Modélisation et mise en œuvre de processus collaboratifs ad hoc

Komlan Akpédjé Kedji

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Université de Toulouse 2 Le Mirail

Cotutelle internationale avec :
Université Mohammed V Souissi

Présentée et soutenue par :
Komlan Akpédjé KEDJI

Le 05 Juillet 2013

Titre :
Modélisation et mise en œuvre de processus collaboratifs ad hoc

---

École doctorale et discipline ou spécialité :
ED MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

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

Directeurs de Thèse :

M. Bernard Coulette	Professeur	Université Toulouse 2 Le Mirail
M. Mahmoud Nassar	HDR	Université Mohamed V Souissi

Rapporteurs :

M. Bouchaïb Bounabat	Professeur	Université Mohamed V Souissi
M. Khalid Benali	HDR	Université de Lorraine
Mme. Marie-Pierre Gervais	Professeur	Université Paris Ouest

Autre membre du jury :

Redouane Lbath	Maître de Conférences	Université de Toulouse 1 Capitole

Table des matières

Table des figures

Liste des tableaux

Remerciements

Gratitude is when memory is stored in the heart and not in the mind.
– Lionel Hampton

À M. Bernard Coulette, mon directeur de thèse, pour m'avoir initié à la recherche, et toujours trouvé le moyen de me motiver, ce dont j'ai bien souvent eu besoin.

À M. Redouane Lbath, mon encadrant, pour sa patience durant toutes ces années où je lui donné largement l'occasion d'en user.

À M. Mahmoud Nassar, mon codirecteur de thèse, pour avoir cru en moi même quand j'hésitais à faire de même.

À Mme. Marie-Pierre Gervais, M. Khalid Benali, et M. Bouchaïb Bounabat, qui ont eu la gentillesse d'accepter d'évaluer mon travail.

À Mme. Hanh Nhi Tran et Sophie Ebersold au contact de qui, à force de longues discussions sur le projet Galaxy, j'ai appris de précieuses leçons sur la recherche.

À Minh Tu Ton That, Mohamed Chouaiech, et Mohamed El Maatla, avec qui j'ai eu le plaisir de collaborer sur mon travail de thèse lors de leurs stages à l'Université du Mirail.

À Mahmoud El Hamlaoui, mon voisin de bureau, voisin de résidence, et voisin de mille autres manières.

À Samba, Adil, Youness, Yael, Rahma, Adel, Damien, Jacob, et à tous les doctorants de l'équipe MACAO que j'ai eu la chance de côtoyer pendant ces années.

À mes chers parents, mon précieux frère, et mes sœurs adorées.

À l'ange, au poète, à la conviction, au pacifique, à l'océan, et à la candeur.

Votre existence est une bénédiction.

Résumé

Le développement logiciel est une activité intensément collaborative. Les problématiques habituelles de collaboration (organisation des tâches, utilisation des ressources, communication, etc.) y sont exacerbées par le rythme rapide des changements, la complexité et la grande interdépendance des artéfacts, le volume toujours croissant d'informations de contexte à traiter, la distribution géographique des participants, etc. Par conséquent, la question du support outillé de la collaboration se pose plus fortement que jamais en ingénierie logicielle.

Dans cette thèse, nous abordons la question de la collaboration sous l'angle de la modélisation et de l'exploitation des processus [1] de développement. Ces derniers sont traditionnellement considérés comme une structure imposée sur le développement d'un produit logiciel. Cependant, une part importante de la collaboration en génie logiciel est de nature ad hoc, faite d'activités non planifiées. Afin de faire contribuer les processus logiciels au support de la collaboration, en particulier celle non planifiée, nous nous intéressons à leur fonction de banques d'information sur les éléments clés de cette collaboration et les interactions entre ces derniers.

Notre contribution est, d'une part, un modèle conceptuel du support au développement collaboratif, capable de rendre compte de la structure d'outils classiques comme ceux de gestion de versions ou de gestion de défauts logiciels. Ce modèle conceptuel est ensuite appliqué aux modèles de processus logiciels. Nous définissons ainsi une approche globale d'exploitation des informations de processus pour le support de la collaboration, basée sur les notions centrales de langage de requête d'information et de mécanisme de réaction aux événements.

D'autre part, nous proposons un méta-modèle, CMSPEM (Collaborative Model-Based Software & System Process Engineering Metamodel), qui enrichit le standard SPEM (Software & System Process Engineering Metamodel) avec des concepts et relations nécessaires au support de la collaboration. Ce méta-modèle est outillé avec des outils de création de modèle (éditeurs graphiques et textuels), et un serveur de processus offrant un langage de requêtes basé sur HTTP/REST et un framework de souscription et de réaction aux événements de processus.

Enfin, notre approche conceptuelle a été illustrée et validée, en premier lieu, par une analyse des pratiques inférées à partir des données de développement de 219 projets open source.

1. Le terme "procédé" est quelques fois utilisé dans la littérature, avec un sens similaire. Dans ce travail, nous désignons par processus un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie [ISO 08].

En second lieu, des utilitaires de support à la collaboration (mise à disposition d'informations conceptuelles, automatisation d'actions, extraction d'information sur les contributions individuelles) ont été implémentés à travers le serveur de processus CMSPEM.

Mots clés : processus logiciel, collaboration ad hoc, ingénierie dirigée par les modèles, SPEM, CMSPEM.

Abstract

Software development is an intensively collaborative activity, where common collaboration issues (task management, resource use, communication, etc.) are aggravated by the fast pace of change, artifact complexity and interdependency, an ever larger volume of context information, geographical distribution of participants, etc. Consequently, the issue of tool-based support for collaboration is a pressing one in software engineering.

In this thesis, we address collaboration in the context of modeling and enacting development processes. Such processes are traditionally conceived as structures imposed upon the development of a software product. However, a sizable proportion of collaboration in software engineering is ad hoc, and composed of unplanned activities. So as to make software processes contribute to collaboration support, especially the unplanned kind, we focus on their function of information repositories on the main elements of collaboration and the interactions of such elements.

Our contribution, on the one hand, is a conceptual model of collaborative development support, which is able to account for popular tools like version control systems and bug tracking systems. This conceptual model is then applied to software processes. We hence define a global approach for the exploitation of process information for collaboration support, based on the central notions of query language and event handling mechanism.

On the other hand, we propose a metamodel, CMSPEM (Collaborative Model-Based Software & System Process Engineering Metamodel), which extends SPEM (Software & System Process Engineering Metamodel) with concepts and relationships necessary for collaboration support. This metamodel is then tooled with model creation tools (graphical and textual editors), and a process server which implements an HTTP/REST-based query language and an event subscription and handling framework.

Our approach is illustrated and validated, first, by an analysis of development practices inferred from project data from 219 open source projects. Second, collaboration support utilities (making contextual information available, automating repetitive actions, generating reports on individual contributions) have been implemented using the CMSPEM process server.

Keywords : software process, ad hoc collaboration, model driven engineering, SPEM, CMSPEM.

Chapitre 1

Introduction

Le présent travail de thèse, intitulé “Modélisation et mise en œuvre de processus collaboratifs ad hoc”, s’est déroulé dans le cadre du projet GALAXY, et d’une cotutelle entre les universités de Toulouse-2 Le Mirail et de Mohammed V Souissi à Rabat.

GALAXY (2009-2012) est un projet de l’Agence Nationale de la Recherche (ANR) qui a rassemblé des laboratoires de recherche (MACAO-IRIT, SARA-LAAS, ATLANMOD-INRIA, MOVE-LIP6) et des partenaires industriels (SOFTEAM, AKKA Technologies, AIRBUS) autour de la problématique du développement collaboratif de systèmes complexes selon une approche IDM (Ingénierie Dirigée par les Modèles).

Les équipes de recherche impliquées dans le présent travail de thèse sont les suivantes :

- Équipe MACAO (Modèles, Aspects, et Composants pour les Architectures à Objets), laboratoire IRIT (Institut de Recherche en Informatique de Toulouse), France.
- Équipe IMS (Ingénierie des Modèles et Systèmes), laboratoire SIME (Systèmes d’Informations Mobiles et Embarqués), Maroc.

1.1 Contexte

Cette thèse se place dans le triple contexte de la collaboration en génie logiciel, des processus de développement logiciel, et de l’ingénierie dirigée par les modèles.

1.1.1 La collaboration en génie logiciel

Le développement logiciel est une activité intensément collaborative [Dewa 93, Robi 00, Avri 10, Dewa 10, Robi 10, Scac 10]. Des experts métier et des informaticiens de différentes spécialités (concepteurs, développeurs, designers, testeurs, administrateurs de bases de données, etc.) unissent et harmonisent leurs efforts pour faire aboutir les projets logiciels [Cher 08]. En plus des considérations habituelles qui motivent la collaboration (succession de tâches,

utilisation de ressources, etc.), en ingénierie logicielle, la complexité et la grande interdépendance des artéfacts demandent une collaboration étroite entre ceux qui les produisent et ceux qui les utilisent [Whit 07].

Afin de gérer cette collaboration, les intervenants dans un projet logiciel ont besoin de certaines capacités. Dans la littérature [Booc 03, Herb 07, Swam 08, Jime 09, Omor 10] , il est question par exemple de la capacité à savoir ce qui est fait (par les autres) autour de soi, afin de pouvoir prendre en compte le contexte de son travail. Cette information additionnelle sur les occupations des autres est non seulement difficile à trouver, mais peut aussi facilement distraire quand elle n'est pas correctement synthétisée et personnalisée. D'autres problèmes relatifs à la collaboration sont ceux de la connaissance partagée [Cram 01, Jime 09], de la structure de communication [Herb 07, Jime 09], de la gestion des conflits entre modifications [Jime 09, Cicc 08, Srip 08], etc. La complexité induite par la collaboration vient donc s'ajouter à la complexité inhérente au développement logiciel [Broo 87].

1.1.2 Les processus logiciels

Depuis le travail fondateur [Oste 87] de L. Osterweil, les processus logiciels ont progressivement été reconnus comme des facteurs clés dans le succès des projets logiciels. Divers modèles de cycle de vie de projet logiciel ont été proposés [Boeh 88, Royc 89, Gord 95, Jaco 99], afin de décrire les activités réalisées dans un projet logiciel, ainsi que leur ordre. Diverses méthodes de développement, regroupées sous la bannière du "développement agile", ont ensuite été élaborées [Abra 02, Boeh 02, Beck 99, Cock 01], avec pour but d'être moins lourdes que les précédentes approches et de prendre en compte l'organisation ad hoc, en se focalisant d'avantage sur des pratiques quotidiennes que des plans généraux rigides.

Une des extensions importantes des travaux sur les processus logiciels est celle du support au processus, avec les PSEEs (Process Centered Software Engineering Evironment). Une première génération de PSEEs a été fondée sur une vision purement normative du modèle de processus [Kais 90, Sutt 91, Junk 95, Band 94], et la génération suivante [Ben 94, Bolc 96, Allo 96, Kais 97, Dami 98, Cugo 99] a proposé diverses manières d'introduire de la flexibilité, en permettant par exemple aux développeurs de dévier des spécifications du processus [Gruh 02a].

1.1.3 L'ingénierie dirigée par les modèles

L'ingénierie dirigée par les modèles (IDM) est une approche de développement qui propose d'élever les modèles au rang de concept unificateur [Bezi 05] de l'ingénierie logicielle, et se positionne comme une nouvelle étape de la montée en abstraction après la révolution des compilateurs de code [Kurt 02]. Pour ce faire, l'IDM propose une nouvelle classe d'abstractions (méta-modèle, transformation, etc.), dont certaines directement liées au domaine métier d'un logiciel en construction [Booc 04, Tolv 04, Weig 06].

L'IDM a été appliquée à diverses préoccupations de génie logiciel. En particulier, l'IDM a contribué à rendre les modèles de processus logiciels rigoureux et sujets à la manipulation automatique, afin de devenir plus productifs que contemplatifs [Bezi 04]. Un des efforts pouvant illustrer cette approche est celui de la norme SPEM [OMG 02, OMG 07] (Software & Systems

Process Engineering Metamodel) de l'OMG (Object Management Group), et ses différentes extensions [Bend 07, Elln 10, Diaw 11], qui fournissent, entre autres, des briques réutilisables pour la modélisation des processus logiciels.

1.2 Problématique et proposition

Le présent travail de thèse s'articule autour de la question du support au développement collaboratif avec les processus logiciels. Pour ce faire, d'une part, nous nous intéressons à l'adaptation des formalismes de description de processus logiciel à la collaboration. D'autre part, nous traitons de l'exploitation des processus logiciels dans le support du développement collaboratif.

1.2.1 Adaptation des formalismes de description de processus à la collaboration

La problématique particulière de la collaboration n'a pas reçu de traitement satisfaisant dans les approches de modélisation de processus logiciel, traitement pourtant nécessaire pour le support de la collaboration [Polt 09]. Dans la norme SPEM par exemple, la question de l'affectation de ressources ou de la réalisation d'une même tâche par plusieurs personnes est considérée comme étrangère à la modélisation de processus, relevant de la mise en œuvre des processus. Par conséquent, les professionnels de l'informatique se retrouvent obligés de répondre aux problèmes posés par la collaboration, avec un support faible ou inexistant des modèles de processus.

Les approches de modélisation existantes [OMG 07, OMG 09] ne sont cependant pas complètement dénuées de concepts liés à la collaboration. En effet, le séquencement de tâches par exemple relève de la collaboration, dans la mesure où il représente en même temps la dépendance entre les personnes qui exécutent ces tâches, et entre les éventuels artéfacts qui y sont manipulés. Cependant, ces formes de collaborations, que nous désignerons par collaboration "faible", restent insuffisantes pour décrire les réalités du développement logiciel.

D'autres formes de collaboration, dites ici "fortes", nécessitent, pour être représentées, des concepts de granularité plus fine que ceux de rôle, produit, et tâches utilisés dans SPEM par exemple. Par exemple, de riches relations (stratégie de fusion de modifications par exemple) lient les efforts de deux développeurs ayant le même rôle, et collaborant sur le même produit (un module de code par exemple), même s'il n'est pas possible de distinguer ces deux développeurs sur un modèle de processus basé sur les rôles. Ces formes de collaboration impliquent une interdépendance plus importante entre les intervenants d'un projet, d'où la qualification de collaboration "forte".

Cette distance prise par la modélisation de processus, par rapport aux questions spécifiques relevant de la mise en œuvre des processus logiciels, est cependant prévisible. En effet, une qualité majeure des processus logiciels est la capacité d'être réutilisés d'un projet à l'autre, d'être génériques. Cette généricité n'est obtenue qu'au prix de l'éloignement des problématiques de granularité plus fine comme celles de la collaboration.

Cependant, la collaboration est une activité fondamentalement humaine. Les participants à un projet logiciel collaborent avec d'autres humains, pas avec des activités, ni des produits, ni des dates, ou encore moins des machines. Tous ces concepts ne sont que des moyens pour résoudre des préoccupations qui se posent fondamentalement entre individus. Un besoin fondamental pour répondre à ces préoccupations est donc de pouvoir représenter chaque individu en tant que concept indépendant. Le concept de rôle a beau être utile pour avoir une vue d'ensemble du processus, dans les interactions de tous les jours, les participants ne traitent pas avec le concept plutôt diffus de rôle, mais bien avec d'autres individus.

La vision managériale des participants à un projet, en tant que ressources remplaçables, dont la caractéristique primordiale est le rôle qu'ils jouent dans le projet, est loin d'être celle utilisée par les participants dans leur travail de tous les jours. On s'adresse à *Bob* qui a écrit telle classe qui pose problème lors de l'intégration, et non pas à un "Développeur" qui a participé à l'activité "Codage". On se demande "qui a fait telle chose", "qui est en train de faire telle chose", "qui fera telle chose", "qui a touché à tel artéfact en dernier", "qui peut renseigner sur telle tâche", etc. De plus, la notion de responsabilité personnelle sur les tâches est capitale dans l'industrie, rendant incontournable la notion de tâche spécifique à un acteur. C'est aussi seulement avec une limite claire entre le travail d'un participant, et celui des autres, que les relations de collaboration peuvent être explicitées. C'est en ayant un concept explicite de personne physique (auquel viendront naturellement s'ajouter les concepts annexes de tâche confiée à une personne spécifique, ou d'artéfact possédé par un individu) qu'on pourra aider vraiment des développeurs qui collaborent sur un projet logiciel. Autrement, on parle une autre langue, qui, même si elle peut être utile au chef de projet, complique inutilement la vie des participants.

Enfin, les processus logiciels sont généralement considérés comme des plans, des indications prescriptives, à mettre en œuvre dans un projet. Une conséquence immédiate est que les modèles de processus n'ont généralement pas la capacité de prendre en compte les changements inattendus de planification et les événements non prévus qui peuvent se produire lors de la mise en œuvre d'un projet. Or, la capacité à représenter de l'information à un niveau de granularité plus fin que celui disponible dans les modèles de processus classiques, et celle à intégrer de l'information nouvelle sont capitales pour le support de la collaboration.

Le premier objectif de cette thèse est donc d'étendre les formalismes de représentation de processus avec un support faible de la collaboration comme SPEM, pour représenter des concepts liés à la collaboration "forte". Nous munissons aussi ces nouveaux concepts d'une sémantique dynamique appropriée à l'évolution ad hoc caractéristique des projets logiciels. Le méta-modèle ainsi défini est outillé par des éditeurs graphique et textuel, ainsi qu'un système de visualisation dynamique d'extraits de processus.

1.2.2 Exploitation des processus dans le support au développement collaboratif

La question de la mise en œuvre des processus logiciel a reçu beaucoup d'attention dans la littérature, surtout à travers les différentes propositions d'environnements de génie logiciel centrés sur les processus. Cependant, le problème de l'adoption de ces environnements par l'industrie du logiciel reste entier, pour diverses raisons dont la prise en main par des personnes non-expertes dans le domaine des processus, la capacité à s'adapter à des changements

inattendus, l'intégration avec les autres outils de développement, etc. [Pene 91, Ossh 00, Wick 04].

Le second objectif de ce travail est de reconsidérer la contribution des processus au support du développement collaboratif. L'approche traditionnelle donne aux outils de support aux processus le rôle d'orchestrateurs des autres outils utilisés dans le développement collaboratif. Nous explorons la possibilité de considérer les modèles de processus comme des sources d'information sur un projet logiciel, au même titre que d'autres préoccupations comme les données de gestion de défauts logiciels, de gestion de version, ou encore de communication. De manière pratique, nous étudions l'effet de la mise à disposition des informations de processus, afin qu'elles puissent être exploitées par d'autres outils de développement logiciel.

Dans cette perspective, nous considérons un modèle de processus comme une représentation changeante d'un processus collaboratif en cours de déroulement, et utilisons un tel modèle pour contribuer au support logiciel offert à la collaboration. Notre hypothèse de recherche est qu'une telle approche réduira considérablement les divergences qui sont la règle entre les modèles de processus (figés) et le déroulement (hautement dynamique) des projets logiciels, et que ceci permettra d'améliorer significativement le support offert aux utilisateurs dans des situations de collaboration, ainsi que l'adoption des outils de support aux processus logiciels.

Pour valider cette hypothèse, nous proposons un modèle conceptuel du support au développement collaboratif, et exploitons ce modèle pour concevoir et implémenter divers outils comme des éditeurs et un serveur de suivi et de réaction à l'évolution d'un processus. L'approche est validée par un ensemble d'utilitaires de support à la collaboration, implémentés ou spécifiés, qui manipulent l'information disponible dans les modèles de processus collaboratifs.

1.3 Organisation du mémoire

Le présent document est organisé comme suit :

Chapitre 1 : Introduction
Il s'agit du présent chapitre, qui présente le contexte du travail, les questions de recherche, l'approche adoptée, et l'organisation du mémoire.

Chapitre 2 : État de l'art
La revue de la littérature a été conduite selon 4 axes clés :

- L'ingénierie dirigée par les modèles (IDM), qui constitue la cadre méthodologique de notre démarche. Cette orientation est due, en premier lieu, à la problématique traitée par le projet GALAXY (Développement collaboratif de systèmes complexes selon une approche IDM). D'autre part, l'IDM est le cadre général du standard SPEM, sur lequel se base notre contribution. Enfin, les prototypes réalisés dans le cadre du présent travail l'ont été avec une approche IDM.
- Les processus logiciels, qui forment l'angle sous lequel nous avons traité la question de la collaboration en génie logiciel. Nous rappelons les contributions majeures à la formalisation et à la mise en œuvre des processus logiciels.

– La collaboration en génie logiciel, qui est notre thème général. Ici, les contributions à la compréhension et au support de la collaboration sont analysées, selon les axes suivants : la modélisation de la collaboration, les stratégies de collaboration, les technologies habituelles de support à la collaboration, et les outils notables de support à la collaboration.

– La modélisation des processus collaboratifs, qui synthétise les contributions précédentes à l'utilisation des processus logiciels pour répondre à des problématiques de collaboration.

Ce chapitre se termine par une classification des différentes approches étudiées, et une évaluation des outils notables de support à la collaboration.

Chapitre 3 : Conceptualisation du support au développement collaboratif

Parmi les outils de développement collaboratifs existants, les outils de gestions de version et ceux de gestion de défauts logiciels sont les plus utilisés. Nous nous appuyons sur une analyse de la conception de ces outils pour mettre en évidence une architecture implicite de support à la collaboration. Cette architecture est ensuite explicitée en un modèle conceptuel, qui est utilisé pour expliquer différentes caractéristiques désirables des outils existants de support à la collaboration.

Nous nous posons ensuite la question de la contribution des modèles de processus au support de la collaboration. D'une part, nous montrons comment les choix de conception traditionnellement faits pour les outils basés sur les processus freinent leur adoption et réduisent leur utilité pratique. D'autre part, nous appliquons le modèle conceptuel précédent aux outils de support de processus, afin de définir une conception alternative dont l'objectif est d'offrir un meilleur support à la collaboration.

Chapitre 4 : Formalisation des aspects statiques de la collaboration

Ce chapitre examine la norme SPEM à la lumière de l'analyse faite dans le chapitre 3, puis explicite et défend les extensions nécessaires afin de pouvoir exploiter un modèle de processus SPEM pour supporter le développement collaboratif.

Une extension de SPEM, CMSPEM (Collaborative Model-based Software & Systems Process Engineering Metamodel), est introduite, et ses concepts et relations statiques sont définies et illustrées.

Chapitre 5 : Formalisation des aspects dynamiques de la collaboration

La norme SPEM ne définit pas de modèle comportemental. Pour les besoins de l'architecture identifiée dans le chapitre 3, nous introduisons ici un modèle comportemental basé sur les événements, et des notions additionnelles de parenté et de remontée d'événement. Une classification des événements natifs du méta-modèle CMSPEM est aussi présentée.

Chapitre 6 : Implémentation

Ce chapitre décrit l'implémentation des éléments fondamentaux de l'architecture de support au développement collaboratif identifié dans le chapitre 3. Il s'agit de deux éditeurs de modèles CMSPEM réalisés sous Eclipse, et d'un serveur de processus CMSPEM communiquant avec d'autres outils de développement via une interface HTTP/REST. Pour chacune de ces réalisations, une description de l'implémentation est faite, ainsi qu'une explication de la contribution de leurs fonctionnalités majeures à l'architecture globale du chapitre 3.

Chapitre 7 : Validation

Le présent travail de thèse est validé selon deux axes. Une première validation est faite sur les éléments clés (liens profonds et crochets) du modèle conceptuel du chapitre 3. Cette validation s'appuie sur des données de 219 projets open source, afin de mettre en évidence les prévisions faites par le modèle conceptuel de support au développement collaboratif. La seconde validation s'intéresse à l'application pratique des outils présentés au chapitre 6 pour supporter des situations pratiques de collaboration. Pour ce faire, des intégrations avec des outils existants sont réalisées, et des pistes additionnelles d'exploitation des outils développés sont abordées.

Chapitre 8 : Conclusion

La conclusion du manuscrit récapitule la question de recherche et la contribution faite. Elle présente aussi un retour critique sur le travail conceptuel effectué, ainsi que celui d'implémentation. Enfin, des pistes d'extension de la contribution sont identifiées.

Chapitre 9 : Publications

Le présent travail de thèse a fait l'objet de publications dans diverses conférences et journaux. Ce chapitre liste ces publications, et résumé les idées essentielles qui y sont développées.

Chapitre 10 : Annexes

Les annexes du présent document contiennent des éléments clés de l'implémentation des prototypes qu'il nous n'avons pas jugé approprié d'inclure dans le texte du chapitre 6. Il s'agit des concepts clés du méta-modèle SPEM, des règles OCL décrivant la sémantique statique du méta-modèle CMSPEM, de la grammaire de la syntaxe textuelle du méta-modèle CMSPEM, des récits utilisateur pour le serveur CMSPEM, et de la documentation de l'API CMSPEM.

Chapitre 2

État de l'art

2.1 Introduction

La description de la collaboration dans le cadre du génie logiciel est une première étape nécessaire pour son support automatisé. D'une part, un langage de description de la collaboration est utile pour comprendre et améliorer les habitudes de travail actuelles. D'autre part, un tel langage sert à mettre en place et faciliter de nouvelles manières de travailler.

Un travail de recherche considérable existe sur la collaboration, et illustre la grande variété des conceptions possibles de la collaboration. Des méthodes de management jusqu'à l'ingénierie des processus, il y a un riche éventail de conceptualisations qui touchent divers aspects relatifs à la collaboration comme les acteurs, les rôles, les activités, les produits, la communication, etc.

Dans le cadre de ce travail, nous nous intéressons à la contribution des processus logiciels au support de la collaboration. Par conséquent, nous étudions en premier lieu les formalismes proposés pour modéliser les processus de développement logiciels. D'autre part, nous nous intéressons à la question de la modélisation de la collaboration dans le cadre des processus logiciels.

Ce chapitre est un état de l'art des travaux existants sur la collaboration (section 2.2), les processus logiciels (section 2.3), et les processus collaboratifs (section 2.4). L'objectif principal est d'identifier les approches et les formalismes pertinents, et les préoccupations auxquelles répondent les outils existants.

Les travaux analysés sont évalués sur la base, d'une part, de leur prise en compte des différents aspects de la collaboration, et, d'autre part, de leur efficience (pouvoir d'analyse et de prédiction des formalismes, assistance fournie par les outils).

2.2 La collaboration

La collaboration assistée par ordinateur est l'utilisation des ordinateurs pour améliorer la capacité des humains à collaborer. La collaboration est, en des termes simples, l'action

de travailler conjointement avec d'autres personnes [Part 09]. Elle a été définie comme une activité synchrone, coordonnée, qui est le résultat d'une tentative continue de construire et maintenir une conception partagée d'un problème et de sa solution [Rosc 94].

La collaboration peut être vue comme une technique pour permettre à un groupe d'individus d'être plus efficaces que la somme de leurs efficacités individuelles, auquel cas on parle d'intelligence collective [Weis 05]. Cependant, en général, la collaboration se focalise sur les problèmes qui surviennent dans un travail de groupe, la situation étant imposée par la taille du projet, une seule personne ne pouvant posséder toute la connaissance nécessaire [Boss 10, Whit 07].

L'effort de collaboration peut être compliqué quand les soucis de communication, de reconnaissance d'effort, de confiance, etc. engendrent beaucoup de surcharge cognitive chez les participants. En ingénierie logicielle, ces problèmes sont encore plus complexes à cause de la distribution géographique fréquente des équipes, de la quantité d'information partagée en jeu, de la communication entre des personnes ayant des expertises différentes, et de la grande complexité et interdépendance des artéfacts [Mist 10].

2.2.1 Modélisation de la collaboration

Dans [Polt 09], Poltrock et al. identifient huit approches pour modéliser la collaboration, dont quatre prescriptives et quatre autres descriptives.

Les approches prescriptives sont les suivantes :

- Les diagrammes d'activité UML qui montrent le séquencement des activités ainsi que les rôles responsables de ces activités.
- Les modèles de workflow qui sont conceptuellement similaires aux diagrammes d'activité UML (rôles, activités, dépendances entre tâches), mais incluent aussi toute une suite de fonctionnalités de gestion d'erreurs et d'exceptions, de manière à pouvoir utiliser les moteurs de workflow pour le travail de coordination.
- La théorie de la coordination [Malo 03] qui se concentre sur la gestion des dépendances (flux, partage, ou adaptation) entre activités.
- GOMS (goals, operator, method, selection rules) [Card 83] qui est une méthode pour représenter les actions d'un utilisateur interagissant avec un ordinateur.

Les approches descriptives sont les suivantes :

- Grounded theory [Glas 67], qui est une méthode générale consistant à observer, enregistrer, et analyser la réalité, afin d'en extraire une théorie. Elle peut être appliquée à la collaboration afin de construire une meilleure compréhension de ses mécanismes.
- Les modèles temporels qui se fondent sur l'observation selon laquelle les humains sont des créatures d'habitude et de rythme. L'objectif est donc d'observer la chronologie de ce qu'ils font, afin d'en extraire des motifs.

– Les réseaux sociaux qui sont utilisés pour identifier la configuration ou circulation de la confiance, l'information, et la motivation.
– La perspective *langage-action* [Scho 01] qui se concentre sur les conversations, mettant en relief les réponses possibles à des questions ou sollicitations.

Dans la suite de cette section, nous présentons chacune de ces approches. Nous omettons les diagrammes d'activités UML vu qu'ils sont largement connus en génie logiciel, et les modèles de workflow qui sont non seulement largement connus, mais sont aussi principalement utilisés pour les processus métiers. De plus, nous décrivons un méta-modèle de description de la collaboration proposé par Hawryszkiewycz [Hawr 05], et qui relève de l'approche descriptive.

