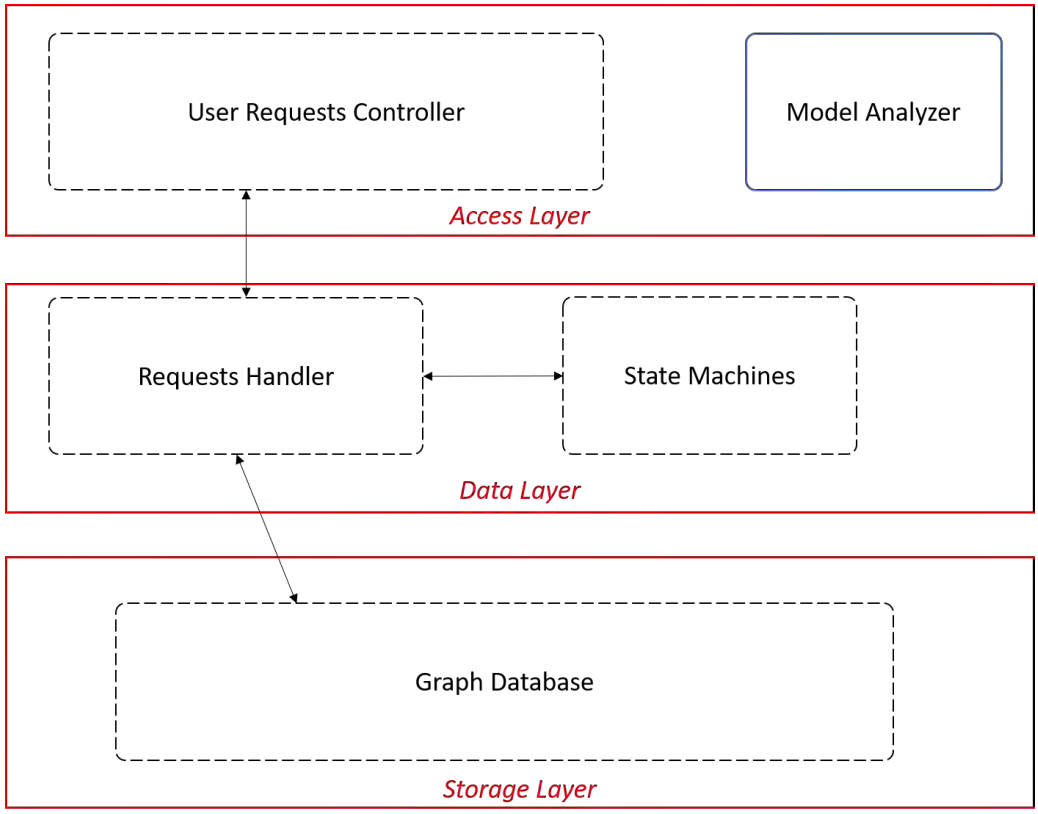


FIGURE 6.3: Couches d'implémentation du serveur CPE.

Le serveur CPE est le coeur du prototype. Il comporte les APIs permettant l'exécution du modèle de processus via des protocoles HTTP. Il a été développé en Angular 6 avec un backend en NodeJS. Les modèles de processus ECPML sont stockés au format XML. Le serveur lit un modèle de processus en XML pour avoir la liste des tâches et des produits à instancier dans

le système. Le serveur permet l'instanciation et l'exécution d'un modèle de processus. Pour ce faire, il dispose d'APIs dévelopées en NodeJS permettant :

— la lecture d'un modèle avec un analyseur XML
— le stockage de l'ensemble des instances des éléments du processus (CTI, STI, WPI, Actor) dans une base de données orientée graphe (*Instances Store*) avec Neo4J. Ce choix permet d'avoir une vision précise des relations qu'une instance de tâche possède avec les autres instances. En outre, la performance d'accès aux données est plus élevée comparée à une base de données relationnelle
— la mise à jour des éléments dans *Instances Store* c'est-à-dire les relations entre les instances des tâches et les liens avec les instances de produit.

En implémentant les machines à état des éléments du processus, le serveur est en mesure de contrôler les transitions entre états. Ce contrôle permet de réagir aux différents événements qui surviennent (affectation d'un acteur, démarrage d'une instance) et d'exécuter les actions adéquates.

6.3.2 Analyseur de modèles

L'analyseur de modèles (*Model Analyzer*) permet l'analyse d'un modèle issu de la solution d'un patron de collaboration pour la récupération des différentes relations : *control-flow* et *data-flow*, ainsi que l'analyse d'un modèle de processus pour la récupération des éléments du processus. Il est utilisé à chaque fois que le project manager choisit un modèle de processus à instancier ou choisit un patron de collaboration pour instancier une tâche collaborative. Par rapport aux patrons, le fonctionnement de l'analyseur est décrit comme suit. Il récupère les STIs incluses dans le patron de collaboration. Pour chaque STI, si celle-ci a un successeur, le work sequence entre les deux est récupéré. Ce work sequence est alors appliqué deux à deux aux différentes STIs. L'analyseur est aussi chargé d'exécuter l'action *applyPattern()* décrite dans le chapitre 5.

Par la suite, pour chaque instance de tâche du patron de collaboration, les paramètres formels sont récupérés ainsi que la direction ("in", "out","inout"). Comme pour le control-flow, le data-flow récupéré est appliqué aux STIs avec les paramètres effectifs du modèle de processus.

6.3.3 Interface d'exécution

L'interface d'exécution (représenté par le composant *User Requests Controller*) est le portail par lequel passent les acteurs du système pour l'exécution du processus. L'interface a été développée en tant qu'application web avec Angular. Elle est composée de plusieurs pages telles que :

— *la page de soumission du modèle* (figure 6.4) : c'est par cette page que se fait la proposition et l'analyse du modèle de processus afin d'en récupérer les éléments. L'instanciation du processus se fait après la soumission.

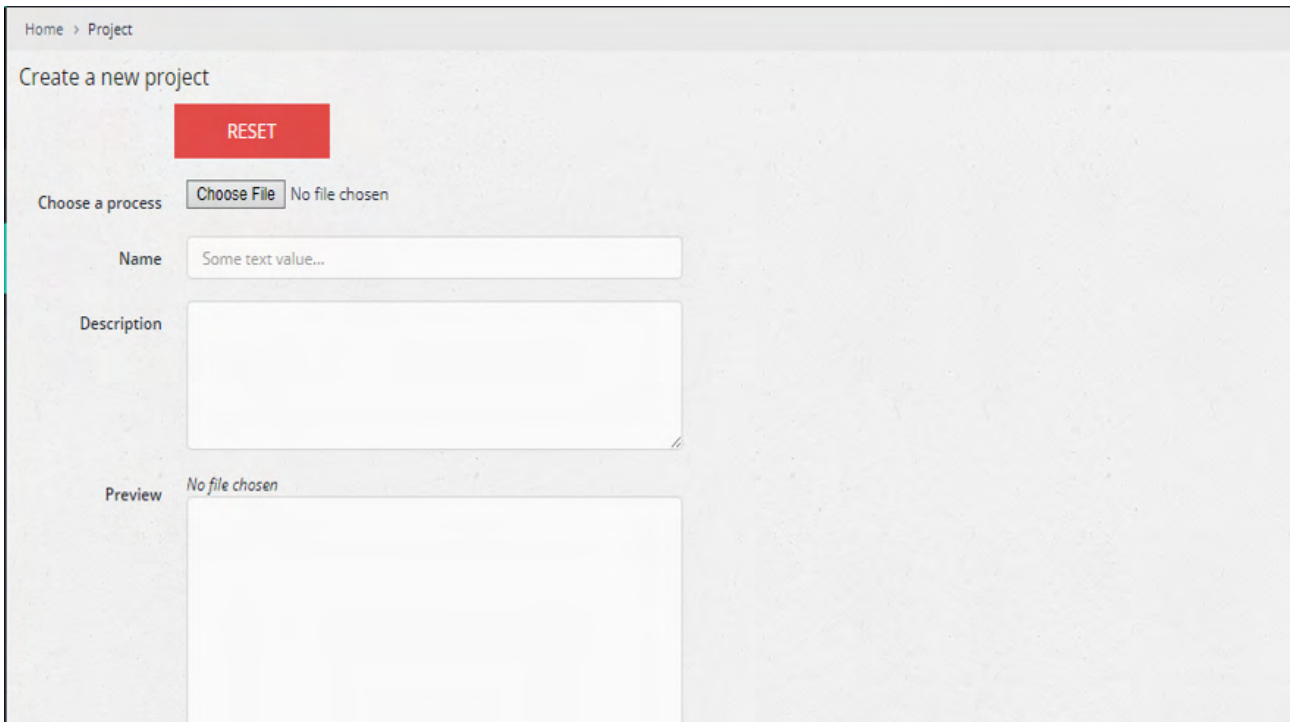


FIGURE 6.4: Page de soumission du modèle de processus.

— *la page des tâches* : cette page (figure 6.5) illustre l'ensemble des instances de tâches après l'instanciation. C'est à ce niveau que se fait l'assignation des ressources en plus du choix des paramètres effectifs de chaque instance de tâche. Un panneau permet de choisir le patron de collaboration parmi un ensemble proposé.

— *la page de contrôle d'une tâche* : dans cette page (figure 6.6), les différentes informations concernant la tâche sont affichées. Elle propose les fonctionnalités permettant le démarrage ou l'arrêt de l'exécution d'une tâche mais aussi l'évolution d'une STI en CTI.

Dans la section 6.4, nous montrons d'autres écrans de l'interface d'exécution correspondant à

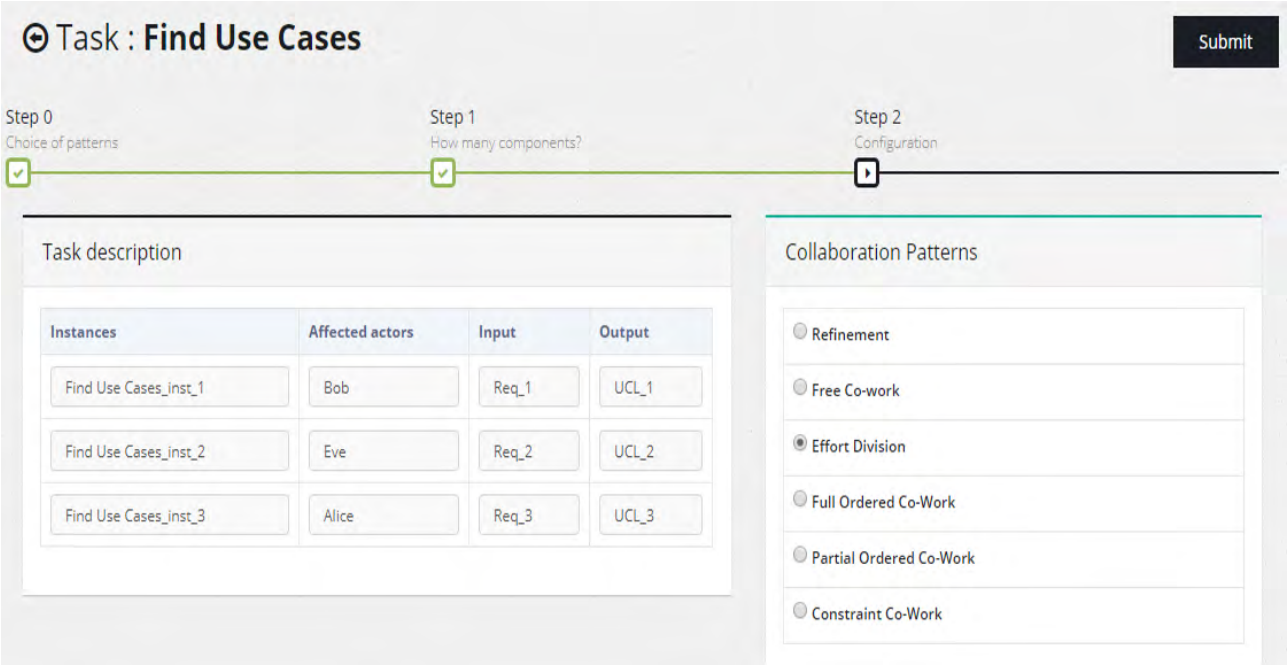


FIGURE 6.5: Page des tâches avec le choix des ressources et d'un patron de collaboration.

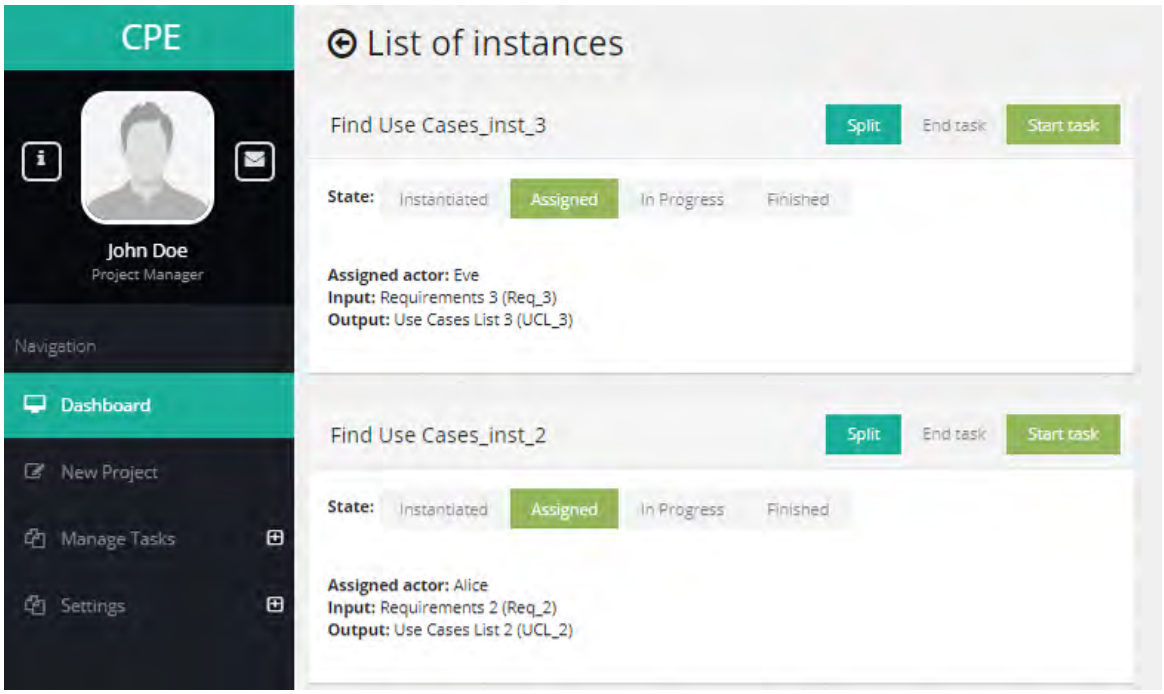


FIGURE 6.6: Page de contrôle d'une tâche.

l'étude de cas concrète développée.

6.4 Etude de cas

La validation expérimentale de notre approche a été effectuée à travers une étude de cas portant sur la mise en oeuvre d'un processus collaboratif représentatif. Pour illustrer cette étude de cas et valider l'implémentation décrite dans les sections précédentes, nous avons décidé de simuler l'exécution de ce processus avec l'outil CPE. La figure 6.7 ci-dessous décrit la démarche à appliquer pour mettre en oeuvre un processus avec CPE. Nous avons choisi un processus collaboratif dans le domaine de la conception logicielle. Ce processus, RUP-A inspiré de RUP (Rational Unified Process Adapted) (Kruchten, 2004), fournit les différentes activités d'analyse et de conception d'un projet logiciel. Ce processus a été modélisé en ECPML.

Pour avoir un contexte réel qui justifie les choix de stratégies et donc de patrons de collaboration, nous avons choisi une application concrète sur laquelle nous allons dérouler le processus RUP-A.

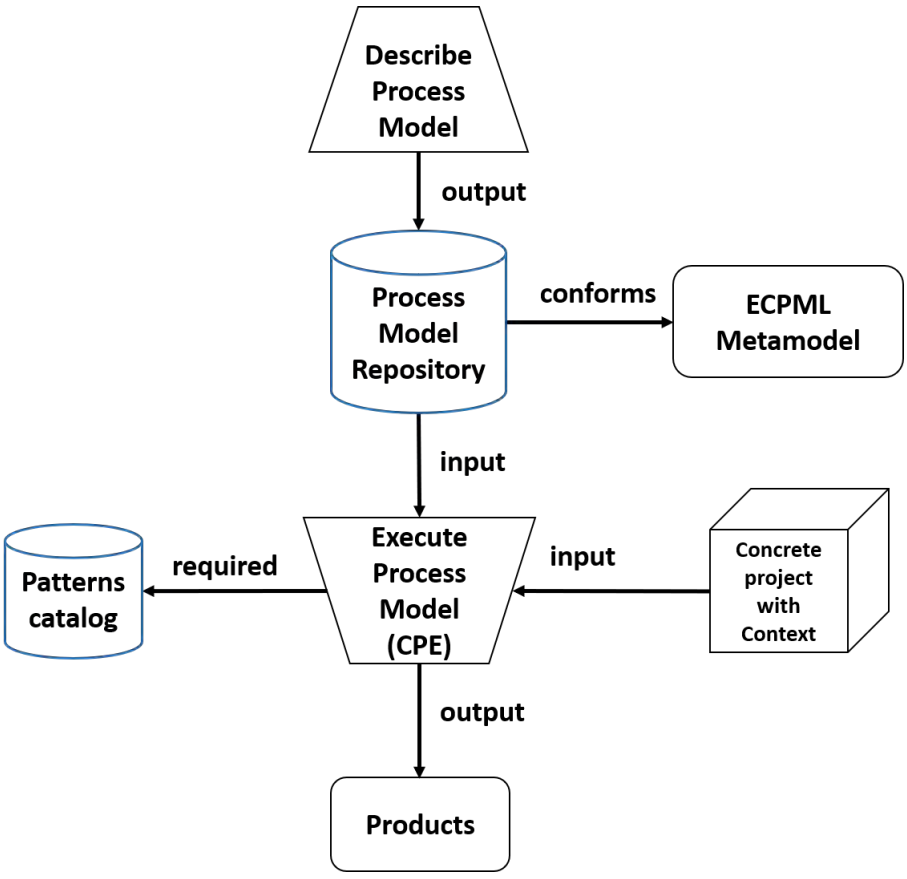


FIGURE 6.7: Démarche de mise en oeuvre d'un processus dans CPE.

6.4.1 Description de l'étude de cas

L'étude de cas consiste en la simulation de l'exécution du processus RUP-A pour analyser et concevoir un **système de gestion d'une mission** à l'étranger.

6.4.1.1 Présentation du processus RUP

RUP est un processus logiciel orienté objet bien connu décrivant les phases d'analyse et de conception d'un système avec UML sous forme d'itérations. Nous l'avons choisi car c'est un standard qui a été largement utilisé comme tel ou à travers des variantes dans des projets logiciels de taille et complexité variables. Il a pour but d'assurer la production de logiciels suivant des critères de qualité, de délai et de budget (Jacobson et al., 1999).

RUP présente un certain nombre de caractéristiques parmi lesquelles :

- Il permet d'améliorer la productivité de l'équipe de projet en regroupant tous les membres autour d'un langage commun.
- C'est un processus configurable, adaptable à la taille de l'équipe. Il convient donc effectivement aussi bien aux grandes qu'aux petites organisations.
- Il permet de créer et maintenir des modèles au lieu de se focaliser sur la production de documents.

RUP est un processus itératif composé de 4 phases : *Inception*, *Elaboration*, *Construction*, *Transition*. Si l'on considère la phase d'inception, les activités d'une itération de cette phase sont les suivantes (figure 6.8) :

- Manage the iteration
- Initiate a Project
- Define the System
- Determine Architectural Feasability

Dans cette étude de cas, nous mettons l'accent sur l'activité *"Define the system"* qui est composée de deux sous-activités :

- Find actors and use cases
- Develop a vision

Dans le reste de cette section, nous considérons l'activité *"Find actors and use cases"* qui est collaborative dès lors que le système à concevoir n'est pas simpliste.

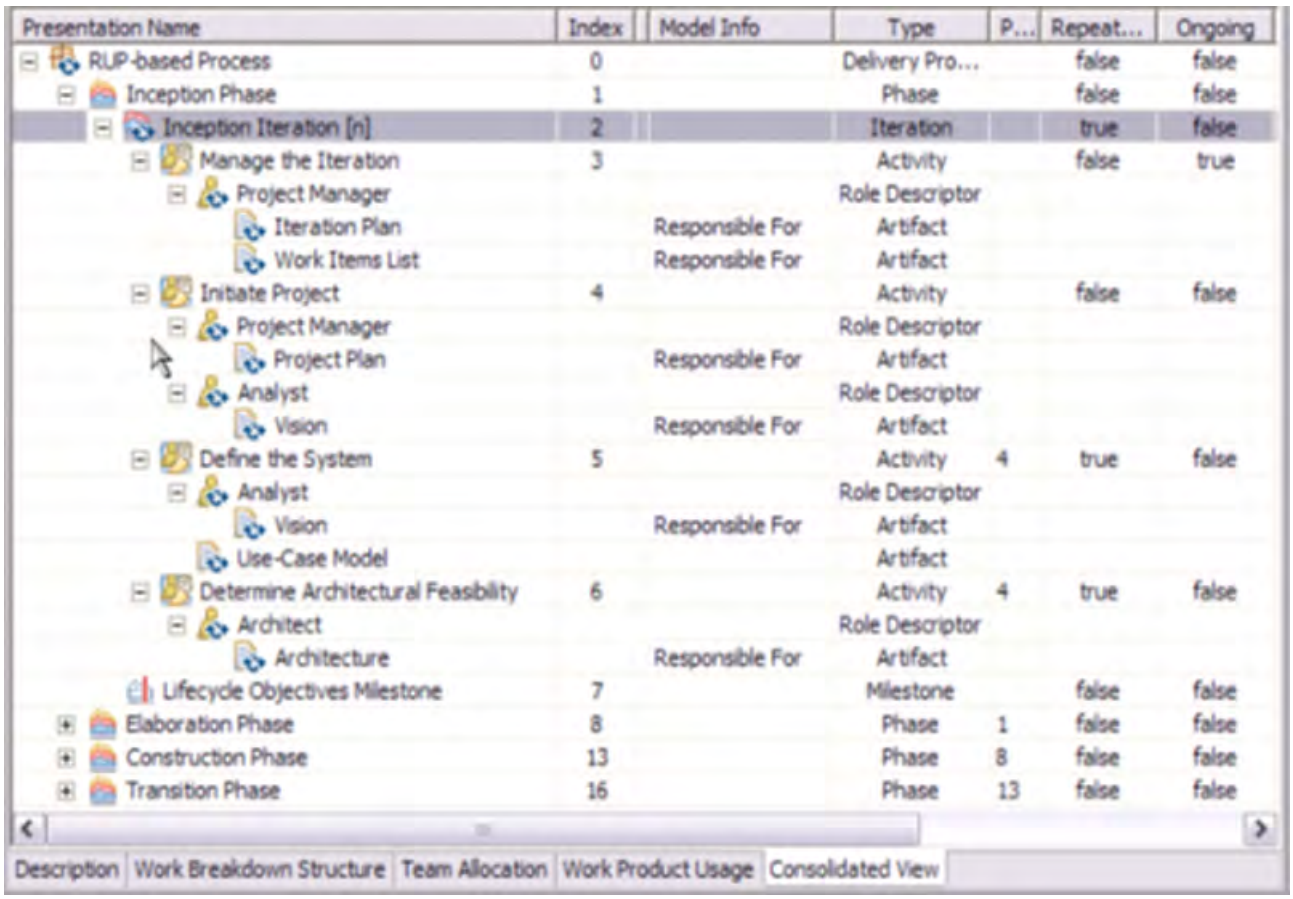


FIGURE 6.8: Différentes phases et activités du processus RUP.

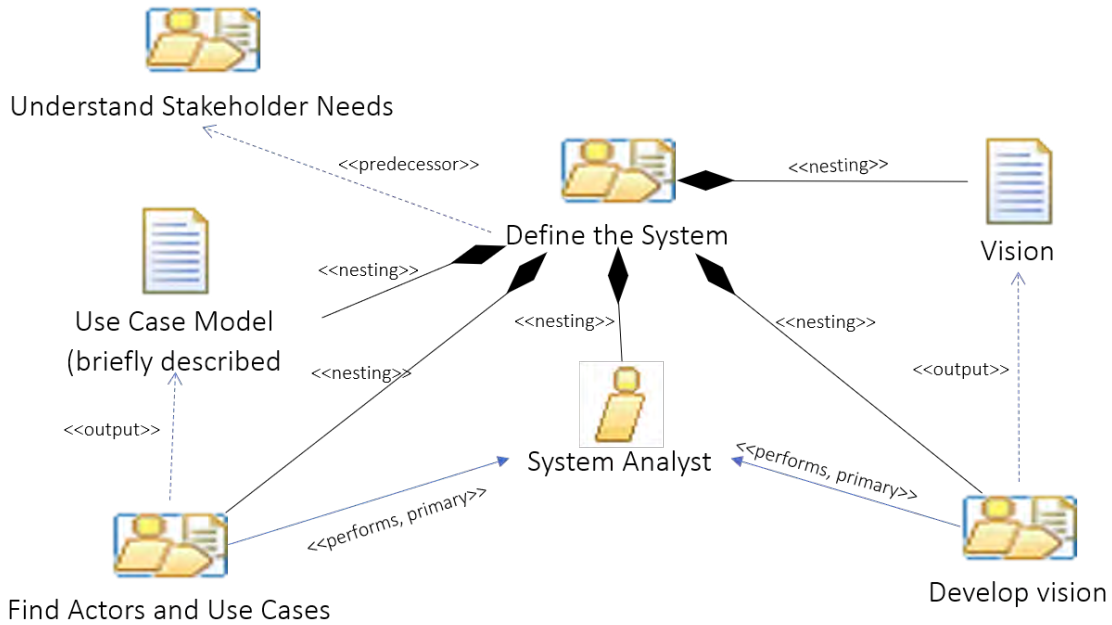


FIGURE 6.9: L'activité "Define the system" d'une itération RUP.

Suite aux expérimentations faites sur des projets avec divers groupes d'étudiants, nous avons légèrement adapté RUP sous une forme appelée *RUP-A (RUP Adapted)*. L'adaptation consiste essentiellement à ajouter la tâche *"Review the Use case model"* et à identifier les tâches collaboratives.

6.4.1.2 Tâches collaboratives de RUP-A

Dans RUP-A, l'activité *"Find actors and use cases"* est composée des tâches suivantes qui sont toutes potentiellement collaboratives :

- Find actors
- Find use cases
- Build the Use cases diagram
- Describe Use cases
- Review the Use cases model

L'exécution de ce processus donne donc lieu à des tâches collaboratives (désignées par CT dans la suite) comme vu précédemment dans ce manuscrit (voir chapitre 3).

Nous commençons par décrire les tâches collaboratives identifiées ci-dessus, exceptée *"Find actors"* que l'on peut considérer comme mono-instance dans un premier temps.

Pour chaque CT, nous décrivons : le *but général*, les *rôles* en charge de l'exécution, les *produits* en entrée-sortie. Le nombre d'instances d'une CT dépend généralement de la taille et de la complexité du système et résulte d'une décision du chef de projet qui tient compte du contexte : *ressources disponibles*, *contraintes temporelles*, ...

Tâche Collaborative : **Find Use Cases**

Objectif : identifier les cas d'utilisation (description informelle) en fonction de la compréhension des besoins des utilisateurs/clients

Rôle : Analyste

Produits en entrée : Acteurs du système, Exigences du système

Produits en sortie : Liste des cas d'utilisation identifiés

Tâche Collaborative : **Build the Use Cases Diagram**

Objectif : produire le diagramme UML des cas d'utilisation du système

Rôle : Analyste

Produits en entrée : Liste des cas d'utilisation identifiés (les acteurs du système étant inclus)

Produits en sortie : Diagramme des cas d'utilisation

Le nombre d'instances de cette CT dépend de la taille de la liste des cas d'utilisation en entrée, et bien sûr des contraintes en termes de ressources (dont le temps). L'exécution des instances peut se faire en séquentiel (le plus logique comme des raffinements successifs) ou en parallèle (moins évident pour un diagramme).

Tâche Collaborative : **Describe Use cases**

Objectif : Détailler la description des cas d'utilisation (identifiés par la tâche "Find use cases")

Rôle : Analyste

Produits en entrée : Exigences, Diagramme des cas d'utilisation

Produits en sortie : Scénarios des cas d'utilisation (nominal et exception) et diagrammes de séquence associés

Le nombre d'instances dépend de la décision du project manager en accord avec les participants à la tâche précédente et les analystes disponibles, les délais. Il peut aussi correspondre au nombre d'acteurs du système. Le travail peut se faire en séquentiel si le temps alloué à cette tâche le permet, mais le plus logique est de le faire faire en parallèle par plusieurs analystes.

Tâche Collaborative : **Review the Use Cases model**

Objectif : Faire une relecture du modèle des cas d'utilisation comprenant les acteurs, le diagramme de cas d'utilisation et les cas d'utilisations décrits

Rôle : Analyste (ou relecteur)

Produits en entrée : Modèle de cas d'utilisation composé des acteurs, des cas d'utilisation décrits et du diagramme de cas d'utilisation

Produits en sortie : document de relecture

Le nombre d'instances de cette CT dépend, entre autre, du nombre et de la complexité des cas d'utilisation à relire, de la disponibilité des relecteurs, du délai fixé, ...

Après avoir décrit les tâches collaboratives de RUP-A, nous montrons comment RUP-A peut s'appliquer à un système concret.

6.4.1.3 Utilisation de RUP-A pour la conception de l'application Gestion de Missions

Le système concret choisi ici est un système de gestion de missions à l'étranger. Considérons un tel système pour un employé d'une entreprise ou d'un institut de recherche, dans le but par exemple d'assister à une conférence scientifique.

Les acteurs (rôles) d'un tel système sont : *le missionnaire (l'employé)*, *la secrétaire (de l'institut d'appartenance du missionnaire)*, *le gestionnaire (financier)*.

Le missionnaire émet une demande (par mail) à la secrétaire du l'entreprise qui va préparer la mission et faire l'interface avec le gestionnaire financier. Une telle demande comporte un lieu, une date de départ, une date de retour, une justification du déplacement (par exemple : notification d'acceptation d'un article), un mode de transport souhaité (par exemple : avion), et un budget maximum.

La secrétaire prépare la mission en traitant la demande reçue du missionnaire. Pour cela elle cherche un billet d'avion, un hôtel, récupère le montant de l'inscription à la conférence, et fait une simulation financière. Si le montant de la mission dépasse le budget disponible, elle contacte le missionnaire en expliquant l'impossibilité de traiter sa demande, et le processus s'arrête. Si le budget est disponible, elle réserve le billet d'avion et l'hôtel. Elle produit ensuite un bon de commande contenant l'identité du missionnaire, le nom de son entreprise, les réservations faites pour le transport et le logement avec les montants associés, et le montant de l'inscription à la conférence. Elle envoie ce bon de commande au gestionnaire, en indiquant le numéro du compte de l'entreprise à débiter. On suppose que le missionnaire est déjà enregistré dans la base de donnée des missionnaires avec ses coordonnées bancaires.

Le gestionnaire établit, à l'aide du logiciel de comptabilité, un ordre de mission (OM), effectue l'achat du billet d'avion auprès de la compagnie d'aviation choisie, et paye l'inscription à la conférence. Il envoie l'OM au missionnaire qui devra l'avoir avec lui durant la mission.

Le missionnaire effectue ensuite la mission aux dates prévues dans l'OM. Il assiste à la conférence,

paye ses nuitées d'hôtel, et garde ses justificatifs de dépense (facture hôtel, factures des transports effectués dans le pays, repas).

Au retour de mission, le missionnaire remplit et signe un état de frais et le transmet, avec les justificatifs de sa mission, au gestionnaire. Celui-ci réalise le remboursement des frais par virement sur le compte du missionnaire, en débitant le compte de l'entreprise préalablement identifié.

Dans la suite, nous allons montrer comment nous utilisons le processus RUP-A pour la définition du système à concevoir. Les différentes tâches du RUP-A seront modélisées avec ECPML puis nous allons décrire leurs exécutions dans CPE.

6.4.1.4 Contexte d'application de la gestion de missions

Le contexte définit les éléments du projet qui influent sur la collaboration. Pour rappel, nous avons identifié trois éléments : la structuration des produits en entrée/sortie, l'intention et la disponibilité des ressources. Dans la suite, nous considérons que les acteurs jouant le rôle d'analyste, Alice, Bob et Eve sont tous disponibles. Nous décrivons donc le contexte concernant la structuration de chaque produit.

Produit "Requirements" : Les exigences du système représentent le cahier de charges décrit à la section 6.4.1.3. Ce document est décomposé en trois parties correspondant aux actions effectuées par chaque participant de la gestion d'une mission.

Produit "Use Cases List" : Etant donné qu'elle est issue des exigences, la liste des cas d'utilisation sera composée de plusieurs parties. Chaque partie correspond aux cas d'utilisation effectués par un acteur (missionnaire, sécrétaire, gestionnaire).

Produit "Use Cases Diagram" : Le système étudié ici étant assez simple, le diagramme de cas d'utilisation peut être non-composite et donc regrouper les cas d'utilisation listés en un seul diagramme.

Produit "Use Cases Description" : la description des scénarios des cas d'utilisation dépend des exigences du système. De ce fait, elle pourra être décomposée en autant de parties que le produit Requirements.

6.4.2 Simulation de l'exécution de l'étude de cas

Pour dérouler l'étude de cas, nous appliquons le processus RUP-A présenté ci-dessus. Dans les sections précédentes, nous avons identifié des tâches collaboratives qui pour rappel sont : *"Find actors"*, *"Find use cases"*, *"Build the Use cases diagram"*, *"Describe Uses cases"* et *"Review the Use cases model"*. La figure 6.10 illustre les différentes tâches du processus en ECPML.

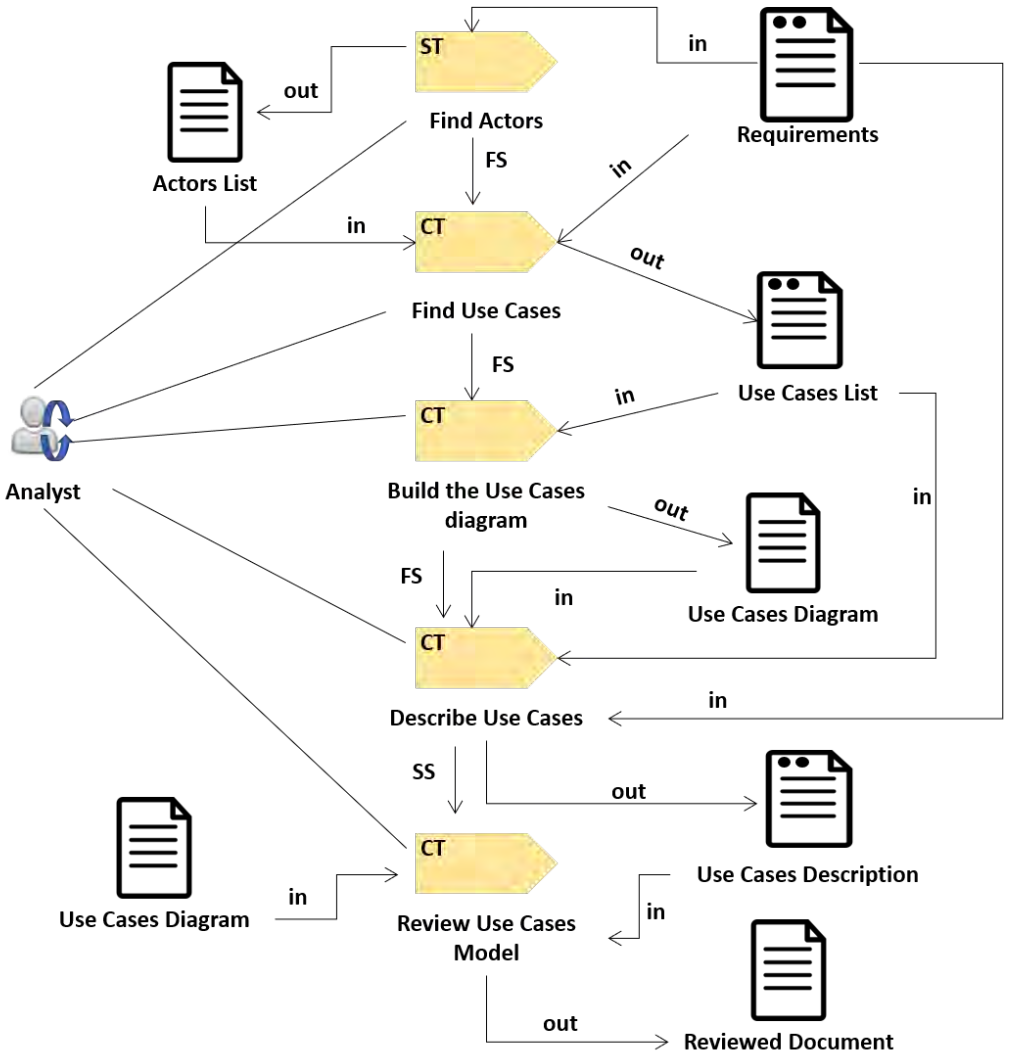


FIGURE 6.10: Modélisation en ECPML des tâches de l'activité de RUP-A "Find Actors and Use Cases".

Les différentes tâches de ce processus sont exécutées séquentiellement avec l'ordonnancement *"FinishToStart"* par le rôle *"Analyst"* sauf pour les deux dernières tâches qui sont exécutées avec l'ordonnancement *"StartTostart"*. Les tâches considérées ici sont collaboratives sauf la première tâche, *"Find Actors"*, qui sera réalisée par un seul acteur. De ce fait, nous les représentons avec la syntaxe concrète d'une tâche collaborative en ECPML.

Dans la suite, nous nous concentrons sur l'exécution des trois tâches collaboratives du processus.

6.4.2.1 Find Use Cases

Lors de l'exécution, le project manager commence d'abord par l'instanciation de la première tâche du process, "Find use cases". Au cours de cette instanciation, différents choix sont faits : *le nombre d'instances de la tâche* et *le patron de collaboration* utilisé pour le déploiement.

Dans la suite, nous considérons que le produit en entrée de cette tâche, *Requirements* et le produit en sortie, *Use Cases List*, sont composites. Nous supposons par exemple que les exigences ("Requirements") sont découpées en trois modules (ou composants) correspondant aux fonctionnalités de chaque acteur. La figure 6.11 présente ce découpage.