2.2.1.1 Théorie de la coordination

La théorie de la coordination [Malo 94, Malo 03] a été proposée pour étudier la coordination, considérée comme la gestion des dépendances entre activités, de manière interdisciplinaire. En effet, ce champ de recherche rassemble des idées du génie logiciel, du management, de la recherche opérationnelle, de l'économie, de la linguistique, et de la psychologie. Par conséquent, la théorie est générique et très abstraite à dessein, afin de pouvoir s'appliquer aussi bien à des systèmes humains qu'à des systèmes informatiques ou biologiques.

La théorie de la coordination consiste à caractériser les différentes sortes de dépendances, et à identifier les différents processus de coordination qui peuvent être utilisés pour les gérer [Malo 94].

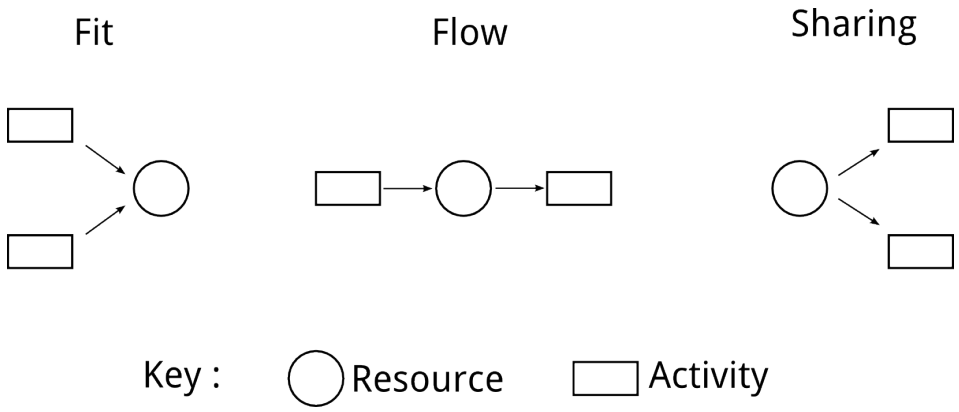

FIGURE 2.1 – Dépendances de base de la théorie de la coordination

Un processus de coordination désigne une organisation de travail utilisée pour gérer une certaine dépendance. Si des activités nécessitent la même ressource partagée par exemple, cette dépendance peut être gérée par le processus "premier arrivé/premier servi". La théorie s'intéresse, entre autres, aux processus de coordination pour gérer des ressources partagées, les relations "producteur/consommateur", les contraintes de simultanéité, et les dépendances "tâche/sous-tâche". Ces différents processus ont été classés en 3 grands groupes : "flow", "sharing", et "fit" [Malo 03] (voir figure 2.1). Une dépendance de type "flow" est caractérisée par

une activité qui produit une ressource dont a besoin une autre activité, le type “sharing” correspond à l’utilisation de la même ressource par plusieurs activités, et le type “fit” a lieu quand plusieurs activités produisent la même ressource (les parties produites devant “s’emboiter” correctement).

2.2.1.2 GOMS

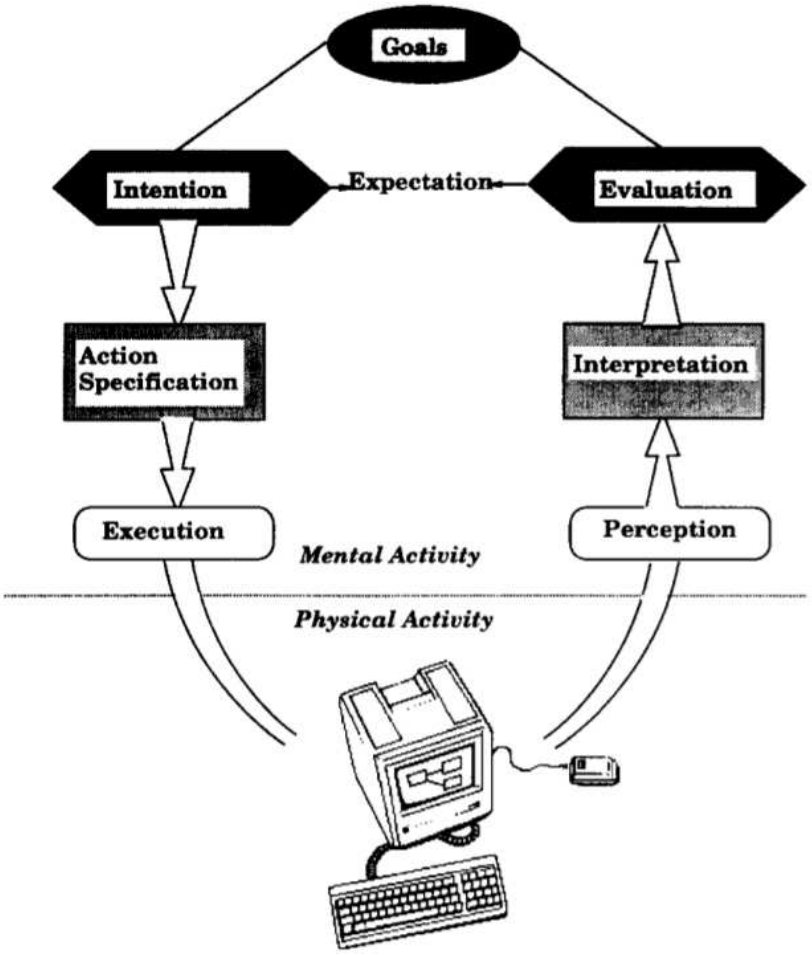

FIGURE 2.2 – Étapes de la réalisation d’une activité utilisateur selon le modèle GOMS

Les travaux sur l’interaction homme-machine se sont aussi intéressés à la collaboration, notamment à travers le projet GOMS [Card 83, Olso 90, Carr 97]. Le projet GOMS a proposé un framework permettant d’analyser systématiquement les buts, méthodes, et actions qui composent les interactions homme-machine routinières. Il se distingue par la prise en compte des structures cognitives qui sous-tendent les comportements manifestes. En effet, GOMS modélise le cycle de décision, depuis la perception jusqu’à l’action, et la connaissance nécessaire pour réaliser une tâche, avec suffisamment de détail pour prédire le comportement humain sur des tâches réelles [Olso 90].

Selon le modèle GOMS, un utilisateur commence par percevoir une activité sur son écran.

Il évalue si cette activité correspond à ses attentes, vu le but qu'il s'est donné. Il forme alors une intention de la prochaine action, trouve la manière de réaliser cette action sur le système avec lequel il interagit (dans une documentation par exemple), puis exécute les mouvements moteurs requis. Ceci produit une nouvelle activité sur l'écran, et le cycle recommence (voir figure 2.2).

Pour modéliser la collaboration avec GOMS, des modèles décrivant les interactions de chaque participant avec le système logiciel utilisé sont dans un premier temps créés. Ensuite, des méthodes sont ajoutées au système, pour représenter les interactions d'un participant avec ses collègues. Ainsi, dans le cycle de la figure 2.2, les actions possibles peuvent inclure "diffuser un message" ou "envoyer un document à un autre participant". Le modèle complet peut alors être simulé, afin de prédire les performances de l'équipe lors de la mise en œuvre de l'activité modélisée. Cependant, Kieras et al. [Kier 04] ont trouvé de larges écarts entre les performances prédites et celles réelles, parce que les participants ne collaboraient pas toujours exactement comme prévu, et que les styles d'interaction et de communication variaient beaucoup d'une équipe à l'autre.

2.2.1.3 Grounded Theory

Grounded Theory est une méthode pour développer une compréhension de comment et pourquoi un certain comportement a lieu dans une circonstance particulière [Glas 67]. La méthode se base sur l'observation, la prise de notes, la collection de données, l'annotation, et le tri. Elle vient des sciences sociales, et a été à l'avant garde de la "révolution qualitative" qui encourage la collecte et l'analyse inductive de données sur un phénomène étudié. Des interprétations analytiques des données recueillies sont construites au fur et à mesure, et ces interprétations guident les collectes suivantes, collectes qui servent à raffiner les interprétations, ainsi de suite [Char 03].

L'application de la méthode "Grounded Theory" à la collaboration en ingénierie logicielle consiste à observer les participants à un projet logiciel dans leurs occupations de tous les jours, et tenter d'inférer la structure qui sous-tend leur comportement. Ainsi, "Grounded Theory" n'est pas un modèle de collaboration, mais une approche pour construire un tel modèle.

Un exemple d'utilisation de la méthode "Grounded Theory" pour analyser la collaboration est donné dans [Polt 09]. Un processus de gestion de changement a été observé, les participants interviewés, et des artéfacts collectés. Ceci a permis de constater que les coordinateurs utilisaient un classeur Excel avec des informations détaillées sur chaque changement, y compris la personne qui l'a créé, la personne qui l'a relu, son état actuel, la décision du comité de contrôle de changement, ainsi que les raisons d'un éventuel retard. L'existence d'un tel classeur fut une surprise, dans la mesure où il dupliquait l'information automatiquement générée par le système de workflow utilisé dans le projet. Ceci a révélé que les coordinateurs avaient relativement peu de confiance dans le système de gestion de changement. Une telle observation, comme préconisé dans l'approche "Grounded Theory", permet de mettre à jour le modèle de collaboration en cours de développement, et guide les prochaines observations à recueillir sur le terrain (identifier les habitudes de mise à jour du classeur, par exemple).

2.2.1.4 Les modèles temporels

Les modèles temporels sont produits par un ensemble d'approches [Bego 03, Fish 04, Huds 02] qui s'appuient sur le constat selon lequel le comportement des participants à un projet comporte des motifs, ou rythmes, comme leurs heures habituelles d'arrivée au bureau, leurs temps de pauses, les réunions régulières auxquelles ils participent, leurs déplacements entre divers sites de travail, etc. La recherche sur le travail collectif en présentiel a montré que les participants partagent un certain sens de ces motifs, et utilisent leur vigilance par rapport à ces rythmes de travail pour coordonner leurs travaux et se construire des attentes sur la disponibilité de leurs collègues [Zeru 85, Redd 02]. L'idée générale des modèles temporels est donc d'observer l'évolution temporelle de différentes activités humaines, afin d'en extraire des motifs. Par exemple, l'observation des dates de modifications d'un document, associée aux différents intervenants sur le document, donne des indications sur l'organisation de travail utilisée lors de la révision collaborative d'un document.

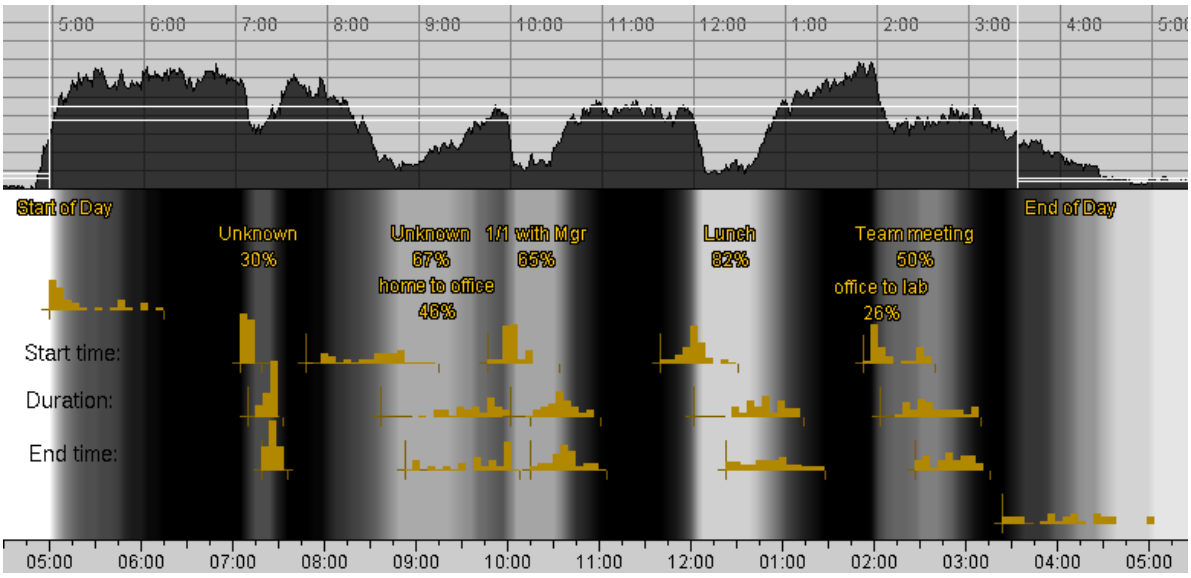

FIGURE 2.3 – Exemple de modèle temporel représentant des niveaux d'activité

L'utilisation de cette connaissance des habitudes temporelles des collègues est sérieusement perturbée dans une équipe distribuée. En effet, il devient difficile d'obtenir les informations sur les habitudes temporelles des collègues. Dans ces cas, le modèle temporel individuel de chacun des participants va être exploité, afin de prédire son comportement futur. Ces prédictions sont mises à disposition des collègues, sous forme d'outil d'inférence de statut par exemple [Bego 03] ("au bureau", "sorti pour un moment", "parti manger", "en réunion", etc.).

La figure 2.3 (extraite de [Bego 03]) montre un modèle temporel de niveau d'activité, construit à partir de données d'utilisation d'ordinateur, d'envoi d'emails, de conversation téléphonique, de calendriers, et de capteurs de présence. La partie supérieure de la figure montre la variation du niveau global d'activité, du début jusqu'à la fin de la journée. La partie inférieure reprend ces niveaux sous forme de zones claires (peu d'activité) et de zones sombres (activité intense), sur lesquelles sont superposées des statistiques sur les périodes d'inactivité.

D'une part, on y trouve la distribution de probabilité des heures de début et de fin de journée. D'autre part, pour chaque période de pause dans la journée, la figure montre la distribution de probabilité des heures de début et de fin, et de la durée de la pause.

2.2.1.5 Les réseaux sociaux

Dans [Cros 04], Rob Cross et Andrew Parker, deux consultants pour le "Knowledge and Organizational Forum" d'IBM, ont analysé une soixantaine de réseaux informels dans des organisations à travers le monde. Ils ont trouvé que les réseaux de transmission de confiance, de connaissance, et de motivation sont critiques pour le fonctionnement d'une organisation. De plus, ces réseaux sont souvent très différents de ceux correspondant aux hiérarchies organisationnelles officielles et aux processus métier.

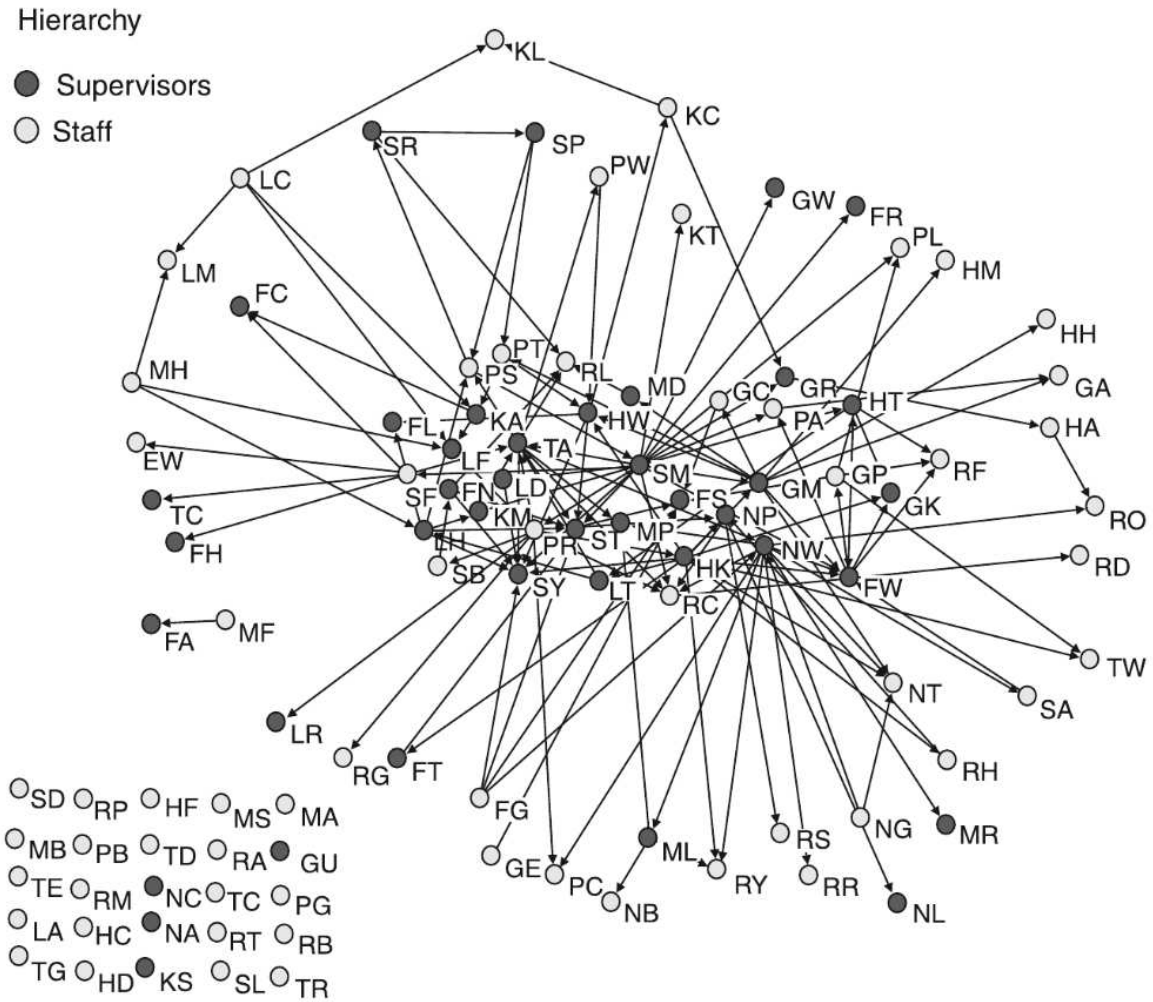

FIGURE 2.4 – Relations dé-énergisantes entre des responsables et des ingénieurs d'une entreprise

Un réseau social est un graphe orienté dans lequel les nœuds représentent des individus,

et les arcs représentent le flux entre individus d'une entité, qui peut être de la confiance, de l'information, ou encore de la motivation. La construction de ces réseaux à partir d'interviews, permet de détecter des problèmes organisationnels de communication ou de coordination, et de proposer des changements pour y remédier (rapprocher un groupe de personnes, éloigner un participant problématique d'un groupe, etc.).

Un exemple d'étude de réseaux sociaux est celui sur les relations énergisantes [Bake 03] où il est question d'évaluer l'impact des relations interpersonnelles sur la sensation subjective d'énergie. Le concept d'énergie vient de la psychologie et de la sociologie, et est défini comme "un type d'excitation affective positive, qu'une personne peut ressentir comme émotion – réponse courte à un événement spécifique – ou humeur – états affectifs plus longs qui ne sont pas forcément des réponses à un événement spécifique" [Quin 05]. La figure 2.4 est un réseau social construit pour matérialiser les relations dé-énergisantes dans une entreprise. Un arc allant de A vers B signifie que l'individu A est démotivé par la présence de l'individu B. Les personnes avec plusieurs arcs entrants (TA, SY, RC, NW, FW) constituent donc un risque pour l'entreprise, dans la mesure où beaucoup de leurs collèges sont démotivés par leur présence. L'analyse d'un tel réseau social permet, entre autres, de réorganiser les équipes pour que le plus grand nombre de personnes se sente à l'aise, ou alors de lancer des initiatives ciblées pour améliorer les relations interpersonnelles.

2.2.1.6 La perspective langage-action

La perspective langage-action [Wino 86, Scho 01, Colo 02] se base sur les conversations pour analyser la collaboration dans un environnement de travail. Par exemple, quand une personne demande à un collègue de réaliser une tâche, le collègue peut accepter, décliner, ou faire une proposition alternative. Chacune des réponses met sur la table différentes alternatives de conversation pour le demandeur. Les conversations peuvent servir à clarifier, à orienter, ou à donner de nouvelles possibilités, et contiennent donc des "actions". La perspective langage-action se base sur le constat fondamental selon lequel chacun de ces types de conversation a une structure régulière, qui peut être analysée, formalisée, et implémentée dans des outils de coordination. Ainsi, la perspective langage-action considère le langage, non pas comme un simple véhicule pour l'information, mais comme un moyen d'action.

Assertive	Commissive	Directive	Expressive	Declarative
assess diagnose evaluate identify needs inform observe record	evaluate investigate review set goal	investigate order refer request prescribe	query rule out suspect	admit discharge transfer

FIGURE 2.5 – Exemple d'application de la perspective langage-action à la médecine

Utiliser la perspective langage-action dans la modélisation de la collaboration revient à

identifier, pour un domaine donné, les termes clés structurant les conversations, et se rapportant à des actions. La figure 2.5 tirée de [Scho 99] montre une telle application pour le domaine médical. Ces termes sont rangés dans les cinq catégories de fonctions "illocutoires" [1] proposées par Searle [Sear 85] : l'assertion (le locuteur se positionne par rapport à la valeur de vérité d'un énoncé), l'engagement (le locuteur fait la promesse d'une action future), l'ordre (énoncé ayant pour objectif de faire réaliser quelque chose par celui/celle qui l'écoute), l'expression (le locuteur fait part de ses attitudes et émotions par rapport à une proposition), et la déclaration (changer la nature de la réalité pour qu'elle soit en accord avec ce qui est dit, comme prononcer une sentence ou déclarer deux personnes mariées).

Ce termes identifiés sont ensuite reliés aux catégories de personnel (médecins, infirmiers, etc.) qui les utilisent, ainsi qu'aux circonstances particulières (visite médicale, admission de patient, opération) et aux interlocuteurs avec lesquels ils sont employés. Ce croisement d'information permet de mettre en évidence des organisations de travail. Par exemple, un tel croisement peur révéler qu'une évaluation ("évaluate", dans la catégorie "Commissive") faite par un infirmier ne peut être envoyé qu'à un autre infirmier ou qu'un docteur et un infirmier peuvent tous les deux recevoir des requêtes ("request", dans la catégorie "Directive").

2.2.1.7 Le méta-modèle de collaboration de Hawryszkiewycz

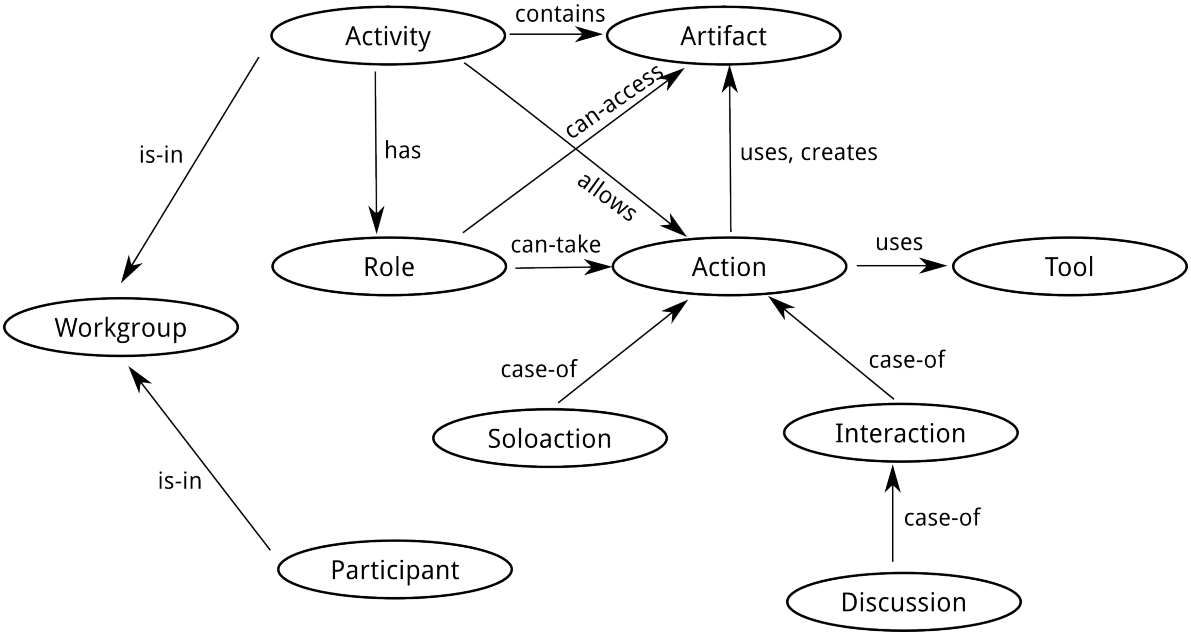

FIGURE 2.6 – Concepts du méta-modèle de collaboration de Hawryszkiewycz

Dans [Hawr 05], Hawryszkiewycz a identifié les concepts sémantiques contenus dans diverses approches de modélisation de la collaboration, puis les a combinés pour former un méta-modèle de description de systèmes collaboratifs. Les approches étudiées incluent les modèles de coordination (en majorité les workflows, qui mettent l'accent sur la séquence de

1. Une fonction illocutoire est, dans la théorie du langage de John Austin [Aust 75], le message porté par un énoncé, au delà de son sens immédiat.

tâches), les modèles communautaires (partage de connaissances explicites), et les modèles sociaux. Hawryszkiewycz montre que les formalismes standard (UML, flux de données, modèles entités-relations) conviennent aux processus stables, où le flux de transactions est défini à l'avance, et les structures de données invariantes.

Le méta-modèle proposé utilise les concepts de groupe de travail, de rôle, de participant, d'action, d'activité, et d'artéfact pour décrire la collaboration (voir figure 2.6). De plus, une série de commandes a été créée sur la base de ces concepts ("ajouter un rôle à un groupe de travail" par exemple). Ces commandes doivent être mises à disposition de l'utilisateur final, comme c'est le cas dans l'outil LiveNet proposé. Enfin, le concept d'agent logiciel est utilisé pour aider les utilisateurs à faire correspondre la sémantique du méta-modèle à la sémantique de leur domaine métier (les agents peuvent observer leur environnement, et déclencher des changements).

2.2.2 Problèmes et approches de collaboration

La présente sous-section explore les divers problèmes qui se posent dans une situation de collaboration, et les approches existantes pour les résoudre et améliorer l'efficacité du travail collaboratif.

2.2.2.1 Problèmes courants

Cramton a analysé les problèmes courants rencontrés par une équipe géographiquement dispersée travaillant sur un projet collaboratif. "Maintenir une connaissance partagée" s'est révélé être la préoccupation centrale. Les autres préoccupations sont les difficultés à communiquer et à garder les informations de contexte, les différences dans la vitesse d'accès à l'information, et l'interprétation de la signification du silence [Cram 01].

Dans [Herb 07], Herbsleb explore les problèmes de coordination qui se posent dans une équipe géographiquement dispersée. Il explique comment une communication déficiente, le manque d'information, et les incompatibilités peuvent rendre la coordination difficile. Une emphase particulière est placée sur "comment réaliser la complémentarité entre l'organisation et l'architecture du produit", une idée qui peut être reliée à la loi de Conway : "la structure d'un système est à l'image de la structure de communication de l'organisation qui l'a conçu" [Conw 68].

2.2.2.2 Approches de collaboration

La collaboration peut avoir plusieurs significations, dont [Gold 02] :

- La collaboration conceptuelle
- La collaboration pratique
- La collaboration éducationnelle

La collaboration conceptuelle (partage de responsabilité, information, et leadership) et la collaboration pratique (décomposition du travail, intégration de résultats, et gestion de la

différence de niveaux d'expertise) sont les plus pertinents pour la plupart des projets. Bien que la collaboration éducationnelle (s'entraider pour améliorer les habitudes de travail) soit importante pour les entreprises, elle n'est pas prise en compte dans tous les projets, comme par exemple le projet Galaxy.

Quel que soit le sens donné à la collaboration, les projets globalement collaboratifs mettent en jeu quatre types de travaux [Robi 00] :

- Le travail collaboratif obligatoire
- Le travail collaboratif sollicité
- Le travail collaboratif ad hoc
- Le travail individuel

Le travail collaboratif obligatoire (réunions régulières planifiées avant le début du projet) et le travail collaboratif sollicité (revues et réunions de planifiées en cours de projet) sont d'un intérêt moyen pour le développement logiciel, dans la mesure où ils sont effectués avec des outils qui ne sont pas propres au développement logiciel. La collaboration ad hoc (courtes séances de travail, généralement en tête à tête, qui souvent précèdent ou suivent de longues séances de travail individuel, et servent à décomposer le travail ou fusionner les contributions) et le travail individuel sont plus pertinents pour le développement logiciel. Il faut noter que la collaboration ad hoc est la forme la plus fréquente de travail collaboratif [Perr 94, Robi 00].

Dans son sens le plus général, le travail collaboratif demande un facilitateur – celui en charge du rôle de coordination. Briggs et al. [Brig 06] ont identifié des techniques qu'un groupe peut utiliser pour créer certains motifs prévisibles de collaboration, sans l'intervention d'un facilitateur. Ces techniques sont appelées des thinkLets, et peuvent servir de briques de base pour la création de processus collaboratifs [Kolf 06]. Le concept de thinkLet est fondamental dans le domaine de l'ingénierie de la collaboration. Chaque thinkLet peut produire une variation prévisible et bien définie d'un des motifs principaux de collaboration [Brig 06] :

- Générer : Diminuer le nombre de concepts dans le répertoire de concepts partagés par le groupe.
- Réduire : Diminuer le nombre de concepts considérés comme nécessitant plus d'attention par le groupe.
- Clarifier : Augmenter la compréhension partagée des concepts et des mots et phrases utilisés pour les exprimer.
- Organiser : Augmenter la compréhension des relations entre les concepts sur lesquels travaille le groupe.
- Évaluer : Augmenter la compréhension des valeurs relatives des différents concepts considérés.
- Créer du consensus : Augmenter le nombre de membres du groupe disposés à s'engager pour une proposition.