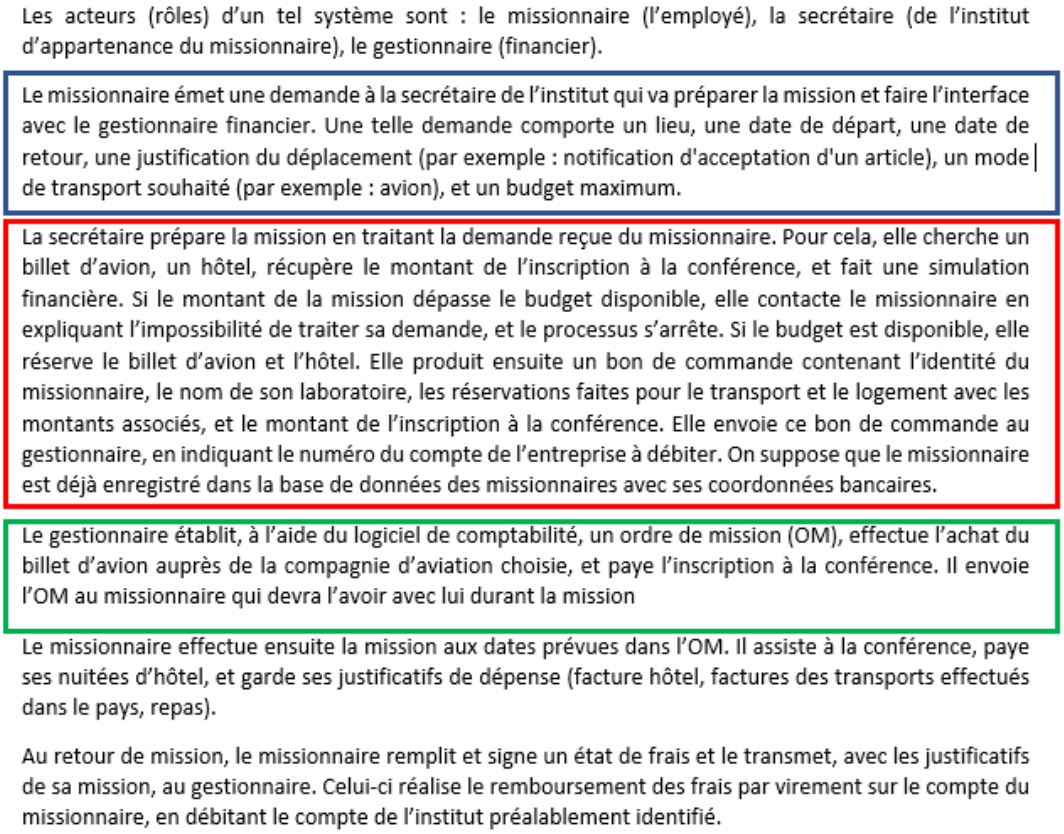


FIGURE 6.11: Découpage en composants du cahier de charges.

On suppose que le nombre de composants en entrée est égal au nombre de composants en sortie. De ce fait, la liste des cas d'utilisation ("Use Cases List") est composée de trois parties correspondant aux fonctionnalités de chaque partie du produit en entrée. Compte tenu de

cette structuration des produits, nous nous retrouvons dans le cas 7 (P_1 et P_2 composites sans dépendances) décrit au niveau du chapitre 4. Ce contexte rentre dans le champ d'application du patron *Effort Division*, qui peut donc être utilisé pour l'exécution de cette tâche. Chaque analyste identifie une liste de cas d'utilisation décrivant les exigences du composant qui lui a été confié.

La figure 6.12 décrit l'exécution de la tâche *Find Use Cases*. Le patron de collaboration utilisé, *Effort Division*, permet l'exécution de chaque instance sans ordonnancement avec les autres instances.

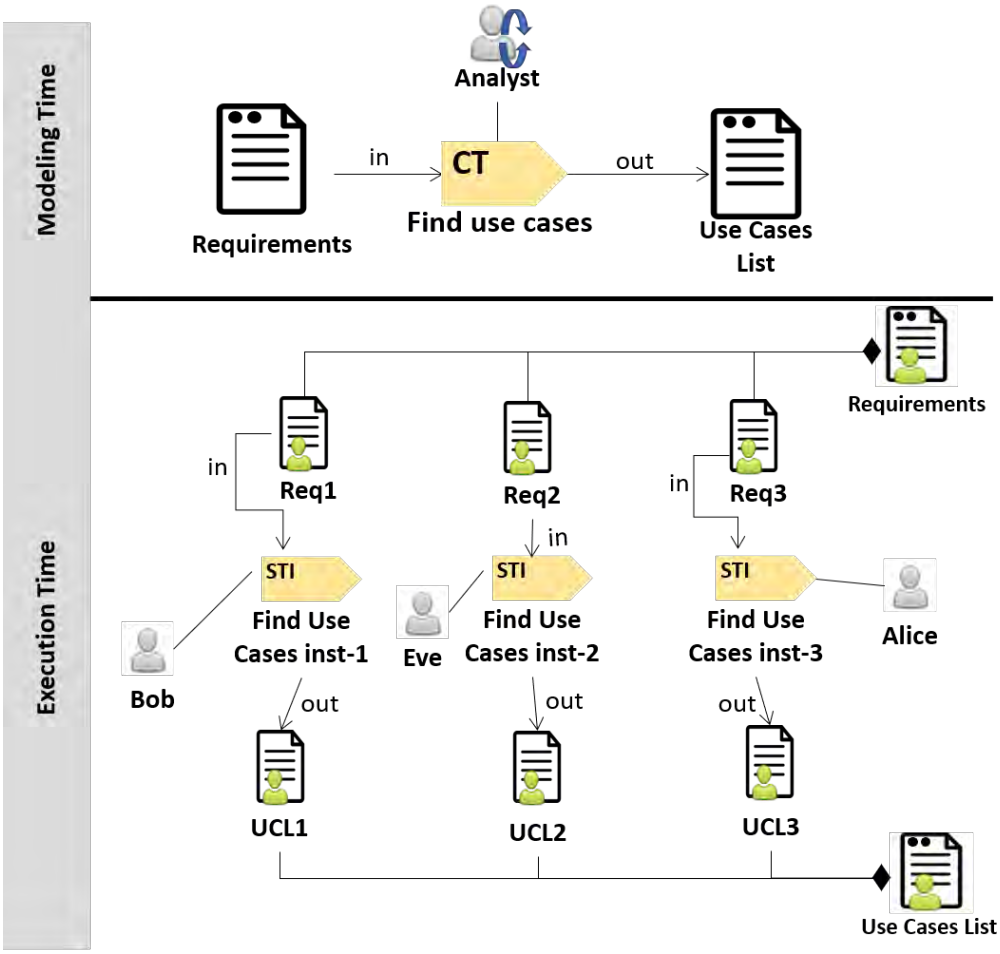


FIGURE 6.12: Illustration de l'instanciation de la tâche *Find use cases* avec le patron *Effort Division*.

De ce fait, Alice peut exécuter sa tâche en prenant une partie du produit en entrée sans attendre la fin de l'exécution des autres acteurs. Le choix de la partie à prendre peut se faire par affectation du project manager ou par un choix délibéré de l'acteur. Après exécution des instances de tâches, nous nous retrouvons avec le produit en sortie présenté sur la figure 6.13.

Dans cette figure, nous retrouvons les trois composants conformément à la décomposition des exigences.

FIGURE 6.13: Liste des cas d'utilisation obtenu en sortie de l'exécution de la tâche "Find Use Cases".

6.4.2.2 Build the Use Cases diagram

L'exécution de la tâche *"Build the Use Cases Diagram"* prend comme entrée la liste des cas d'utilisation identifiés. Elle produit en sortie le diagramme de cas d'utilisation. Etant donné que la liste des cas d'utilisation n'est pas grande pour ce système, le produit en sortie dans ce scénario n'est pas considéré comme composite.

Le produit en entrée de cette tâche est la liste des cas d'utilisation ("Use Cases List") définie précédemment et qui est composite. Compte tenu de la structuration des produits, il est possible de faire l'exécution en parallèle conformément au cas 5 (P_1 composite et P_2 non-composite) défini au chapitre 4. Dans cette situation, le patron appliqué est *Free Co-Work*. Considérant les trois instances de la tâche, le premier analyste, Bob, réalise une version du diagramme des cas d'utilisation en traitant les cas d'utilisation du composant 1, le second, Eve, participe au diagramme en ajoutant les cas d'utilisation du composant 2, Alice fait de même et ajoute les cas d'utilisation du composant 3.

La figure 6.14 illustre l'instanciation de la tâche *Build the Use Cases diagram* avec le patron choisi. Après exécution de cette tâche, le diagramme de cas d'utilisation obtenu est présenté à la figure 6.15.

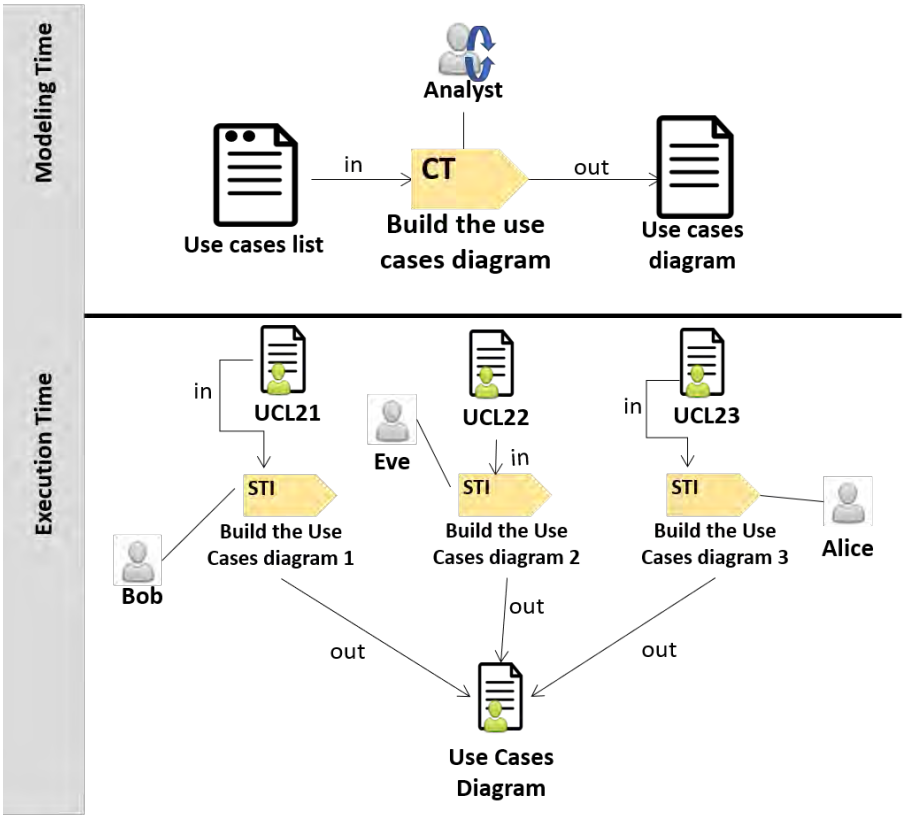


FIGURE 6.14: Illustration de l'instanciation de la tâche *Build the use cases diagram* avec le patron *Refinement*.

6.4.2.3 Describe Use Cases

Cette tâche permet de faire la description semi-formelle des cas d'utilisation. Elle prend en entrée le diagramme de cas d'utilisation produit à la tâche *Build the Use Cases diagram* ainsi que les exigences du système de gestion de mission (*Requirements*). Le produit en sortie est le document de scénarios des cas utilisation contenant le diagramme de séquence de chaque cas d'utilisation. Il est nommé ici *Use Cases Description* et est considéré composite étant donné que chaque acteur fait la description de la partie des exigences qui l'intéresse.

Le produit en entrée et le produit en sortie de cette tâche sont composites. Cette structuration des produits correspond au cas 7 comme précédemment. Dans ce contexte, nous pouvons à nouveau utiliser le patron de collaboration *Effort Division*. Comme pour la tâche Find use

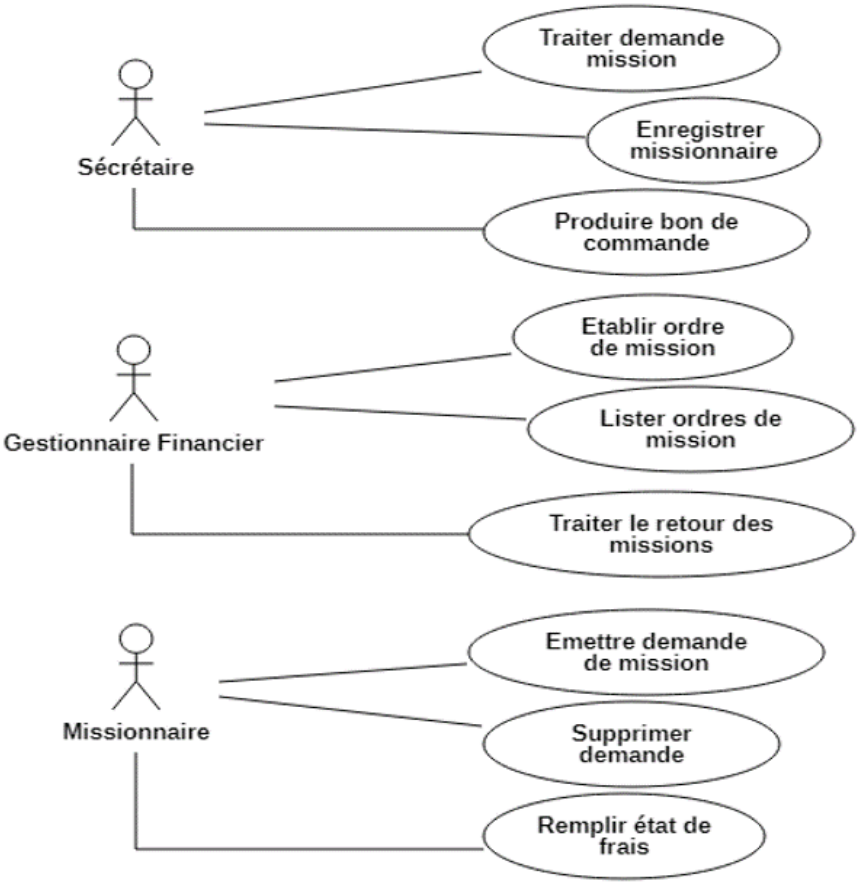


FIGURE 6.15: Diagramme de cas d'utilisation obtenu en sortie de l'exécution de la tâche *Build the use cases diagram* avec le patron *Free Co-Work*.

Cases, chaque analyste exécute une instance de tâche indépendamment des autres. La figure 6.16 illustre le résultat de l'instanciation avec le patron choisi. Dans cette figure, le produit en entrée est composée des trois composants du produit *Requirements*, *Req1, Req2* et *Req3*, ainsi que du produit *Use Cases diagram*.

Pour ne pas surcharger la présentation de cette étude de cas, nous ne donnons pas la description du produit concret *"Use Cases Description"* qui contient les scénarios et diagrammes de séquence associés aux cas d'utilisation du produit d'entrée *"Use Cases Diagram"*. Pour les mêmes raisons de taille de cette étude de cas, nous ne décrivons pas l'exécution de la tâche collaborative *"Review Use Cases Model"* qui suit le même principe que les autres tâches collaboratives présentées.

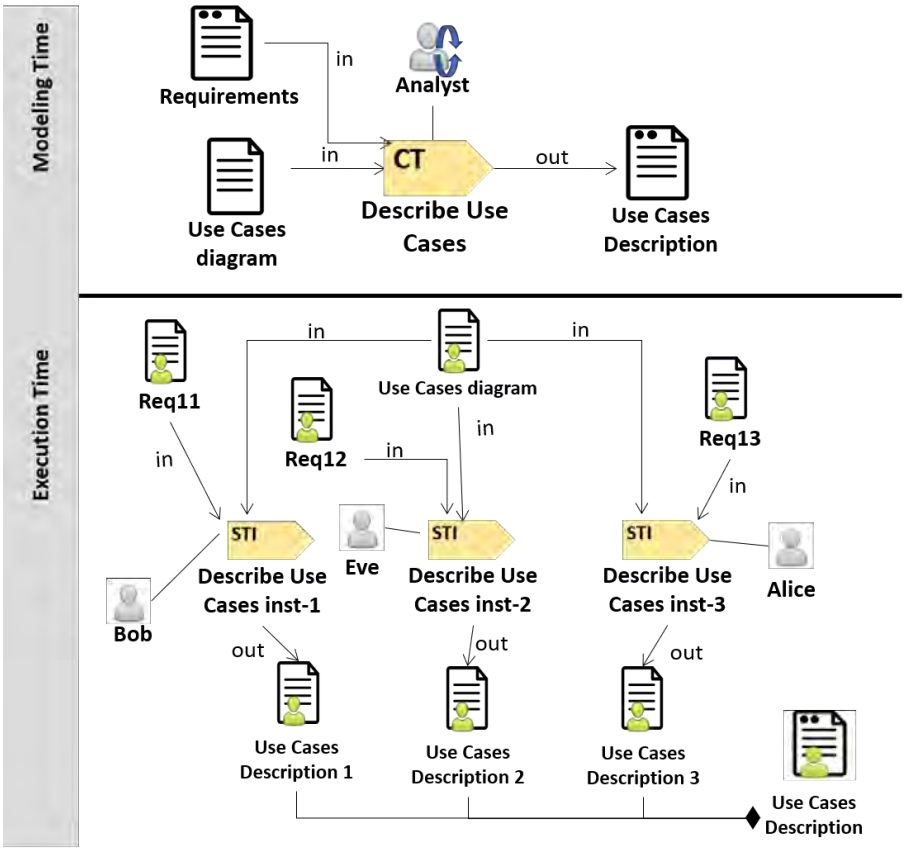


FIGURE 6.16: Illustration de l'instanciation de la tâche *Describe use cases* avec le patron *Effort Division*.

6.4.3 Illustration de l'exécution avec l'outil CPE

Dans cette section, nous décrivons l'exécution du processus de notre étude de cas "Gestion d'une mission" avec CPE, en nous appuyant sur la simulation décrite ci-dessus. Nous allons illustrer les différentes étapes de notre démarche, de la soumission d'un modèle de processus jusqu'au suivi de l'exécution par le project manager. Nous rappelons que les produits concrets de l'application de gestion d'une mission ne sont pas gérés par CPE mais par des outils externes. De ce fait, ils apparaissent comme des références dans les interfaces montrées ci-après.

6.4.3.1 Instanciation du processus

Le processus considéré est celui représenté par la figure 6.10. Il décrit les différentes tâches de l'activité "Find actors and Use Cases" décrites ci-dessus permettant d'aboutir à la définition des cas d'utilisation de l'application de gestion d'une mission.

Le modèle de processus considéré ici est représenté en XML comme montré sur la figure 6.17.

En effet, comme décrit dans les sections précédentes, notre prototype prend, actuellement, un modèle de processus au format XML [1] qui est ensuite analysé pour en déduire les tâches et produits à instancier dans le système.

```
<Process type="process" id="P1" name="RUP">
    <Task type="SingleTask" name="Find Actors" id="FA1ST" linkToSuccessor="1" />
    <Task type="CollaborativeTask" name="Find Use Cases" id="FUC1CT" linkToSuccessor="2"
    linkToPredecessor="1"/>
    <Task type="CollaborativeTask" name="Build the Use Cases Diagram" id="BUC1CT" linkToSuccessor="3"
    linkToPredecessor="2" />
    <Task type="CollaborativeTask" name="Describe Use Cases" id="DUC1CT" linkToSuccessor="4"
    linkToPredecessor="3" />
    <Task type="CollaborativeTask" name="Review Use Cases Model" id="RUC1CT" linkToPredecessor="4" />

    <TaskSequence id="1" predecessor="FA1ST" successor="FUC1CT" linkKind="FinishToStart"/>
    <TaskSequence id="2" predecessor="FUC1CT" successor="BUC1CT" linkKind="FinishToStart"/>
    <TaskSequence id="3" predecessor="BUC1CT" successor="DUC1CT" linkKind="FinishToStart"/>
    <TaskSequence id="4" predecessor="DUC1CT" successor="RUC1CT" linkKind="StartToStart"/>

    <WorkProduct name="Requirement" id="REQ" isComposite="true"/>
    <WorkProduct name="Use Cases List" id="UCL" isComposite="true"/>
    <WorkProduct name="Use Cases Diagram" id="UCD" isComposite="false"/>
    <WorkProduct name="Uses Cases Description" id="SED" isComposite="true"/>
    <WorkProduct name="Reviewed Document" id="REV" isComposite="false"/>

    <TaskParameter id="1" task="FUC1CT" product="REQ" direction="in"/>
    <TaskParameter id="2" task="FUC1CT" product="UCL" direction="out"/>

    <TaskParameter id="3" task="BUC1CT" product="UCL" direction="in"/>
    <TaskParameter id="4" task="BUC1CT" product="UCD" direction="out"/>

    <TaskParameter id="5" task="DUC1CT" product="UCD" direction="in"/>
    <TaskParameter id="6" task="DUC1CT" product="SED" direction="out"/>

    <TaskParameter id="7" task="RUC1CT" product="UCD" direction="in"/>
    <TaskParameter id="8" task="RUC1CT" product="REV" direction="out"/>
</Process>
```

FIGURE 6.17: Modèle du processus RUP-A décrit en XML.

La première étape de l'exécution consiste en l'instanciation du modèle de processus soumis. Lors de la soumission du modèle dans l'interface illustrée par la figure 6.4, l'ensemble des liens décrits dans le modèle sont générés par CPE.

Après cela, la liste des tâches est présente au niveau de notre plateforme web avec l'état *created* en attendant l'instanciation (voir figure 6.18). De cette interface, nous pouvons voir qu'il est possible d'instancier une tâche (lien *"Instantiate"*), de voir ses instances (*"See instances"*) ou de changer de patron (*"Change pattern"*) sauf pour la première tâche, "Find Actors", qui n'est pas collaborative.

1. A terme CPE pourra lire en entrée des modèles de processus décrits en ECPML

FIGURE 6.18: Liste des tâches du processus RUP-A.

6.4.3.2 Instanciation de la tâche collaborative : Find Use Cases

Lors de l'instanciation de la tâche collaborative *Find Use Cases*, nous choisissons le patron *Effort Division* comme vu dans la simulation ci-dessus et illustré par la figure 6.12.

Les patrons de collaboration étant des modèles de processus, ils sont aussi décrits au format XML pour être analysés au niveau de la plateforme. Nous présentons dans l'annexe A la description de chaque patron au format XML. Le project manager peut par la suite choisir les trois composants du produit de sortie à développer (voir figure 6.19), ce qui donne le nombre d'instances de la tâche collaborative. Dans cette figure, le project manager choisit aussi les produits en entrée de chaque instance de la tâche, c'est-à-dire les différentes parties du document d'exigences illustrées à la figure 6.11.

Après application du patron *Effort Division*, la figure 6.20 illustre le résultat obtenu avec les différents liens entre instances de tâche collaborative sous la base de données Neo4J.

Chaque noeud *UCL (pour Use Case List)* correspond à une partie du produit de sortie présentée à la figure 6.13. Nous retrouvons les acteurs (Alice, Bob, Eve) pour chaque instance de la tâche collaborative Find Use Cases.

6.4.3.3 Démarrage de l'exécution et contrôle des instances de tâche

La figure 6.21 montre le démarrage de l'exécution d'une instance de tâche. Cette figure illustre le fait que, en raison du choix du patron Effort Division, il n'y a pas d'ordonnancement entre les différentes instances. De ce fait, chaque tâche peut être exécutée au moment voulu par l'acteur

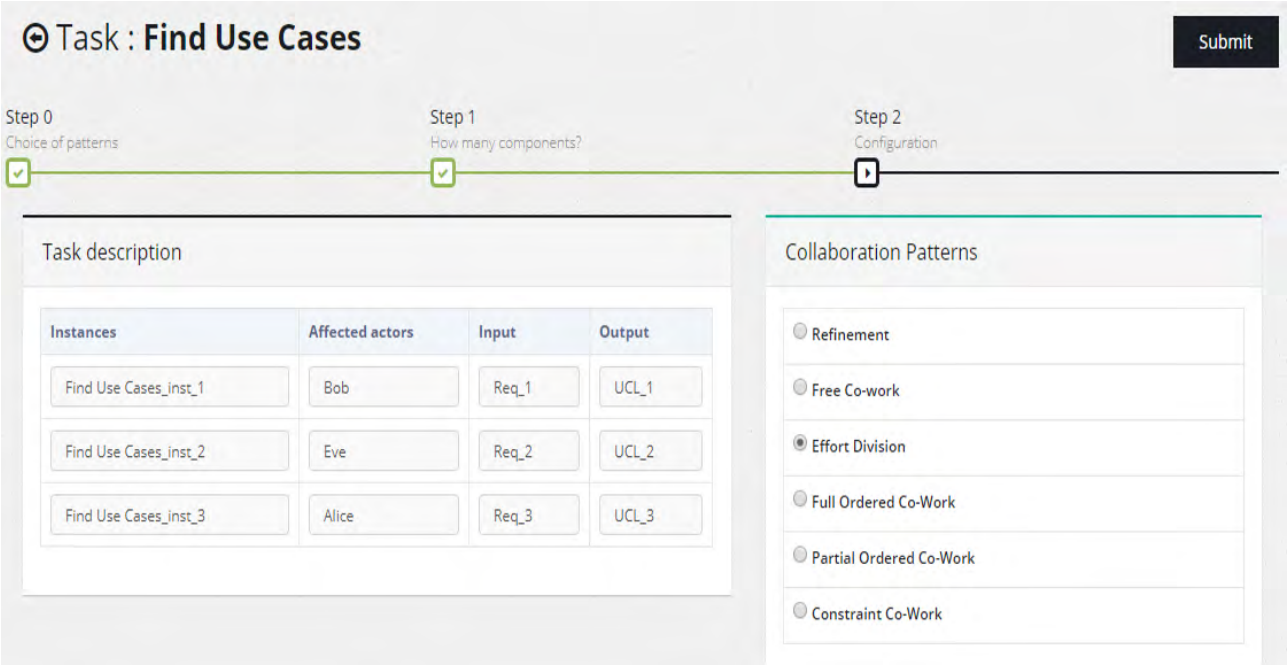


FIGURE 6.19: Choix d'un patron de collaboration et assignation des acteurs et produits pour la tâche Find use cases.

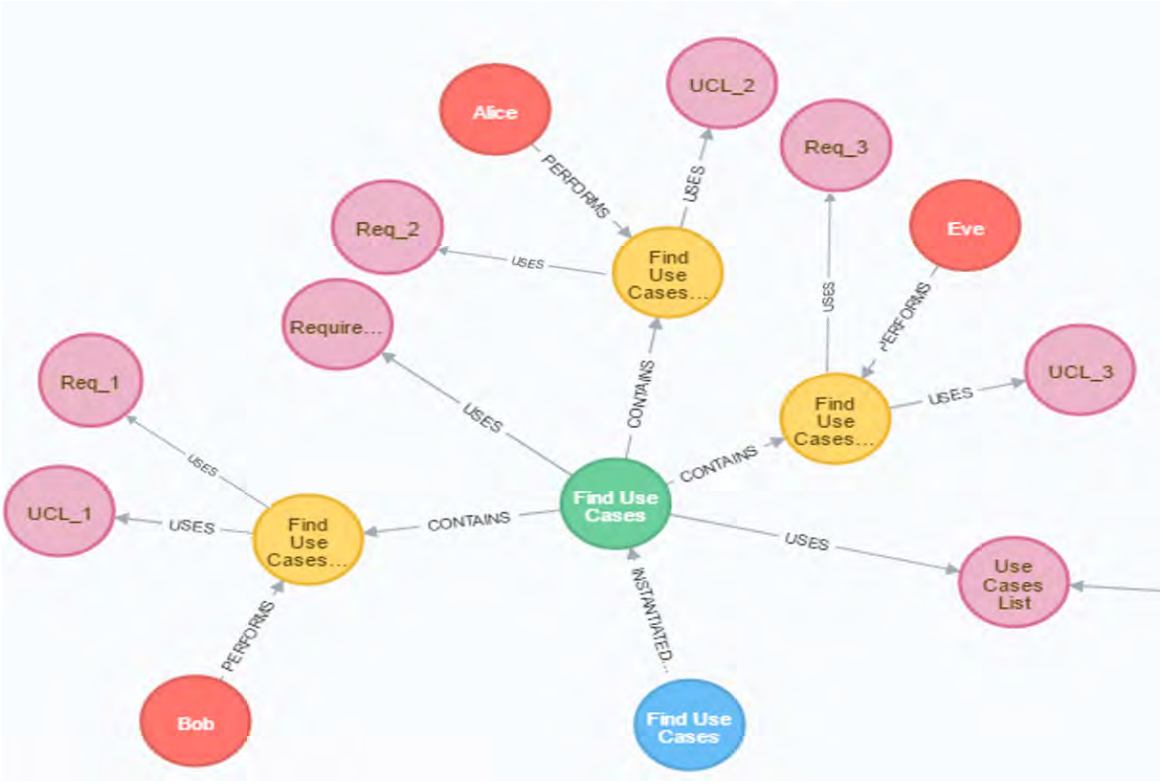


FIGURE 6.20: Résultat sous forme de graphe de l'instanciation de la tâche Find Use cases.

assigné. Dans cet exemple, Alice peut exécuter l'instance qui lui est affectée sans incidence sur les deux autres instances restantes.

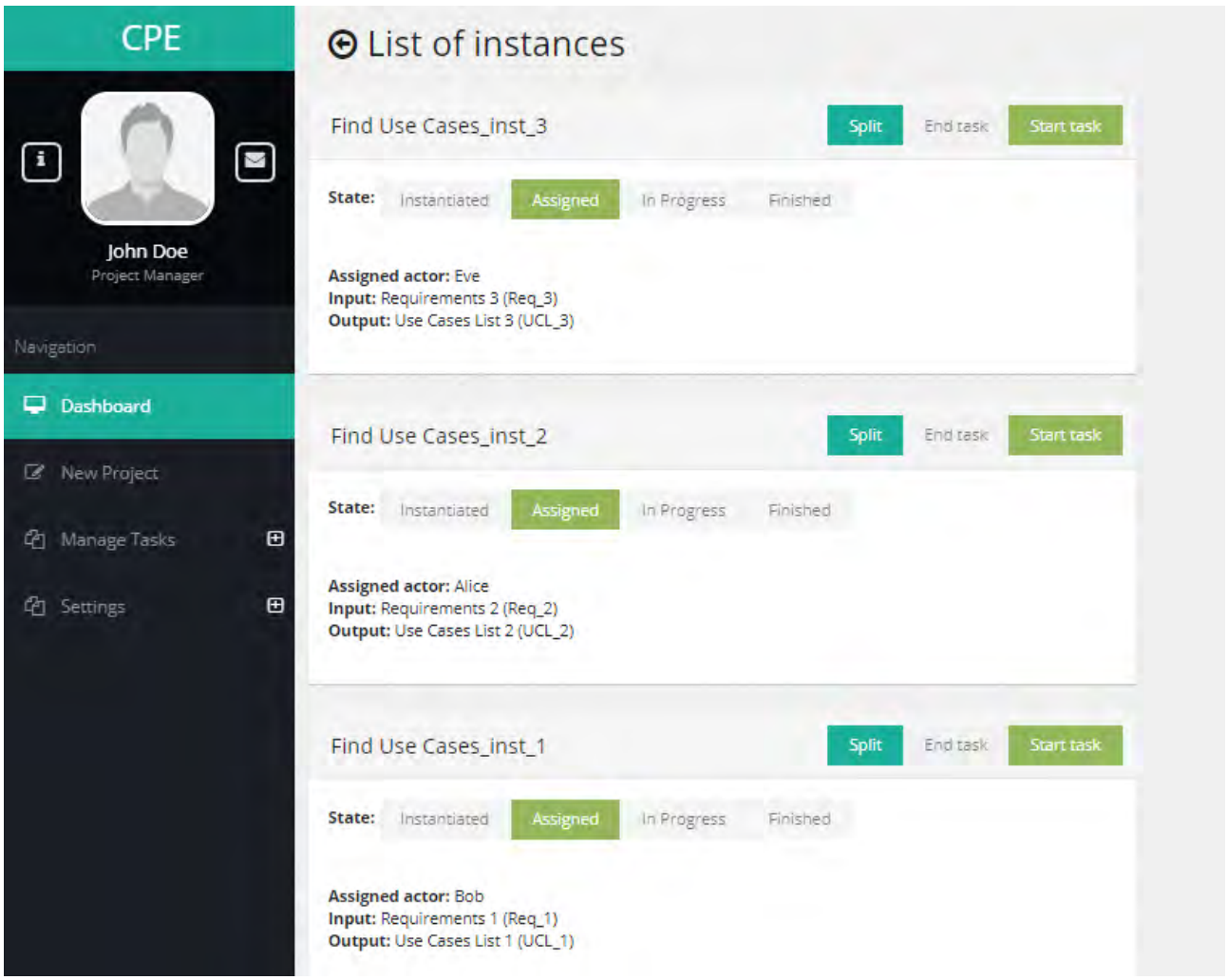


FIGURE 6.21: Liste des instances de la tâche Find Use Cases avec leur état courant.

Dans cette figure, nous retrouvons pour chaque instance de tâche, le produit en entrée et le produit en sortie réalisé conformément à la simulation faite. Etant en parallèle, chaque tâche peut démarrer grâce au bouton *Start task*. L'option est aussi donnée au project manager de rendre une instance collaborative avec la commande *split* (cf. figure 6.21).

6.4.3.4 Instanciation et exécution de la tâche collaborative : Build the Use Cases diagram

Après exécution de la tâche Find Use Cases, la tâche Build Use Cases diagram utilise la liste des cas d'utilisation (figure 6.13) pour produire le diagramme de cas d'utilisation. Le contexte souhaité ici est que le produit en entrée est composite et le produit en sortie est non-composite. De ce fait, le patron choisi est le patron de collaboration *Free Co-work* comme présenté par la

figure 6.14.

L'instanciation de cette tâche suit les mêmes étapes que la tâche précédente. La figure 6.22 présente la liste des instances obtenues après instanciation avec les produits en entrée (les composants de Use Cases List) et le produit en sortie (l'unique Use Cases diagram). Dans cette

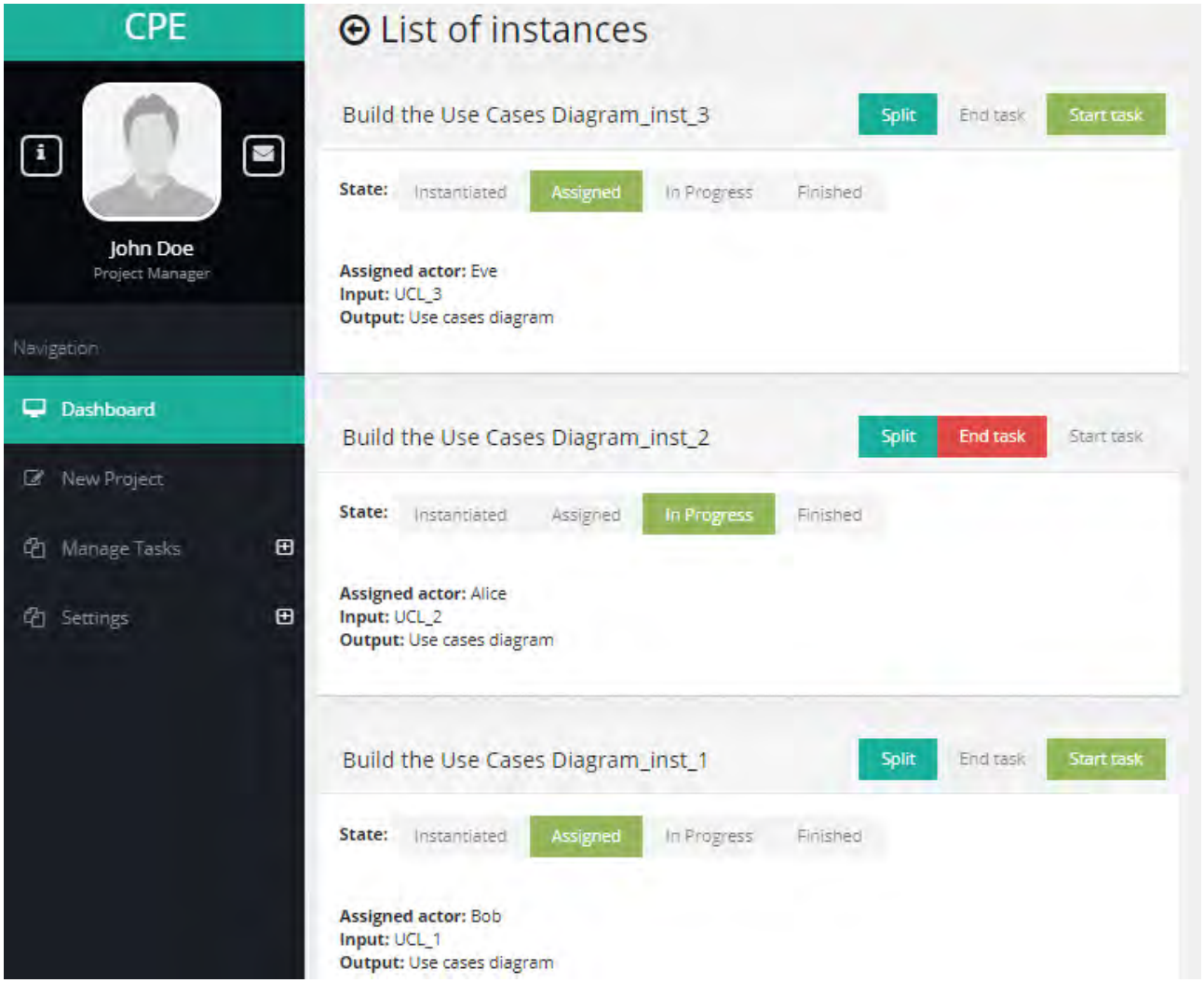


FIGURE 6.22: Liste des instances de la tâche Build the Use Cases Diagram avec leur état courant.

figure, nous constatons que la seconde instance, *Build the Use Cases Diagram _inst_2* a démarré son exécution et donc est à l'état *InProgress*. Après exécution des trois instances de tâche, le diagramme de cas d'utilisation obtenu est celui representé sur la figure 6.15.

6.4.3.5 Instanciation et exécution de la tâche collaborative : Describe the Use Cases

L'exécution de la tâche *Describe the Use Cases* se fait après celle de la tâche *Build the Use Cases diagram*. Durant cette exécution, les produits en entrée utilisés sont les exigences du système (composite) et le diagramme de cas d'utilisation (non-composite) obtenu précédemment. Le produit en sortie, *use cases description*, est composite conformément à la simulation faite précédemment. De ce fait, le patron utilisé ici *Effort Division* permet une exécution en parallèle des différentes instances décrites dans la figure 6.23.