Une simple classification des activités quotidiennes de collaboration peut être faite par rapport au temps et à l'espace. Cette structure distribue les activités en quatre cadrants conceptuels [Cook 07] :

- Colocalisé et synchrone (exemple : réunions en présentiel)
- Colocalisé et asynchrone (exemple : modification de document au bureau)
- Distribué et synchrone (exemple : discussion instantanée)
- Distribué et asynchrone (exemple : email)

2.2.3 Technologies de support à la collaboration

Dans les divers outils proposés, dans la recherche aussi bien que dans l'industrie, pour supporter le travail collaboratif, certaines technologies clés reviennent fréquemment. Un important effort de recherche et de développement a contribué à perfectionner ces technologies au cours des années. La présente sous-section explore certaines des plus pertinentes.

2.2.3.1 Gestion de notification

La gestion de notification ou de vigilance porte sur la manière dont les développeurs sont informés de la présence et des actions de leurs pairs. Elle sert à réduire le vide créé par la distribution géographique des équipes de développement, en fournissant aux développeurs des informations de contexte sur leur travail. Les stratégies de notification vont des notifications en temps-réel jusqu'à l'information manuellement partagée par email par exemple.

La problématique principale de la gestion de notification est la pertinence de l'information. Avec une approche naïve, les développeurs peuvent être submergés par un déluge d'informations sur les activités en cours dans le projet. Mais ceci n'est pas efficace, si l'information n'est pas utile au développeur particulier qui le reçoit, ou de trop bas niveau pour être prise en compte. Par exemple, savoir qu'une méthode a été renommée à l'autre bout du monde est bien moins utile que d'être informé d'une refactorisation en cours de tout un module.

Les outils de vigilance à l'information peuvent être classés en trois catégories [Cook 07] :

- Vigilance par rapport aux vues. Cette catégorie contient les modifications d'artéfacts qui ne correspondent pas immédiatement ou directement au modèle sémantique du projet. Cela inclut la proximité physique, la modification de vues (comme les changements de disposition), les modifications textuelles et les voisinages de code.
- Vigilance par rapport au modèle sémantique. Ceci inclut les changements sémantiques, les rapports d'impact, les métriques logicielles, et les résultats de cas de test.
- Vigilance par rapport au workflow. Toutes les modifications d'artéfact qui ne sont pas liées au modèle sémantique du projet peuvent être classées dans cette catégorie. Il s'agit par exemple de changement dans le catalogue de défauts logiciels ou de changements dans la documentation.

Les systèmes de gestion de version traditionnels ne génèrent des notifications que lorsqu'un développeur enregistre son travail[2]. Les outils populaires comme SVN et Git offrent la possibilité de réagir à un commit, ce qui peut être utile pour bâtir un framework de notification. Avec juste un système basique de gestion de version, les développeurs doivent normalement vérifier manuellement la disponibilité de nouveaux changements dans le dépôt du projet.

Dans [Cook 03], Cook et al. décrivent CAISE, un framework d'ingénierie logicielle collaborative. CAISE est basé sur une architecture client-serveur. Le serveur maintient un modèle continuellement mis à jour du projet, et chaque éditeur local y envoie les modifications faites par les développeurs. A cet effet, le serveur a un arbre sémantique à jour de tous les artéfacts, et peut automatiquement détecter les dépendances entre ces artéfacts.

2.2.3.2 Gestion des mises à jour

Gérer les mises à jour consiste à s'assurer que toutes les implications d'un changement sur un artéfact sont prises en compte. La gestion de mises à jour s'appuie sur la détection de changement et la résolution de dépendances.

En génie logiciel, la détection de changement est traditionnellement faite au niveau des fichiers, en utilisant les dates de modification. Autrement dit, les changements ne sont pris en compte que lorsqu'un fichier est enregistré. Il n'y a, en général, pas de support des changements au niveau de l'éditeur. Autrement dit, les changements dans la mémoire de travail d'un éditeur n'ont pas d'impact sur les autres artéfacts, jusqu'à ce que le document soit enregistré. Certains environnements de développement intégrés, comme Eclipse, peuvent vérifier le code au fur et à mesure qu'il est saisi (pour la coloration syntaxique par exemple), et recharger du code à chaud dans une machine virtuelle[3]. En règle générale, les éditeurs permettent de configurer un script à exécuter chaque fois qu'un fichier est enregistré.

Diverses versions de l'utilitaire Unix `make` ont été utilisées pour gérer la propagation des modifications. Avec `make`, les dépendances sont généralement statiques, et spécifiées comme des relations entre fichiers. Des outils plus récents comme `ant` ont des règles internes qui couvrent les besoins du développement Java (compilation, exécution de tests, packaging, etc.). Un environnement de développement intelligent peut exploiter la sémantique du langage utilisé pour améliorer le suivi des modifications (un éditeur Java par exemple peut détecter qu'un changement dans une interface ou une implémentation affecte un autre artéfact).

La gestion de mises à jour peut être faite à la volée (autrement dit, en cours d'édition d'un artéfact) ou lorsque l'artéfact est enregistré. La gestion de mises à jour à la volée demande la coopération de l'éditeur (ou de l'environnement de modélisation). Dans CAISE [Cook 03] par exemple, un arbre sémantique représentant l'artéfact manipulé est maintenu à jour sur le serveur qui exécute des moteurs de détection de conflits en temps-réel, et envoie les résultats d'analyse aux éditeurs. Dans ModelBus [Srip 08], certaines commandes de l'éditeur sont interceptées (chargement, sérialisation, navigation, etc.) pour pouvoir faire de la détection de changement.

2. On parle de "commit" dans les systèmes de gestion de version.
3. Recharger le code dans une machine virtuelle est typiquement fait seulement après l'enregistrement du fichier.

2.2.3.3 Fusion

Fusionner les contributions est une étape nécessaire lorsqu'un verrouillage optimiste est utilisé (autrement dit, lorsque les modifications concurrentes sur un même artéfact sont autorisées). Étant donné que plus d'une personne peut travailler sur un artéfact à la fois, il faut trouver le moyen d'intégrer plus tard leurs modifications. Cette problématique est raisonnablement résolue pour les formats textuels, puisqu'il existe de nombreux outils pour calculer et exporter des différences entre fichiers dans un format structuré.

La fusion s'appuie principalement sur les outils de calcul de différences. Il y a eu divers efforts pour concevoir des algorithmes de calcul de différences (ou de deltas) pour les modèles [Mehr 05, Srip 08]. Le développement de ces algorithmes est surtout compliqué par le fait que les modèles sont des artéfacts à base de graphes, plutôt que linéaires.

La fusion nécessite de pouvoir identifier les changements intervenus sur un artéfact. Un changement sur un fichier peut être identifié de diverses manières :

- Avec une comparaison ligne à ligne. Ceci est utilisé pour le contrôle de version appliqué au code, et est généralement suffisant, bien qu'il ne puisse pas toujours suffire pour identifier la sémantique d'un changement (renommage, déplacement, etc.).
- Identification à base d'heuristiques dans les graphes. Cette stratégie a été utilisée pour les modèles, mais n'est pas optimale : il peut y avoir des faux-positifs et des faux-négatifs.
- Identification à base d'ID (code d'identification) dans les graphes. Pour ce faire, on assigne un code d'identification unique à chaque élément de modèle. Cette approche est utilisée dans ModelBus [Srip 08], et demande un adaptateur spécifique pour chaque éditeur, puisqu'il nécessite l'introduction d'identificateurs externes dans le modèle.
- Avec l'aide d'un éditeur. Dans les cas où il est possible de le faire, cela produit une meilleure capture de la sémantique du changement. En effet, un éditeur peut identifier par exemple un large effort de refactoring comme un seul changement sémantique, au lieu d'une série de modifications déconnectées, simplement parce que l'outil de refactoring utilisé est fourni par l'éditeur.

2.2.4 Outils de collaboration assistée par ordinateur

La collaboration assistée par ordinateur traite de "la manière dont les activités collaboratives et leur coordination peut être supportée par le biais de systèmes informatisés" [Cars 99]. Elle combine une compréhension de la manière de travailler des individus dans un groupe avec les technologies clés de réseaux informatiques, du matériel physique associé, de logiciel, de services, et de techniques. Un environnement de travail collaboratif supporte les individus dans leurs travaux individuels et collectifs.

Les applications ou services suivants sont considérés comme faisant partie d'un environnement de travail collaboratif : la messagerie électronique, la messagerie instantanée, le partage d'applications, la vidéoconférence, la gestion des espaces de travail collaboratif et de documents, le partage d'agenda, la gestion des tâches et des workflows, l'édition collaborative sur

les wikis, et les blogs avec les billets catégorisés par groupes, communautés, ou autres concepts supportant la collaboration.

Dans la présente sous-section, nous nous intéressons au support de la collaboration en génie logiciel. En premier lieu, nous étudions des environnements qui regroupent des outils de support à la collaboration, autrement dit, des environnements "intégrés" de support à la collaboration. En second lieu, nous nous focalisons sur les outils de gestion de versions, cette préoccupation étant au cœur des activités quotidiennes de développement. Ceci permet d'obtenir une vue détaillée et représentative du support outillé à la collaboration en génie logiciel.

2.2.4.1 Environnements de développement intégré collaboratifs

Nous décrivons ici un exemple industriel, Jazz [Booc 03], qui est l'un des efforts les plus récents pour construire un environnement de développement logiciel collaboratif. Nous abordons aussi trois propositions académiques couvrant aussi bien la collaboration sur le code (Syde [Hatt 10], CoDesign [Youn 10]) que sur les modèles (ModelBus [Srip 08]).

2.2.4.1.1 IBM-Jazz Dans l'article "Collaborative development environments" [Booc 03], Grady Booch et Alan W. Brown, de Rational Software Corporation (détenu par IBM), ont présenté la vision qui se matérialisera plus tard avec la plate-forme Jazz. La vision a germé de l'observation selon laquelle la production de logiciel est une activité fondamentalement collaborative, dans laquelle "toutes les parties prenantes d'un projet – même séparées par le temps ou la distance – peuvent négocier, faire du brainstorming, discuter, partager de la connaissance, et généralement travailler ensemble pour réaliser une tâche, le plus souvent pour créer un livrable exécutable et ses artéfacts liés" [Booc 03]. Un environnement virtuel qui peut héberger et faciliter les interactions précédemment mentionnées est appelé "environnement de développement collaboratif" (CDE : Collaborative Development Environment).

Bien que Booch et Brown reconnaissent que les divers outils supportant de telles interactions ont toujours existé (depuis la messagerie électronique aux forums de discussion, outils de gestion de défauts logiciels ou de versions, etc.), ils avancent que "les communautés de pratique sont fragiles et ne peuvent fleurir qu'avec un dosage adéquat de technologie et d'expérience utilisateur [...] C'est la nature de support intégré d'un CDE qui le distingue de la variété de produits fonctionnels disparates typiquement utilisés aujourd'hui".

Le but ultime est donc de créer une "surface lisse" qui peut éliminer ou automatiser les activités non créatives, et encourager des modes créatifs et intenses de communication entre les différentes parties prenantes d'un projet. Si le but d'Eclipse est d'améliorer la productivité de l'individu, un environnement de développement collaboratif a pour but d'améliorer la productivité d'une équipe en tant qu'un tout.

Après une étude de produits notables qui ont certaines des caractéristiques voulues, les auteurs proposent une liste des fonctionnalités qu'un environnement de développement collaboratif doit avoir, organisée en trois grands groupes :

– Des fonctionnalités de coordination (gestion de calendrier, planification, gestion d'événements, tableaux de bord, métriques, recherche, etc.)

– Des fonctionnalités de collaboration (discussions, réunions, discussion instantanée, sondages, tableaux blancs partagés, etc.)
– Des fonctionnalités de construction de communautés (protocoles et rituels, publication personnelle de contenu, administration personnelle de projets, champ d'action et leadership, etc.)

IMB a lancé la plate-forme Jazz (`http://jazz.net`) pour implémenter la vision précédente. IBM présente ainsi Jazz : "Jazz est conçu pour transformer la manière dont les gens travaillent ensemble pour construire du logiciel, rendant la livraison de logiciel plus collaborative, productive, et transparente. Jazz peut être conçu comme un framework extensible qui intègre et synchronise dynamiquement les personnes, les processus, et les artéfacts associés aux projets de développement logiciel." Jazz est vu comme l'évolution naturelle de la plate-forme Eclipse. La plate-forme Jazz est une fondation technique sur laquelle s'appuient un ensemble de produits :

– Rational Team Concert (suivi de tâches, gestion de versions, intégration continue, support des processus)
– Rational Quality Manager et Rational Test Lab Manager (gestion de cycle de vie d'application, test)
– Rational Requirements Composer (définition d'exigences)
– Rational Project Conductor (gestion de projet et de ressources)
– Rational Insight (mesure et gestion de performance pour améliorer les projets et processus)
– Rational Build Forge (compilation et gestion de publication de logiciel)
– Rational Asset Manager (gestion des artéfacts techniques et métier réutilisables)

Le code source de Jazz n'est pas disponible en open source. Cependant, une version "express" de Rational Team Concert est disponible pour les petites équipes (au plus dix membres). Les documentations d'API sont librement disponibles, et peuvent être utilisées pour ajouter des extensions aux différentes composantes de la plate-forme Jazz. Actuellement, il n'y a aucun support explicite pour la métamodélisation dans Jazz.

2.2.4.1.2 CoDesign CoDesign [Youn 10] est un projet développé conjointement par Infosys Technologies Limited (Bangalore, Inde) et l'Université de Caroline du Sud, et décrit comme un "framework hautement collaboratif de modélisation logicielle collaborative". CoDesign est construit autour de l'observation selon laquelle, dans les infrastructures distribuées de collaboration s'appuyant sur les systèmes de gestion de versions, les conflits peuvent être détectés seulement après un commit, lorsqu'il est en général trop tard pour une solution automatisée et/ou efficace. Il est donc utile de détecter les conflits en temps-réel, et de notifier les designers concernés, afin de permettre une collaboration fluide dans les équipes géographiquement dispersées [Youn 10].

La contribution principale de CoDesign est un "framework extensible de détection de conflits pour la modélisation collaborative". Le framework est conçu pour supporter l'intégration sans effort de moteurs de détection de conflits et d'outils de modélisation tiers, avec

des adaptateurs appropriés. Les outils de modélisation envoient des événements correspondant aux actions d'édition à un serveur, qui utilise les moteurs de détection de conflits pour identifier les actions non compatibles. Le conflit est automatiquement résolu si possible ; sinon, des notifications sont renvoyées aux outils de modélisation, qui en informent les concepteurs.

Trois catégories principales de conflits sont reconnues par CoDesign :

- Conflits de synchronisation : un designer réalise une action d'édition qui n'a pas de sens dans le contexte d'une autre action d'édition réalisée par un collègue distant (comme ajouter un attribut à une classe qui a été supprimée). Ceci se produit quand l'événement correspondant à une action distance n'est pas reçu à temps.
- Conflits syntaxiques : des règles syntaxiques du méta-modèle (comme les cardinalités) sont violées par l'effet cumulé de modifications concurrentes.
- Conflits sémantiques : des éditions concurrentes qui résultent en une violation d'une règle de cohérence, externe au méta-modèle et explicitement spécifiée avec OCL par exemple.

Le téléchargement de CoDesign est disponible sur requête [4], et le framework a été intégré avec Drools pour la détection des conflits de synchronisation, le vérificateur de méta-modèle GME pour les conflits syntaxiques, et le vérificateur OCL GME pour les conflits sémantiques.

2.2.4.1.3 Syde Syde [Hatt 10] est un outil de l'Université de Lugano en Suisse, qui, comme CoDesign, détecte les conflits en temps-réel dans un environnement de développement distribué. Contrairement à CoDesign, Syde se focalise sur le code source (par opposition aux modèles) et est implémenté comme un plugin qui exploite les riches possibilités de refactoring et de manipulation d'arbre syntaxique Java disponibles sous Eclipse.

L'outil a une architecture client-serveur, avec un serveur central qui stocke les arbres syntaxiques (AST : Abstract Syntax Tree) de chaque espace de travail de développeur. Le serveur héberge aussi des moteurs de détection de conflit qui fonctionnent en comparant les différents ASTs. Les ASTs sont mis à jour avec l'information envoyée depuis les ordinateurs de chaque développeur par des plugins spécialisés. Ces informations de modification, contrairement aux calculs de différences des outils courants de gestion de versions, ne sont pas basées sur les lignes, mais plutôt sur les ASTs, et capturent ainsi plus d'information sémantique. Mieux, les puissantes capacités de refactoring d'Eclipse permettent de capturer des changements substantiels à un projet comme une seule information conceptuelle. Ceci permet de minimiser le nombre d'information de contexte [Hatt 10].

Une contribution majeure de Syde (à côté des changements basés sur les AST) est la variété de fonctionnalités de vigilance. Un plugin peut aider les développeurs à visualiser les dernières classes modifiées, avec une couleur indiquant le développeur à l'origine du changement, et une taille indiquant le degré d'activité sur cet élément par exemple (nuage de mots). Un autre plugin peut montrer les changements récents et leurs auteurs en tant qu'annotations ou infobulles directement dans l'éditeur Eclipse.

4. http://softarch.usc.edu/~ronia/codesign/#download

2.2.4.1.4 ModelBus ModelBus [Srip 08] est un “framework, basé sur les modèles, d’intégration d’outils”, développé à l’origine dans le cadre du projet MODELWARE [IST 04], et maintenu actuellement dans le cadre du projet MODELPLEX [IST 06]. Le projet MODELWARE (2004-2006) a rassemblé un consortium d’universitaires et industriels européens afin d’améliorer les outils, processus, et standards de l’ingénierie dirigée par les modèles. Le projet MODELPLEX (2006-2009), pour sa part, est consacré au développement d’une solution ouverte pour l’ingénierie des systèmes complexes, avec pour objectif d’améliorer la qualité et la productivité.

D’une part, ModelBus est une couche d’interopérabilité qui permet à un outil de modélisation d’utiliser les services d’un autre outil. Cette fonctionnalité est basée sur RCP, avec une sémantique d’appel-par-copie-restauration (autrement dit, une copie du modèle est envoyée à un outil, l’outil met à jour la copie, la version modifiée est renvoyée, et remplace l’original). ModelBus permet à des outils d’envoyer seulement une partie du modèle, afin d’améliorer les performances (moins de données à copier) et le contrôle d’accès (envoyer seulement la partie que le destinataire a la permission de voir/modifier). Les fragments de modèle sont définis par un nœud racine, et une collection de nœuds qui peuvent être inclus dans le fragment : seuls les nœuds qui peuvent être atteints à partir de la racine appartiennent à la collection et sont inclus.

D’autre part, ModelBus est un environnement distribué d’édition collaborative de modèles ; conçu d’après un système de gestion de versions centralisé comme SVN ou CVS. Les modèles sont stockés dans un dépôt central comme des fichiers XMI, ce qui permet de réutiliser une partie des fonctionnalités de CVS, un système de gestion de versions basé sur les fichiers. Les développeurs copient le dépôt (ou une partie) vers leur espace de travail local, font des modifications, puis envoient les mises à jour vers le dépôt central.

La fusion et la résolution de conflits dans ModelBus est basée sur les deltas, qui sont des modèles contenant les changements entre une paire de modèles : le modèle original et le modèle modifié. ModelBus offre des fonctionnalités de calcul de deltas (génération des modèles delta), de détection et résolution de conflits (basée sur la comparaison de deltas), et d’intégration de deltas (en d’autres termes, la fusion). Ces fonctionnalités sont implémentées au niveau MOF, en utilisant Eclipse EMF, ce qui rend ModelBus compatible avec tout modèle basé sur MOF.

Le calcul de deltas dans ModelBus s’appuie sur l’association d’un identifiant unique (un UUID en fait) à chaque nœud dans un modèle. ModelBus peut désigner de manière non ambigüe un nœud de modèle en utilisant le chemin (relatif) du modèle, et l’identifiant du nœud, comme dans “module1/model2.xmi#node-uuid”. Les liens entre les nœuds sont représentés en utilisant les chemins et les identifiants, ce qui permet à ModelBus de détecter des suppressions, ajouts, modifications, et même des déplacements d’éléments entre modèles (refactoring).

La détection de conflits est traitée en comparant les deltas deux à deux. Chaque fois qu’un développeur envoie ses mises à jour sur le dépôt central, deux deltas sont calculés :

– un premier delta, d1, entre la version actuelle du modèle dans l’espace de travail du

développeur, et la version que le développeur a utilisée comme point de départ de son travail

– un second delta, d2, entre la version actuelle du modèle dans le dépôt, et la version que le développeur a utilisée comme point de départ de son travail (ces deux versions pouvant être différentes si un autre développeur a envoyé ses modifications sur le même modèle entre-temps)

Les deux modèles delta, d1 et d2, sont comparés pour détecter les conflits (multiplicités et sens des associations, en majorité liés aux liens inter-modèles). ModelBus peut résoudre automatiquement les conflits (en donnant la priorité au dernier changement) ou laisser le développeur décider de manière interactive ce qu'il faut faire avec chaque conflit. De plus, ModelBus offre une API qui peut être utilisée pour implémenter une stratégie personnalisée de résolution de conflits.

ModelBus fait une distinction entre le gestionnaire d'espace de travail, qui communique avec le dépôt, et l'adaptateur d'outil, qui fait le lien entre l'outil de modélisation et le gestionnaire d'espace de travail. Les adaptateurs d'outils gèrent les identifiants, en conservant une correspondance entre les identifiants et des emplacements mémoire, et en offrant aux outils de modélisation des fonctions de chargement et de sérialisation de modèles, afin que la gestion des identifiants soit transparente. L'adaptateur d'outil peut aussi implémenter un "chargement de modèle pouvant passer à l'échelle" en retardant le chargement d'un modèle en mémoire jusqu'à ce qu'un lien inter-modèle menant à ce modèle ait été suivi. Ceci est seulement possible quand l'éditeur offre des crochets d'interception de liens de navigation vers des modèles non chargés en mémoire, ce qui est le cas avec EMF.

2.2.4.2 Gestion de versions

La gestion de versions, aussi appelée gestion de configuration logicielle ou encore contrôle de révision, est la gestion des mises à jour de l'information stockée sur un support informatique dans des fichiers (source code, documents, etc) [Wiki].

En ingénierie logicielle, la gestion de versions est utilisée pour traiter certains problèmes récurrents, qui, bien qu'étant traditionnellement relatifs au code source, sont pratiquement identiques pour les modèles :

– Comment Alice, en tant que nouvelle employée de la compagnie, peut inspecter l'historique du projet auquel elle va contribuer, afin d'étudier les changements précédents et les raisons pour lesquelles ils ont été introduits ?

– Si Alice découvre un bug dans le code source, comment peut-elle identifier quand (et par qui) il a été introduit ?

– Comment Alice peut-elle obtenir et exécuter les précédentes versions du logiciel en cours de développement, afin de caractériser plus précisément le bug ?

– Si Bob introduit, par erreur, une série de changements qui rendent le logiciel en cours de développement non fonctionnel, comment peut-il retourner à une version précédente, fonctionnelle ?

– Quand Alice développe des fonctionnalités expérimentales, comment peut-elle éviter d'interférer avec les améliorations que Bob est en train d'ajouter au code source ?
– Comment Alice peut-elle intégrer ses contributions si le développement expérimental est concluant ?
– Comment l'entreprise peut-elle suivre les responsabilités des changements aux documents dans un effort collaboratif, pour des besoins de communication et de traçabilité ?
– Comment Alice peut-elle rester au courant des derniers changements continuellement introduits par Bob dans le code source ?

Les systèmes de gestion de versions existants sont basés sur deux modèles principaux : le modèle centralisé et le modèle distribué.

2.2.4.2.1 Le modèle centralisé Les plus anciens systèmes de gestion de version étaient en majorité centralisés, avec un serveur utilisé comme dépôt central (la copie officielle du code source), chaque développeur travaillant sur une copie locale. CVS, Subversion et Perforce (commercial) sont des exemples notables de système de gestion de versions centralisé.

Dans un système de gestion de version centralisé, un projet débute comme un gabarit, configuré sur le serveur. Un administrateur donne aux développeurs la permission de faire des commits (autrement dit, le droit de contribuer au code source) sur le dépôt central. Chaque développeur télécharge (check-out) le code sur sa machine locale, fait les modifications nécessaires, puis envoie ses modifications locales au serveur (commit).

Un développeur peur verrouiller un fichier sur le serveur (empêchant effectivement les autres développeurs de le modifier, soit un verrouillage pessimiste), le modifier, puis envoyer ses modifications vers le serveur (commit). Ceci prévient effectivement les conflits.

Si le développeur oublie de poser un verrou, un autre développeur peut avoir modifié le fichier et fait un commit entre temps. Dans ce cas, le serveur essaye de fusionner (voir la section 2.2.3.3) les deux modifications, et notifie l'auteur du dernier commit en cas d'échec, afin qu'une résolution manuelle puisse avoir lieu avant le commit.

2.2.4.2.2 Le modèle distribué Au début des années 2000, les systèmes de gestion de versions distribués (DCVS : Distributed Version Control Systems) ont commencé à apparaître, notamment Monotone et Arch. Ces nouveaux systèmes ne s'appuient pas sur un serveur central ; mais plutôt, chaque développeur a une copie complète du projet ainsi que son historique. Chaque copie du projet a un statut équivalent à celui des autres [5], et les développeurs collaborent en échangeant des "changesets" (unités logiques de modifications). Des exemples notables de systèmes de gestion de versions distribués sont Git, Mercurial, Bazaar, et Darcs.

Dans un système de gestion de version distribué comme Git, un projet commence comme un prototype initial, dans le dépôt privé d'un développeur particulier. Ce développeur met à disposition une URL de dépôt (protocole HTTP ou GIT), afin que d'autres développeurs puissent copier (cloner) le travail de leur collègue. Lorsqu'un autre développeur clone le dépôt,

5. Du moins, techniquement. En effet, les développeurs peuvent se mettrent d'accord pour considérer une copie comme la copie principale, et l'utiliser pour les diffusions publiques du code source.

le contenu du dépôt, ainsi que toute son historique (autrement dit, les versions précédentes) sont rapatriés. Quand un développeur fait des changements, il fait un commit sur son dépôt local, ce qui ne nécessite pas d'accès réseau. Il peut alors envoyer un "pull request" (une demande d'intégration de ses modifications) à n'importe quel autre développeur, qui peut alors récupérer les changements.

2.2.4.2.3 Comparaison des modèles centralisés et distribués De manière générale, le modèle centralisé présente un avantage pour les larges fichiers binaires. Vu que les différences entre des versions de ces fichiers ne peuvent pas être simplement calculées comme avec les fichiers au format textuel, l'espace de stockage utilisé peut grandir très vite. Le modèle centralisé, parce qu'il stocke l'historique du projet à un seul endroit (sur le serveur), est plus efficace, en terme d'utilisation d'espace disque. De plus, la nature centralisée de ces systèmes permet d'assigner plus facilement des responsabilités aux membres de l'équipe fichier par fichier, et de pouvoir automatiquement vérifier ces politiques d'accès. Un commit peut être refusé par exemple, parce que l'auteur a modifié des fichiers qu'il n'a pas le droit de manipuler.

Les outils distribués souffrent d'une plus grande utilisation d'espace disque, à cause de leur nature distribuée, et n'incluent généralement pas de contrôle d'accès natif. Cependant, les systèmes distribués, parce qu'ils ne nécessitent pas de connexion réseau, rendent beaucoup d'opérations, en particulier celles qui manipulent l'historique, très rapides. Plus important, ils encouragent un style différent de développement. Les commits dans ce style de développement sont "bon marché" vu qu'aucun accès réseau n'est nécessaire et sont moins angoissants pour le développeur puisqu'ils sont locaux [6]. Ceci produit un historique de plus fine granularité [7], ce qui rend les bugs plus faciles à trouver ou analyser. La création de branches, qui est une opération onéreuse de copie dans un système centralisé, devient une opération locale bon marché dans un système distribué, ce qui encourage l'expérimentation. De plus, la flexibilité des systèmes distribués permet toute sorte d'organisation logique hiérarchique, où la centralisation peut avoir lieu à plusieurs niveaux, par opposition à l'approche "un seul dépôt pour les gouverner tous" des systèmes centralisés.

2.2.4.2.4 Modèles et gestion de version Une des problématiques du stockage des modèles dans un système de gestion de version est le fait que les modèles peuvent être stockés dans des fichiers binaires. La fusion de modification de fichiers au format binaire est souvent très difficile (voire impossible). La réutilisation de systèmes existants de contrôle de version demande donc le développement d'algorithmes innovants de calcul de différences et de fusion pour les modèles.

Un format populaire pour le stockage d'artéfacts de modélisation est le XMI, qui est un format basé sur XML. Un des algorithmes notables de détection de changements pour XML est X-Diff [Wang 03]. Contrairement aux précédents algorithmes de détection de changements pour XML, X-Diff est seulement sensible aux relations nœud parent - nœud enfant, et pas à l'ordre des nœuds adjacents. Les auteurs défendent le fait que cette stratégie est plus appropriée pour les applications de base de données. Pour les modèles, cela signifie que X-Diff

6. Un développeur peut réorganiser ses commits avant de les publier.
7. Résumé généralement par la devise "commit early, commit often".

conclura correctement qu'un modèle n'a pas changé, si des attributs ont juste été réordonnés dans une classe.

Le calcul de delta dans ModelBus [Srip 08] s'appuie sur l'association d'un identifiant unique (un UUID) à chaque élément de modèle. Ceci permet une détection très précise de changement, au prix d'exiger la modification des routines de chargement et de sérialisation de modèle, afin de préserver les identifiants dans les fichiers XMI entre des manipulations par des outils de modélisation.