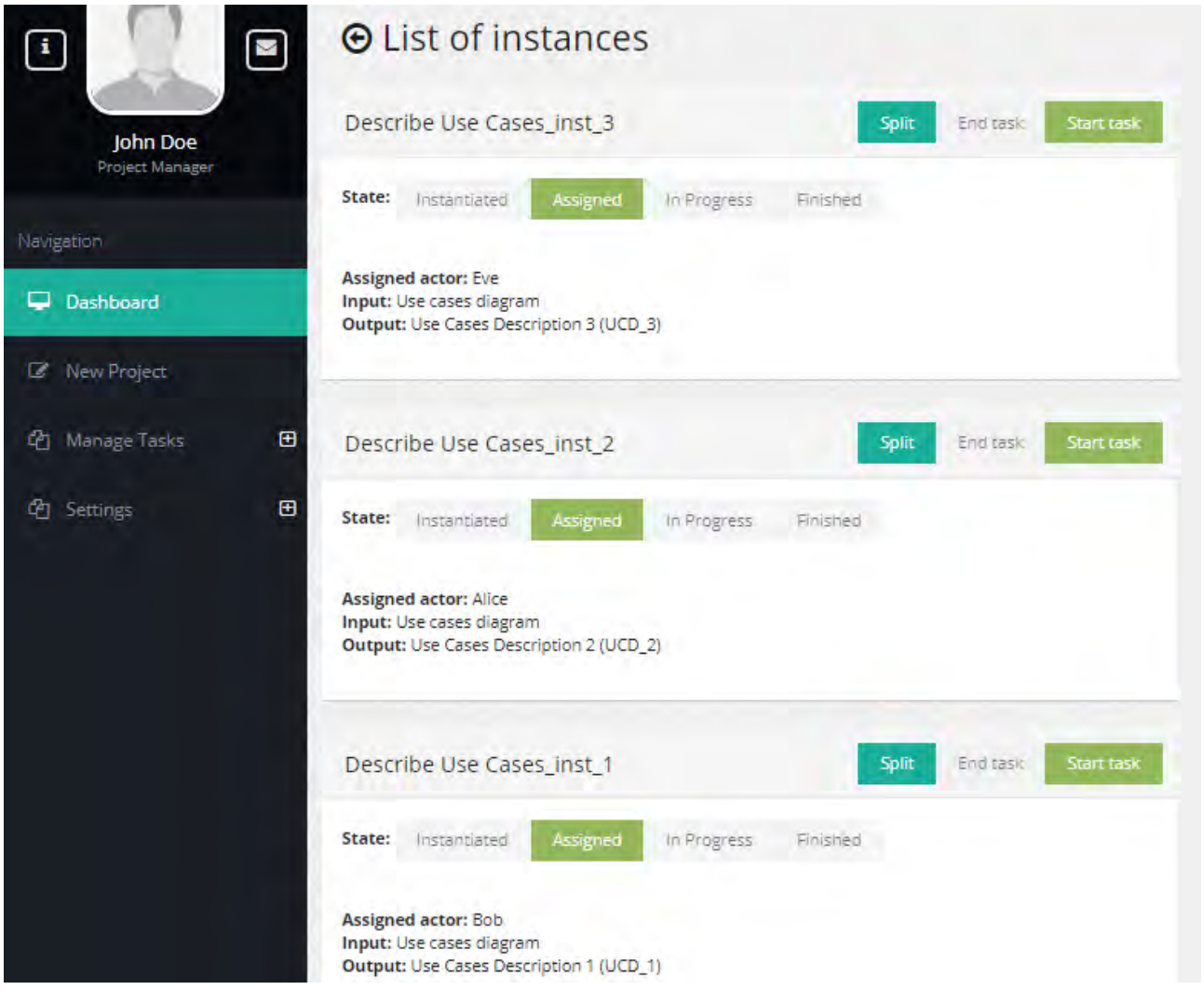


FIGURE 6.23: Liste des instances de la tâche Describe Use Cases avec leur état courant.

6.4.3.6 Evolution d'une instance de la tâche Find Use Cases en tâche collaborative

Durant l'exécution, comme nous l'avons vu dans le chapitre 3, il peut arriver que l'une des instances d'une tâche ait besoin de devenir collaborative pour diverses raisons, par exemple pour avoir recours à l'expertise d'un acteur supplémentaire. Dans notre cas, supposons qu'Alice ait besoin d'un autre analyste, Victor, pour l'aider dans sa tâche. La commande *split* permet de décomposer l'instance de tâche afin de la rendre collaborative, et donc nécessite patron de collaboration pour son déploiement. Si nous considérons que Alice et Victor vont travailler en séquentiel, Alice démarrant et Victor utilisant le produit fourni par Alice pour l'affiner, nous pouvons appliquer le patron *Refinement*. L'évolution d'une STI en CTI se fait suivant les mêmes étapes que l'instanciation d'une tâche. Cela veut dire qu'il faut aussi assigner un patron de collaboration à la nouvelle CTI et choisir les produits en entrée comme en sortie. Ces étapes sont présentées sur la figure 6.24. Cette nouvelle CTI prend comme entrée la deuxième composante des exigences (requirements 2).

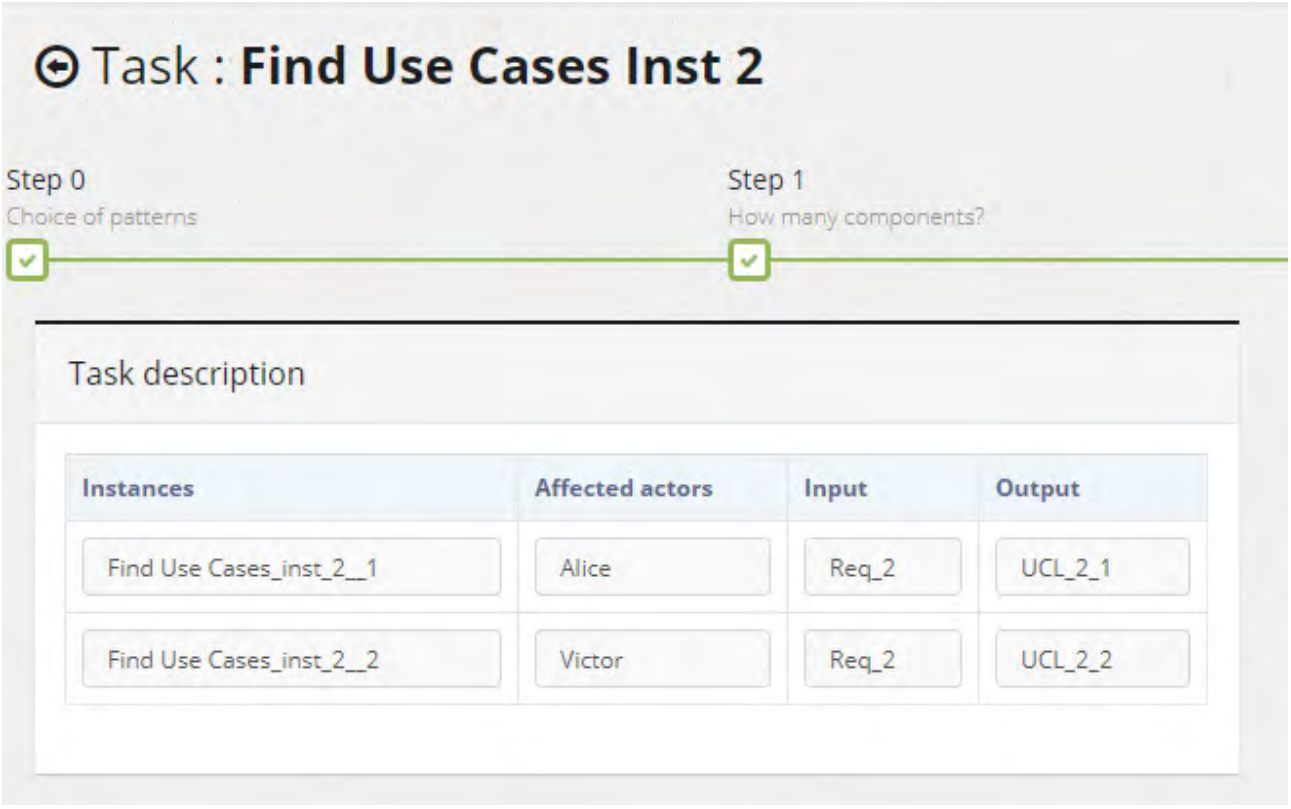


FIGURE 6.24: Instanciation de la nouvelle tâche collaborative issue de l'évolution de Find Use Cases inst_2.

La nouvelle interface d'affichage des instances obtenue est illustrée par la figure 6.25. Dans

cette figure, nous pouvons voir la nouvelle tâche collaborative *Find use cases inst_2* à droite avec de nouvelles options permettant notamment de voir la liste de ses instances après son instanciation.

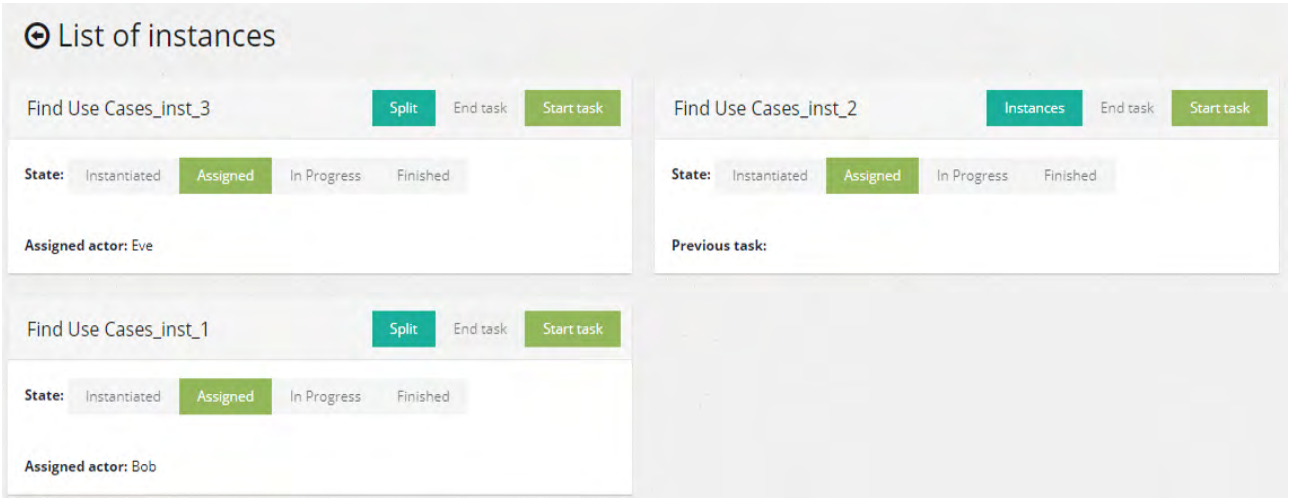


FIGURE 6.25: Nouvelle liste des instances après évolution en CTI d'une instance de la tâche Find Use cases.

La liste des instances de cette nouvelle tâche collaborative ainsi obtenue est représentée sur la figure 6.26.

Etant donné le patron refinement, qui implique un ordonnancement séquentiel, Victor ne peut démarrer sa tâche tant qu'Alice n'a pas fini la sienne. Cela est indiqué par la couleur grisée du bouton "Start task" au niveau de l'instance de tâche de Victor et le libellé *"Previous task"* qui précise qu'effectivement la tâche précédant la tâche de Victor est bien celle d'Alice. Après exécution de cette nouvelle tâche collaborative, le produit de sortie obtenu est la seconde partie de la liste des cas d'utilisation avec la participation de Victor maintenant (voir figure 6.27).

6.5 Discussion et limites de la validation

L'objectif de ce chapitre était de valider notre approche à travers un outil support opérationnel et une étude de cas significative. Pour l'étude de cas, nous avons utilisé le processus RUP-A qui est un processus collaboratif représentatif et qui a fait ses preuves dans le domaine de la conception logicielle. Nous avons montré comment certaines tâches de ce processus pouvaient être exécutées de manière collaborative dans le cadre d'une application de gestion de missions.

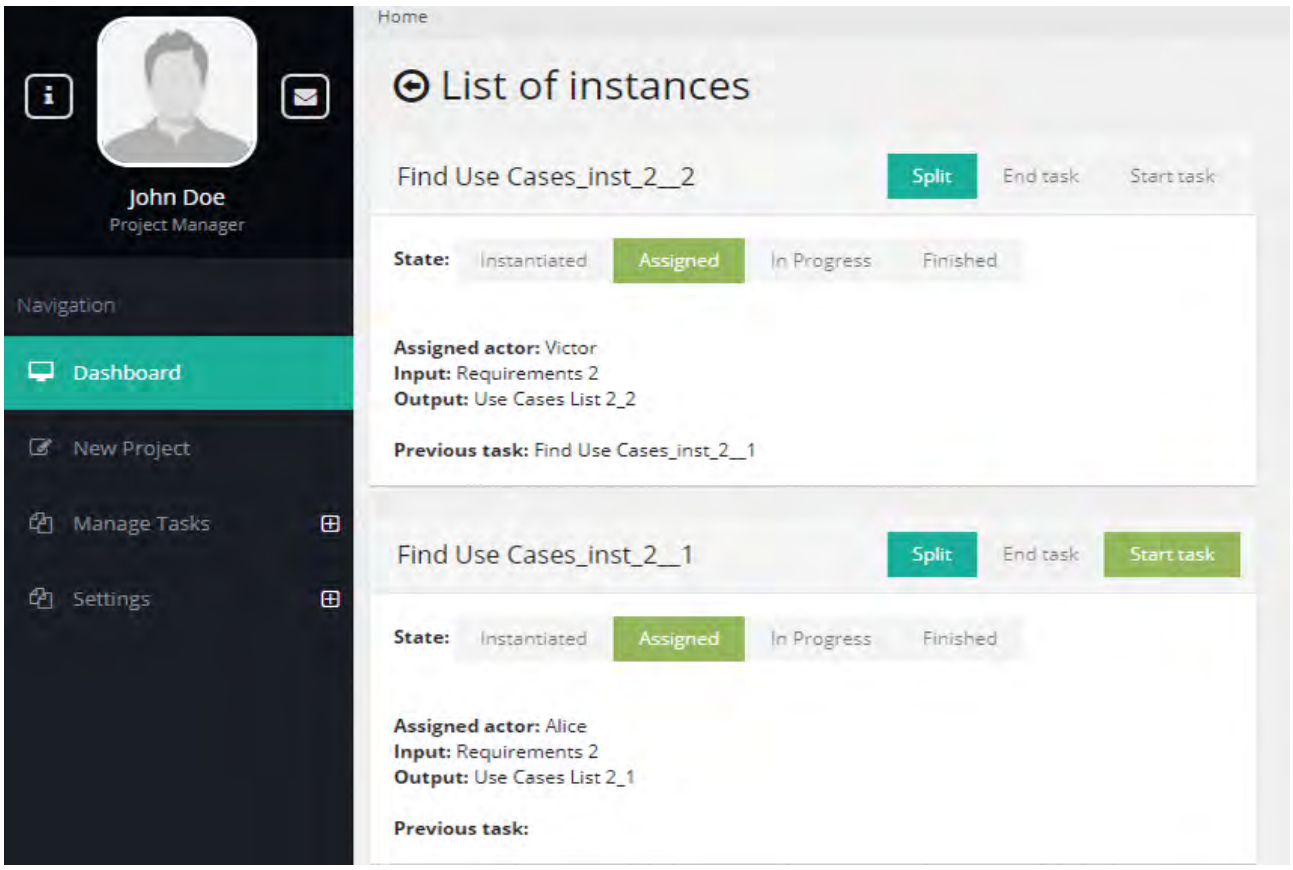


FIGURE 6.26: Liste des instances de la nouvelle tâche collaborative après évolution.

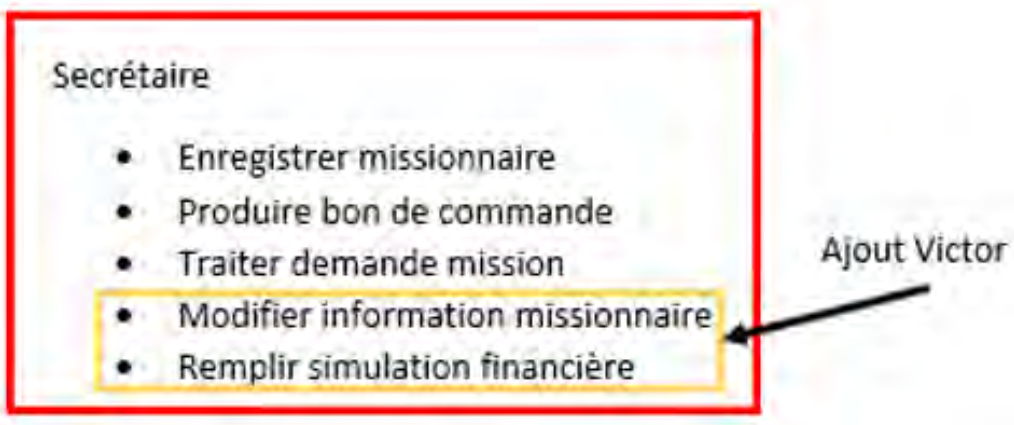


FIGURE 6.27: Résultat obtenu en sortie de l'exécution de la nouvelle tâche collaborative.

Grâce à ECPML, l'exécution des différentes tâches du RUP-A peut se faire de façon flexible. En effet, il n'est pas nécessaire dès le début de définir de façon rigide tout le cheminement de l'exécution. Le focus est mis sur le contexte d'exécution permettant de connaître la nature des produits d'entrée et de sortie et la disponibilité des acteurs.

Les principales fonctionnalités de CPE ont été implémentées. Nous avons proposé un algorithme

pour l'instanciation d'une tâche collaborative par l'intermédiaire d'un patron de collaboration. Toutefois, cet algorithme est limité aux stratégies de collaboration actuellement définies. De plus, notre approche n'a pas été testée avec un réel projet industriel. Cela tient notamment à la difficulté d'obtenir des informations détaillées concernant le processus de déroulement d'un projet industriel.

6.6 Conclusion

Ce chapitre a tout d'abord présenté l'implémentation réalisée pour supporter notre approche. Le prototype CPE a été développé, avec pour objectif de supporter les différentes étapes de l'exécution d'un processus. La principale étape au niveau de ce prototype est l'instanciation d'une tâche collaborative avec un patron de collaboration. Après instanciation, chaque instance évolue conformément à une machine à états définissant son comportement comme défini dans le chapitre 5.

L'utilisation des patrons montre la pertinence d'avoir une approche flexible. Grâce à cela, nous sommes capables de gérer dynamiquement les relations entre les STIs d'une tâche collaborative et de les faire évoluer en cours d'exécution. Cette évolution suit les différents changements de contexte qui peuvent se produire (exemples : décomposition d'un produit pour le rendre composite, défection d'un acteur, ajout d'un acteur). Une fois la tâche instanciée, nous pouvons gérer l'exécution des différentes STIs en accord avec l'ordonnancement défini par le patron.

Une étude de cas a été développée pour valider notre démarche. Cette étude de cas a consisté à dérouler un sous-ensemble du processus RUP-A pour la conception d'une application de gestion de missions. L'activité de RUP-A sur laquelle nous nous sommes concentrée est *Find Actors and Use Cases*. Nous avons déroulé cette tâche au sein de CPE pour illustrer notre approche.

Chapitre 7

Conclusion

Sommaire

Ce travail de thèse a porté sur la modélisation et l'exécution flexible des processus collaboratifs. La problématique abordée dans ce travail de recherche concerne la mise en oeuvre de la collaboration forte au sens où plusieurs acteurs jouant le même rôle collaborent pour exécuter une tâche donnée. Traditionnellement, la façon de collaborer est détaillée dans le modèle de processus. Cela a pour conséquence un manque de flexibilité durant l'exécution. En effet, cette solution ne permet pas de s'adapter aux changements de contexte. De ce fait, il est important de proposer une gestion à la fois fine et flexible des tâches collaboratives. Pour traiter cette problématique, nous avons identifié deux questions de recherche principales : (1) Comment représenter une stratégie de collaboration qui définisse, à l'exécution, les relations entre les instances d'une tâche collaborative ? (2) Comment permettre à une tâche collaborative de choisir dynamiquement sa stratégie de collaboration durant l'exécution, pour s'adapter aux changements de contexte ?

Notre travail a porté sur une conceptualisation de la collaboration (question de recherche 1) au sein d'un processus, et son exécution flexible permettant de choisir dynamiquement la stratégie de collaboration (question de recherche 2).

Dans les sections suivantes, nous résumons les contributions de cette thèse (section 7.1). Nous

présentons aussi les limites de ce travail (section 7.2). Enfin, des perspectives ou directions de recherche sont proposées afin d'étendre ce travail de recherche (section 7.3).

7.1 Contributions de notre travail

L'étude que nous avons mené a été conduite sur trois axes : (1) proposer une formalisation de la collaboration à travers le langage de modélisation et d'exécution de processus *ECPML* ; (2) identifier différents contextes d'exécution et un ensemble de patrons de collaboration associés ; (3) proposer une sémantique opérationnelle permettant l'exécution flexible de processus collaboratifs.

Un langage de modélisation et d'exécution de processus

Notre contribution dans le cadre du premier axe a été présentée dans le chapitre 3. Elle a consisté à définir un langage de modélisation et d'exécution de processus collaboratifs, *ECPML (Executable Collaborative Process Modeling Language)*. Pour cela, nous nous sommes inspirés de SPEM pour définir les concepts permettant de représenter un processus. Nous avons jugé pertinent d'introduire des notions permettant de gérer la multi-instanciation d'une tâche, en particulier la notion de tâche collaborative (*CollaborativeTask*). Chacun de ces éléments est instancié durant l'exécution du processus. Pour pouvoir contrôler finement l'exécution d'un processus, nous avons introduit dans ECPML un paquetage dédié décrivant les instances des éléments de processus exécutables.

Patrons de collaboration pour une exécution flexible de processus

Les relations entre les éléments instanciés d'un processus collaboratif ne sont pas toujours figées et peuvent changer en fonction du contexte d'exécution d'un projet. En conséquence, nous avons identifié différentes stratégies permettant de mettre en oeuvre la collaboration dans divers contextes. Les contextes d'exécution varient en fonction du projet mais aussi des tâches d'un projet. C'est dans ce cadre que porte notre deuxième contribution dans ce travail de thèse.

Parmi les éléments de contexte d'une tâche, nous considérons la structuration des produits en entrée, l'intention derrière la tâche et la disponibilité des ressources. Ces éléments ont été détaillés dans le chapitre 4, où suivant les variations possibles, nous avons défini dix cas de contexte. Pour chaque cas des stratégies d'exécution ont été formalisées en tant que patrons de

collaboration. Chaque patron constitue donc un modèle comportemental permettant de décider des relations entre les instances d'une tâche collaborative en tenant compte des éléments de contexte.

Une sémantique opérationnelle à base de machines à états

Notre contribution dans le cadre du troisième axe porte sur une sémantique opérationnelle pour formaliser le cycle de vie des éléments d'un processus collaboratif.

La sémantique opérationnelle est fondée sur des machines à états dont certaines transitions prennent en paramètre des patrons de collaboration. Les actions de certaines transitions utilisent ces patrons pour définir dynamiquement les relations intra-instances d'une tâche collaborative avec le mécanisme de late-binding.

Sur la base de cette sémantique, nous avons défini le comportement du "process engine" pour exécuter des processus modélisés avec ECPML. Dans cet objectif, nous avons défini des algorithmes pour les phases d'instanciation et d'assignation du processus ainsi que pour l'évolution des instances de tâche du processus.

Un environnement supportant l'exécution par utilisation de patrons de collaboration

Pour concrétiser et valider notre approche, nous avons réalisé le prototype CPE (*Collaborative Process Engine*) qui permet le déploiement et l'exécution d'un processus. CPE a été développé entièrement en JavaScript, sous la forme d'une plateforme web. D'une part, la plateforme intègre des fonctionnalités permettant d'analyser un modèle de processus et d'en tirer les informations nécessaires.

CPE fournit trois fonctions principales :

— instancier un processus à partir d'un modèle de processus,
— instancier des tâches collaboratives avec des choix de patrons de collaboration,
— dérouler l'exécution du processus suivant les patrons de collaboration choisi et le séquencement des différentes instances de tâche du processus.

7.2 Limites de notre approche

Un point limitant de notre approche est la prise en compte partielle du contexte d'exécution. Dans notre approche, nous considérons les aspects les plus évidents qui influencent le choix de la stratégie de collaboration. Mais il est très difficile d'avoir une exhaustivité au niveau des scénarios possibles. Cependant, la définition de patrons de collaboration dépendant d'un contexte défini, il est possible de définir de nouveaux éléments de contexte qui pourront orienter la stratégie de collaboration. Par exemple, en considérant, pour une tâche collaborative donnée, la différence entre le nombre de composants du produit en entrée et le nombre de composants du produit en sortie, de nouveaux contextes pourraient être identifiés et servir à définir de nouveaux patrons de collaboration.

Une autre limitation de notre approche est le fait qu'elle ne prend pas en compte la charge d'un acteur dans la réalisation d'une tâche. La prise en compte de la charge permettrait de définir de manière plus précise quand un acteur peut passer à l'état *performing*. L'estimation de la durée de la tâche pourrait aussi impacter le cycle de vie d'un acteur.

Même si notre proposition présente un caractère générique et facilement adaptable, la validation par des cas d'étude industriels reste la plus appropriée pour mesurer les points forts et les points faibles d'une approche telle que la nôtre et sa capacité à passer à l'échelle. Notre cas d'étude de validation a été limité à une simulation d'un processus qui certes est utilisé dans l'industrie, mais qui ne permet pas de produire une réelle validation expérimentale.

Enfin, d'un point de vue outillage, CPE est un prototype qui reste à améliorer. Par exemple, le seul profil d'accès, à ce jour, est celui du project manager. Il est nécessaire de permettre aux autres acteurs du processus de pouvoir suivre l'exécution du processus et, par exemple, de commencer, par eux même, l'exécution de leur instance de tâche.

7.3 Perspectives

Le travail effectué dans cette thèse peut être poursuivi dans plusieurs directions. La plupart de nos perspectives portent sur le traitement des limites mentionnées ci-dessus.

Sur le plan conceptuel, nous envisageons d'approfondir les points suivants :

Des éléments de contexte pour des patrons de collaboration. Nous avons mis en évidence deux types d'ordonnancement principaux : *séquentiel* et *parallèle*. Cependant vu le nombre limité de patrons de collaboration défini, il serait intéressant de définir de nouveaux éléments de contexte influant sur la collaboration. Un exemple serait la différence entre le nombre de composants des produits en entrée et en sortie ou encore la complexité ou la durée d'une tâche.

La définition d'autres patrons de collaboration pour couvrir, par exemple, des processus autour de l'agilité. Il serait intéressant de définir de nouveaux patrons de collaboration pour l'exécution de modèles de processus dans plus de contextes. Ainsi, la définition de nouveaux patrons permettrait d'étendre l'utilisation de notre approche.

La définition de métriques pour caractériser l'apport d'une approche d'exécution collaborative comme la nôtre. Ces métriques serviront à mesurer l'impact de l'utilisation de cette approche. Nous pourrons ainsi produire des mesures quantitatives sur le gain apporté par la collaboration. Cela servirait à justifier d'avantage l'importance de nos contributions et l'intérêt de notre approche pour une utilisation industrielle.

La mise en place d'un moteur de recommandation. Automatiser le choix d'un patron de collaboration permettrait d'aider le project manager durant l'exécution. Pour ce faire, il est nécessaire de formaliser la définition d'un contexte de projet. Cette formalisation permettrait d'avoir un DSL (*Domain-Specific Language*) de définition de contexte. Ce langage permettrait de bénéficier d'une souplesse dans la définition de nouveaux éléments de contexte.

Dans cette perspective, il serait possible d'utiliser des algorithmes de recommandation se basant sur le contexte d'utilisation d'un patron.

Sur le plan de l'outillage, nous envisageons les travaux suivants :

L'interconnexion avec des outils de développement. Une autre perspective à explorer pourrait porter sur l'intégration d'outils de développement existants du style versioning. En effet, en intrégrant des outils comme *GIT*, il serait possible de créer des branches (ou dépôts) pour chaque acteur d'une tâche. Les différentes branches permettront aux acteurs d'avoir un espace de travail personnel. Cette création de branches pourait se faire à l'instanciation de la tâche.

La réalisation d'un éditeur de modèles de processus. Pour éviter d'avoir à utiliser le format XML pour les modèles de processus, il serait intéressant d'intégrer un éditeur

au niveau de CPE. L'éditeur permettrait de pouvoir modéliser directement et graphiquement un processus collaboratif en utilisant le langage ECPML.

Annexes

A Représentation XML des patrons de collaboration

Cette annexe fournit les formalisations XML des patrons de collaboration que nous avons définis. Pour rappel les patrons que nous avons définis au chapitre 4 sont au nombre de six :

- Le patron *Refinement*
- Le patron *Free Co-Work*
- Le patron *Constraint Co-Work*
- Le patron *Effort Division*
- Le patron *Full Ordered Co-Work*
- Le patron *Partial Ordered Co-Work*

A.1 Patron Refinement

La figure 7.1 montre le format XML du patron de collaboration *Refinement*. Pour rappel, ce patron est utilisé pour créer ou modifier un produit par raffinage successif.

A.2 Patron Free Co-Work

La figure 7.2 montre le format XML du patron de collaboration *Free Co-Work*. Ce patron est utilisé pour créer un produit à plusieurs sans avoir de contraintes sur l'ordre d'exécution.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Process SYSTEM "Pattern.dtd">
<Process type="patron" name="Refinement">
    <TaskInstance name="t11" id="t11" linkToSuccessor="1" />
    <TaskInstance name="t12" id="t12" linkToPredecessor="1" />

    <TaskInstanceSequence id="1" predecessor="t11" successor="t12" linkKind="FinishToStart"/>

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>
    <WorkProductInstance name="P2" id="p2" isComposite="false"/>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>

    <TaskInstanceParameter id="3" taskinstance="t11" product="p2" direction="out"/>
    <TaskInstanceParameter id="4" taskinstance="t12" product="p2" direction="in"/>
    <TaskInstanceParameter id="5" taskinstance="t12" product="p2" direction="out"/>
</Process>
```

FIGURE 7.1: Représentation en XML du patron de collaboration Refinement.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Process type="patron" id="P1" name="Free Co-work">
    <TaskInstance name="t11" id="t11" />
    <TaskInstance name="t12" id="t12" />

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>
    <WorkProductInstance name="P2" id="p2" isComposite="false"/>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>

    <TaskInstanceParameter id="3" taskinstance="t11" product="p2" direction="out"/>
    <TaskInstanceParameter id="4" taskinstance="t12" product="p2" direction="out"/>
</Process>
```

FIGURE 7.2: Représentation en XML du patron de collaboration Free Co-Work.

A.3 Patron Constraint Co-Work

La figure 7.3 montre le format XML du patron de collaboration *Constraint Co-Work*. Ce patron est une variante de *Free Co-Work* où des contraintes portent sur le début ou sur la fin de la tâche.

A.4 Patron Effort Division

La figure 7.4 illustre le patron *Effort Division*, au format XML. Ce patron est utilisé lors de la production d'un produit de sortie composite.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Process type="patron" id="P1" name="Constraint Co-work">
    <TaskInstance name="t11" id="t11" linkToSuccessor="1"/>
    <TaskInstance name="t12" id="t12" linkToPredecessor="1"/>

    <TaskInstanceSequence id="1" predecessor="t11" successor="t12" linkKind="StartToStart"/>

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>
    <WorkProductInstance name="P2" id="p2" isComposite="false"/>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>

    <TaskInstanceParameter id="3" taskinstance="t11" product="p2" direction="out"/>
    <TaskInstanceParameter id="4" taskinstance="t12" product="p2" direction="out"/>
</Process>
```

FIGURE 7.3: Représentation en XML du patron de collaboration Constraint Co-Work.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Process SYSTEM "Pattern.dtd">
<Process type="patron" id="" name="Effort Division">
    <TaskInstance name="t11" id="t11"/>
    <TaskInstance name="t12" id="t12"/>

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>

    <WorkProductInstance name="P2" id="p2" isComposite="true" nestedInstances="[p21,p22]">
        <WorkProductInstance name="P21" id="p21" isComposite="false"/>
        <WorkProductInstance name="P22" id="p22" isComposite="false"/>
    </WorkProductInstance>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>

    <TaskInstanceParameter id="3" taskinstance="t11" product="p21" direction="out"/>
    <TaskInstanceParameter id="4" taskinstance="t12" product="p22" direction="out"/>
</Process>
```

FIGURE 7.4: Représentation en XML du patron de collaboration Effort Division.

A.5 Patron Full Ordered Co-Work

La figure 7.5 représente le patron *Full Ordered Co-Work* au format XML. Il sert à la production de différents composants d'un produit composite avec des composants dépendants.

A.6 Patron Partial Ordered Co-Work

La figure 7.6 représente le patron *Partial Ordered Co-Work* au format XML. Il sert à la production de différents composants d'un produit avec des dépendances entre certaines parties de ce produit.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Process SYSTEM "Pattern.dtd">
<Process type="patron" id="" name="Full Ordered Co-Work">
    <TaskInstance name="t11" id="t11" linkToSuccessor="1" />
    <TaskInstance name="t12" id="t12" linkToPredecessor="1"/>

    <TaskInstanceSequence id="1" predecessor="t11" successor="t12" linkKind="FinishToStart"/>

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>

    <WorkProductInstance name="P2" id="p2" isComposite="true" nestedInstances="[p21,p22]">
        <WorkProductInstance name="P21" id="p21" isComposite="false"/>
        <WorkProductInstance name="P22" id="p22" isComposite="false" impactedWPI="p21"/>
    </WorkProductInstance>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>

    <TaskInstanceParameter id="3" taskinstance="t11" product="p21" direction="out"/>
    <TaskInstanceParameter id="4" taskinstance="t12" product="p21" direction="in"/>
    <TaskInstanceParameter id="5" taskinstance="t12" product="p22" direction="out"/>
</Process>
```

FIGURE 7.5: Représentation en XML du patron de collaboration Full Ordered Co-Work.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Process SYSTEM "Pattern.dtd">
<Process type="patron" id="" name="Partial Ordered Co-Work">
    <TaskInstance name="t11" id="t11" linkToPredecessor="1"/>
    <TaskInstance name="t12" id="t12" linkToPredecessor="1"/>
    <TaskInstance name="t13" id="t13"/>

    <TaskInstanceSequence id="1" predecessor="t11" successor="t12" linkKind="FinishToStart"/>

    <WorkProductInstance name="P1" id="p1" isComposite="false"/>

    <WorkProductInstance name="P2" id="p2" isComposite="true" nestedInstances="[p21,p22,p23]">
        <WorkProductInstance name="P21" id="p21" isComposite="false"/>
        <WorkProductInstance name="P22" id="p22" isComposite="false" impactedWPI="p21"/>
        <WorkProductInstance name="P23" id="p23" isComposite="false"/>
    </WorkProductInstance>

    <TaskInstanceParameter id="1" taskinstance="t11" product="p1" direction="in"/>
    <TaskInstanceParameter id="2" taskinstance="t12" product="p1" direction="in"/>
    <TaskInstanceParameter id="3" taskinstance="t13" product="p1" direction="in"/>

    <TaskInstanceParameter id="4" taskinstance="t11" product="p21" direction="out"/>
    <TaskInstanceParameter id="5" taskinstance="t12" product="p21" direction="in"/>
    <TaskInstanceParameter id="6" taskinstance="t12" product="p22" direction="out"/>
    <TaskInstanceParameter id="7" taskinstance="t13" product="p23" direction="out"/>
</Process>
```

FIGURE 7.6: Représentation en XML du patron de collaboration Partial Ordered Co-Work.

B Compléments sur la description des patrons de collaboration

Pour chaque tâche collaborative, il existe différents contextes d'exécution dans un projet donné. Chacun de ces contextes a été présenté dans la section 4.2 sous forme de dix cas qui donnent lieu à diverses stratégies de collaboration. Chacun des cas correspond à une configuration des produits en entrée et sortie d'une tâche. En effet, la structuration d'un produit (composite ou non, avec des dépendances partielles/totales entre ses composants le cas échéant) impacte fortement le choix du patron de collaboration à appliquer pour mettre en oeuvre une tâche collaborative. Ici, nous détaillons les modèles d'exécution des patrons pour les cas de contextes qui n'ont pas été présentés dans le chapitre 4.

B.1 Patron Refinement : Cas 6

Dans la figure 7.7, nous présentons le patron *"Refinement"* dans le contexte où le produit en entrée est composite avec des dépendances et le produit en sortie est non-composite. Le modèle d'exécution obtenu est un enchaînement séquentiel induit par les dépendances entre les composants du produit p_1 en entrée, avec un séquencement FS entre les STIs.

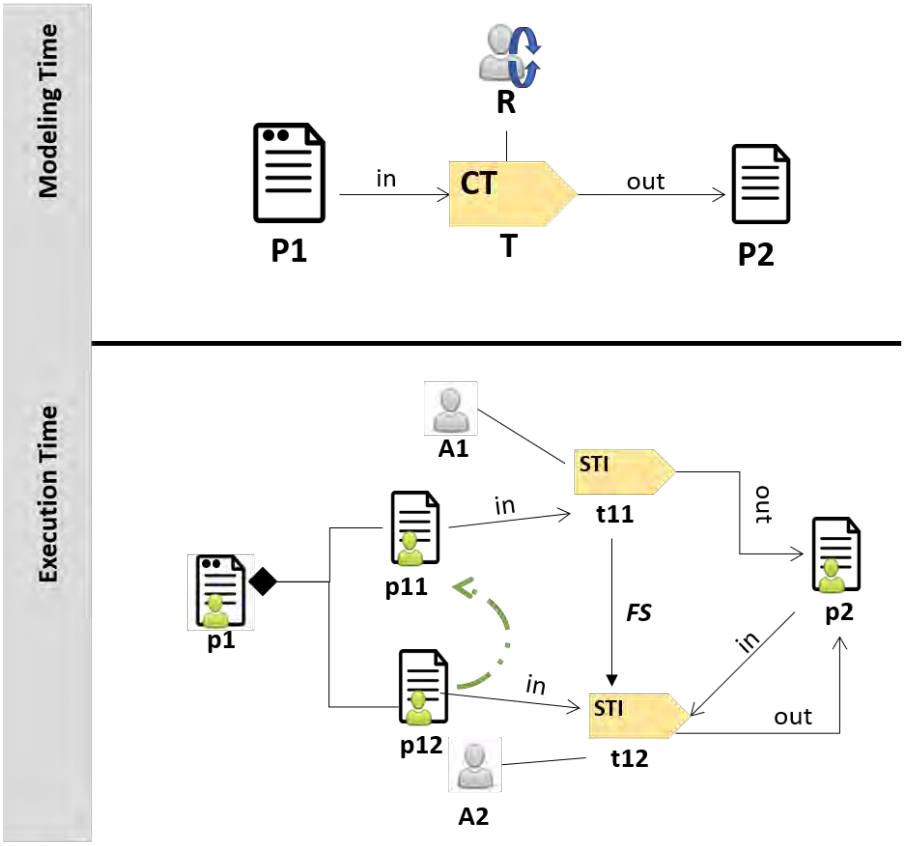


FIGURE 7.7: Patron de collaboration Refinement avec un produit en entrée composite avec des dépendances.

B.2 Patron Free Co-Work : Cas 5

La figure 7.7 présente le patron *"Free Co-Work"* dans le contexte où le produit en entrée est composite et le produit en sortie est non-composite. Le modèle d'exécution proposé est une exécution des STIs en parallèle, chacune ayant en entrée un composant du produit p_1.