Dans [Mehr 05], Mehra et al. décrivent l'implémentation d'un plugin de calcul de différences et de fusion dans l'outil Punamu META-Case. L'approche utilise des identifiants uniques pour les éléments de modèles comme dans ModelBus, mais travaille sur des modèles Java des diagrammes. Par exemple, pour détecter les différences entre deux diagrammes, ils sont chargés en mémoire comme des graphes d'objets Java. Ces graphes sont comparés pour identifier les différences, qui sont traduites en des commandes d'édition Punamu (l'éditeur). Ces commandes représentent l'ensemble des changements qui doivent être appliqués à un diagramme pour obtenir l'autre. Ceci permet à Punamu de présenter un riche feedback visuel des changements, puisqu'il travaille avec la représentation propre à l'outil de ses actions d'édition, au lieu du format de stockage XMI.

Enfin, la collaboration sur les modèles sans les verrous demande une approche efficace de versionnement de graphe. Une des approches existantes, basée sur la détection de conflits, est présentée dans [Taen 10].

2.3 Les processus logiciels

Depuis le travail fondateur [Oste 87] de L. Osterweil, le domaine des processus logiciels a connu des progrès significatifs. Dans cette section, nous nous intéressons particulièrement aux approches de modélisation et de support de processus.

2.3.1 Objectifs de la modélisation des processus logiciels

La modélisation des processus logiciels peut être reliée à la "crise du logiciel" [Buxt 70], qui fut l'un des tout premiers constats des difficultés à créer des systèmes logiciels corrects, en respectant les temps et les délais [Broo 87]. Une des solutions proposées pour répondre à ce constat est la formalisation des processus logiciels. Dans [Kont 98], les buts les plus fréquents de la formalisation des processus logiciels sont identifiés comme suit :

- Support de la certification de processus
- Support de la compréhension et de la communication
- Support de l'harmonisation et de la standardisation
- Support de la planification et du contrôle de projet
- Facilitation de la mise en œuvre de projet
- Support de la surveillance de processus
- Facilitation de la réutilisation de processus

– Facilitation de la simulation de processus et de l'analyse
– Support de la gestion de processus et de l'amélioration

Une séparation conceptuelle entre la gestion de processus et la gestion de projet a été introduite dans [Kont 98]. Les auteurs notent que les processus sont plus abstraits que les projets. Dans la terminologie objet, on pourrait dire que les processus sont des classes, et que les projets en sont les instances. L'ingénierie des processus modifie les processus, qui sont utilisés comme des gabarits pour les plans de projet. Ces plans guident la réalité, dont le feedback est utilisé pour contrôler le plan de projet et améliorer l'ingénierie des processus. Les plans de projet sont définis en utilisant le cahier des charges, les objectifs, les contraintes, les calendriers, et l'allocation de ressources. L'ingénierie des processus vise l'amélioration de la productivité, de la consistance, et de la prévisibilité.

La pertinence de la modélisation des processus logiciels pour l'amélioration de la qualité des produits logiciels a été établie par des académiques et des industriels [Rubi 93, Conr 94, Hart 00, Ashr 03]. Cette amélioration est liée au fait que la formalisation des processus logiciels clarifie les préoccupations du projet pour toutes les parties impliquées, et permet de fournir de l'assistance à l'exécution des tâches [Lonc 94].

Malgré les avantages précédents, les processus logiciels sont encore régulièrement en retard, et plus ils sont grands, plus ils sont livrés en retard et dépassent les budgets. Inspiré par les travaux de Peter Drucker en management, [Caus 10] a expliqué cet état de choses par le fait que l'on continue à planifier et gérer le développement logiciel, qui est un travail intellectuel, avec les méthodes de management scientifique dérivées des travaux de Winslow Taylor, bien que ces méthodes aient été conçues pour le travail manuel.

Peter Drucker a en effet suggéré que les "travailleurs du savoir", autrement dit, ceux qui font du travail intellectuel, se gèrent eux mêmes, puisqu'un manager externe a trop peu de visibilité sur ce qu'ils font. De plus, les buts de ces travailleurs (projet techniquement intéressant, environnement confortable, etc.) ne sont pas ceux des managers de projet (livrer des résultats dans les délais et le budget prévus) [Caus 10].

2.3.2 Modélisation des processus logiciels

Un modèle de processus est une représentation abstraite des activités de production de logiciel et de leurs relations[Bart 03]. Les modèles de processus peuvent être considérés comme des objets logiciels [Oste 87]. Ils ont donc des cycles de vie, et peuvent être spécifiés, conçus, implémentés, et déployés. Les étapes sont des unités de travail, qui peuvent être combinées en tâches et activités, et être associés à des rôles (qualifications nécessaires pour accomplir une tâche). Lors de l'exécution des tâches, des artéfacts (livrables) sont produits [Bart 03].

2.3.2.1 Méthodes de développement

Les toutes premières approches proposées pour conceptualiser les activités de développement logiciel sont connues sous le nom de "modèles de cycle de vie logiciel". Ces approches structurent le développement logiciel en un ensemble de phases rigidement définies, et souvent séquentielles. Les modèles de cycle de vie sont une réponse au besoin, dans les années

1960, de développer des systèmes logiciels de grande envergure à une époque où les grands conglomérats industriels étaient fréquents [Elli 04]. Divers modèles de cycle de vie de projet logiciel ont été proposés afin de décrire les activités réalisées dans un projet logiciel, ainsi que leur ordre. Certains exemples classiques sont le modèle en spirale [Boeh 88], le modèle de la cascade [Royc 89], le prototypage rapide [Gord 95], et le processus unifié [Jaco 99].

Diverses méthodes de développement, sous la bannière du "développement agile" [Abra 02, Boeh 02, Cock 01], ont ensuite été proposées avec pour but d'être moins lourdes que les précédentes approches, en se focalisant beaucoup plus sur des pratiques quotidiennes que des plans généraux rigides. Les méthodes agiles s'appuient sur des équipes auto-organisées et interdisciplinaires, et encouragent des réponses rapides au changement, des livraisons continues, ainsi qu'une planification adaptative. Les exemples majeurs sont Lean Software Development [Popp 03], Extreme Programming [Beck 99], Crystal Clear [Cock 05, Cock 01], SCRUM [Risi 00], et Kanban [Ande 04].

2.3.2.2 Standards

Divers efforts de standardisation ont été faits sur les processus logiciels. Parmi les approches notables, on peut citer SPEM et ISO/IEC 24744.

2.3.2.2.1 SPEM La norme SPEM [OMG 02, OMG 07] (Software & Systems Process Engineering Metamodel) définie par l'OMG fournit, entre autres, des briques réutilisables pour la modélisation des processus logiciels. Une première version de la norme [OMG 02] a été publiée en 2000. La version 2.0 de la norme, publiée en 2008, rend la norme compatible avec UML 2.0 [OMG 05], définit un nouveau schéma XML, ajoute des fonctionnalités d'échange de représentation graphique, rend la norme plus modulaire afin de pouvoir être utilisée partiellement, et introduit de nouvelles extensions (SPEM 2.0 Base plug-in). La norme SPEM se définit comme un méta-modèle et un cadre conceptuel pour modéliser, interchanger, documenter, gérer, et présenter des méthodes de développement et des processus. La norme est fondée sur les concepts clés de rôle, d'activité, et de produit pour décrire les processus de développement.

SPEM 2.0 propose des concepts d'une part pour constituer une librairie réutilisable de définitions, et d'autre part, pour représenter et gérer des processus de développement utilisant ces définitions. Les définitions, regroupées dans le paquetage "Method Content", décrivent pas à pas comment des objectifs de développement sont atteints, de manière indépendante du placement de ces objectifs dans un processus. Les processus (paquetages "Process Structure" et "Process With Methods") réutilisent ces définitions, et les relient en des séquences partiellement ordonnées adaptées à des projets spécifiques. La figure 2.8 illustre le lien entre les méthodes et les processus dans SPEM 2.0. Les concepts relatifs aux méthodes, et ceux relatifs au processus, peuvent être rattachés à des descriptions textuelles par les concepts du paquetage "Managed Content". Un élément typique du paquetage "Managed Content" est "Guidance", qui sert à fournir des indications de travail comme des templates, des check-lists, etc. La figure 2.7 est une vue simplifiée des concepts clés de SPEM 2.0, ainsi que leur répartition entre les processus, les méthodes, et les éléments du paquetage "Managed Content".

La norme SPEM s'organise autour des concepts centraux de `TaskUse` (unité de travail à réaliser), `RoleUse` (ensemble de qualifications nécessaires à un sous-ensemble de participants

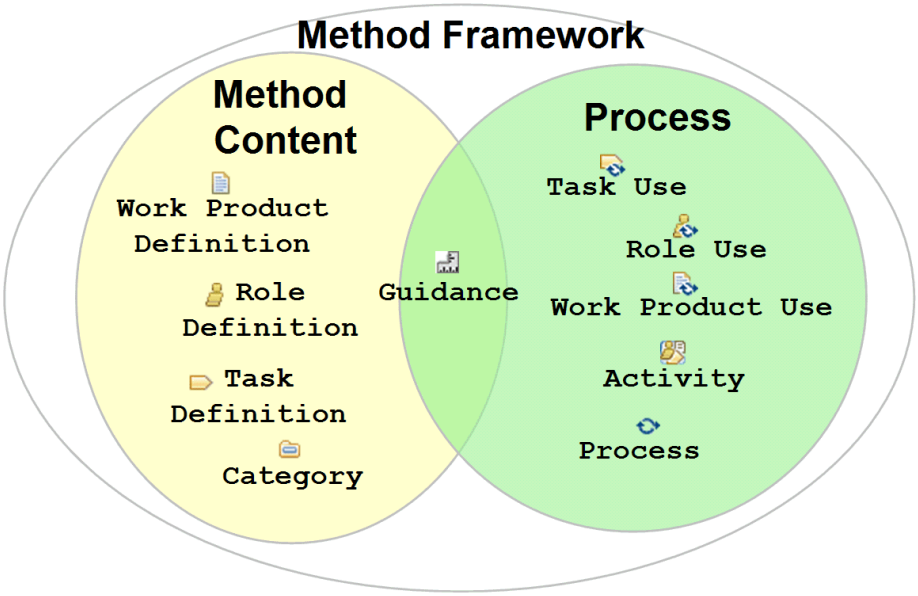

FIGURE 2.7 – Concepts clés du méta-modèle SPEM 2.0

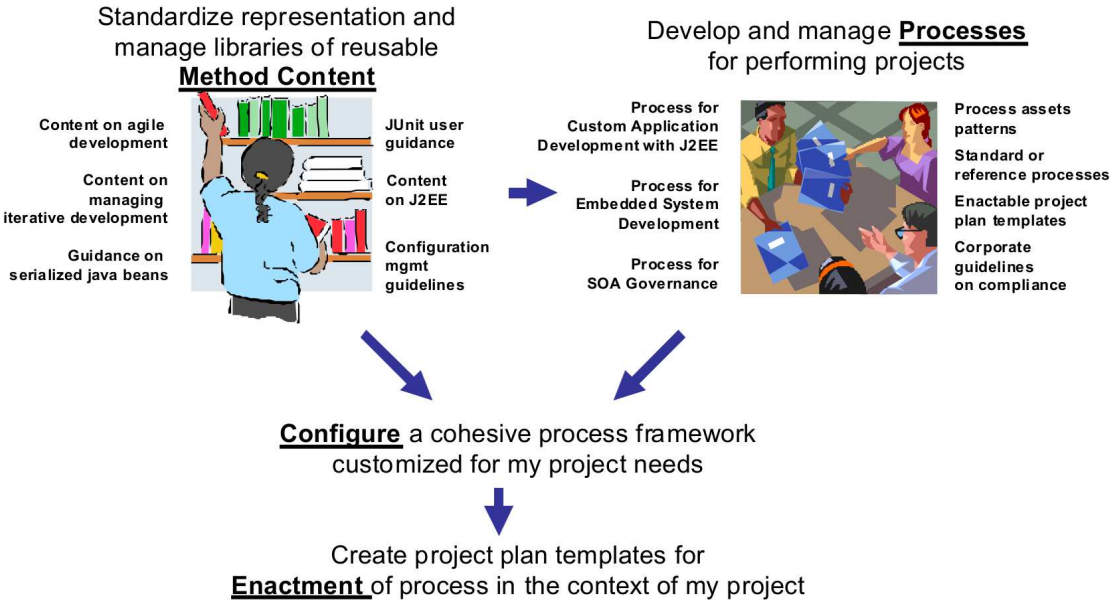

FIGURE 2.8 – Framework conceptuel du standard SPEM 2.0

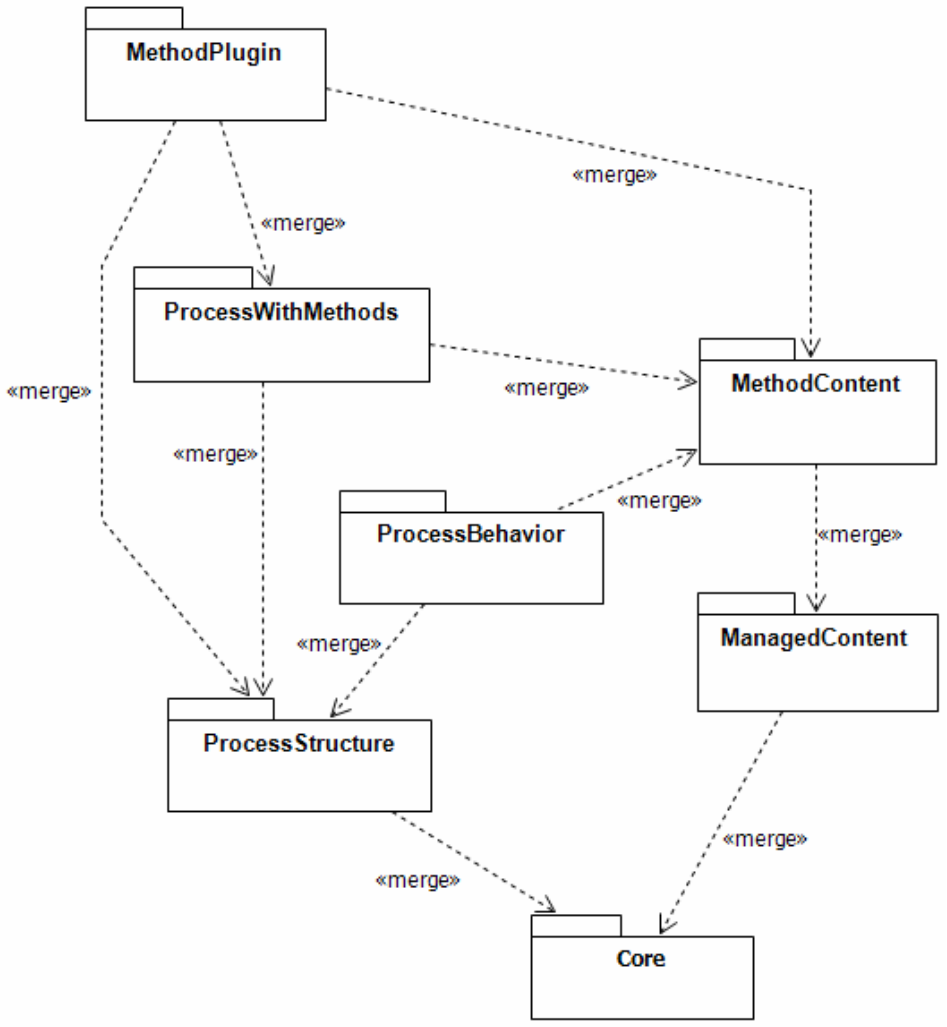

FIGURE 2.9 – Paquetages du méta-modèle SPEM 2.0

à un projet), et de `WorkProductUse` (produit à créer comme résultat d'une tâche). De plus, le concept d'`Activity` est utilisé comme conteneur, et comme espace de nommage servant à regrouper des `RoleUse`, `TaskUse`, `WorkProductUse`, ansi que les concepts servant à les mettre en relation.

SPEM 2.0 a été conçue pour être utilisée dans n'importe quel cycle de vie logiciel. Elle fournit un ensemble de mécanismes d'extension (Method Plug-in) permettant de personnaliser le contenu d'une définition (Method Content) sans modifier l'original, en définissant les différences (contributions, remplacements, suppressions). En particulier, il est possible de réutiliser des bouts de processus à divers endroits dans la définition d'un processus, avec la notion de "Process Pattern". SPEM n'inclut cependant pas de modèle comportemental natif, fournissant simplement des concepts pour attacher un modèle comportmental externe (Process Behavior). Enfin, un paquetage "Core" rassemble les concepts de base, communs à tous les paquetages (voir figure 2.9).

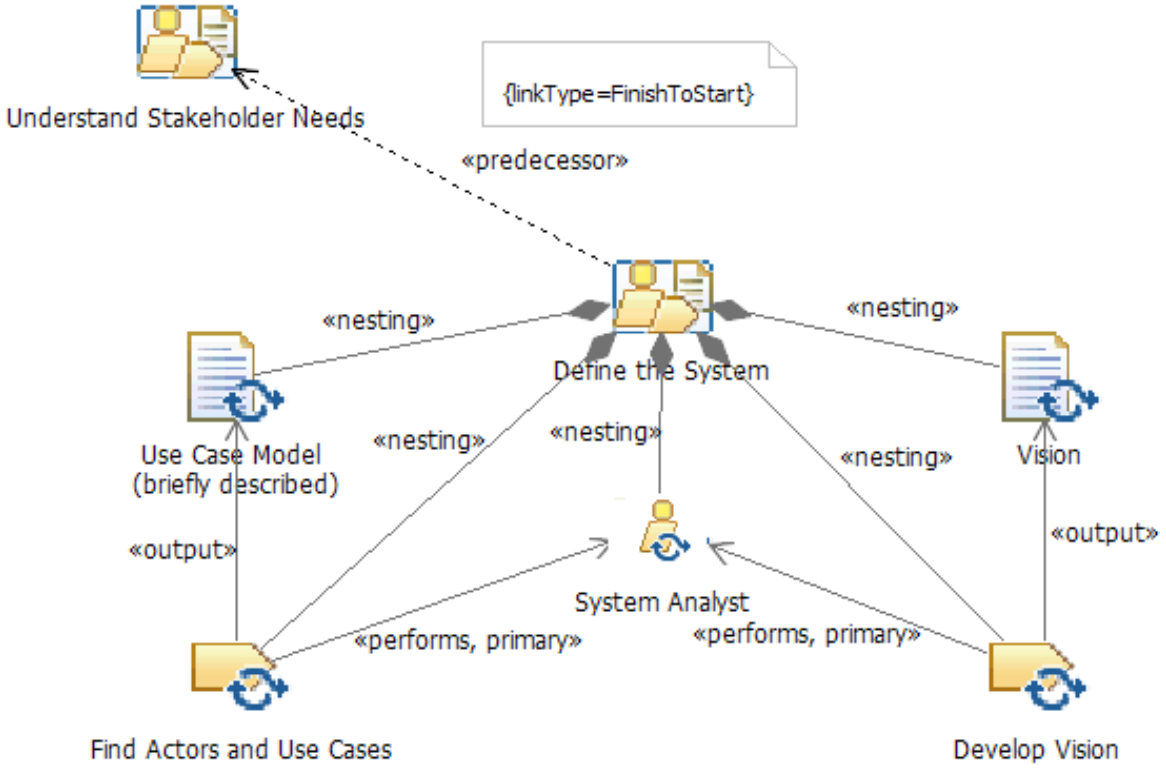

FIGURE 2.10 – Exemple de définition d'activité avec SPEM 2.0

La figure 2.10 montre un exemple de modèle SPEM 2.0, décrivant une activité (`Activity`) "Define the System", composée des tâches (`TaskUse`) "Find Actors and Use Cases" et "Develop Vision", toutes deux réalisées par le rôle (`RoleUse`) "System Analyst". Ces tâches on respectivement pour sorties les produits (`WorkProductUse`) "Use Case Model" et "Vision". L'activité "Define the System" est liée à une autre activité, "Understand Stakeholder needs", par un lien de précédence de type "FinishToStart". Ce lien spécifie (`WorkSequence`) que l'activité "Understand Stakeholder needs" doit être terminée avant que l'activité "Define the System" ne

commence.

Divers efforts d'extension de la norme SPEM on été réalisés. Par exemple, [Porr 06] a défini un formalisme basé sur les réseaux de Petri, visant une définition précise des modèles de processus IDM et le suivi de leur exécution. L'approche ne s'intéresse cependant, ni à l'allocation des ressources, ni à la définition de rôles, et se concentre sur les étapes de processus (décrits avec des liens, des modèles, des flux, et des ressources). L'extension xSPEM [Bend 07, Bend 09] permet, elle, de créer des modèles de processus SPEM exécutables en précisant les types et quantités des ressources affectées au projet, les contraintes de temps sur les activités, et les différents états du processus qu'on souhaite observer lors de son exécution. Le méta-modèle xSPEM est muni d'outils de vérification formelle et de simulation des processus. SPEM4MDE [Diaw 10] raffine les notions de `WorkProductUse` et d'`Activity`, afin de les adapter à l'ingénierie dirigée par les modèles. Ainsi, les modèles de processus SPEM4MDE peuvent prendre en compte, de manière native, les produits qui sont des modèles, et des activités qui sont des transformations de modèles potentiellement automatiques. Pour sa part, eSPEM [Elln 10] remplace le mécanisme natif d'ajout de modèle comportemental prévu dans SPEM par une alternative plus flexible, qui est utilisée pour greffer à SPEM un modèle comportemental basé sur les machines à état UML, afin de permettre une exécution automatisée.

2.3.2.2.2 ISO/IEC 24744 Un autre effort de standardisation notable est celui du standard ISO/IEC 24744 (Software Engineering – Metamodel for Development Methodologies) [Hend 08]. Ses divers domaines, "metamodel", "method", et "endeavour" (pour méta-modèle, méthode, et effort ; voir figure 2.11), correspondent approximativement aux niveaux méta-modèle, modèle, et instance de processus de SPEM. Le domaine "méta-modèle" est constitué de structures conceptuelles standardisées, le domaine "méthode" contient des éléments méthodologiques utilisés dans les projets réels (méthodes, outils, règles de style, etc.), et le domaine "effort" représente les processus tels qu'utilisés par des participants à un projet. Bien que le standard ISO/IEC 24744 soit orienté vers l'ingénierie logicielle, rien dans sa structure ne l'empêche d'être appliqué à l'ingénierie des systèmes ou à d'autres domaines [Hend 08].

ISO/IEC 24744 a la possibilité de représenter un concept défini dans le domaine méta-modèle directement dans le domaine effort (le niveau instance de SPEM), et remplace la relation "instance de" utilisée en UML par des relations de représentation. En effet, la norme est basée sur le concept de "powertype" [Odel 94]. Dans la figure 2.12 par exemple, TaskKind et Task sont du domaine "méta-modèle". Ensemble, ces deux classes forment un "powertype", qui peut être instancié en deux opérations. Un objet régulier (domaine effort, correspondant au niveau instance de SPEM) est obtenu [8] en instanciant la classe TaskKind, et une classe régulière WriteCode est obtenue en héritant de la classe Task. Les deux éléments résultant de cette double opération (délimités par l'éllipse sur la figure 2.12) forment un "clabject" [Gonz 07b]. Ce style particulier de modélisation rend la norme adaptée aux méthodes agiles et à la prise de décision progressive dans les projets logiciels [Gonz 07a].

Une vue d'ensemble des concepts d'ISO/IEC 24744 est disponible dans la figure 2.13. La norme s'appuie sur les concepts clés de "template" (donnant les grandes lignes d'une tâche, d'un produit, ou d'un producteur) et de "resource" (spécifiant des contraintes, des plans, des

8. L'objet a son nom souligné dans la figure 2.12.

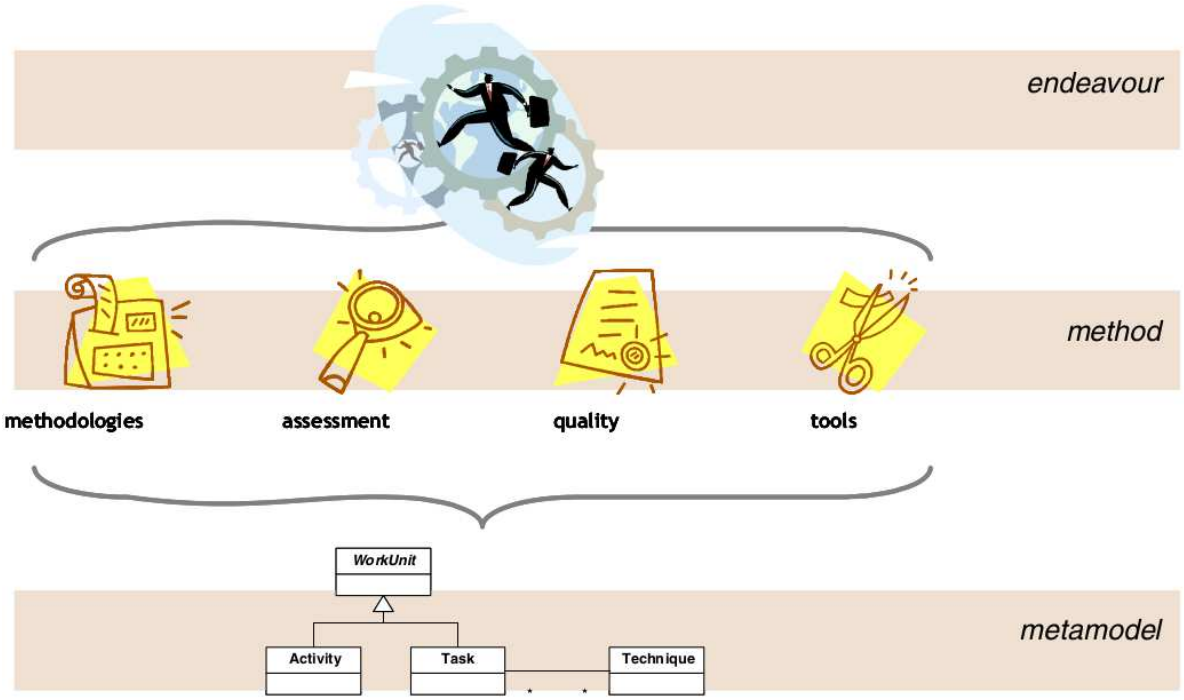

FIGURE 2.11 – Niveaux de modélisation de la norme ISO/IEC 24744

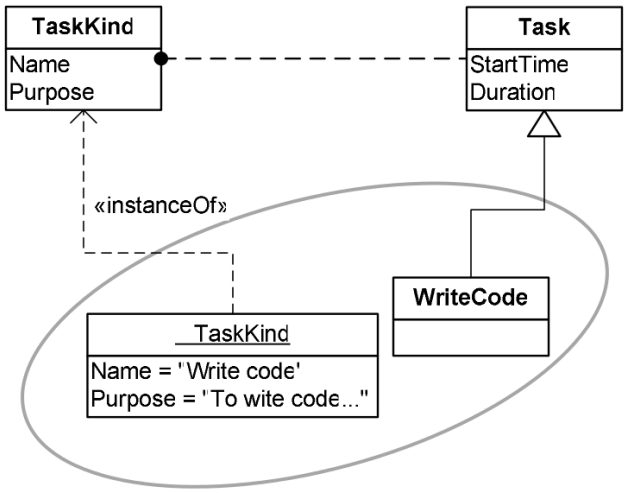

FIGURE 2.12 – Le concept de "power type" dans la norme ISO/IEC 24744

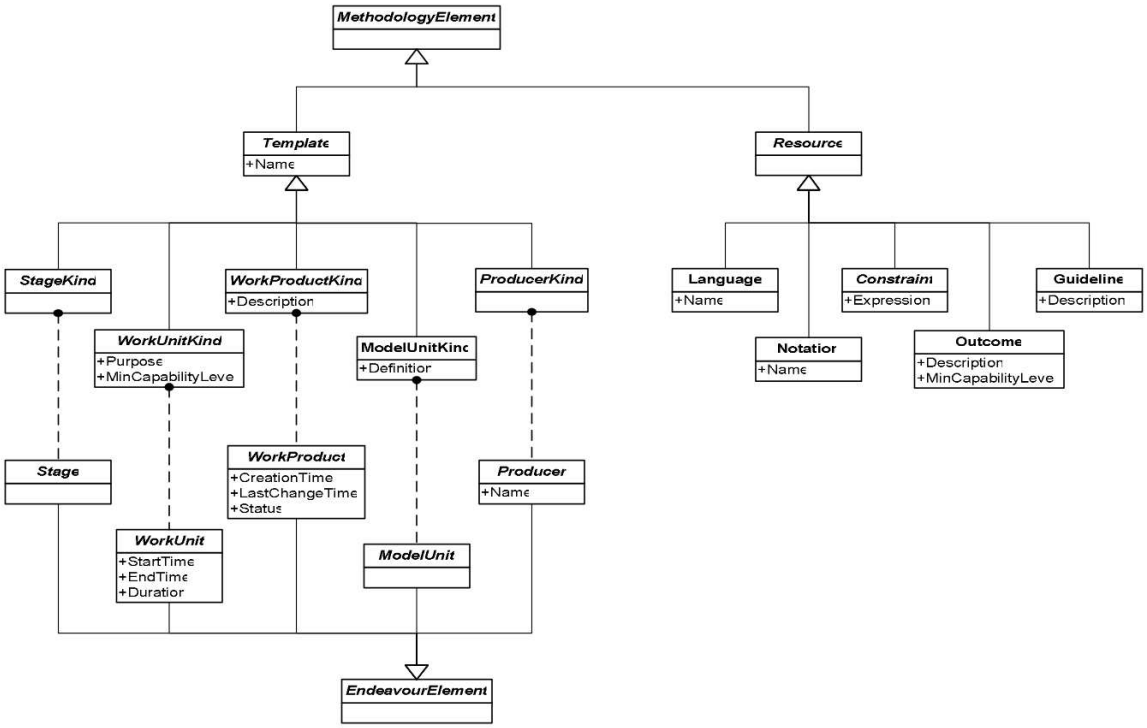

FIGURE 2.13 – Vue d'ensemble de la norme ISO/IEC 24744

résultats abstraits, etc.). ISO/IEC 24744 n'est pas destinée à la création de processus global comme SPEM, mais plutôt de morceaux de processus, formant une librairie dont les éléments sont à assembler en début de projet. Cette manière de procéder est dénommée "ingénierie des méthodes" par la norme, et distinguée de l'ingénierie des processus. En effet, l'ingénierie des méthodes s'intéresse aussi à la modélisation des outils utilisés et des personnes impliquées [Hend 06].