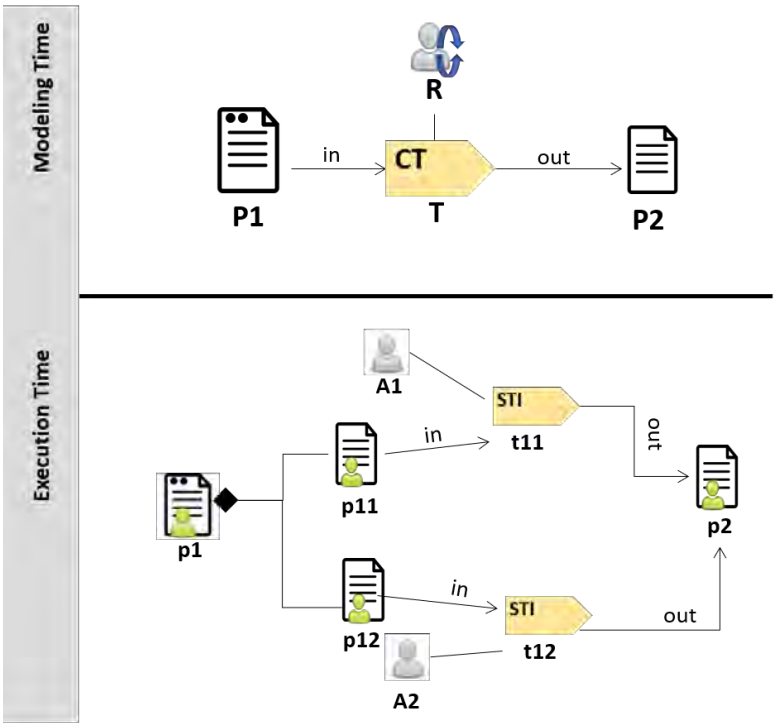


FIGURE 7.8: Patron de collaboration Free Co-Work avec un produit en entrée composite.

B.3 Patron Effort Division : Cas 7 et Cas 10

Les figures 7.9 et 7.10 présentent les cas 7 et 10 du patron *"Effort Division"* où les deux produits sont composites. De plus, dans le cas 7, le produit en entrée comporte des dépendances. Pour le cas 7 comme pour le cas 10, le modèle d'exécution proposé est une exécution des STIs en parallèle, chacune ayant en entrée un composant du produit p_1 et produisant un composant du produit p_2.

B.4 Patron Full Ordered Co-Work : Cas 8 et Cas 9

Les figures 7.11 et 7.12 présentent les cas 8 et 9 du patron *"Full Ordered Co-Work"*. Dans le cas 8, les deux produits sont composites mais avec des dépendances uniquement au niveau du produit en sortie. Dans le cas 9, les deux produits sont aussi composite avec des dépendances de part et d'autre. Pour le cas 8 comme le cas 9, le modèle d'exécution proposé est une exécution des STIs en séquentiel induit par les dépendances entre les composants du produit p_2. Chaque STI utilise un composant du produit p_1 en entrée et produit un composant du produit p_2 en sortie.

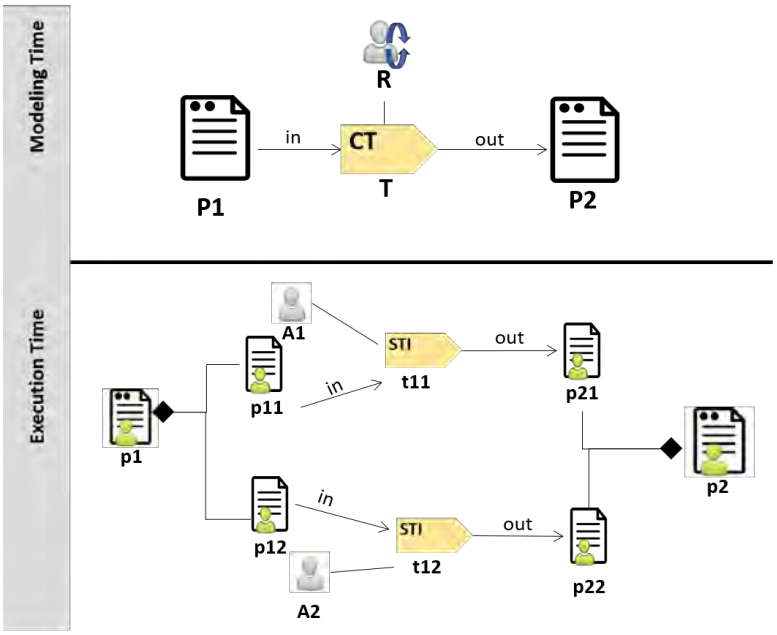


FIGURE 7.9: Patron de collaboration Effort Division avec les deux produits composite sans dépendances.

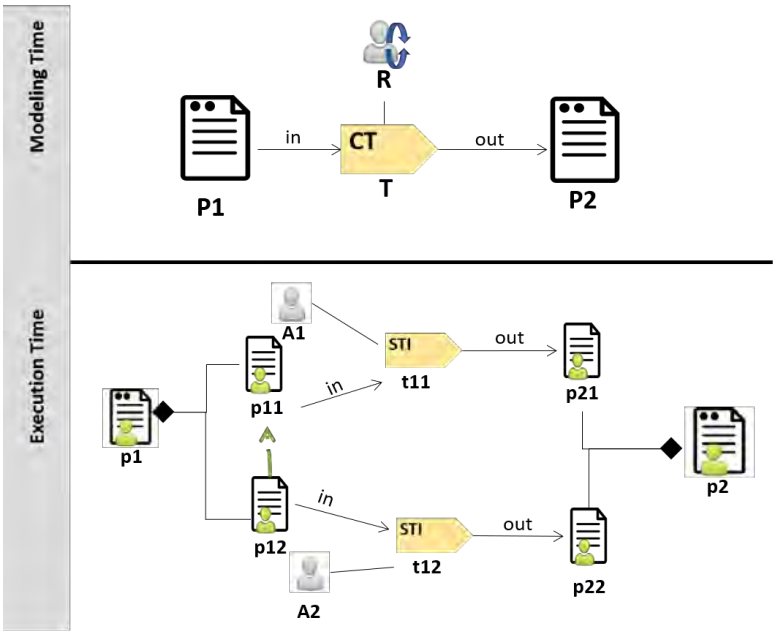


FIGURE 7.10: Patron de collaboration Effort Division avec un produit en entrée composite avec des dépendances et un produit en sortie composite avec dépendances.

B.5 Patron Partial Ordered Co-Work : Cas 8

La figure 7.13 présente le cas 8 du patron *"Partial Ordered Co-Work"*. Ici, les deux produits sont composites et il existe des dépendances partielles au niveau des composants du produit en sortie. Le modèle d'exécution obtenu est une exécution en séquentielle des STIs manipulant les

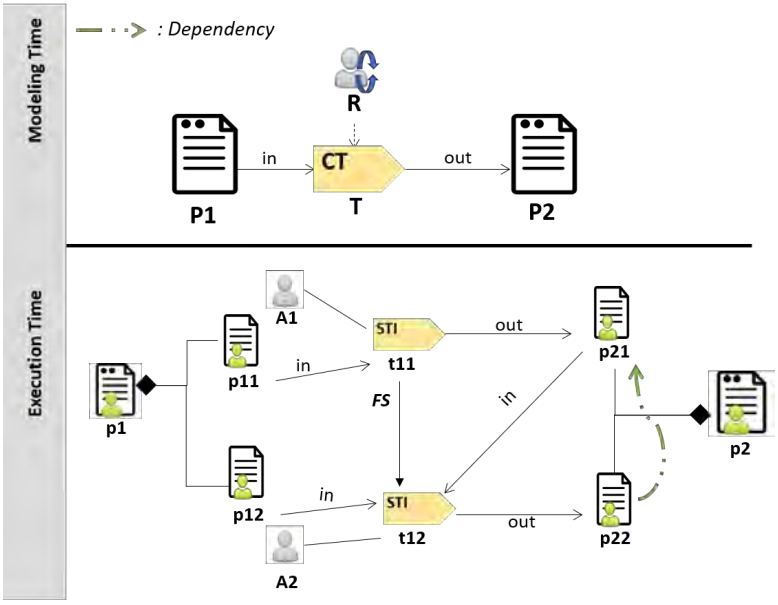


FIGURE 7.11: Patron de collaboration Full Ordered Co-Work avec un produit en sortie composite avec des dépendances et un produit en entrée composite.

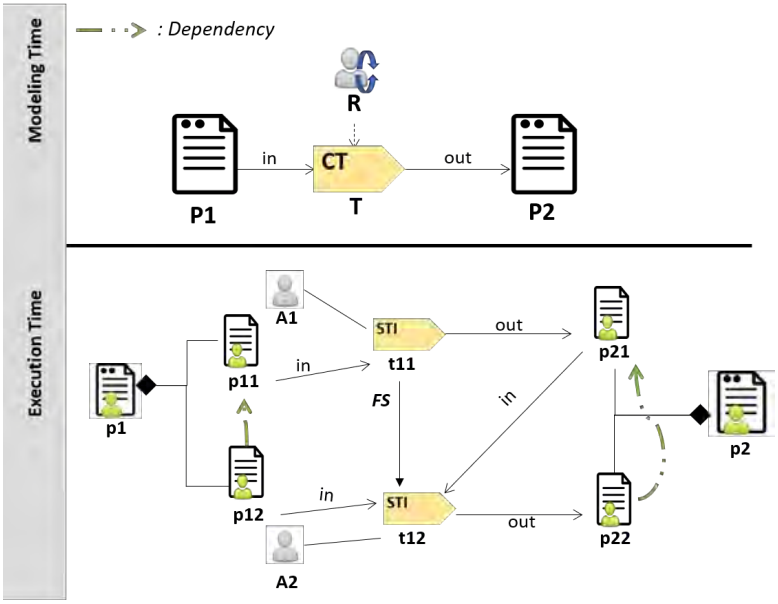


FIGURE 7.12: Patron de collaboration Full Ordered Co-Work avec un produit en entrée et sortie composite avec des dépendances.

composants ayant des dépendances et une exécution parallèle des autres STIs.

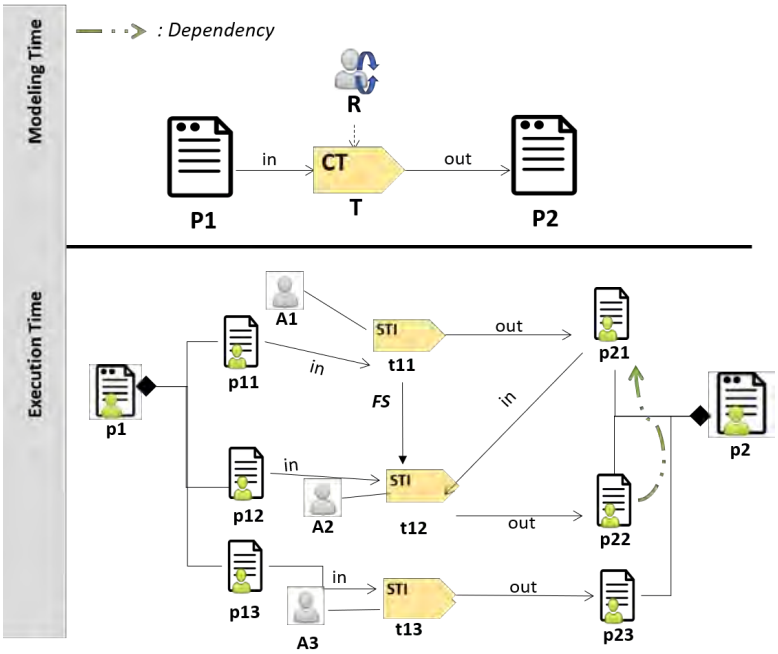


FIGURE 7.13: Patron de collaboration Partial Ordered Co-Work avec un produit en sortie composite avec des dépendances partielles.

C Diagramme de séquences de CPE

Cette annexe décrit les diagrammes de séquence des cas d'utilisation de CPE. La description de ces d'utilisation a été fait à la section 6.1.

C.1 Upload Process Model

La figure 7.14 décrit le séquencement des actions pour la soumission d'un modèle de processus dans CPE.

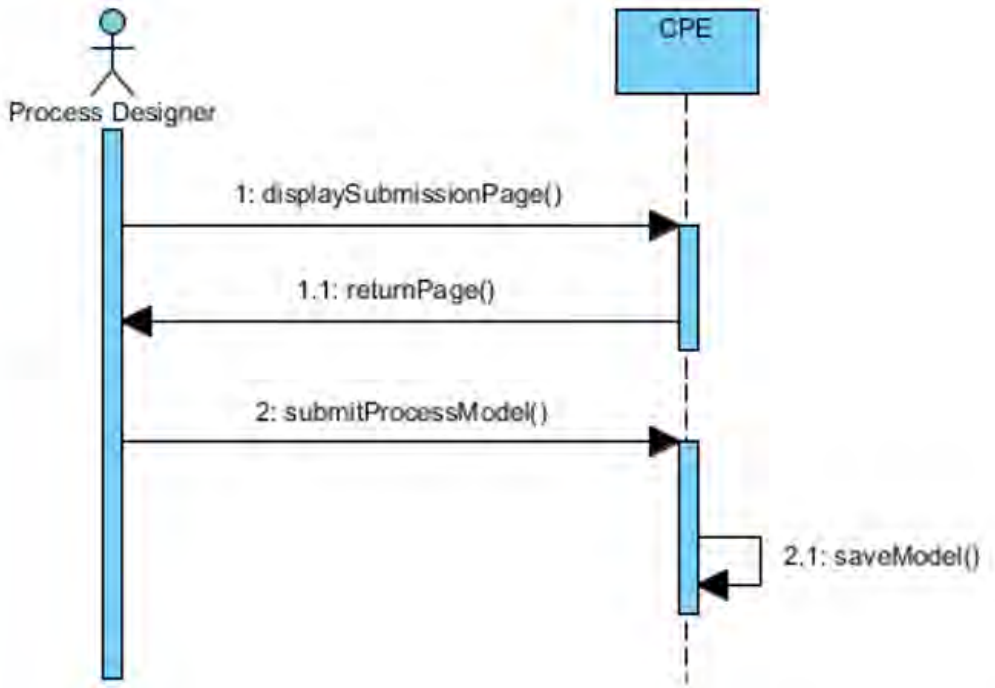


FIGURE 7.14: Diagramme de séquence système du cas "Upload Process Model".

C.2 Generate Process Instance

La figure 7.15 décrit les actions pour la génération de l'instance de processus. L'acteur de ce cas d'utilisation est le project manager.

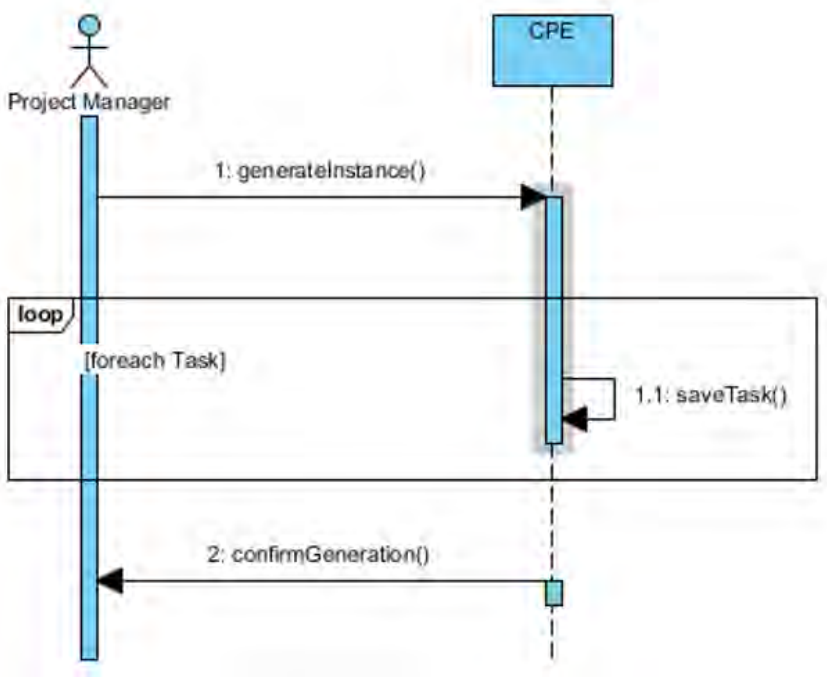


FIGURE 7.15: Diagramme de séquence système du cas "Generate Process Instance".

C.3 Apply Collaboration Pattern

La figure 7.16 décrit les actions permettant le choix d'un patron de collaboration par le project manager.

C.4 Assign Resource to Task

L'affectation d'une ressource à la tâche se fait application d'un patron de collaboration. Elle est décrite sur la figure 7.17.

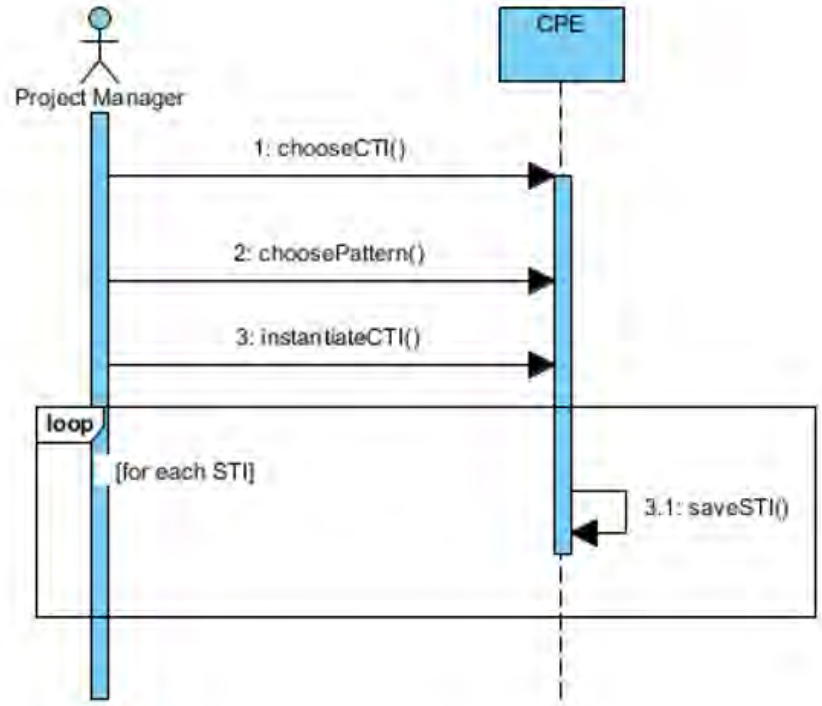


FIGURE 7.16: Diagramme de séquence système du cas "Apply a Collaboration Pattern".

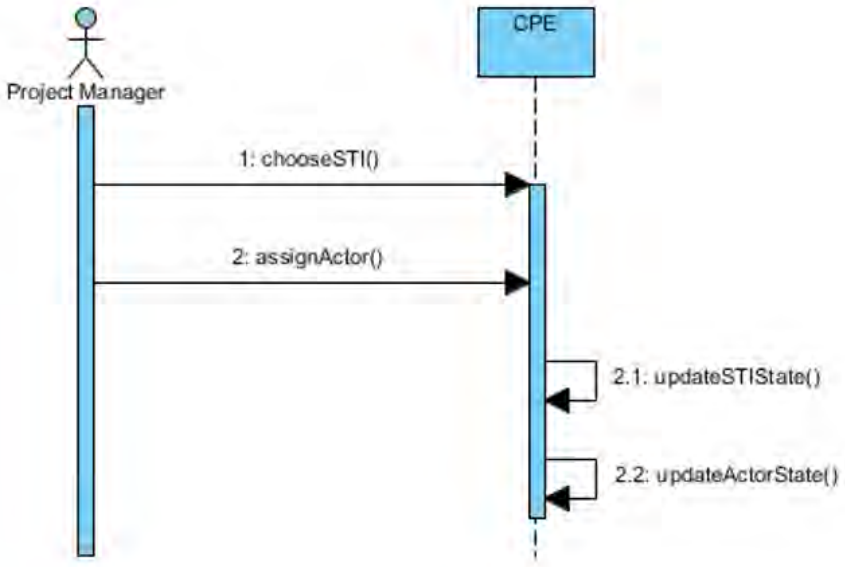


FIGURE 7.17: Diagramme de séquence système du cas "Assign Resource to Task".