2.3.2.3 Flexibilité des processus logiciels

L'utilisation pratique des modèles de processus a montré le besoin de flexibilité dans le respect des contraintes spécifiées dans le modèle de processus [Such 87, Lonc 90, Robi 91]. Ainsi, divers efforts ont été entrepris dans ce sens, dont le cadre conceptuel d'instanciation flexible de processus dans [Kill 10]. Le but est de fournir une assistance pas à pas dans l'instanciation de modèles de processus génériques et complexes, tout en s'assurant que l'instanciation respecte certaines règles (syntaxiques et organisationnelles).

Une approche alternative est de détecter, analyser, et traiter les déviations d'exécution de processus par rapport au processus modélisé. Dans [Kabb 08] par exemple, une observation du déroulement d'une mise en œuvre de processus est capturé dans un modèle dit "modèle observé" qui contient un historique des états du développement et des actions utilisateur. Le modèle observé et le modèle d'origine sont alors traduits en une représentation utilisant la logique des prédicats. Les décisions de traitement de déviation sont alors prises à l'aide d'un

modèle de tolérance basé sur des prédicats logiques.

2.3.3 Support des processus logiciels

Une des applications importantes des travaux sur les processus logiciels est celle du support au processus, avec les PSEEs (Process Centered Software Engineering Evironment). Les PSEEs sont des environnements de développement basés sur un modèle de processus logiciel. Ils stockent des définitions de processus en termes d'actions à réaliser, d'artéfacts à produire et à transformer par ces actions, des personnes qui doivent réaliser ces actions, souvent identifiés par leurs rôles, et les contraintes d'exécution comme les précédences entre ces actions [Bart 03]. Ils coordonnent aussi les outils de développement utilisés en accord avec le processus, et assistent les développeurs dans l'exécution du processus. Un PSEE peut être considéré comme un cas particulier d'environnement de génie logiciel assisté ou ASEE (Assisted Software Engineering Environment), ce type d'environnement s'intéressant à la modélisation et à la gestion, autant de l'information que des processus logiciels [Lonc 91].

Une première génération de PSEEs a été fondée sur une vision purement normative du modèle de processus [Kais 90, Sutt 91, Junk 95, Band 94], et la génération suivante [Ben 94, Bolc 96, Allo 96, Kais 97, Dami 98, Cugo 99] a proposé diverses manières d'introduire de la flexibilité, en permettant aux développeurs de dévier des spécifications du processus logiciel [Gruh 02a].

La mise en œuvre des processus logiciels a très vite montré qu'il n'était pas toujours possible d'avoir des modèles de processus complets avant la mise en œuvre. Des efforts ont donc été faits pour permettre de modifier les modèles de processus lors de leur exécution. Par exemple, Kobialka [Kobi 04] a montré la pertinence de modèles de processus incomplets, qui peuvent être complétés lors de l'exécution, en modifiant directement l'instance de processus. La proposition est basée sur l'observation selon laquelle il n'est pas pratique de prétendre que toutes les activités dans les projets logiciels doivent être complètement définies. Ceci implique donc le besoin d'un support incrémental de processus, qui est réalisé dans [Kobi 04] en utilisant des déclencheurs et des contraintes. Dans [Elln 10], un concept de "TaskScheduler", associé à chaque activité, est ajouté au méta-modèle SPEM. Un "TaskScheduler" permet de créer dynamiquement des tâches non prévues dans le modèle de processus, tout en respectant les contraintes de précédence de l'activité à laquelle il est associé.

Dans [Grud 01], Grudin et al. proposent une méthodologie pour la conception d'environnements collaboratifs, qui est à mi-chemin entre l'approche descendante qui consiste à introduire toute l'information au début, et celle ascendante qui consiste à fournir simplement un cadre conceptuel vide. Dans [Wits 10], cette approche est utilisée pour concevoir un formalisme de "patron de tâche". Les patrons de tâche sont définis comme des abstractions de tâche qui fournissent de l'information et de l'expérience généralement pertinente pour la réalisation de la tâche. "Abstraction" ici désigne les caractéristiques communes d'une famille de tâches similaires, et qui ont les mêmes buts en des circonstances similaires. Les patrons de tâche sont instanciés, en associant des personnes physiques aux positions définies dans le patron de tâche. Les personnes concernées peuvent accepter ou refuser l'affectation. Un utilisateur peut améliorer un patron de tâche lors de son exécution. Les améliorations sont stockées localement, automatiquement réutilisées pour les prochaines instances du patron de tâche, et

peuvent être publiées pour que d'autres personnes les réutilisent.

Les PSEE sont traditionnellement considérés comme responsables de l'intégration des outils dans l'environnement de travail, ce qui fait de l'intégration une des préoccupations majeures du support des processus [Gruh 02a].

Dans [Kipe 87], des idées pionnières sur l'intégration d'outils de développement ont été discutées : la coopération (intégration de contrôle), la communication (intégration de données), et la similarité (intégration de présentation). Cette étude a aussi identifié les paramètres avec lesquels les capacités d'intégration d'un outil peuvent être mesurées : la granularité, la cohésion, et l'harmonie. La terminologie hiérarchique actuelle d'intégration de plate-forme, de présentation, de données, de contrôle, et de processus a été introduite dans [Wass 90], et une structuration par couches des outils logiciels en a été dérivée (dépôt partagé, gestion d'objets, fonctionnalité, interface utilisateur, présentation).

Barghouti [Barg 94] a décrit comment Provence, un PSEE, minimise son intrusivité, en surveillant les informations dont il a besoin au niveau du système de fichiers, au lieu de nécessiter une intégration de haut niveau. Cependant, ceci résout seulement le problème de l'adaptation des outils pour qu'ils soient contrôlés par un PSEE. En effet, Provence ne met pas l'information de processus à disposition des autres outils.

Dans [Ambr 97], Ambriola et al. ont conduit une étude extensive des PSEEs. Les fonctionnalités majeures des PSEEs sont listées, et diverses implémentations sont discutées. Cette étude touche à deux points pertinents pour notre travail : l'intégration de contrôle et l'intégration de données. Cependant, toutes les approches décrites (OIKOS, EPOS, et SPADE) considèrent ces formes d'intégration seulement quand l'information passe d'outils tiers vers le PSEE. En d'autres mots, les approches évaluées invoquent les autres outils ou accèdent aux données des autres outils. Cependant, elles ne permettent pas à des outils tiers d'accéder aux données du PSEE.

Traitant de l'état de l'art dans l'intégration d'outils en 2004, [Wick 04] analyse divers sujets liés à l'intégration d'outils, parmi lesquels l'intégration basée sur les processus. Les auteurs notent comment le manque de flexibilité et d'adaptabilité empêchent ces solutions d'être offertes sur le marché et utilisées. Dans une étude ultérieure [Wick 07], les auteurs proposent un agenda de recherche pour l'intégration d'outils, et notent que la gestion de configuration logicielle, la gestion de défauts logiciels, et la gestion de changement, sont bien mieux intégrés que la gestion de projet, la gestion des cahiers de charges, l'analyse, le design, et l'implémentation. Les auteurs suggèrent que ceci est dû à des décisions métier, par exemple, parce qu'une équipe n'a précédemment pas réussi à livrer des composants logiciels pour une version spécifique du système en cours de construction. De même, dans son plan de route pour la collaboration en ingénierie logicielle [Whit 07], Whitehead propose des infrastructures de collaboration qui supportent aussi bien l'intégration de données que de contrôle. Il défend aussi une meilleure intégration entre les logiciels de bureau et ceux basés sur le web, ces derniers étant de plus en plus utilisés en ingénierie logicielle.

Dans [Mati 12], sept PSEEs sont comparés (voir figure 2.14), et leurs fonctionnalités classées avec des besoins intrinsèques aux PSEEs (mise en œuvre, consistance, flexibilité, etc.), des critères dérivés des critiques émises par les auteurs à l'encontre des PSEEs (déviations,

Criteria Group	PSEE Name / Criterion Name	DOSDE	VRPML SE	CASDE	Transforms	SPACE	ADAMS	MD Integrated
PSEEs' Requirements	Enactment Support	3/3	3/3	2/3	3/3	3/3	3/3	3/3
	Software Team Distribution	1/2	2/2	2/2	0/2	2/2	2/2	0/2
	Consistency Management	0/3	2/3	2/3	0/3	3/3	3/3	0/3
	Process Flexibility	0/3	1/3	3/3	2/3	3/3	3/3	3/3
	Process Evolution	0/3	3/3	1/3	3/3	0/3	3/3	0/3
	Security	0/1	1/1	0/1	0/1	1/1	1/1	0/1
	Mobility	0/3	0/3	0/3	0/3	0/3	0/3	0/3
Critique Results	Process Deviations Support	12/30	6/30	18/30	21/30	2/30	3/30	12/30
	Human-Dimension Support	12/24	2/24	5/24	12/24	18/24	18/24	12/24
	New Technology Adoption	3/3	0/3	1/3	2/3	2/3	0/3	2/3
General Requirements	Coverage	2/6	0/6	1/6	3/6	5/6	6/6	4/6
	Change Management	2/4	1/4	1/4	3/4	1/4	4/4	2/4
	Traceability	0/2	0/2	0/2	1/2	2/2	2/2	0/2
	Interoperability	0/6	0/6	2/6	5/6	4/6	2/6	6/6

FIGURE 2.14 – Comparaison de PSEEs par Matinnejad et al. Chaque critère est évalué à l'aide d'un ensemble de fonctionnalités qui le définissent. Pour chaque fonctionnalité, le degré de support de la fonctionnalité par le PSEE est évalué sur une échelle à quatre niveaux (0/3 : le PSEE ne mentionne pas la fonctionnalité, 1/3 : le PSEE fournit des points d'extension pour incorporer la fonctionnalité, 2/3 : le PSEE supporte partiellement la fonctionnalité, 3/3 : le PSEE supporte totalement la fonctionnalité) ou sur une échelle à trois niveaux (0/2 : le PSEE ne supporte explicitement aucune partie de la fonctionnalité, 1/2 : le PSEE couvre partiellement la fonctionnalité, 2/2 : le PSEE couvre complètement la fonctionnalité). La note associée à un critère est obtenue en sommant les notes des fonctionnalités le composant.

dimension humaine, adoption de technologie nouvelle), et des besoins généraux. Parmi les besoins généraux, se trouve la capacité du PSEE à offrir des points d'extension pour l'intégration d'outils. Cependant, tous les projets bien notés sur ce critère (SPACE, Transforms, et "Model-Driven Integrated Approach") sont spécialisés dans le support des processus utilisant l'approche IDM. Ainsi, les capacités d'intégration de ces PSEEs sont des résultats naturels du contrôle qu'ils ont sur le genre d'outils utilisés dans ce style de développement (éditeurs de modèles, moteurs de transformation, etc.). En d'autres termes, la facilité relative avec laquelle ces PSEEs peuvent s'intégrer aux outils externes est une conséquence directe du fait que ces outils externes ont été conçus explicitement pour supporter des processus IDM. Par conséquent, ce style d'intégration ne s'applique pas à des outils génériques d'ingénierie logicielle.

2.4 Processus collaboratifs

Cette section discute des différentes approches de conceptualisation de processus collaboratifs, avec une emphase sur le support et l'amélioration de la collaboration. Le concept de collaboration peut être appliqué à tout le processus, ou à des activités particulières. Dans le premier cas, la collaboration est réduite à la coordination des efforts individuels des participants. Dans le second cas, la collaboration décrit comment un groupe de participants unissent leurs efforts pour accomplir une seule tâche, et s'applique plus à la collaboration ad hoc.

2.4.1 La collaboration en tant que coordination d'activités

Toute définition de processus qui utilise le concept de "rôle" (ou un concept similaire) peut être qualité de collaboratif, dans la mesure où les différents rôles fournissent un effort collectif sur un seul projet. Ainsi, les définitions de processus supportent généralement une forme de routage de travail entre participants, ce qui est l'essence de la collaboration telle que décrite dans cette sous-section. L'étude suivante est utilisée pour illustrer cette approche.

[Sari 91] a posé les fondations de la modélisation de processus collaboratifs en identifiant les besoins pour un processus collaboratif :

- Router le travail entre les différents participants selon le rôle qu'ils jouent dans le processus.
- Fournir du contexte pour le travail fait par un participant.
- Autoriser le jugement humain dans la détermination de la séquence d'actions système nécessaires pour effectuer une tâche, et décider quand la tâche est terminée.
- Supporter les processus mutables, autrement dit, le processus réel doit pouvoir dévier de sa description.
- Supporter le suivi de processus, afin de pouvoir interroger l'état des différentes tâches.
- Être suffisamment ouvert et flexible pour ne pas être limité à des applications et services système particuliers.

L'auteur propose un service orienté-objet de gestion de processus et de documents entre l'utilisateur et les services système. Ce service est basé sur le modèle de processus collaboratif conçu pour satisfaire aux exigences précédentes.

Le concept principal est celui de "job", une activité collaborative multi-personne. Un "job" décrit les dépendances temporelles entre tâches, qui sont des unités de travail qui peuvent avoir des relations de précédence entre elles. Chaque tâche est assignée à un rôle qui représente une personne (humain ou programme) qui réalise la tâche. Chaque tâche (et "job") possède des références aux documents et autres objets applicatifs qui forment l'espace de travail ou le contexte de réalisation de la tâche.

L'état du système est modifié au fur et à mesure que les utilisateurs demandent une tâche pour travailler et la libèrent quand ils ont terminé. Il faut noter que le système a la responsabilité d'invoquer l'application appropriée (éditeur) sur le document approprié afin que l'utilisateur puisse réaliser sa tâche.

2.4.2 La collaboration en tant que réflexion de groupe sur une activité

La collaboration peut avoir lieu à l'intérieur d'activités particulières, quand elles sont assignées à un groupe de personnes. Dans la présente sous-section, nous considérons les processus collaboratifs où l'on peut définir comment ces activités collaboratives sont réalisées.

Dans [Lonc 96], J. Longchamp introduit CPCE, un environnement collaboratif centré sur les processus. CPCE est conçu pour des processus collaboratifs basés sur les problèmes, autrement dit, des processus avec plusieurs participants réalisant un travail créatif qui progresse principalement via des décisions collectives, avec des interactions asynchrones par défaut, et prenant place dans un environnement collaboratif où le produit en cours de construction, l'historique du processus, et les justifications des décisions de conception sont tous disponibles. Cette approche est basée sur la notion de "résolution collective de problème", dont les autres types d'activité sont des spécialisations. Les concepts clés sont ceux de "problème", "opinion", "argument", "phase", et "rôle", avec un ensemble de relations entre elles, et des actions possibles pour chaque rôle.

NGPM (Next Generation Process Model), une adaptation collaborative du modèle de développement logiciel en spirale a été proposée par Boehm et al. [Boeh 94]. Dans le modèle original en spirale, chaque cycle commence par un risque, une évaluation des forces et faiblesses du dernier prototype (si disponible) et la définition des exigences pour le prochain prototype. Cette phase demande la collaboration de toutes les parties prenantes. Dans [Boeh 94], l'accent est mis sur cette phase, et un sous-processus collaboratif basé sur la Théorie W [Boeh 89] est défini pour s'assurer que la phase se termine avec succès. La Théorie W (win-win) demande l'identification des conditions gagnantes de chaque participant, autrement dit, quel aspect du dernier prototype et quelles exigences pour le prochain prototype sont souhaitables. Ces conditions sont alors comparées l'une à l'autre, et les situations perdant-perdant (risques) et gagnant-perdant (désaccords) sont résolues. Le but ultime de NGPM est de s'assurer que les contraintes appliquées au produit sont homogènes. NGPM a été appliqué avec succès au projet STARS du département américain de la défense (DoD).

Dans [Brag 07] est défini un processus collaboratif basé sur le concept de thinklet[Kolf 04] venant de l'ingénierie collaborative (CE : Collaborative Engineering), pour le cas d'une stratégie de développement multi-organisationnelle. Ceci démontre que les thinklets peuvent être combinés pour construire des processus complets. Même si le but principal des thinklets est

de produire des processus collaboratifs répétables, l'exemple est pertinent pour démontrer comment un processus peut être construit de bout en bout pour supporter des activités collaboratives. Les auteurs analysent des besoins de développement stratégique, et identifient les thinklets (de la librairie de thinklet du CE) qui sont appropriés. L'approche a aussi été appliquée pour obtenir du feedback innovant d'utilisateurs finaux sur des systèmes d'informations avancés basés sur le web [Brag 08].

Richardson et al. dans [Rich 10] se sont intéressés au problème du développement logiciel mondial, c'est-à-dire, les projets logiciels collaboratifs avec une équipe virtuelle répartie dans le monde entier, avec une approche processus. Ils ont identifié vingt-cinq conditions qui doivent être prises en compte dans la construction d'équipes mondiales virtuelles (équipes géographiquement distribuées avec des tâches interdépendantes) en utilisant des données de trois cas d'étude réalisés sur une période de neuf ans. Ces conditions ont été utilisées pour implémenter "Global Teaming", un domaine de processus logiciel similaire en structure à CMMI.

Global Teaming a deux objectif spécifiques qui sont réalisés avec un ensemble de pratiques :

- Définition de la gestion globale de projet. Ceci comprend des tâches générales de management de projet, ainsi que le management de tâches (structure organisationnelle et allocation de tâches entre les différents sites), de connaissances, et de compétences.
- Définition de la gestion des sites. Les sous-buts incluent les procédures opérationnelles (communication, réunions, et résolution de conflit), la collaboration entre les sites (frontières de possession de produits, interfaces d'échange d'entrées, sorties, et produits, listes d'engagements et plans de travail liés aux produits et interfaces entre équipes).

Dans le contexte du "Global Studio Project" (GSP), SIEMENS a réalisé, ces dernières années, des expériences sur le développement industriel de logiciel à l'échelle mondiale. Le projet [Avri 07, Avri 10] consiste à simuler, avec des étudiants répartis dans le monde, le développement à l'échelle mondiale, afin d'identifier les pratiques courantes de collaboration entre différents sites distribués. Dans [Avri 10], Avritzer et al., commentant le GSP, notent que "les projets de développement logiciel multi-sites sont optimisés pour des patrons de communication entre les ingénieurs informatiques travaillant dans les divers sites physiques". Ils décrivent ensuite deux processus courants utilisés dans le projet GSP :

- Le modèle de l'établi étendu. Dans ce processus, les leaders du projet (manager en chef, responsable de l'ingénierie de besoins, architecte en chef) sont tous sur un même site central, et distribuent le travail aux groupes d'exécution qui gravitent autour d'eux. Ce modèle est appliqué quand l'expertise technique est rare et disponible seulement sur un seul site. Le modèle demande beaucoup de travail préparatoire (besoins et architecture) par l'équipe centrale avant que les autres ne puissent contribuer.
- Le modèle de "système de systèmes". Ici, l'équipe globale est constituée de multiples équipes chacune étant spécialisée pour un domaine, regroupée géographiquement, et ayant des experts à sa disposition. Contrairement au modèle précédent, la communication entre les équipes n'est pas coordonnée par une équipe centrale (qui existe toujours).

Ceci passe mieux à l'échelle, vu que l'équipe centrale n'est pas envahie par les tâches de coordination.

Pour chacune de ces approches, Avritzer et al. donnent des exemples de modèles de processus (décrits avec une notation informelle de boites et de flèches).

2.5 Conclusion

La présente revue de la littérature révèle une grande diversité de conceptions de la collaboration. Vu que des validations empiriques ne sont pas toujours disponibles pour les différentes approches, cette section est dédiée à la classification (au lieu d'une évaluation en bonne et due forme) des contributions étudiées selon les critères définis ci-après.

2.5.1 Critères de classification

Le premier critère de classification est la manière dont la collaboration est envisagée : comme une opportunité, ou comme un mal nécessaire. La collaboration est une opportunité quand elle est utilisée pour accomplir ce qui n'aurait pas pu être fait en travaillant de manière individuelle (ou alors, de manière moins efficace). Par opposition, la collaboration est un mal nécessaire quand elle est imposée uniquement par la quantité de travail à faire, et que l'objectif principal et de la gérer, la réduire, ou même l'éliminer.

Les formalismes de modélisation de la collaboration peuvent aussi différer par leur nature prescriptive (modélisation de *ce qui doit être fait*) ou descriptive (modélisation de *ce qui est en train d'être fait*).

Une autre distinction peut être faite sur la base des concepts qui structurent la collaboration. Les candidats sont les rôles, les artéfacts, les activités, les actions, et la communication.

Un dernier ensemble de critères clarifie les buts poursuivis et les contraintes imposées par les différentes approches. Les buts peuvent en plus permettre de séparer les approches orientées management (estimations de temps, détection de problèmes) et celles qui s'intéressent plus aux développeurs (vigilance et facilité de contact [Swam 08], informations de contexte).

2.5.2 Résultats de la classification

En dehors du domaine de l'intelligence collective [Weis 05], la collaboration a tendance à être considérée comme un problème à résoudre [9]. Ceci peut être lié à la difficulté inhérente à la construction d'une compréhension partagée, qui est nécessaire pour la collaboration.

Parmi les formalismes étudiés, le groupe suivant relève de l'approche prescriptive :

- ThinkLets [Kolf 04]
- SPEM [OMG 07]

9. [Swam 08] étant une exception notable à cette approche courante.

– ISO/IEC 24744 [Hend 08]
– BPEL [OASI]
– Théorie de coordination [Malo 03]
– Environnement collaboratif centré sur les processus [Lonc 96]
– Next Generation Process Model [Boeh 94]
– Global Teaming [Rich 10]

Par contre, les études suivantes adoptent une approche descriptive :

– Métamodèle de description de processus collaboratifs [Hawr 05]
– La perspective langage-action [Scho 01]
– Processus pour le développement logiciel multi-site [Avri 10]

Le tableau 2.1 montre les concepts centraux utilisés pour modéliser la collaboration dans chacun des formalismes étudiés. On peut constater que les approches varient énormément dans l'utilisation de ces concepts : des concepts considérés comme fondamentaux dans certaines approches sont complètement absents d'autres approches. Ceci suggère que la collaboration est une question complexe, multiforme, rendant potentiellement illusoire et réductrice la quête d'un formalisme unique, traitant tous ses aspects. Il semble donc plus pertinent de caractériser la collaboration dans un contexte particulier, et de se demander quel ensemble de concepts est nécessaire pour en rendre compte. Nous procédons ainsi dans le reste de ce travail, notamment dans le chapitre 3.

	Activités	Actions	Artéfacts	Rôles	Ressources	Dépendance	Rythme	Influence	Interaction
UML/SPEM	✓		✓	✓		✓			
ThinkLets		✓		✓					✓
ISO/IEC 24744	✓		✓	✓	✓				
GOMS	✓			✓	✓	✓			
Perspective langage-action		✓		✓					✓
Théorie de la coordination	✓				✓	✓			
Modèles temporels	✓						✓		
Réseaux sociaux				✓				✓	✓
Métamodèle de Hawryszkiewycz	✓	✓	✓	✓					✓

TABLE 2.1 – Concepts utilisés pour modéliser la collaboration

Le tableau 2.2 montre le but principal d'un ensemble d'outils ou d'études sur la collaboration. *Vigilance* fait référence au fait d'être informé de ce que font les autres participants du projet. *Contact* désigne la capacité à trouver et à échanger avec les personnes avec lesquelles l'on a besoin de collaborer. *Productivité* représente l'efficacité gagnée dans des situations pratiques de collaboration. *Gestion de projet* matérialise la disponibilité de l'information (états, problèmes possibles, etc.) sur ce que font les participants dans un effort collectif.

On peut constater qu'IBM-JAZZ est une des rares approches à offrir une gamme relativement bien fournie de fonctionnalités de vigilance. Cependant, il faut noter que cette capacité d'IBM-JAZZ à offrir de l'information de contexte (celle relative aux processus y compris) est en grande partie due au fait que les divers composants de la plateforme ont été créés pour communiquer, et non pas à cause de leur capacité individuelle à s'intégrer à des systèmes tiers. En effet, Eclipse, Rational Team Concert, Rational Quality Manager, etc., sont tous des produits IBM.

De plus, le support pour le contact (la capacité à retrouver la personne la plus appropriée pour répondre à une question qu'on se pose) est assez faible dans les outils existants. Une explication possible est que cette information se trouve typiquement dans le modèle de processus (affectation de ressources) dont il n'est souvent pas simple d'extraire de l'information.

	Vigilance	Contact	Productivité	Gestion de projet
UML/SPEM	aucun	aucun	moyen	fort
ThinkLets	faible	faible	fort	moyen
IBM-JAZZ	fort	moyen	moyen	fort
ModelBus	moyen	aucun	moyen	faible
CoDesign	moyen	aucun	moyen	faible
Syde	moyen	aucun	moyen	faible

TABLE 2.2 – Buts et contraintes de la collaboration

2.5.3 Manques identifiés dans la littérature

Un important travail de recherche qui reste à faire nous semble consister à identifier un petit ensemble de concepts appropriés pour décrire la collaboration ad hoc en génie logiciel, dans la perspective du support outillé. En effet, les approches existantes s'appuient chacune sur des concepts différents (séquence d'activités, actions, rôles, communication, etc.) pour modéliser la collaboration, mais aucune dédié à la collaboration ad hoc ne prend en compte la contrainte du support outillé. En effet, le support outillé nécessite de reconnaître le fait que les outils de génie logiciel soient utilisés en concertation, ce qui implique que des rapprochements doivent pouvoir être faits entre les concepts qui sous-tendent les modèles manipulés par ces différents outils. Cette contrainte nous semble importante pour l'adoption des outils de support à la collaboration, vu qu'elle diminue par définition la surcharge cognitive introduite par un nouvel outil.

Des travaux menés sur la contribution des PSEEs à l'intégration des outils de développement collaboratif, on peut noter la rareté de la mise à disposition des informations de processus. En effet, les PSEEs ont tendance à être conçus comme "responsables de l'intégration", et non comme "participants à l'intégration". Ainsi, pour le support de la collaboration, il reste compliqué d'exploiter les informations de processus, ce qui isole les outils de support de processus dans l'environnement de travail.

Un autre champ de recherche est l'influence du contact facile et de la disponibilité des outils de vigilance [Swam 08] sur la collaboration. Bien que des études et implémentations abondent, la contribution de ces facteurs à l'émergence de comportements collaboratifs n'est toujours pas claire.

Une dernière direction d'exploration est comment les outils collaboratifs peuvent être rendus "invisibles", afin de réduire la surcharge cognitive des utilisateurs. La vision derrière le projet IBM-Jazz (*collaboration fluide* [Booc 03]) est un pas dans cette direction. Ceci est crucial pour une large adoption de la collaboration assistée par ordinateur.

Dans le présent travail de thèse, nous nous intéressons, en premier lieu aux concepts manquants pour décrire un processus collaboratif avec la norme SPEM (voir chapitres 4 et 5). En second lieu, nous traitons de l'exploitation de l'information de processus dans le support au développement logiciel collaboratif (voir chapitres 6 et 7). Afin de justifier ces deux principales directions, nous avons réalisé une étude du support outillé à la collaboration en génie logiciel, qui a permis de définir un modèle conceptuel du support au développement collaboratif (voir chapitre 3).

Chapitre 3

Conceptualisation du support au développement logiciel collaboratif

3.1 Introduction

La problématique centrale de cette étude est le support au travail collaboratif grâce à l'exploitation des processus de développement. Afin de déterminer la stratégie adéquate pour faciliter l'exploitation des informations de processus, il est pertinent de déterminer comment sont exploitées les autres informations sur le développement collaboratif. Ceci revient à étudier les entités qui stockent, gèrent, et exploitent cette information : les outils de développement existants. Le but d'une telle étude est d'expliciter les stratégies de conception [1] qui rendent possible une telle exploitation de l'information, afin de les appliquer au cas particulier des informations de processus.

Les informations de processus peuvent être exploitées en créant de nouveaux outils les utilisant, ou en facilitant leur utilisation par les outils existants. L'étude menée dans ce chapitre servira à montrer la pertinence de la seconde stratégie, par opposition à la première qui est l'option traditionnellement utilisée.

Ce chapitre commence par décrire le fonctionnement général des outils courants de support à la collaboration, en s'appuyant sur l'exemple d'un outil de gestion de versions, identifiant au fur et à mesure ses briques constitutives. Ces différentes briques sont ensuite généralisées pour former un modèle conceptuel générique du développement collaboratif, qui est ensuite appliqué à des exemples relatifs à l'assurance qualité et aux environnements de travail orientés tâches. Le modèle conceptuel est enfin appliqué aux processus pour dégager les grandes lignes de l'approche proposée par la thèse et une comparaison est faite avec l'approche sur laquelle se basent les PSEEs (Process Centered Software Engineering Environment).

1. Dans le présent chapitre, le terme "conception" est utilisé pour rendre le terme anglais "design", qui désigne l'ensemble des compromis réalisés dans la spécification et la planification de la réalisation d'un produit logiciel.

3.2 Analyse d'un système existant de support à la collaboration : la gestion de versions

Les outils de gestion de défauts logiciels, de contrôle de versions, de communication (chat, mailing-list, etc.), d'intégration continue, et les bases de connaissance sont des exemples d'outils de collaboration courants en ingénierie logicielle[Lanu 10]. Un des outils les plus utilisés de support de développement logiciel collaboratif est le logiciel de gestion de versions. Les différentes implémentations d'un tel logiciel forment deux grandes classes : les systèmes dit centralisés, et ceux dit décentralisés ou distribués.

Une installation d'un système de gestion de versions est nommée dépôt ("repository"). Les systèmes de type **centralisé** limitent les échanges d'information entre dépôts à ceux mettant en jeu un dépôt spécial (appelé "serveur") et tout autre dépôt (appelé "client"). Les systèmes **distribués**, plus récents, permettent l'échange d'information entre deux installations arbitraires, chaque installation étant traitée pareillement du point de vue du logiciel. Une équipe de développeurs peut décider, dans un système distribué, de traiter un ou plusieurs dépôts comme "centraux", afin d'imposer un flux particulier d'information, et donc une certaine organisation du travail.

Dans cette étude, nous nous intéresserons à un système distribué, Git[2]. Ce choix est uniquement motivé par le fait que, de par la richesse de son écosystème, Git permet une illustration parlante de la plupart des concepts abordés.

3.2.1 Préoccupation de développement collaboratif

La gestion de versions, en génie logiciel, fait partie de la gestion de configuration, dont l'objectif est de gérer la description technique d'un système, et l'évolution de cette description dans le temps.

En génie logiciel, un logiciel de gestion de versions gère en grande partie du code source, mais aussi des modèles, des diagrammes, des fichiers de configuration, etc. Pour simplifier la description à suivre, la plupart des exemples porteront sur la gestion du code source, qui est de loin le cas le plus fréquent. Ceci n'enlève rien à la généralité de la discussion, les problématiques de gestions de version abordées s'appliquant de manière quasiment identique à tous ces types d'artéfact.

La préoccupation à laquelle répondent les systèmes de gestion de versions est celle du suivi de l'évolution du code source d'un système logiciel en cours de développement ou de maintenance. Il s'agit donc de garder une trace des données (le code source) qui changent, des personnes responsables de ces changements (les développeurs), des moments auxquels interviennent ces changements, et des divers liens sémantiques entre ces changements (fusion, copie, réécriture, etc.).

Pour gérer cette préoccupation, un système de gestion de versions comme Git garde la trace d'un ensemble d'informations, constituées de données et d'événements pertinents.

2. http://git-scm.com/

3.2.2 Données

Un système de gestion de versions enregistre un certain ensemble de données sur l'évolution du code. Dans le cas particulier de Git, les données gérées sur le code source sont, entre autres :

- Le contenu du code source, autrement dit, la suite de caractères qui représentent le code, dans divers fichiers, sur un médium de stockage physique.
- L'organisation du code source. Il s'agit des noms et emplacements des différents fichiers et répertoires qui contiennent le code source. Suivant le système de gestion de versions, des informations additionnelles comme les dates de modifications et les permissions sont aussi gérées [3].
- Les révisions apportées au code source ("commit"), dans le temps. Autrement dit, en plus de la dernière version de chaque fichier, est conservée tout l'historique des révisions apportées aux fichiers. Une révision peut contenir des ajouts, suppressions, ou changements de lignes dans des fichiers, ou encore des ajouts ou suppressions de fichiers.
- Les personnes responsables des différents changements apportés au code source. Généralement, l'identité des auteurs est représentée par leur adresse email.
- Les relations entre les différentes révisions réalisées sur le code source. Par exemple, un ensemble de révisions peut représenter une évolution uniforme du code lors de l'implémentation d'une fonctionnalité spécifique. Une telle suite logique de révisions est appelée "branche". Une révision peut aussi représenter le regroupement de deux branches différentes, auquel cas on parle d'une "fusion" ("merge").

  Dans le cas de Git, cette information est représentée sous forme d'arbre (graphe acyclique orienté). Chaque nœud du graphe représente [4] un ensemble de changements (une révision ou "commit"), et les arcs représentent les différentes relations de parenté entre commits (précédence simple ou fusion).
- Les différents dépôts ("remotes") qui contiennent aussi le code du même projet logiciel, leur adresse, et les relations que ces dépôts ont avec le dépôt courant.

Conceptuellement, un système de gestion de versions comme Git peut donc être vu comme une solution logicielle qui gère un ensemble de données pertinentes pour la préoccupation qui est la sienne. Ces données sont structurées avec un ensemble de concepts cohérents, appropriés pour la préoccupation, et les différentes relations entre ces concepts. Dans Git, les concepts d'unité d'information (fichier / "blob"), d'arborescence de répertoires ("tree"), de commit, etc., et les relations comme la parenté entre commits, sont utilisés pour structurer les données. Ces concepts et relations constituent donc, en un sens, un **modèle** des données de gestion de versions.

3. Dans Git, seul le contenu des fichiers, et leur permission d'exécution sont gérés.

4. Cette description est simplifiée. Le stockage interne de Git est en réalité un graphe acyclique orienté dont les nœuds peuvent être des entités appelés "object", de 4 types : "blob" (le contenu d'un fichier), "tree" (une arborescence de répertoires), "commit" (un "tree", représentant l'état du dépôt à une date donnée, ainsi que des informations additionnelles comme l'auteur de cette sauvegarde et une description), et "tag" (un nom et une description librement associés à un "commit").

3.2.3 Événements

Tout changement qui survient dans un système informatique peut être considéré comme un événement. Un événement peut être décrit par l'instant d'occurrence, le type d'événement (qui définit les propriétés que l'événement doit avoir), et les données (valeurs de propriétés) associées au changement précis.

Dans un système de gestion de versions, un "commit" est une donnée composée d'un ensemble de changements, l'identifiant du développeur responsable de ces changements, et les commentaires associés. Le terme "commit" est cependant aussi utilisé au sens d'événement [5]. Un événement "commit" a pour données associées les informations enregistrées dans la donnée "commit", et un instant associé qui est la date à laquelle ce commit a été créé.

La distinction entre donnée et événement pourrait sembler superflue au vu de la description précédente. En effet, un instant peut être considéré comme une autre donnée, rendant le concept distinct d'événement, a priori, inutile. Cette distinction prend cependant toute son importance, lorsque l'on essaye de donner à l'utilisateur final la possibilité de spécifier un traitement à faire, de manière récurrente, à certains instants précis. Par exemple, si l'on souhaite vérifier l'adhérence de nouvelles modifications faites sur le code aux conventions de style en vigueur dans un projet, il faut pouvoir lancer automatiquement ce traitement au moment d'un commit.

En l'absence d'un concept d'événement, vu qu'on dispose seulement de la possibilité de lire l'information sur les commits existants, il est nécessaire de consulter la liste des commits, d'en faire une copie, afin de déduire, à chaque consultation, si de nouveaux commits ont été effectués ou pas. Cette consultation régulière (on parle de "pooling") est inutilement coûteuse en ressources et imprécise dans la mesure où l'on ne se rend compte de la présence de nouveaux commits qu'à la prochaine consultation. La notion distincte d'événement permet donc de spécifier à l'avance les traitements voulus, afin que le système de gestion de versions les exécute au moment approprié.

Pour chaque événement, il est possible, en principe, de réaliser des traitements avant et après la prise en compte [6] de chaque occurrence de l'événement. Ces traitements sont configurables et généralement désignés par "hook" [7]. Le type d'un hook précise l'événement associé, et le moment auquel il faut l'exécuter (juste avant ou juste après l'événement). Le moment d'exécution constitue ainsi un "point d'accroche" dans le système de gestion de versions, auquel on peut greffer des traitements tiers. Git permet de configurer des hooks de type pre-commit (juste avant qu'un commit ne soit enregistré) et post-commit (juste après qu'un commit ne soit enregistré). Pour définir des traitements pour un type de hook, il suffit de les placer dans un fichier exécutable, portant un nom conventionnel associé à l'événement, et défini par Git. Par exemple, un traitement à exécuter avant chaque commit sera défini dans le script `.git/hooks/pre-commit` [8]. Les différents paramètres associés à l'occurrence d'événement en

5. Pour faire la distinction, dans cet exposé, on parlera de "donnée commit" et d'"événement commit".
6. La prise en compte d'un événement peut simplement consister à l'enregistrer dans un historique. Un traitement personnalisé peut être réalisé avant ou après cet enregistrement.
7. C'est le cas dans Git.
8. `.git` est un répertoire présent dans tous les dépôts Git, et contient des métadonnées concernant le dépôt. Il

cours de traitement sont passés au script via des variables d'environnement [9].

Les événements et les points d'accroche associés sont définis s'il est pertinent d'introduire des traitements tiers à ces points. Remarquons qu'en général, les points d'extensions d'avant un événement (pre-commit par exemple) peuvent servir à vérifier des pré-conditions définies par l'utilisateur : Git vérifiera donc la valeur de retour du script associé, et ne procédera à l'enregistrement du commit que si la valeur indique un succès. Ceci permet par exemple de rejeter automatiquement un commit qui ne respecte pas les règles de style.

Les événements ne servent donc pas qu'à introduire des traitements à certains instants précis, mais aussi, à garantir certaines règles de cohérence sur les données enregistrées.

3.2.4 Utilitaires de consultation

Les différentes informations enregistrées par un outil de gestion de version peuvent être lues avec un ensemble d'utilitaires, chacun adapté à un cas d'utilisation donné.

Dans Git, 144 utilitaires sont disponibles pour manipuler l'information de gestion de versions. Des exemples d'outils pour la lecture sont git-log (lecture de l'historique), git-checkout (lecture des fichiers contenant le code géré dans Git), git-diff (lecture des différences entre deux versions), git-grep (recherche d'information dans le code), etc.

Chacun de ces utilitaires offre une vue partielle du modèle géré par Git, dans une variété de formats de sortie. Ces utilitaires étant des outils accessibles en ligne de commande, ils peuvent être facilement invoqués par d'autres outils via un appel système, pour demander une information particulière relative à la gestion de versions, et l'obtenir dans un format (textuel) facile a manipuler.

Il ne s'agit cependant pas juste d'une exportation de différents aspects ou de différentes vues du modèle de données de gestion de version interne de Git. Vu que ces utilitaires acceptent des paramètres pour spécifier l'information précise recherchée, ils forment ensemble, avec les utilitaires de base d'une ligne de commande, un moyen de référer une unité précise d'information de gestion de versions. Par exemple, dans chaque dépôt git, la commande "`git rev-list -all -author "user@example.com" -since "2012-12-01" -until "2012-12-31" | wc -l`" représente une manière de faire référence à l'information précise *"le nombre de révisions introduites par l'utilisateur identifié par `user@example.com` dans le mois de Décembre 2012"* [10]. Il faut noter que cette référence est complètement indépendante des détails d'implémentation utilisés par Git pour garder la trace de cette information. Les outils de lecture d'information de Git, combinés aux outils de base de la ligne de commande, peuvent donc être considérés comme un **langage de requêtes** d'accès à des informations relativement précises de contrôle de versions.

s'agit de diverses configurations, et des données sur les versions précédentes.

9. Le script peut aussi utiliser les commandes Git de lecture des données de gestion de versions pour inspecter l'état du système au moment du commit.

10. L'exécution d'une telle commande renvoie le nombre recherché, et rien d'autre.

3.2.5 Éditeurs

De même qu'un ensemble d'utilitaires permet de lire les données de gestion de versions, un autre ensemble d'utilitaires permet de les créer et de les modifier. Par exemple, Git met à disposition, entre autres, git-commit (enregistrement d'une nouvelle révision), git-pull (copie de révisions réalisées dans d'autres dépôts), git-merge (création d'une révision qui est la fusion de deux autres), git-rebase (réécriture d'un ensemble de révisions existantes), etc. Ces utilitaires sont des éditeurs, dans la mesure où ils permettent de manipuler l'information de gestion de versions.

Ces utilitaires forment une interface, qui permet à Git de garantir la cohérence des informations de gestion de versions, tant qu'on ne touche aux données de gestion de versions que via les utilitaires prévus. Ici encore, les utilitaires permettent des modifications relativement précises, par un tiers, de l'information de gestion de versions. Ceci permet par exemple de modifier le message associé à une révision particulière.

3.2.6 Interface d'accès via le réseau

Les outils de collaboration ont en général, naturellement, plusieurs installations. Ces installations peuvent correspondre simplement aux différents utilisateurs. Dans le cas de Git par exemple, chaque utilisateur a sa copie personnelle du dépôt correspondant au projet en cours de développement. Un aspect important de la gestion de versions est la communication entre ces dépôts, principalement pour échanger des révisions.

Divers protocoles sont utilisés pour les échanges entre dépôts de gestion de versions. Avec Git par exemple, un dépôt est capable de communiquer avec un autre dépôt local présent dans un autre répertoire, ou un dépôt distant via un protocole propre (appelé lui aussi "git") ou les protocoles SSH (Secure Shell) et HTTP (Hypertext Transfer Protocol). En particulier, la possibilité de communiquer via HTTP permet d'utiliser Git dans toutes sortes d'environnements industriels, qui peuvent avoir diverses restrictions sur les protocoles autres que HTTP (en filtrant des ports dans leur firewall) par mesure de sécurité.

3.2.7 Outils de support au développement

Tout un écosystème d'outils de support au développement exploite les données et événements mis à disposition par les systèmes de gestion de version comme Git. Il faut remarquer que ces outils n'ont pas forcément à avoir comme préoccupation principale la gestion de versions. Il suffit que le problème qu'ils résolvent ait *besoin* de l'information de gestion de versions. Ainsi, contrairement aux utilitaires de consultation et de manipulation décrits plus haut, les outils de support au développement ne sont pas forcément restreints à l'information de gestion de versions.

Les outils basés sur Git, et qui relèvent strictement de la gestion de version, sont en général des outils de génération de statistiques et de diverses visualisations sur l'historique du projet. Dans la mesure où un outil est propre à la préoccupation de gestion de versions, il peut être considéré comme un utilitaire de haut niveau, exploitant les utilitaires de bas niveau mis à disposition par Git pour consulter et modifier l'information de gestion de versions. Par

exemple, un outil de visualisation graphique de l'historique est une version de plus haut niveau de l'utilitaire natif git-log, qui sert à extraire toutes sortes d'informations sur l'historique. De même, un outil graphique pour réorganiser des répertoires, et appliquer le changement à toute l'historique du projet, est une interface de haut niveau pour les fonctionnalités mises à disposition par un utilitaire de bas niveau comme git-filter-branch.

D'autres outils, dédiés plus généralement au support du développement logiciel collaboratif, peuvent faire appel aux utilitaires Git. Ceci représente d'ailleurs leur utilisation la plus courante. Par exemple, le fait, dans une équipe de développement, de s'assurer que le code introduit dans chaque révision suit certaines conventions où passe certains tests peut être implémenté comme un outil indépendant, invoqué par un hook de type pre-commit. Il s'agit dans ce cas d'une problématique d'assurance qualité, résolue, en partie, grâce à l'exploitation des événements de gestion de versions. Il faut noter qu'un tel outil peut librement utiliser d'autres informations sans lien avec la gestion de version pour réaliser son travail.

3.3 Conceptualisation

Dans la section précédente, une étude du système de gestion de versions Git a été effectuée, afin de mettre en évidence les concepts majeurs qui sous-tendent sa conception et comment ces concepts contribuent au support du développement collaboratif. La présente section propose un modèle conceptuel du support au développement collaboratif (voir figure 3.1), s'appuyant sur l'étude précédente, et généralisé de manière à rendre compte de tout système contribuant au support du développement collaboratif.

3.3.1 Vue d'ensemble

L'idée de base de la présente thèse est que le support au développement collaboratif consiste à identifier des **problèmes** de développement collaboratif[11], et à créer des **outils** qui y apportent des solutions. Un outil de développement collaboratif exploite des informations qui peuvent êtres classées en **données** et **événements**. Chacune de ces informations est liée à une **préoccupation** de développement collaboratif, et est gérée par un **gestionnaire de préoccupation**, qui met à disposition un ensemble d'**utilitaires** pour lire (formant un **langage de requêtes**) et modifier (sous forme d'**éditeurs natifs**) ces informations. Un gestionnaire de préoccupation peut mettre à disposition ses utilitaires de lecture et de modification via une **interface d'accès via le réseau**. Un ensemble d'outils de support à la collaboration peuvent être rassemblés en un seul exécutable qui offre, entre autre, une présentation uniformisée, constituant ainsi un **environnement de développement intégré**.

3.3.2 Préoccupation de développement collaboratif

La notion de "séparation des préoccupations" a été introduite par E.W. Dijkstra[Dijk 82]. Il s'agit *"d'étudier en profondeur un aspect particulier d'un sujet, de façon isolée, à cause de sa*

11. L'identification des problèmes de développement collaboratif n'est pas traitée dans cette thèse. Notre objectif est de fournir un support conceptuel pour les outils répondant à des problèmes de développement collaboratif donnés.

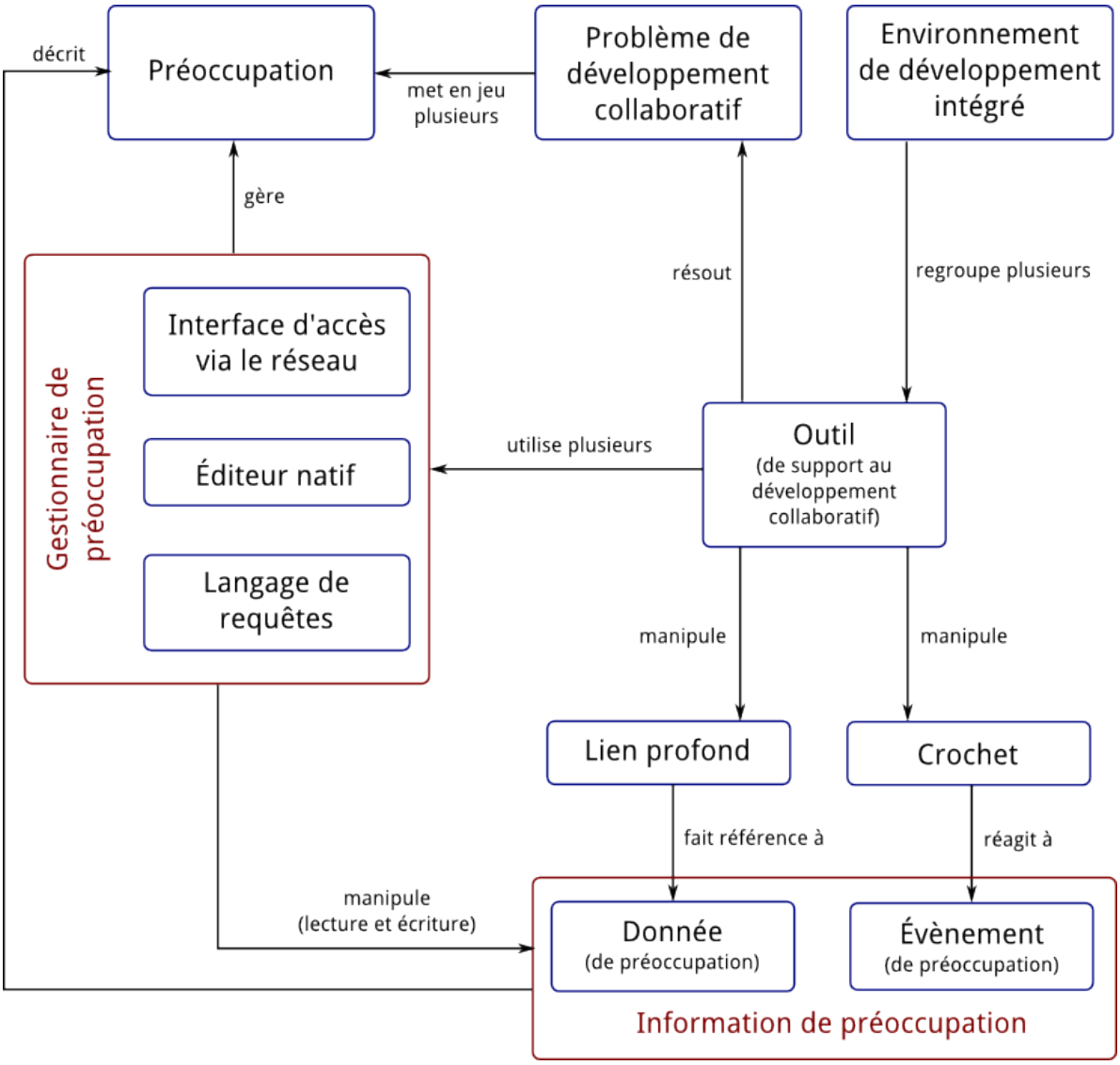

FIGURE 3.1 – Modèle conceptuel du support au développement collaboratif

propre cohérence, sachant pertinemment qu'on ne s'occupe que d'un des aspects". Dijkstra précise qu'*"il ne s'agit pas d'ignorer les autres aspects du sujet, mais de reconnaître et mettre en valeur le fait que du point de vue de l'aspect considéré, les autres sont hors de propos"*. La modularité est ainsi une conséquence de la séparation des préoccupations, et est nécessaire pour étudier un sujet ou construire un système complexe.

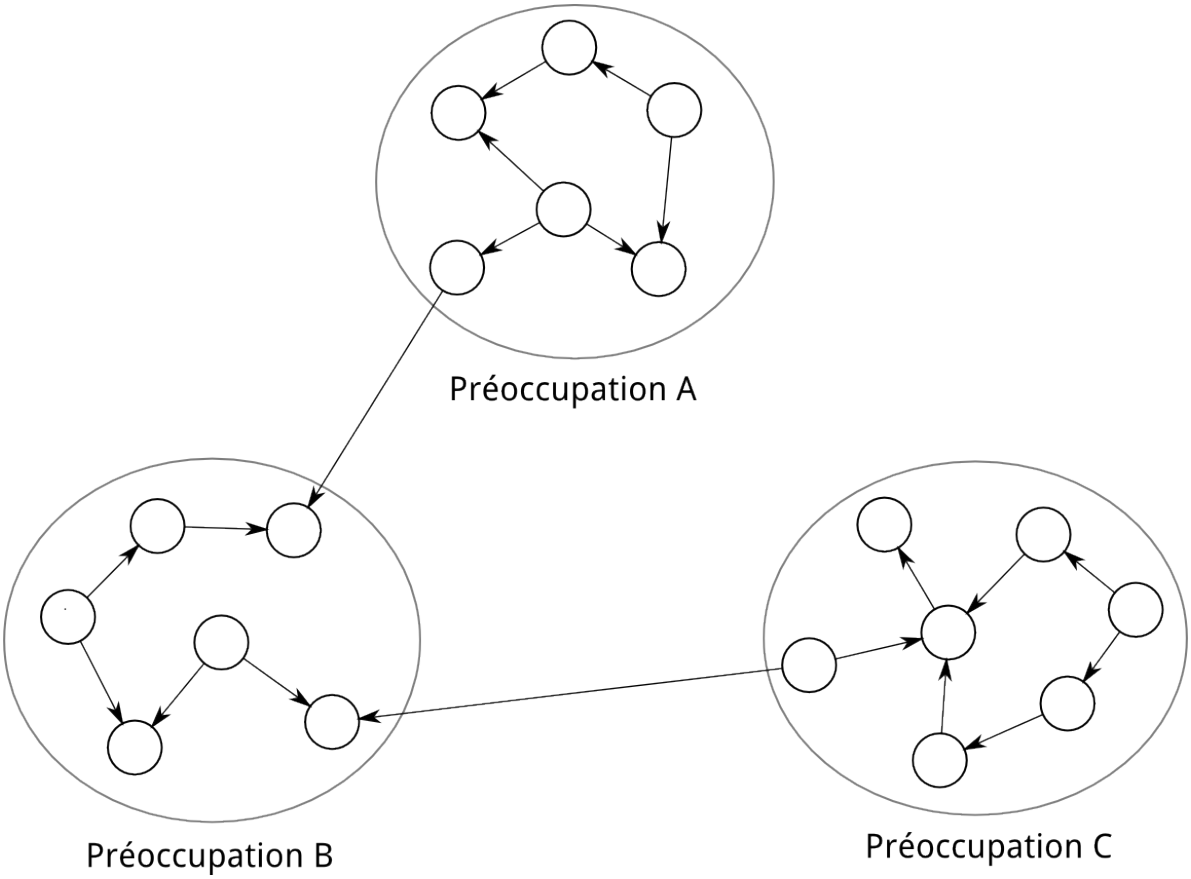

FIGURE 3.2 – Dépendances entre les concepts de trois préoccupations de développement collaboratif

Pour appliquer cette notion de "séparation des préoccupations" au développement collaboratif, il est nécessaire d'expliciter ce qu'est une préoccupation dans le cadre du développement collaboratif. De manière générale, une préoccupation est un sous-ensemble de concepts. Ces derniers sont choisis de telle sorte que les dépendances entre eux soient considérablement plus fortes, globalement, que les dépendances qu'ils peuvent entretenir avec des concepts extérieurs à la préoccupation. Autrement dit, si les concepts décrivant un sujet sont représentés par des nœuds d'un graphe, et les dépendances entre concepts par des arcs, une préoccupation est un sous-graphe qui contient relativement plus d'arcs internes que d'arcs sortants ou entrants (voir figure 3.2).

Une préoccupation de développement collaboratif est donc un ensemble de concepts, liés au développement collaboratif, qui ont suffisamment de cohésion interne pour être traités ensemble. Un exemple évident est l'évolution du code source d'un système logiciel, commu-

nément appelé “contrôle de versions”, qui a été étudié dans la section précédente. D’autres exemples sont la gestion des défauts logiciel (ou bugs), l’intégration continue, la gestion de la documentation, etc.

3.3.3 Gestionnaire de préoccupation

Une préoccupation de développement étant identifiée, un système logiciel peut être mis en place pour gérer l’information relative à cette préoccupation. Un tel système logiciel, le gestionnaire de préoccupation, peut se présenter sous diverses formes : un seul exécutable, un ensemble d’exécutables (utilitaires), ou encore un serveur disponible sur le réseau (voir figure 3.1).

Le rôle d’un gestionnaire de préoccupation est d’organiser l’information relative à la préoccupation. Ces informations sont constituées de données et d’événements, décrits dans les sections à suivre. Le gestionnaire de préoccupation a aussi pour rôle de mettre à disposition des fonctionnalités de lecture et de modification de l’information qu’il gère. Dans le cas de la gestion de versions par exemple, Git (la suite d’utilitaires) est un gestionnaire de préoccupation.

3.3.4 Données et liens profonds

Les données relatives à une préoccupation (voir figure 3.1) sont un ensemble de concepts et de relations qui décrivent, à tout instant donné, l’information disponible sur cette préoccupation. Les données forment une abstraction de l’aspect de la collaboration à laquelle s’intéresse la préoccupation et constituent, dans ce sens, un **modèle**. Un tel modèle peut être plus ou moins formalisé, depuis de simples informations textuelles dans un fichier dans un format ad hoc propre au gestionnaire de préoccupation et peu formalisé, jusqu’à un modèle s’appuyant sur un méta-modèle avec une sémantique définie.

Peu importe le niveau de formalisation des données d’une préoccupation, elles sont structurées [12] par les concepts familiers de la préoccupation. Pour la gestion de version par exemple, il s’agit des fichiers, des versions, des différences entre versions, des fusions, etc. Dans le cas de la gestion de défauts logiciels, il s’agira plutôt des tickets, des priorités, des dépendances, etc.

La notion de lien profond ou “deep-link” vient du web, et plus précisément de l’hypertexte. Un document hypertexte peut contenir des liens vers d’autres documents hypertexte. Conceptuellement, un lien fait référence à l’information contenue dans le document sur lequel il pointe. Dans un système hypertexte comme le web, un document hypertexte peut contenir une information précise, ou peut agir comme une table de matières, et contenir principalement des liens vers d’autres documents. La page d’accueil d’un site web contient typiquement des liens vers les autres pages, ou les grandes catégories. Un lien qui pointe vers une page d’accueil fait donc référence à une information moins précise, moins spécifique, qu’un lien qui

12. Les concepts plus fins, de bas niveau, peuvent ne pas être liés au domaine. Par exemple, bien qu’un système de gestion de versions comme Git stocke l’information sous forme de graphe, l’utilisation courante des utilitaires git ne porte pas sur les concepts de nœud et d’arc, mais sur des concepts propres à la gestion de versions comme les révision/commits, les fichiers, les messages de commit, les fusions, etc.

pointe vers une page particulière. On parle de **deep-link** quand le lien pointe vers une page particulière, et donc vers une information précise.

Un document hypertexte **exploite** l'information disponible dans d'autres documents hypertexte, en incluant des liens vers ces documents. Si les documents hypertexte étaient réduits à inclure des liens seulement vers des collections de documents (comme la page d'accueil d'un site web), leur utilité ou efficacité serait grandement réduite. En effet, il faudrait inclure, à côté de chaque lien, une description de comment retrouver le document hypertexte précis auquel on fait référence dans la collection. Un deep-link, par contre, élimine cette contrainte, en permettant d'utiliser (en y faisant référence) l'information précise dont on a besoin. Autrement dit, plus une collection d'informations dans des documents hypertextes a une granularité fine, et qu'il est possible d'associer une adresse (et donc un lien) à chacune des unités d'information, plus il est facile d'exploiter cette collection dans d'autres documents hypertexte.