D Compléments sur la définition des algorithmes

Cette annexe la définition des algorithmes *createQualification()* et *deleteQualification()* du cycle de vie d'un acteur.

D.1 createQualification()

Il permet la création d'une liaison entre un acteur et un rôle donné.

Algorithm 24: createQualification().

```
   Input: Actor a, Role role ;
1  begin
2  |  a.Qualification[a.Qualification.length] = role ;
3  end
```

D.2 deleteQualification()

Il permet la suppression d'une liaison entre un acteur et un rôle donné.

Algorithm 25: deleteQualification().

```
   Input: Actor a ;
1  begin
2  |  a.Qualification = null ;
3  end
```

Liste des publications

Publications de Mamadou Lakhassane CISSE issues de sa thèse

Conférence internationales avec actes et comités de programme

1. ***Mamadou Lakhassane Cisse***, *Hanh Nhi Tran, Samba Diaw, Bernard Coulette, and Alassane Bah. "Using Patterns to parameterize the execution of Collaborative Tasks". In Proceedings of the 28th IEEE International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE-2019). Capri, Italy, 2019.*

2. ***Mamadou Lakhassane Cisse***, *Hanh Nhi Tran, Samba Diaw, Bernard Coulette, and Alassane Bah. "A pattern-based process management system to flexibly execute collaborative tasks". In Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering. Heraklion, Crete, Greece, 2019.*

3. ***Mamadou Lakhassane CISSE***, *Hanh-Nhi TRAN, Samba DIAW, Bernard COULETTE, Alassane BAH. "Collaborative Processes Management : From Modeling to Enacting". In Proceedings of the IEEE 22nd International Conference on Computer Supported Cooperative Work in Design (CSCWD). pages 461-466. Nanjing, 2018.*

4. ***Mamadou Lakhassane CISSE***. *"Collaborative Processes Behavioral Modeling". In*

Proceedings of the 33rd ACM/SIGAPP Symposium on Applied Computing. Student Research Abstract, pages 1702-1703. ACM. Pau, 2018.

Poster dans une conférence internationale avec comité de programme

1. *Mamadou Lakhassane CISSE "A project management tool for flexible collaboration". EuroScience Open Forum. Toulouse, July 2018.*

Publications de Mamadou Lakhassane CISSE connexes à sa thèse

Revue internationale avec comité de lecture

1. *Samba Diaw,* ***Mamadou Lakhassane Cisse****, Alassane Bah. "Tailoring and Instantiation of MDE Process Models : A Y Model-Based approach". Global Journal of Engineering Science and Researches, vol. 4, no. 7, pages 13–25, July 2017.*

Conférences internationales avec actes et comité de programme

1. *Samba Diaw,* ***Mamadou Lakhassane Cisse****, and Alassane Bah. "An Y MDE Approach for Enactable Software Process Models Generation". In Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1 : MODELSWARD, ISBN 978-989-758-283-7, pages 511-518. Madeira, 2018.*
2. *Samba Diaw,* ***Mamadou Lakhassane Cisse****, and Alassane Bah. "Using the SPEM 2.0 kind-based extension mechanism to define the SPEM4MDE metamodel". In Proceedings of the International Conference on Computing for Engineering and Sciences, pages 63-69. ACM. Istanbul, 2017.*

Bibliography

Acuña, S. T. and Ferré, X. (2001). Software process modelling. In *Proceeding of the World Multiconference on Systemics, Cybernetics and Informatics, ISAS-SCIs 2001, July 22-25, 2001, Orlando, Florida, USA, Volume I : Information Systems Development*, pages 237–242. IIIS.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. (1977). *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press.

Ariouat, H., Andonoff, E., and Hanachi, C. (2016). Do process-based systems support emergent, collaborative and flexible processes ? comparative analysis of current systems. In *Proceedings of the 20th International Conference KES-2016, York, UK, 5-7 September 2016*, pages 511–520. Elsevier.

Bendraou, R., Combemale, B., Crégut, X., and Gervais, M. (2007a). Definition of an executable SPEM 2.0. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), 5-7 December 2007, Nagoya, Japan*, pages 390–397. IEEE Computer Society.

Bendraou, R., Jézéquel, J., Gervais, M., and Blanc, X. (2010). A comparison of six UML-based languages for software process modeling. *IEEE Transactions on Software Engineering*, 36(5) :662–675.

Bendraou, R., Sadovykh, A., Gervais, M., and Blanc, X. (2007b). Software process modeling and execution : The UML4SPM to WS-BPEL approach. In *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2007), August 28-31, 2007, Lübeck, Germany*, pages 314–321. IEEE Computer Society.

Bennani, S., Ebersold, S., Hamlaoui, M. E., Coulette, B., and Nassar, M. (2019). A collabora-

tive decision approach for alignment of heterogeneous models. In *Proceedings of the 28th IEEE International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE 2019, Naples, Italy, June 12-14, 2019*, pages 112–117. IEEE.

Briggs, R. O., Kolfschoten, G. L., de Vreede, G., and Dean, D. L. (2006). Defining key concepts for collaboration engineering. In *Proceedings of the 12th Americas Conference on Information Systems, AMCIS 2006, Acapulco, Mexico, August 4-6, 2006*, page 17. Association for Information Systems.

Charoy, F., Guabtni, A., and Faura, M. V. (2006). A dynamic workflow management system for coordination of cooperative activities. In *Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006*, pages 205–216. Springer.

Cisse, M. L., Tran, H. N., Diaw, S., Coulette, B., and Bah, A. (2018). Collaborative processes management : from modeling to enacting. In *Proceedings of the 22nd IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2018, Nanjing, China, May 9-11, 2018*, pages 461–466. IEEE.

Cisse, M. L., Tran, H. N., Diaw, S., Coulette, B., and Bah, A. (2019). Using patterns to parameterize the execution of collaborative tasks. In *Proceedings of the 28th IEEE International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises, WETICE 2019, Naples, Italy, June 12-14, 2019*, pages 106–111. IEEE.

Combemale, B., Garoche, P., Crégut, X., Thirioux, X., and Vernadat, F. (2007). Towards a formal verification of process model's properties SIMPLEPDL and TOCL case study. In *Proceedings of the Ninth International Conference on Enterprise Information Systems ICEIS 2007, Volume EIS, Funchal, Madeira, Portugal, June 12-16, 2007*, pages 80–89.

Crégut, X. and Coulette, B. (1996). Assistance environment for object oriented development. In *TOOLS 19, Actes conference TOOLS EUROPE 96, Paris,*, pages 61–73. Prentice Hall.

Crégut, X. and Coulette, B. (1998). Emphasizing guidance in the process-centered software engineering environment rhodes. In *Actes conférence internationale IEEE Engineering Computer Based Systems (ECBS'98), Jérusalem.* .

Curtis, B., Kellner, M. I., and Over, J. (1992). Process modeling. *Commuications of the ACM*, 35(9) :75–90.

da Silva, M. A. A., Bendraou, R., Blanc, X., and Gervais, M. (2010). Early deviation detection in modeling activities of MDE processes. In *Model Driven Engineering Languages and*

Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Part II, pages 303–317. Springer.

da Silva, M. A. A., Bendraou, R., Robin, J., and Blanc, X. (2011). Flexible deviation handling during software process enactment. In *Workshops Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOCW 2011, Helsinki, Finland, August 29 - September 2, 2011*, pages 34–41. IEEE Computer Society.

Dadam, P. and Reichert, M. (2009). The ADEPT project : a decade of research and development for robust and flexible process support. *Computer Science - R&D*, 23(2) :81–97.

Diaw, S. (2011). *SPEM4MDE : un métamodèle et un environnement pour la modélisation et la mise en œuvre assistée de processus IDM.* PhD thesis, University of Toulouse II, France.

Diaw, S., Lbath, R., and Coulette, B. (2011). Specification and implementation of SPEM4MDE, a metamodel for MDE software processes. In *Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011*, pages 646–653. Knowledge Systems Institute Graduate School.

Dumas, M., Rosa, M. L., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Management.* Springer.

Dustdar, S. (2004). Caramba - A process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases*, 15(1) :45–66.

Egyed, A., Letier, E., and Finkelstein, A. (2008). Generating and evaluating choices for fixing inconsistencies in UML design models. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*, pages 99–108. IEEE Computer Society.

Ellis, C. A., Gibbs, S. J., and Rein, G. L. (1991). Groupware : Some issues and experiences. *Communications of the ACM*, 34(1) :39–58.

Ellis, C. A. and Wainer, J. (1994). A conceptual model of groupware. In *Proceedings of the Conference on Computer Supported Cooperative Work CSCW '94, Chapel Hill, NC, USA, October 22-26, 1994*, pages 79–88. ACM.

Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., and Philippsen, M. (2010). eSPEM - A SPEM extension for enactable behavior modeling. In *Modelling Foundations and Applications, 6th European Conference, ECMFA 2010, Paris, France, June 15-18, 2010*, pages 116–131. Springer.

Feiler, P. H. and Humphrey, W. S. (1993). Software process development and enactment : Concepts and definitions. In *Proceedings of the Second International Conference on the Software Process, ICSP 1993 : Continuous Software Process Improvement, Berlin, Germany, February 25-26*, pages 28–40. IEEE Computer Society.

Fisichella, M. and Matera, M. (2011). Process flexibility through customizable activities : A mashup-based approach. In *Workshops Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 226–231. IEEE Computer Society.

Fuks, H., Raposo, A. B., Gerosa, M. A., and de Lucena, C. J. P. (2005). Applying the 3C model to groupware development. *International Journal of Cooperative Information Systems (IJCIS)*, 14(2-3) :299–328.

Hawryszkiewycz, I. T. (2005). A metamodel for modeling collaborative systems. *Journal of Computer Information Systems*, 45(3) :63–72.

Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., and Teschke, M. (1999). A comprehensive approach to flexibility in workflow management systems. In *Proceedings of the international joint conference on Work activities coordination and collaboration 1999, San Francisco, California, USA, February 22-25, 1999*, pages 79–88. ACM.

Ignat, C., Oster, G., Fox, O., Shalin, V. L., and Charoy, F. (2015). How do user groups cope with delay in real-time collaborative note taking. In *Proceedings of the 14th European Conference on Computer Supported Cooperative Work ECSCW 2015, 19-23 September 2015, Oslo, Norway*, pages 223–242. Springer.

Izquierdo, J. L. C. and Cabot, J. (2016). Collaboro : a collaborative (meta) modeling tool. *PeerJ Computer Science*, 2 :e84.

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., and Booch, G. (1999). *The unified software development process*, volume 1. Addison-wesley Reading.

Kabbaj, M., Lbath, R., and Coulette, B. (2007). A deviation-tolerant approach to software process evolution. In *Proceedings of the 9th International Workshop on Principles of Software Evolution (IWPSE 2007), in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, Croatia, September 3-4, 2007*, pages 75–78. ACM.

Kabbaj, M., Lbath, R., and Coulette, B. (2008). A deviation management system for handling software process enactment evolution. In *Proceedings of the International Conference on Software Process, ICSP 2008, Leipzig, Germany, May 10-11, 2008*, pages 186–197.

Kedji, K. A., Lbath, R., Coulette, B., Nassar, M., Baresse, L., and Racaru, F. (2014). Supporting collaborative development using process models : a tooled integration-focused approach. *Journal of Software : Evolution and Process*, 26(10) :890–909.

Kedji, K. A., Lbathd, R., Coulette, B., Nassar, M., Baresse, L., and Racaru, F. (2012). Supporting collaborative development using process models : An integration-focused approach. In *Proceedings of the 2012 International Conference on Software and System Process (ICSSP)*, pages 120–129. IEEE, IEEE Computer Society.

Kedji, K. A., That, M. T. T., Coulette, B., Lbath, R., Tran, H. N., and Nassar, M. (2011). Towards a tool-supported approach for collaborative process modeling and enactment. In *Proceedings of the 18th Asia Pacific Software Engineering Conference, APSEC 2011, Ho Chi Minh, Vietnam, December 5-8, 2011*, pages 414–421. IEEE Computer Society.

Kruchten, P. (2004). *The Rational Unified Process : an introduction.* Addison-Wesley Professional.

Lanubile, F. (2009). Collaboration in distributed software development. *International Summer School on Software Engineering, ISSSE 2006-2008*, 5413 :174–193.

Lanubile, F. and Visaggio, G. (2000). Evaluating Defect Detection Techniques for Software Requirements Inspections. *International Software Engineering Research Network (ISERN)*, pages 1–24.

Laurillau, Y. and Nigay, L. (2002). Clover architecture for groupware. In *Proceeding on the ACM 2002 Conference on Computer Supported Cooperative Work CSCW 2002, New Orleans, Louisiana, USA, November 16-20, 2002*, pages 236–245. ACM.

Le Moigne, J.-L. (1999). La modélisation des systèmes complexes. *Paris, Dunod, réed.*

Liptchinsky, V., Khazankin, R., Schulte, S., Satzger, B., Truong, H. L., and Dustdar, S. (2014). On modeling context-aware social collaboration processes. *Information System*, 43 :66–82.

Lonchamp, J. (1993). A structured conceptual and terminological framework for software process engineering. In *Proceedings of the Second International Conference on the Software Process, ICSP 1993 : Continuous Software Process Improvement, Berlin, Germany, February 25-26*, pages 41–53. IEEE Computer Society.

Lonchamp, J. (1998). Process model patterns for collaborative work. In *Proceedings of the 15th IFIP World Computer Congress, Telecooperation Conference, Telecoop'98*, Vienna, Austria.

Malone, T. W. and Crowston, K. (1990). What is coordination theory and how can it help

design cooperative work systems ? In *Proceedings of the Conference on Computer Supported Cooperative Work CSCW '90, Los Angeles, CA, USA, October 7-10, 1990*, pages 357–370. ACM.

Mistrík, I., Grundy, J., van der Hoek, A., and Whitehead, J. (2010). Collaborative software engineering : Challenges and prospects. In *Collaborative Software Engineering*, pages 389–403. Springer.

Mundbrod, N., Beuter, F., and Reichert, M. (2015). Supporting knowledge-intensive processes through integrated task lifecycle support. In *Proceedings of the 19th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2015, Adelaide, Australia, September 21-25, 2015*, pages 19–28. IEEE Computer Society.

Object Management Group (OMG) (2007). Unified modeling language, version 2.5.1. Technical report, Object Management Group.

Object Management Group (OMG) (2008). Software & Systems Process Engineering Meta-Model Specification V2.0. Technical report.

Object Management Group (OMG) (2011). Business Process Model and Notation (BPMN), Version 2.0. Technical report, Object Management Group.

Object Management Group (OMG) (2016). MOF Query/View/Transformation, version 1.3. Technical report.

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007). DECLARE : full support for loosely-structured processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA*, pages 287–300. IEEE Computer Society.

Polancic, G., Jost, G., and Hericko, M. (2015). An experimental investigation comparing individual and collaborative work productivity when using desktop and cloud modeling tools. *Empirical Software Engineering*, 20(1) :142–175.

Raposo, A. B., Magalhães, L. P., Ricarte, I. L. M., and Fuks, H. (2001). Coordination of collaborative activities : A framework for the definition of tasks interdependencies. In *Proceedings of the Seventh International Workshop on Groupware (CRIWG'01), September 6-8, 2001, Darmstadt, Germany, Proceedings*, pages 170–180. IEEE Computer Society.

Rastrepkina, M. (2010). Managing variability in process models by structural decomposition. In *Business Process Modeling Notation - Second International Workshop, BPMN 2010, Potsdam, Germany, October 13-14, 2010*, pages 106–113. Springer.

Reichert, M. and Dadam, P. (1998). Adept$_{\text{flex}}$-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems (JIIS)*, 10(2) :93–129.

Reichert, M., Rinderle, S., and Dadam, P. (2003). ADEPT workflow management system : Flexible support for enterprise-wide business processes. In *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003*, pages 370–379. Springer.

Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer.

Robillard, P. N. and Robillard, M. P. (2000). Types of collaborative work in software engineering. *Journal of Systems and Software*, 53(3) :219–224.

Rosa, M. L., van der Aalst, W. M. P., Dumas, M., and Milani, F. (2017). Business process variability modeling : A survey. *ACM Computing Surveys*, 50(1) :2 :1–2 :45.

Roschelle, J. (1996). Designing for cognitive communication : Epistemic fidelity or mediating collaborating inquiry. *Computers, communication & mental models*, 13 :25.

Rumpe, B. (2014). Executable modeling with UML. A vision or a nightmare ? *Computing Research Repository*, abs/1409.6597.

Schmidt, K. and Bannon, L. J. (1992). Taking CSCW seriously. *Computer Supported Cooperative Work*, 1(1-2) :7–40.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2008a). Process flexibility : A survey of contemporary approaches. In *Advances in Enterprise Engineering I, 4th International Workshop CIAO ! and 4th International Workshop EOMAS, held at CAiSE 2008, Montpellier, France, June 16-17, 2008. Proceedings*, pages 16–30. Springer.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., and van der Aalst, W. M. P. (2008b). Towards a taxonomy of process flexibility. In *Proceedings of the Forum at the CAiSE'08 Conference, Montpellier, France, June 18-20, 2008*, pages 81–84. CEUR-WS.org.

Smatti, M. and Ahmed-Nacer, M. (2014). Dealing with deviations on software process enactment : Comparison framework. In *Proceedings of the 1st International Conference on Advanced Aspects of Software Engineering, ICAASE 2014, Constantine, Algeria, November 2-4, 2014*, pages 108–115. CEUR-WS.org.

Tran, H. N., Coulette, B., and Dong, B. T. (2007). Modeling process patterns and their application. In *Proceedings of the Second International Conference on Software Engineering*

Advances (ICSEA 2007), August 25-31, 2007, Cap Esterel, French Riviera, France, page 15. IEEE Computer Society.

Tran, H. N., Coulette, B., Thu, T. D., and Vu, M. H. (2011). Automatic reuse of process patterns in process modeling. In *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*, pages 1431–1438. ACM.

van der Aalst, W. M. P., Pesic, M., and Schonenberg, H. (2009). Declarative workflows : Balancing between flexibility and support. *Computer Science - Research & Development*, 23(2) :99–113.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003a). Workflow patterns. *Distributed and Parallel Databases*, 14(1) :5–51.

van der Aalst, W. M. P., ter Hofstede, A. H. M., and Weske, M. (2003b). Business process management : A survey. In *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003*, pages 1–12.

van der Aalst, W. M. P. and van Hee, K. M. (2002). *Workflow Management : Models, Methods, and Systems*. Cooperative information systems. MIT Press.

Weiss, A. (2005). The power of collective intelligence. *networker*, 9(3) :16–23.

Yang, Q., Li, M., Wang, Q., Yang, G., Zhai, J., Li, J., Hou, L., and Yang, Y. (2007). An algebraic approach for managing inconsistencies in software processes. In *Software Process Dynamics and Agility, International Conference on Software Process, ICSP 2007, Minneapolis, MN, USA, May 19-20, 2007*, pages 121–133. Springer.

Zamli, K. Z. and Lee, P. A. (2001). Taxonomy of process modeling languages. In *Proceedings of the 2001 ACS / IEEE International Conference on Computer Systems and Applications (AICCSA 2001), 26-29 June 2001, Beirut, Lebanon*, pages 435–437. IEEE Computer Society.