Cette discussion sur l'hypertexte est très révélatrice pour le développement collaboratif. En effet, les problèmes de développement collaboratif ne mettent pas forcément en jeu une préoccupation unique. Un outil de support à la collaboration doit donc, entre autres, exploiter de l'information provenant de diverses préoccupations. Dans la suite de l'analogie hypertexte, il s'agit d'accéder à une information précise dans une autre collection de documents hypertextes, ce qui n'est possible que si cette collection offre un accès **individuel** à cette information. Les gestionnaires d'information on donc besoin d'offrir des deep-links vers des parties précises des données qu'ils gèrent. Les facilités mises à disposition dans ce sens forment un langage de requêtes, décrit plus loin.

3.3.5 Événements et crochets

Pour les besoins de la présente étude, un événement est *une occurrence, dans le cadre d'un projet de développement logiciel collaboratif, significative pour une préoccupation particulière*.

Le fait qu'une occurrence soit significative pour une préoccupation se matérialise par le fait que cette occurrence soit associée à une modification potentielle des données décrivant la préoccupation. En effet, les données étant supposées décrire complètement une préoccupation, toute occurrence qui ne peut pas modifier les données d'une préoccupation est, du point de vue du gestionnaire de préoccupation, indépendante de la préoccupation. Par exemple, dans un système de gestion de versions, une tentative de mise à jour d'un dépôt A, en récupérant les modifications d'un dépôt B, peut n'occasionner aucune modification des données de gestion de versions, si le dépôt A est déjà à jour par rapport au dépôt B. Cependant, la tentative aurait entraîné une modification si les dépôts A et B n'étaient pas déjà synchronisés.

Les données relatives à une préoccupation peuvent se limiter à l'état du système à un instant donné (mises à jour destructives) ou enregistrer de manière cumulative l'historique de toutes les mises à jour (mises à jour en mode "append-only").

Si des mises à jour **destructives** sont utilisées, les événements correspondent à toute donnée nouvelle qui (1) est disponible, du moment que (2) les données existantes n'ont pas encore été modifiées pour la prendre en compte. Si par contre, des mises à jour **cumulatives** sont utilisées, les événements correspondent à toute donnée nouvelle qui (1) est disponible, et

(2) n'a pas encore été ajoutée à l'historique des mises à jour. Dans les deux cas, (1) équivaut à dire que l'occurrence s'est **produite**, et (2) veut dire que l'occurrence n'a pas encore été **traitée**.

Avec les définitions précédentes, un événement est effectivement toute occurrence significative pour une préoccupation donnée, qui s'est produite, et qui n'a pas encore été traitée. Les données d'une préoccupation sont donc la synthèse de tous les événements passés.

Un crochet ou "hook" est un traitement prévu pour réagir à un événement (voir figure 3.1). Un hook peut être exécuté juste avant la prise en compte d'un événement, ou sinon juste après. Dans le cas d'un hook d'avant la prise en compte d'un événement, le résultat d'exécution du hook peut être utilisé pour décider de comment l'événement doit être pris en compte. En particulier, le résultat de l'exécution d'un tel hook peut être utilisé par le gestionnaire de préoccupation pour annuler conditionnellement l'action en cours correspondant à l'événement.

3.3.6 Langage de requêtes

Un gestionnaire de préoccupation met les données de la préoccupation à disposition des applications tierces (voir figure 3.1). Les données peuvent être disponibles via un mécanisme d'exportation de l'ensemble des données, et/ou un mécanisme de sélection d'informations précises contenues dans les données.

L'exportation de l'ensemble des données d'une préoccupation est d'une utilité limitée pour des outils tiers. D'une part, il est peu probable qu'un outil tiers ait besoin de l'ensemble des données, puisque le cas échéant, cet outil est en réalité un utilitaire propre à la préoccupation. En effet, un tel utilitaire doit comprendre un format de données au moins aussi complexe que celui utilisé en interne par le gestionnaire de préoccupation. D'autre part, ce genre de copie d'information à grande échelle pose des problèmes de synchronisation quand les données de préoccupation évoluent.

L'accès sélectif à une information particulière relative à une préoccupation, est, par rapport à l'exportation, un cas d'utilisation plus fréquent et moins problématique. En effet, il s'agit de répondre à une question particulière relative à la préoccupation. Si la préoccupation est la gestion de défauts logiciels par exemple, on peut demander "combien de tickets de bug affectés à l'utilisateur X sont actuellement non-résolus ?". La réponse peut être obtenue en effectuant une requête dans le langage défini par le gestionnaire de préoccupation, si ce langage prévoit une telle requête. Sinon, la réponse peut être déduite de réponses à des requêtes intermédiaires ("quels sont les rapports de bug affectés à l'utilisateur X ?", "le rapport de bug Y est-il non-résolu ?") après des traitements relativement simples.

Pour obtenir une information précise relative à une préoccupation, il faut avoir un moyen de la désigner, de faire référence à cette préoccupation. Par exemple, pour avoir accès à un document hypertexte sur internet, il faut connaître son adresse URL, qui joue le rôle de référence. De même, pour avoir accès à l'information "quel est le message associé au dernier commit" dans un dépôt Git, la commande `git log -1 -pretty=%B` qui récupère une telle information joue le rôle de référence.

Une fois qu'une référence à une information est disponible, il suffit de la **déréférencer** pour avoir accès à l'information. Dans le cas d'un document hypertexte par exemple, il faut introduire l'URL dans un navigateur web. Dans le cas de la commande `git log -1 -pretty=%B`, il faut l'exécuter dans un shell, dans le contexte d'un dépôt Git.

L'ensemble des références possibles à des informations précises relatives à une préoccupation forment un langage de requêtes. L'existence d'un langage de requête, et sa facilité d'utilisation par des outils tiers, contribuent grandement à l'exploitation des données d'une préoccupation par des outils tiers.

Un langage de requête peut avoir plusieurs formes, suivant la technologie de mise en œuvre utilisée. Si l'on utilise des exécutables destinés à être exécutés par un shell ou en tant que sous processus, le langage est formé par les noms des exécutables et les différents paramètres qu'ils acceptent. Si l'on utilise un serveur HTTP par exemple, le langage est formé par les différentes URL disponibles et les différentes méthodes HTTP que l'on peut appliquer à ces URLs.

3.3.7 Éditeurs natifs

Un éditeur de préoccupation gère des données et leur évolution dans le temps. En particulier, il gère la modification de ces données. Les éditeurs natifs sont l'ensemble des utilitaires mis à disposition par le gestionnaire de préoccupation pour créer et mettre à jour les données de la préoccupation (voir figure 3.1). Ces éditeurs sont qualifiés de "natifs" dans la mesure où tout autre outil de manipulation des données de la préoccupation doit leur déléguer la modification effective. Cette interface bien définie de modification permet de garantir la cohérence des données de la préoccupation.

Les éditeurs natifs évitent à tout autre outil d'avoir à comprendre le format natif dans lequel le gestionnaire de préoccupation stocke les données de la préoccupation. Un tel découplage simplifie l'implémentation de tous les autres logiciels interagissant avec le gestionnaire de préoccupation. Par exemple, s'il est trivial d'ajouter un rapport de bug à un gestionnaire de défauts logiciels, l'implémentation de la signalisation de bug est considérablement facilitée pour divers scénarios comme à partir du système en cours d'exécution, à partir d'environnements de développement intégrés, ou encore via des systèmes de gestion du support client.

Les éditeurs permettent aussi une mise à jour sélective, précise, de l'information liée à une préoccupation. Dans un système de gestion de défauts logiciels par exemple, si l'opération d'ajout de commentaire à un rapport de bug existant est simple, ceci simplifie (et donc encourage) la création d'outils tiers, automatisés, qui, sans pour autant comprendre toute la logique de la préoccupation de gestion de défauts logiciels, peuvent offrir de l'assistance aux développeurs. C'est le cas par exemple des robots de détection de rapports de bug dupliqués dans les projets de développement open source de grande envergure.

3.3.8 Interface d'accès via le réseau

Dans le cadre du développement logiciel collaboratif, les outils qui ont besoin de l'information sur une préoccupation particulière ne se trouvent pas forcément sur la même machine

que le gestionnaire de préoccupation. Il est donc nécessaire de fournir une interface d'accès aux données de la préoccupation via un réseau de communication. Cette interface étant une voie d'accès aux éditeurs natifs et au langage de requête, elle permet autant la lecture que la modification (voir figure 3.1).

Les caractéristiques essentielles des éditeurs natifs et du langage de requête doivent rester manifestes pour les outils qui interagissent avec l'interface d'accès réseau. Ceci place un certain nombre de contraintes de conception sur cette interface.

D'une part, cette interface doit pouvoir associer un "nom" à chaque information élémentaire de la préoccupation [13]. Autrement dit, il faut pouvoir, en une transaction avec cette interface, accéder à n'importe quelle donnée, et non pas être obligé d'exécuter tout un processus par étapes. Par exemple, dans l'interface d'accès via le réseau d'un outil de gestion de défauts logiciels, il faut pouvoir accéder aux données d'un ticket par exemple, sans avoir à récupérer toutes les données du projet avant d'y rechercher l'information voulue.

D'autre part, une telle interface doit supporter la sémantique d'invocation des éditeurs natifs. En effet, l'invocation d'un éditeur natif pour modifier des données renvoie une information sur le succès ou l'échec de l'opération, le plus souvent avec des codes d'erreur bien définis, qui décrivent précisément le problème rencontré.

Enfin, l'interface d'accès via le réseau doit être capable de prendre en compte le mécanisme de réaction aux événements (hooks). Pour ce faire, elle doit disposer d'une fonctionnalité d'invocation d'un outil tiers, qui a préalablement souscrit à un événement, tout comme le gestionnaire de préoccupation est capable d'invoquer un exécutable (local) tiers quand un événement se produit. L'interface doit pouvoir aussi interpréter le résultat de l'exécution d'un écouteur d'événement, et le prendre en compte. Par exemple, l'action en cours peut être annulée si l'écouteur d'événement rencontre un problème d'exécution.

3.3.9 Outils de support à la collaboration

Le support au développement collaboratif est fourni par des outils qui adressent des problèmes particuliers. Les outils fournissent leurs services en s'appuyant sur l'information disponible sur les différentes préoccupations de collaboration (voir figure 3.1).

Le constat essentiel sur lequel se base ce travail est que **les problèmes de collaboration ne correspondent pas forcément à une seule préoccupation de collaboration**. Il faut donc un découplage entre les systèmes de gestion d'une préoccupation, et les outils qui peuvent avoir besoin d'exploiter les données de cette préoccupation.

Dans ce sens, il convient de restreindre les gestionnaires de préoccupation à un pur rôle de gestionnaire d'information, de manière comparable aux bases de données dans une application N-tiers. Même si un gestionnaire de préoccupation est fourni avec des outils de support à la collaboration, il convient de distinguer le rôle de *gestion d'information* de celui de *support à la collaboration*. En effet, cette distinction permet de constater que le support à la collaboration met généralement en jeu des informations venant de plusieurs préoccupations, ce

13. Dans la mesure où elle est accessible via le langage de requêtes.

qui a des conséquences importantes pour la conception, aussi bien des outils de support à la collaboration, que des gestionnaires de préoccupation.

Un logiciel qui joue accidentellement les deux rôles (gestionnaire de préoccupation et support à la collaboration) doit donc être conçu de manière à ce que l'un de ces deux rôles ne pénalise pas l'autre. Par exemple, le fait qu'un gestionnaire de préoccupation ait un langage de requête basique (dans la mesure où il ne donne pas accès à certaines informations de préoccupation) peut être le signe qu'il a été conçu dans la logique de servir d'outil de support, sans considération pour le fait que d'autres outils de support peuvent avoir besoin des données qu'il gère.

La différence entre préoccupation et problème de développement collaboratif est liée à la différence de finalité entre un gestionnaire de préoccupation et un outil de support au développement. Ainsi, le rôle du gestionnaire de préoccupation est de gérer (permettre de créer, lire, et modifier) de l'information *descriptive* sur un certain sujet, appelé ici "préoccupation". La finalité du gestionnaire de préoccupation est purement celle d'une infrastructure, et est complètement indépendante des *motifs* pour lesquels cette infrastructure peut être utilisée. Par contre, un outil de développement n'est pas forcément une infrastructure de gestion d'information, mais utilise des infrastructures de gestion d'information pour répondre à des besoins particuliers.

Une préoccupation se caractérise donc par sa généricité, et son caractère descriptif. Les concepts regroupés dans une préoccupation sont cohérents entre eux indépendamment des situations dans lesquelles ont peut les retrouver. Un problème de développement par contre, fait référence à des situations précises, où les informations mises en jeu n'ont que la cohérence circonstancielle due au problème particulier traité. Ces informations viennent chacune d'une préoccupation particulière, qui peut être traitée de façon isolée. Cependant, le regroupement de diverses informations dans la description et la résolution d'un problème, parce que circonstanciel, n'a par définition pas la cohérence interne nécessaire pour être étudié de façon isolée, dans la perspective de l'exploitation de ce regroupement dans d'autres situations.

3.3.10 Environnements de développement intégrés

L'intégration harmonieuse des diverses préoccupations de développement est essentielle pour un support effectif de la collaboration [Whit 07, Ossh 00]. Un environnement de développement intégré est un regroupement cohérent d'outils de collaboration (voir figure 3.1). Ces environnements offrent généralement une intégration de présentation[Kipe 87, Wass 90].

Il faut noter que l'existence d'un environnement intégré ne veut pas forcément dire qu'une préoccupation de développement est considérée comme principale par rapport à tous les autres. Il s'agit plutôt d'une décision pratique de greffer d'autres outils sur un outil avec lequel les utilisateurs finaux passent la majorité de leur temps. Ainsi, un environnement intégré construit autour des processus ne se justifie que si l'activité principale des utilisateurs porte sur le processus en lui même, ce qui est loin d'être le cas pour des développeurs.

Dans la plupart des environnements intégrés à destination de développeurs, l'outil de base est l'éditeur de code. Il est en effet défendable de considérer l'écriture de code comme une

activité sur laquelle il est facile de tisser une bonne partie des autres activités des développeurs (gestion de défauts logiciels, déploiement, documentation, communication, etc.). Ce n'est donc pas surprenant que les éditeurs de code soient régulièrement utilisés comme noyaux dans des solutions industrielles existantes comme IBM Jazz[Booc 03].

Remarquons que bien que les modèles de processus offrent une vue globale d'un projet logiciel, ils ne sont pas des candidats satisfaisants comme base d'un environnement intégré pour développeurs. En effet, le travail *quotidien* des développeurs ne nécessite pas forcément une prise en compte explicite des préoccupations du processus comme la dépendance globale des activités, leur niveau de complétion, l'état des livrables, les ressources affectées aux différentes activités, etc. Par contre, fonder un environnement intégré sur les processus peut être pertinent si l'environnement est destiné à des chefs de projet ou à des ingénieurs qualité. Ce point est abordé en détail dans la section 3.5.

3.4 Application à des outils existants

Dans la présente section, nous présentons des exemples d'outils de support à la collaboration (gestion de versions, gestion de défauts logiciels), et montrons comment le modèle conceptuel présenté à la section 3.3 peut être instancié pour décrire leur fonctionnement.

3.4.1 Gestion de versions

Une présentation du fonctionnement du système de gestion de versions Git a été faite à la section 3.2. Nous prenons ici l'assurance qualité comme exemple de problème de développement collaboratif (voir sous-section 3.3.1), et la gestion de versions en tant que préoccupation de développement collaboratif (voir sous-section 3.3.2), afin de montrer comment la conceptualisation faite dans la section 3.3 s'y applique. Pour ce faire, considérons une équipe de développement logiciel, travaillant sous Git, avec les normes d'assurance qualité suivantes :

- Tout code introduit dans un dépôt doit respecter les règles de formatage de code de l'équipe.
- Toute révision envoyée sur le dépôt officiel doit avoir passé les tests.

Un outil de reformatage de code, configuré avec les règles de l'équipe, est disponible, et les tests sont aussi disponibles. Cependant, pour résoudre ce *problème* d'assurance qualité, on a besoin, entre autres, de certaines *données* (contenu du code et des tests) et *événements* (ajout d'une nouvelle révision, envoi d'une nouvelle révision sur le dépôt officiel) de la préoccupation *"gestion de versions"*.

Git met à disposition un hook `pre-commit` (associé à l'événement *commit*) qui peut être utilisé pour examiner les fichiers modifiés lors de l'ajout d'une nouvelle révision. Il suffit donc d'effectuer les actions suivantes en utilisant ce hook :

- Extraire la liste des fichiers modifiés avec la commande `git-diff-index`, qui fait partie du *langage de requête* de Git.

– Invoquer l'outil de formatage de code (`indent` par exemple) sur les fichiers modifiés.
– Ajouter les fichiers modifiés par l'outil de formatage à la révision en cours d'enregistrement avec `git-add`, qui fait partie des *éditeurs natifs* de Git.

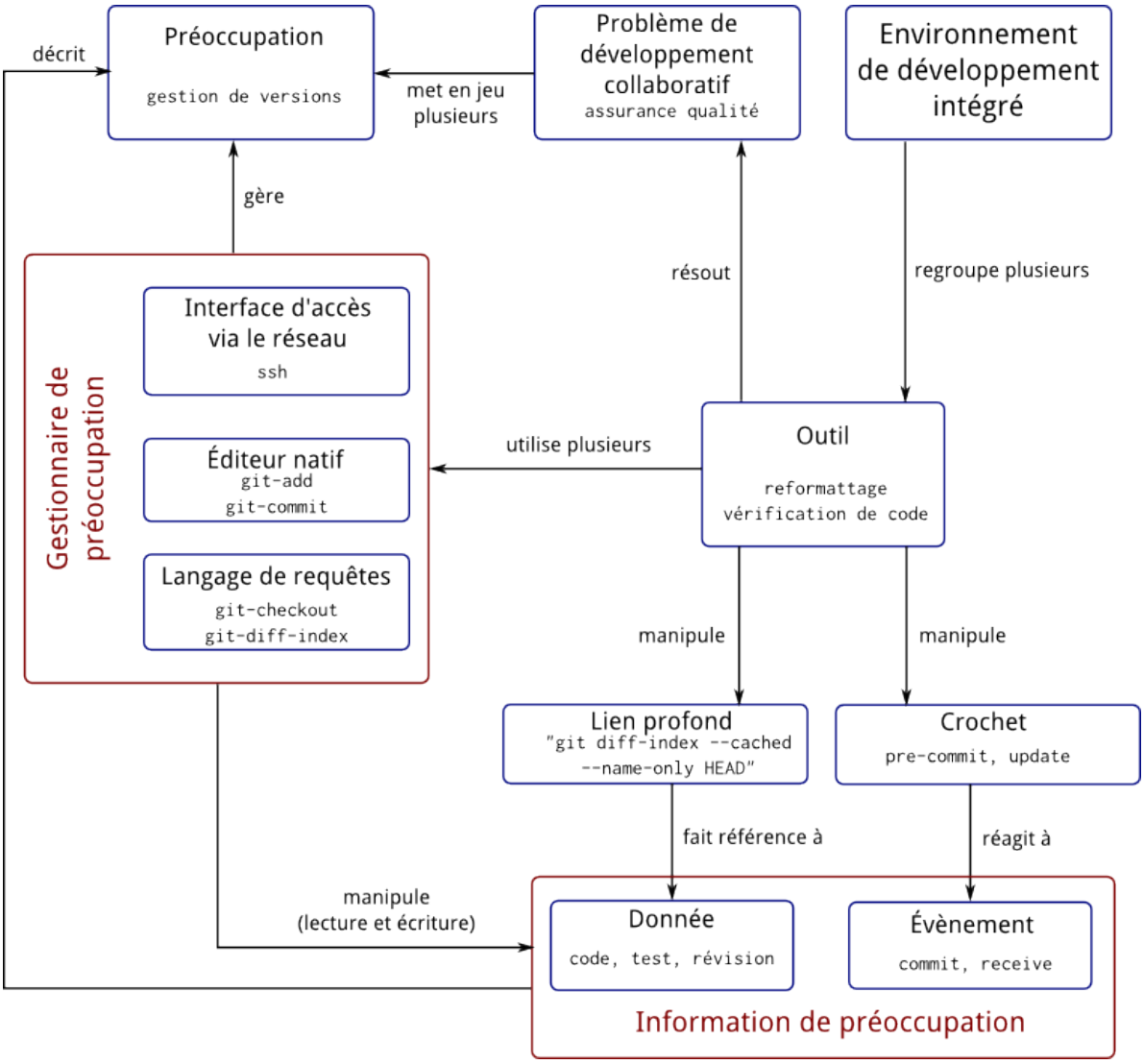

FIGURE 3.3 – Modèle conceptuel du support à la collaboration : application à l'assurance qualité

Un hook `update` (associé à l'événement *receive* : réception d'une nouvelle révision dans un dépôt) est aussi disponible dans Git, et est invoqué juste avant de mettre à jour le dépôt. Ce hook reçoit en paramètre une référence vers la nouvelle révision reçue. Dans cet exemple, la communication entre les dépôts Git est faite via ssh. Pour s'assurer que les tests passent avec succès sur la nouvelle série de révisions, il suffit de réaliser les actions suivantes dans le hook :

– Extraire une copie de la nouvelle révision dans un répertoire temporaire avec `git-checkout`, qui fait partie du *langage de requêtes*.

– Invoquer l'exécutable qui correspond aux tests.
– Selon les résultats des tests, positionner la valeur de retour du hook. Cette valeur de retour est lue par Git, pour décider si la nouvelle révision est acceptée ou pas.

La figure 3.3 montre la correspondance entre le modèle conceptuel et les différents éléments cités dans cet exemple.

3.4.2 Gestion de défauts logiciels

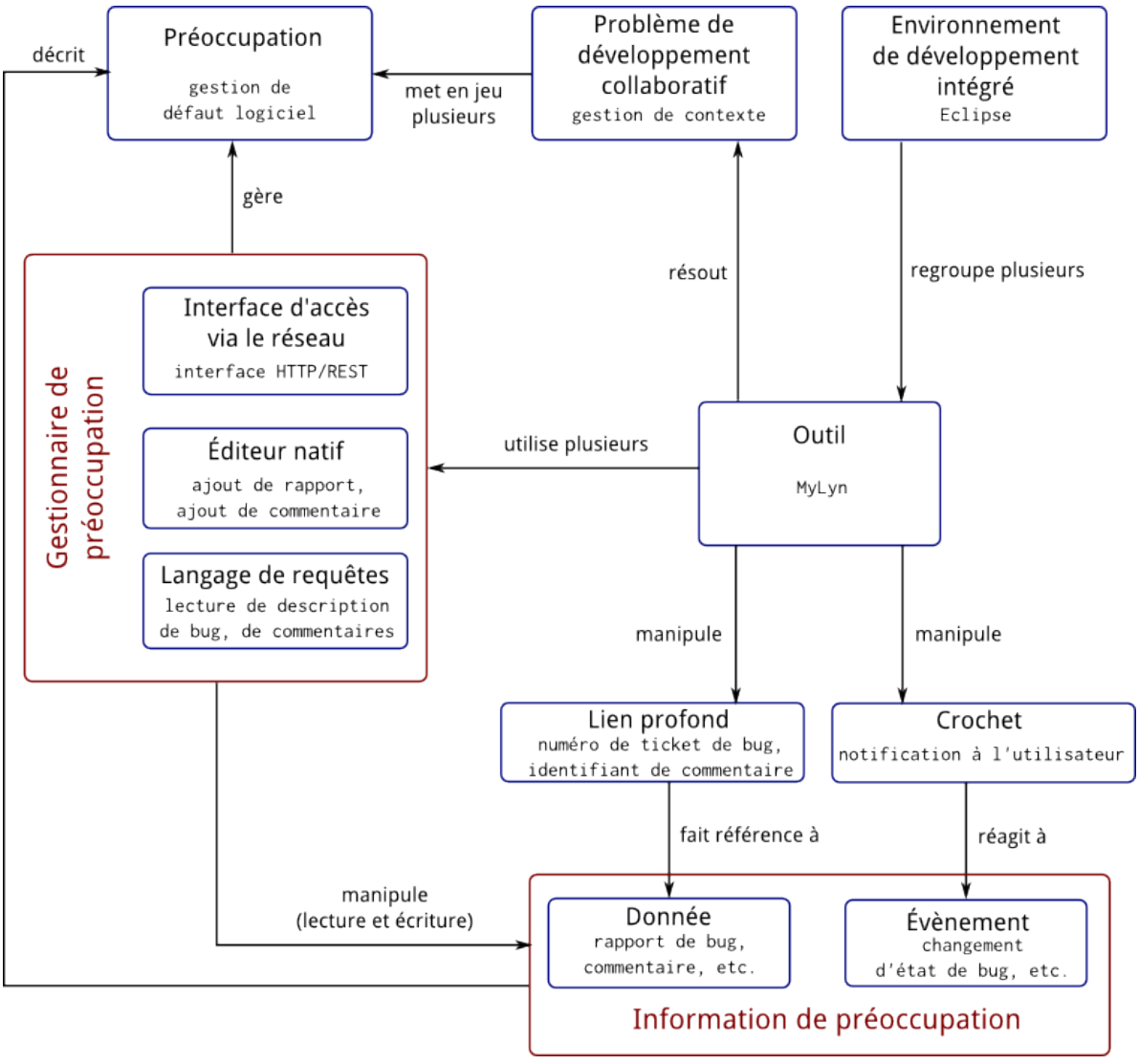

FIGURE 3.4 – Modèle conceptuel du support à la collaboration : application à la gestion de contexte

Nous considérons ici une grande équipe de développement, travaillant sur un projet logiciel complexe. Il se pose dans une telle équipe un problème (voir sous-section 3.3.1) de

gestion de contexte, auquel l'on répond à l'aide de la préoccupation (voir sous-section 3.3.2) de gestion de défauts logiciels. La gestion de contexte est avant tout un problème de collaboration. En effet, le contexte du travail d'un développeur est formé, non seulement par les artéfacts, mais aussi par tous les participants travaillant sur des artéfacts et tâches connexes, ainsi que les actions de ces participants.

Pour assister les développeurs dans la navigation dans le projet, il a été décidé d'adopter une interface basée sur les tâches [Goth 09]. Le projet a adopté l'outil Mylyn d'Eclipse, qui permet aux développeurs d'organiser leur espace de travail autour des tâches. MyLyn utilise des données de gestion de défauts logiciels, qui est une *préoccupation* de développement collaboratif gérée par des outils nommés "bug trackers", dont par exemple BugZilla.

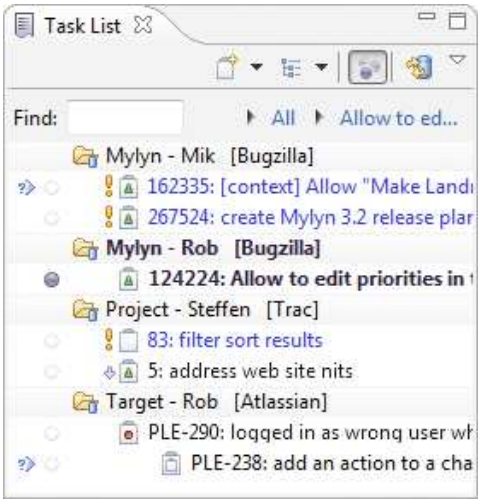

FIGURE 3.5 – Liste de défauts logiciels extraits de BugZilla, dans Mylyn

Mylyn permet de focaliser l'interface d'un éditeur comme Eclipse sur la tâche en cours, en masquant les fichiers non concernés, en mettant à disposition toutes sortes d'informations relatives à la tâche, et en offrant un moyen simple de changer de tâche, et de pouvoir y revenir sans perdre le contexte. Le contexte fourni par un outil comme Mylyn aux développeurs travaillant sur une tâche particulière contient, entre autres, des informations sur les défauts logiciels, leur état de résolution, les commentaires attachés, etc (voir figure 3.5). Cette information est disponible dans BugZilla, et Mylyn est capable d'aller la chercher pour la mettre à disposition dans l'IDE Eclipse. De plus, les développeurs peuvent créer ou modifier des rapports de bug sous BugZilla via Mylyn.

La figure 3.4 montre l'application du modèle conceptuel à l'utilisation des données de défauts logiciels (gérées par BugZilla) dans une interface orienté-tâches comme Mylyn.

3.5 Contribution de la modélisation des processus au support du développement collaboratif

Les processus logiciels sont une préoccupation de développement clairement délimitée. Par conséquent, ils peuvent être étudiés de façon indépendante, ce qui est généralement le

cas autant dans la recherche que dans l'industrie. Dans cette section, nous appliquons d'une part la conceptualisation faite dans la section 3.3 à la préoccupation de *"gestion des modèles de processus"*, afin de dégager un écosystème d'utilitaires adaptés à cette préoccupation. D'autre part, nous comparons l'approche basée sur le gestionnaire de préoccupation à celle des PSEEs, afin de mettre en relief les problèmes de l'approche PSEE.

3.5.1 Critique de l'approche fondée sur la notion de PSEE

Les PSEEs (Process-Centered Software Engineering Environment) sont des environnements de développement intégrés, nés de la recherche sur les modèles de processus. Ces environnements ont eux aussi pour but le support à la collaboration en exploitant les modèles de processus. Nous discutons ici de leurs limites, et montrons comment l'approche adoptée dans cette thèse répond à ces manques.

Comme le nom "Process-Centered Software Engineering Environment" l'indique, les processus sont supposés occuper une place centrale dans un PSEE[Gruh 02b]. Cette orientation fondamentale a des conséquences notables sur le fonctionnement pratique des PSEEs, en termes d'intégration de données et de contrôle d'une part, et de support au développement collaboratif d'autre part.

3.5.1.1 Intégration de données

Un PSEE assiste les développeurs dans la mise en œuvre d'un processus de développement. Pour jouer ce rôle, le PSEE a besoin d'informations sur les activités des développeurs. Dans une solution peu sophistiquée, toutes ces informations seront entrées manuellement dans le PSEE par le chef de projet ou un ingénieur qualité. Par contre, dans des solutions plus évoluées et destinées à de plus grands projets, il est utile que le PSEE puisse automatiquement extraire l'information dont il a besoin sur l'état du projet.

L'information sur l'état d'un projet est disponible dans les outils de développement utilisés. Un outil de gestion de versions par exemple peut renseigner sur l'état des produits. Des informations sur la progression d'une tâche peuvent être obtenues à partir d'un gestionnaire de défaut logiciel par exemple, ou alors d'un outil d'intégration continue qui garde la trace des déploiements.

Certains PSEEs[Kris 93, Jian 06] ont la capacité d'obtenir les informations pertinentes sur le processus à partir d'outils tiers, et de les exploiter. Ceci n'est évidemment possible que parce que ces outils tiers prévoient des mécanismes d'exportation ou d'interrogation des données qu'ils gèrent. Cependant, cette intégration de donnée n'est généralement pas possible dans l'autre sens. S'il peut être possible d'exporter le modèle de processus, il n'est pas possible d'interroger le PSEE pour avoir des informations précises comme l'état d'une tâche particulière, par des moyens automatisés. Autrement dit, l'intégration de données entre un PSEE et les autres outils de développement est à sens unique.

3.5.1.2 Intégration de contrôle

Dans un environnement de travail utilisant un PSEE, les activités de développement sont en général lancées et terminées à partir du PSEE [Zaml 05, Maci 09, Alei 11]. Ceci rend difficile voire impossible l'invocation automatisée de ces commandes à partir d'outils tiers. Autrement dit, les commandes pour contrôler le processus étant du ressort du PSEE, il n'est pas prévu que des outils tiers puissent être à l'origine de ces commandes, via une interface fournie de manipulation de processus. Par conséquent, il n'existe généralement pas d'autre moyen de manipuler le processus ou d'y introduire des informations supplémentaires que d'utiliser l'interface graphique du PSEE.

Les PSEEs lancent les autres outils de développement, quand ces outils sont associés à des événements de processus comme le début d'une activité. Il faut noter que ceci n'est possible que parce que ces autres outils mettent à disposition des moyens de contrôler leur fonctionnement. Dans le cas des éditeurs de texte par exemple, il s'agit de la possibilité d'ouvrir l'éditeur sur un fichier particulier, à une ligne particulière.

En termes d'intégration de contrôle, il y a donc une asymétrie, une intégration dans un seul sens. En effet, le PSEE peut effectuer des actions dans les autres outils de développement, mais l'inverse n'est pas possible, vu que le PSEE n'expose pas une telle fonctionnalité. Ce manque est tout à fait cohérent avec l'approche selon laquelle le processus est au centre du développement logiciel, et toutes les autres préoccupations y sont subordonnées.

Or, il existe des situations tout à fait légitimes où il serait utile de pouvoir contrôler le modèle de processus à partir d'autres outils. Par exemple, si l'enregistrement d'un fichier doit marquer la fin d'une activité, il est utile que l'éditeur de texte puisse être configuré pour notifier le PSEE. De plus, cette notification doit être possible par un mécanisme générique (invocation de sous-programme, communication inter-processus, requête HTTP, etc.) qui ne demande pas de modification substantielle de l'éditeur de texte. Cette solution nécessite que le PSEE expose des fonctionnalités de contrôle du processus, ce qui n'est pas disponible dans un PSEE, puisque non compatible avec la vision du PSEE en tant que centre de contrôle. Cependant, l'absence d'une telle fonctionnalité force les concepteurs de PSEE à adopter des solutions non optimales pour écouter l'environnement de travail, afin de détecter les changements significatifs pour le processus. Par exemple, Abriola et al.[Ambr 97] déplorent le fait que les outils de développement existants n'exposent pas suffisamment de points d'intégrations de contrôle, sans pour autant noter le besoin pour les PSEE de faire de même.

3.5.1.3 Support au développement collaboratif

Un des objectifs d'un PSEE est d'offrir du support au développement collaboratif. Ce support peut se présenter sous diverses formes : tableau de bord sur l'état d'un projet, vérification des pré et post conditions des activités, etc. Il faut cependant noter que dans un PSEE, toutes les fonctionnalités de support au développement qu'il fournit sont natives, et ne peuvent être fournies par des outils tiers via un mécanisme d'extension.

Ceci montre d'une part un manque de séparation entre la préoccupation de gestion de l'évolution du processus, et celle liée, mais distincte, de support au développement collaboratif

basé sur les données et événements du processus. Comme noté plus haut, un tel manque de séparation conduit souvent à une implémentation sous optimale de l'un ou l'autre de ces aspects.

D'autre part, une telle approche accentue les problèmes d'intégration de données et d'événements relevés plus haut. En effet, le PSEE n'étant pas prévu pour fournir l'information de processus aux autres outils, ou pour recevoir de l'information sur le processus à l'initiative d'autres outils, il est cloisonné. En conséquence, l'information de processus est piégée dans le PSEE, et ne peut être exploitée par aucun autre outil. Une telle situation rend difficile, voire impossible l'adoption *progressive* des processus de développement dans des projets collaboratifs.

3.5.2 Approche basée sur la gestion de préoccupation

La préoccupation "gestion de processus" porte sur la définition des modèles de processus de développement, et du suivi de leur évolution dans le temps. Il s'agit de définir l'organisation initiale du travail et prendre en compte les différents changements relatifs aux processus, intervenant au fur et à mesure de l'avancée du projet.

Les données de processus décrivent l'organisation structurelle du travail, ainsi que sa progression dans le temps. L'organisation structurelle du travail porte sur les activités et leurs relations, les produits en entrée et sortie de ces activités, et les ressources humaines affectées à ces activités. Pour décrire la progression du travail dans le temps, il faut pouvoir représenter des informations de plus fine granularité comme l'affectation des tâches aux divers participants, l'état de complétion des différentes tâches, l'état des différents artéfacts, les relations de travail entre tâches, participants, et copies de travail des produits, etc. Le chapitre 4 propose un méta-modèle dont les modèles correspondants peuvent décrire les données précitées.

Les événements de processus correspondent aux occurrences significatives dans la mise en œuvre d'un processus de développement collaboratif. Il s'agit par exemple du début et de la fin des tâches, de la création ou de la finalisation d'artéfact, de la disponibilité ou non de participant, etc. La formalisation des événements ainsi que le mécanisme de souscription et de réaction à ces événements sont décrits dans le chapitre 5.

Le chapitre 6 décrit l'implémentation d'un gestionnaire de préoccupation pour les processus de développement collaboratif. Un langage de requêtes y est présenté, ainsi que sa capacité à supporter les deep-links. Les options de modification du modèle de processus y sont aussi décrites, ainsi que le fonctionnement pratique des hooks. La modification et la lecture de processus sont mises en œuvre, entre autres, par un serveur de processus, qui forme l'interface d'accès via le réseau du gestionnaire de préoccupation.

La description d'un ensemble d'outils de support au développement logiciel, capables d'exploiter l'information mise à disposition par le gestionnaire de processus est incluse dans le chapitre 7.

3.6 Conclusion

Ce chapitre a permis de présenter un modèle conceptuel du support au développement collaboratif, et montré son adéquation à la description de la conception d'outils existants. Ce modèle conceptuel a ensuite été appliqué aux modèles de processus, pour définir les fonctionnalités d'une suite d'applications exploitant les modèles de processus, afin de supporter le développement collaboratif.

L'idée centrale du modèle conceptuel introduit dans ce chapitre est le découplage entre les problèmes de développement collaboratif et les préoccupations de développement collaboratif. Un problème de développement collaboratif peut mettre en jeu plusieurs préoccupations de développement collaboratif. Afin de permettre à tout outil répondant à un problème d'avoir un accès optimal aux informations de chaque préoccupation, il est nécessaire que le rôle de gestionnaire de préoccupation soit clairement distingué. Le gestionnaire de préoccupation s'occupe de mettre à disposition des outils tiers les moyens de lire (grâce à un langage de requête), de modifier (grâce aux éditeurs natifs) les données d'une préoccupation, aussi bien que le moyen de réagir aux événements de la préoccupation (grâce aux hooks).

La terminologie introduite dans ce chapitre sera utilisée dans tout le reste du manuscrit. Les chapitres 4 et 5 traitent de la formalisation des données et événements de processus. Le chapitre 6 traite de la définition et de l'implémentation d'un langage de requêtes pour les modèles de processus, ainsi que d'éditeurs natifs. Enfin, le chapitre 7 montre, entre autres, comment implémenter des outils de support à la collaboration exploitant l'information de processus.

Chapitre 4

Formalisation des aspects statiques de la collaboration

4.1 Introduction

Le chapitre 3 a introduit une conceptualisation du support au développement collaboratif, et montré comment l'appliquer aux modèles de processus. Dans le cadre de cette conceptualisation, ce chapitre a pour objectif de formaliser les **données** relatives à la préoccupation de gestion de processus.

SPEM (Software & Systems Process Engineering Metamodel) est le standard de l'OMG (Object Management Group) consacré à l'ingénierie des processus. Il se décrit à la fois comme un méta-modèle, et un framework conceptuel, qui fournit les concepts nécessaires pour modéliser, documenter, présenter, gérer, échanger, et mettre en œuvre des méthodes et processus de développement [OMG 07].

En tant que standard, le but principal de SPEM est de permettre la description d'une large gamme de processus dans le domaine du développement logiciel et système. Cette emphase sur la généralité explique le silence de la norme, non seulement sur les aspects plus précis et plus variables du développement logiciel, mais aussi sur certaines questions fondamentales pour le développement collaboratif.

Ce chapitre présente les questions laissées ouvertes par SPEM, montre leur intérêt pour la description et le support du développement collaboratif, puis propose, pour traiter ces questions, un ensemble de concepts et de relations formant une extension de SPEM. Ce nouvel ensemble de concepts sera désigné par CMSPEM [1] (Collaborative Model-Based Software & Systems Process Engineering Metamodel [Kedj 11a, Kedj 11b, Kedj 12b]).

1. Le méta-modèle CMSPEM a été élaboré dans le cadre du projet Galaxy, et contient aussi des concepts spécifiques à l'ingénierie dirigée par les modèles (d'où le qualificatif "model-based"). Ces concepts ne sont pas décrits ici, vu qu'ils ne sont pas pertinents pour la problématique de la collaboration.

4.2 Conceptualisation des processus collaboratifs

4.2.1 Caractérisation de la collaboration en génie logiciel

La collaboration a été définie comme “un effort collectif pour construire une compréhension partagée d’un problème et de sa solution” [Rosc 94]. Dans le cadre de cette étude centrée sur les processus de développement, nous parlerons de collaboration chaque fois que deux ou plusieurs participants à un projet travaillent sur la même tâche ou sur le même produit.

4.2.1.1 Intérêt de décrire la collaboration ad hoc

Les formalismes de description de processus existants suivent en général une approche où la définition est clairement séparée de la mise en œuvre, non seulement conceptuellement, mais aussi dans le temps : il y a une phase de définition clairement *antérieure* à la phase de mise en application.

Cette séparation est souvent défendue par les possibilités de réutilisation qu’elle crée, en permettant d’appliquer une seule définition à plusieurs cas d’utilisation. Dans SPEM par exemple, les définitions de base (paquetage `Method Content`) sont séparées de leur utilisation dans des processus génériques (paquetage `Process with Methods`).

Cependant, le travail collaboratif planifié n’est ni la seule forme de collaboration en génie logiciel, ni la plus répandue. Mieux, la collaboration ad hoc [2] est la forme dominante de collaboration en génie logiciel [Cher 08, Robi 00]. Une étude de terrain a montré que la collaboration ad hoc couvre jusqu’à 69% de toutes les formes de travail collaboratif en génie logiciel [Robi 00]. Par opposition à la collaboration planifiée qui a lieu parce qu’elle était à l’ordre du jour, la collaboration ad hoc survient en réaction à un problème nouveau, ou à des complications inattendues dans un problème existant.

Il est donc nécessaire de prendre en compte, dans les formalismes de représentation et de support au travail collaboratif, les formes de collaboration qui échappent à la planification. Pour ce faire, tout modèle de développement collaboratif doit pouvoir évoluer naturellement en fonction de l’avancée effective des travaux lors de la mise en œuvre.

4.2.1.2 Rôle de la granularité des concepts

Á l’utilisation de tout outil de support au développement logiciel est associé une certaine surcharge cognitive, liée à l’intégration de ses concepts et conventions dans le jargon et les habitudes des développeurs. Cette surcharge cognitive doit donc être maintenue au minimum, afin d’offrir une “surface lisse” pour le développement logiciel [Booc 03].

Dans des situations pratiques de développement logiciel (conception, implémentation, test, documentation, etc.), les concepts en jeux pour les développeurs sont les personnes (collègues, clients, etc.), ce que chacune de ces personnes fait (leur contribution aux différentes

2. Courtes séances non planifiées de travail, généralement en tête à tête, qui souvent précèdent ou suivent des séances de travail individuel, et servent à décomposer le travail ou fusionner les contributions. Voir la sous-section 2.2.2.

tâches), et les copies d'artéfacts (fichiers code source, documents, feuilles de calcul, etc.) qui matérialisent ces contributions.

Cependant, les concepts ci-dessus n'apparaissent pas dans la norme SPEM [OMG 07]. Nous proposons donc d'introduire dans SPEM ces concepts nouveaux, qui sont de granularité plus fine que les concepts habituels de rôle, de produit, et d'activité.

4.2.1.3 Rôle des outils existants de support à la collaboration

Les processus ne sont qu'un aspect de la problématique de la collaboration. D'autres questions comme la gestion de configuration, la communication, la gestion des défauts logiciels, etc., rentrent en ligne de compte. Ces aspects additionnels sont couverts par un ensemble d'offres logicielles.

La contribution des processus à la problématique de la collaboration n'est donc pas isolée, mais rejoint un écosystème logiciel existant, qui a déjà une certaine organisation. Une contribution basée sur les processus devra donc s'intégrer suffisamment à l'écosystème existant pour minimiser les problèmes d'adoption.

Cette question est traitée en détail dans le chapitre 3 qui porte sur l'intégration des outils de support à la collaboration. Elle a aussi guidé le choix des concepts structurant le méta-modèle `CMSPEM`.

4.2.2 Concepts manquants dans SPEM pour supporter la collaboration en génie logiciel

Cette section définit, à partir des particularités des processus collaboratifs relevées dans la section précédente, les concepts importants qui n'apparaissent pas dans le méta-modèle SPEM, sur lequel `CMSPEM` est basé.

4.2.2.1 Les participants au projet

Dans un modèle de processus SPEM, on peut retrouver une instance du concept `RoleUse`, nommée "Interaction Designer" par exemple. Lors de la mise en œuvre du projet, il revient au chef de projet d'y affecter autant de participants au projet que nécessaire. Cette approche limite les possibilités d'expression de certaines relations de travail.

En effet, le concept de `RoleUse` permet d'exprimer, en SPEM, des relations entre un ensemble de personnes (avec la même spécialisation), et d'autres concepts du méta-modèle comme `WorkProductUse`, `TaskUse`, ou même une autre instance de `RoleUse`. Cependant, l'impossibilité de distinguer systématiquement, au niveau d'un modèle de processus SPEM, deux participants à un projet (qui peuvent être affectés au même `RoleUse`), interdit de spécifier certaines relations de collaboration.

D'une part, la granularité du concept `RoleUse` interdit de spécifier des relations de travail, qui peuvent être temporaires (dans la mesure où elles sont spécifiques à une tâche), entre

deux participants affectés à la même instance de RoleUse. Considérons un processus de relecture collaborative d'un document au cours d'un projet. Trois personnes, liées au RoleUse "Reviewer", affectées à la relecture d'un document par exemple ne pourront pas représenter les relations de travail qui matérialisent le fait que deux d'entre elles vérifient les informations factuelles de deux sections différentes, pendant que la troisième fait une relecture globale sur la forme. Une solution basée sur SPEM nécessite la spécialisation de "Reviewer" (en "Technical Reviewer" et "Style Reviewer" par exemple), ce qui fait perdre la généricité supposée du concept de RoleUse, qui devient ainsi dépendant d'une tâche particulière, et ne décrit plus un métier, mais la tâche qui a été confiée à un participant particulier.

D'autre part, toute relation de collaboration spécifique à un participant qui partage son rôle avec un autre ne peut être représentée. Il est cependant courant que, par rapport à un processus, deux participants ne soient pas équivalents, parce que l'un d'entre eux a plus d'expérience par exemple. Dans le cadre de la relecture de l'exemple précédent, un participant peut être chargé de vérifier l'adéquation du document avec une norme ou des conventions stylistiques. Cette responsabilité supplémentaire est liée à la familiarité du participant avec la norme ou les conventions en question. Cependant, SPEM ne laisse d'autre choix que de créer un rôle spécialisé (qui ne décrit plus un métier, mais la responsabilité assumée dans une tâche particulière) ou de créer une ambigüité en affectant la responsabilité supplémentaire au RoleUse "Reviewer".

4.2.2.2 La contribution d'un participant à une tâche

Dans un modèle de processus SPEM, une instance de TaskUse correspond à une division de travail, qui pourra être affectée à plusieurs personnes lors de la mise en œuvre. Autrement dit, au niveau du modèle, les contributions de chacun des participants à la tâche ne sont pas différenciées, et l'on ne rend compte que du fait que leurs contributions participent à la réalisation de la tâche.

Cependant, les relations de collaboration de pair à pair sont basées sur la contribution de chaque personne à la tâche. Dans l'exemple de relecture de la section précédente par exemple, c'est parce l'un des participants fait de la relecture technique, et un autre vérifie la forme, que, par exemple, une certaine stratégie de révision est utilisée au lieu d'une autre.

De plus, les autres outils de développement traitant de tâche considèrent les unités de travail affectées à une personne particulière. Un outil de gestion de version identifie le code écrit par chaque participant, un outil de gestion de défauts peut retracer la contribution de chaque individu à la résolution d'un défaut (identification du défaut, vérification du défaut, résolution du défaut, vérification de la solution, etc.), et un outil de gestion de projet peut suivre le travail de chaque participant. L'absence d'une telle granularité dans le modèle de processus rend difficile le dialogue avec ces outils, à cause des pertes obligatoires d'information lors de l'importation d'informations.

4.2.2.3 Les copies de travail d'un participant

Dans un modèle de processus CMSPEM, le concept de WorkProductUse représente une version de référence d'un produit. Il peut s'agir par exemple de la version finale d'un document

de spécification, ou de la version finale d'un livrable (un document, du code, etc.).

Cependant, dans leur travail journalier, les participants à un projet travaillent sur des copies locales de ces documents officiels, et ces copies locales peuvent se trouver dans des états différents. D'ailleurs, une fonction essentielle des outils de gestion de version est de suivre l'évolution de ces copies locales, et de permettre de fusionner les contributions qu'elles contiennent.

Les manipulations faites sur ces copies locales, ainsi que les stratégies utilisées pour faire remonter les modifications vers la version de référence, sont des problématiques de collaboration. Cependant, la granularité des concepts dans SPEM ne permet pas de représenter ces copies, encore moins de spécifier les relations qui les lient.

4.2.2.4 Les relations de collaboration

La dynamique de la collaboration peut être représentée par les relations de travail entre les divers concepts qui entrent en jeu (les participants, les tâches, et les produits). Dans SPEM, des concepts comme `ProcessParameter` (reliant une activité à ses entrées / sorties), `ProcessResponsibilityAssignment` (reliant un produit à des rôles) et `ProcessPerformer` (reliant une activité à des rôles) servent à spécifier des relations entre des instances de concepts différents. D'autres concepts comme `WorkProductUseRelationship` (relation entre produits), et `WorkSequence` (succession temporelle de tâches) servent à mettre en relation deux instances d'un même concept. Enfin, des propriétés d'association comme `usedActivity`, `nestedBreakDownElement`, et `suppressedBreakDownElement` servent à décrire la décomposition hiérarchique des unités de travail (`Activity`).

Les nouveaux concepts introduits dans CMSPEM peuvent eux aussi être liés par des relations. Ces relations sont pourvues d'un mécanisme d'extension tiré de SPEM, ce qui permet, dans un modèle de processus, de les spécialiser afin d'affiner leur sémantique.

4.3 Le méta-modèle CMSPEM

4.3.1 Approche générale

Le méta-modèle CMSPEM (Collaborative Model-Based Software & Systems Process Engineering Metamodel [3]) est une extension de SPEM. Ainsi, tous les concepts et relations présents dans SPEM sont valides dans un modèle de processus CMSPEM. CMSPEM introduit le paquetage `CollaborationStructure`, qui contient les concepts nouvellement définis [4] (figure 4.1). Les règles de bonne formation OCL associées au méta-modèle sont incluses en annexe dans la section 10.2. La description ci-après suit le style utilisé dans la norme SPEM (sections distinctes pour la sémantique, les attributs, les propriétés d'association, et les contraintes), avec une section supplémentaire d'exemple pour chaque concept.

3. Les concepts de CMSPEM relatifs à l'ingénierie dirigée par les modèles ne sont pas décrits ici, vu qu'ils ne sont pas pertinents pour la problématique de la collaboration.

4. Le paquetage CMSPEM :`ProcessStructure` contient les concepts de CMSPEM relatifs à l'ingénierie dirigée par les modèles, et importe les paquetages SPEM::`Core` et SPEM::`ProcessStructure`.

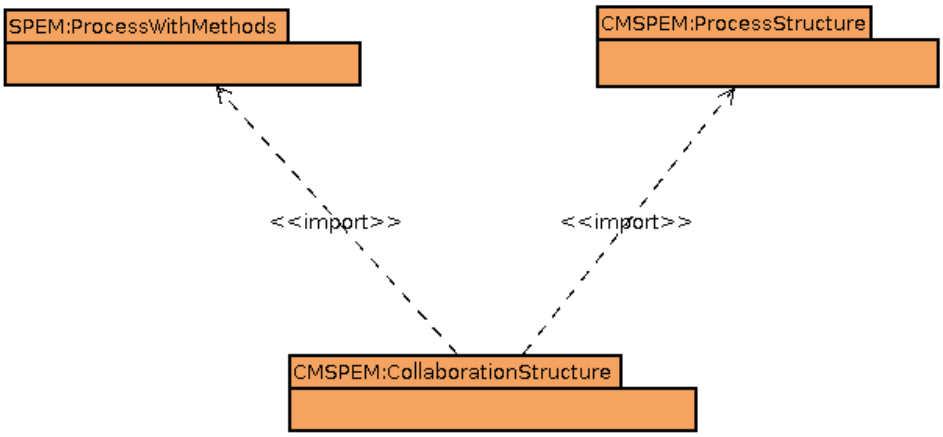

FIGURE 4.1 – Structure de paquetages du méta-modèle CMSPEM

La définition des concepts nouveaux dans CMSPEM fait appel – en les modifiant – aux concepts de RoleUse (de SPEM::ProcessStructure), de WorkProductUse (de SPEM::ProcessStructure) et de TaskUse (de SPEM::ProcessWithMethods). Les principaux concepts introduits sont Actor, ActorSpecificWork, et ActorSpecificArtifact. Les autres concepts sont des relations de collaboration entre les concepts nouvellement introduits.

4.3.2 Concepts repris de SPEM

4.3.2.1 RoleUse

RoleUse (voir annexe 10.1.1) est défini dans SPEM::ProcessStructure. Il est repris dans CMSPEM, avec les ajouts ci-après (figure 4.2).

Propriétés d'association

affectedActor : Actor Cette association lie le RoleUse aux Actors (voir section 4.3.3.1) qui jouent ce rôle dans le processus.

Contraintes

– Si un RoleUse est associé à plus d'un Actor, la propriété hasMultipleOccurences du RoleUse (définie dans SPEM::ProcessStructure::BreakDownElement) doit avoir la valeur true.

4.3.2.2 TaskUse

TaskUse (voir annexe 10.1.2) est défini dans SPEM::MethodContent. Il est repris dans CMSPEM avec les ajouts ci-après (figure 4.3).

Propriétés d'association

`contributingActorSpecificWork : ActorSpecificWork` Cette association lie le `TaskUse` avec les `ActorSpecificWorks` (voir section 4.3.3.2) qui contribuent à sa réalisation.

Contraintes

– Si un `TaskUse` est associé à plus d'un `ActorSpecificWork`, la propriété `hasMultipleOccurences` du `TaskUse` (définie dans `SPEM::ProcessStructure::BreakDownElement`) doit avoir la valeur `true`.

4.3.2.3 WorkProductUse

`WorkProductUse` (voir annexe 10.1.3) est défini dans `SPEM::ProcessStructure`. Il est repris dans CMSPEM, avec les ajouts ci-après (figure 4.4).

Propriétés d'association

`representingActorSpecificArtifact : ActorSpecificArtifact` Cette association lie un `WorkProductUse` avec les `ActorSpecificArtifacts` (voir section 4.3.3.3) qui en sont des copies.

Contraintes

– Si un `WorkProductUse` est associé à plus d'un `ActorSpecificArtifact`, la propriété `hasMultipleOccurences` du `TaskUse` (définie dans `SPEM::ProcessStructure::BreakDownElement`) doit avoir la valeur `true`.

4.3.3 Concepts de base de CMSPEM

Chaque concept nouveau dans CMSPEM est décrit ici avec sa sémantique, ses attributs, et ses propriétés d'association (figures 4.2, 4.3, et 4.4). Dans la présente sous-section et les suivantes, les exemples utilisés pour illustrer les nouveaux concepts se rapportent tous aux activités de test d'un projet fictif de développement logiciel collaboratif.

4.3.3.1 Actor

Super Classes

`ExtensibleElement` (`SPEM::Core`)

Sémantique

`Actor` désigne un participant humain, jouant un ou plusieurs rôles (figure 4.2). Il y a une correspondance un-à-un entre l'ensemble des instances de `Actor` dans un modèle de processus, et l'ensemble des participants à un projet.

Exemple

Le rôle "Testeur" désigne la qualification et la responsabilité qui consistent à définir un plan de test, à l'implémenter, et à suivre son exécution. Au cours du projet, plusieurs personnes physiques peuvent jouer ce rôle, suivant le nombre de fonctionnalités à tester et les délais. Chacun de ces participants est une instance du concept `Actor`.

Attributs

Propriétés d'association

`associatedRoleUse : RoleUse` Cette association représente les rôles que joue un participant particulier au projet.

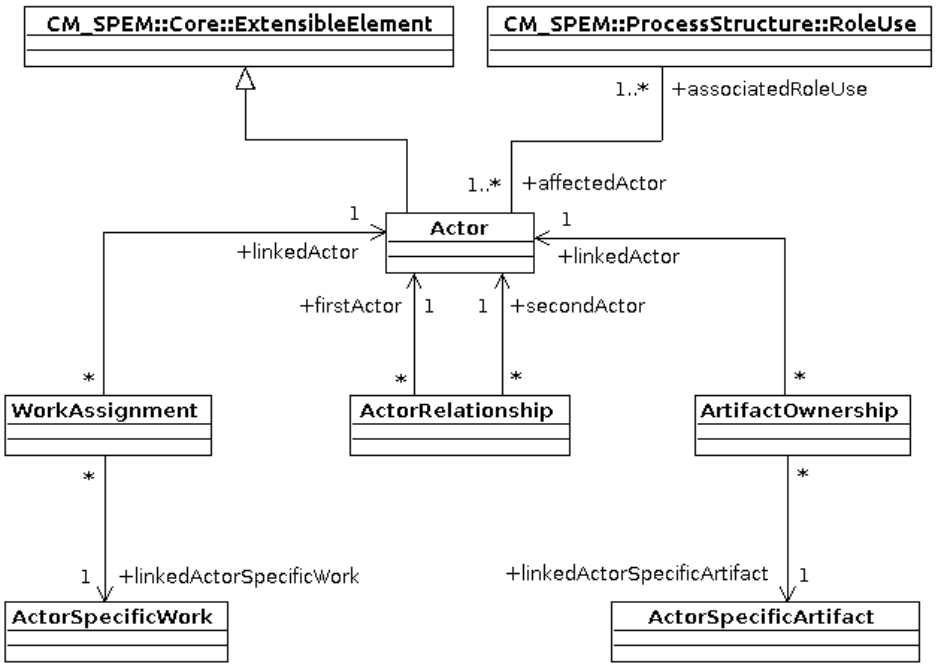

FIGURE 4.2 – Le concept d'`Actor` et ses relations en CMSPEM

4.3.3.2 ActorSpecificWork

Super Classes

`ExtensibleElement` (`SPEM::Core`)
`WorkBreakDownElement` (`SPEM::ProcessStructure`)

Sémantique

Un `ActorSpecificWork` représente la contribution d'un `Actor` à un `TaskUse`(figure 4.3). Par exemple, quand les étapes nécessaires à la réalisation d'un `TaskUse` ont besoin d'être répétées pour trois composants distincts, et que chacun de ces composants est affecté à un `Actor` distinct, trois `ActorSpecificWork` seront crées, pour représenter le travail sur chaque composant.

Exemple

Si un `TaskUse` "Réaliser les tests d'intégration" correspond à l'ensemble des tests d'intégration à faire dans un projet, selon la division du travail envisagée, un `ActorSpecificWork` peut être créé pour chaque ensemble de modules dont il faut tester l'intégration,

ou alors pour un sous-ensemble de modules qui communiquent. Chacun de ces ActorSpecificWork sera alors affecté à un participant jouant le rôle “Testeur” dans le projet. Ceci permet de suivre la progression du TaskUse “Réaliser les tests d’intégration” de manière plus fine, et de pouvoir identifier par exemple le participant responsable d’un délai, ou alors d’établir plus facilement des liens entre les résultats des tests et leur auteur.

Attributs

Propriétés d’association

associatedTaskUse : TaskUse Cette association matérialise le fait qu’un ActorSpecificWork est une contribution au TaskUse lié.

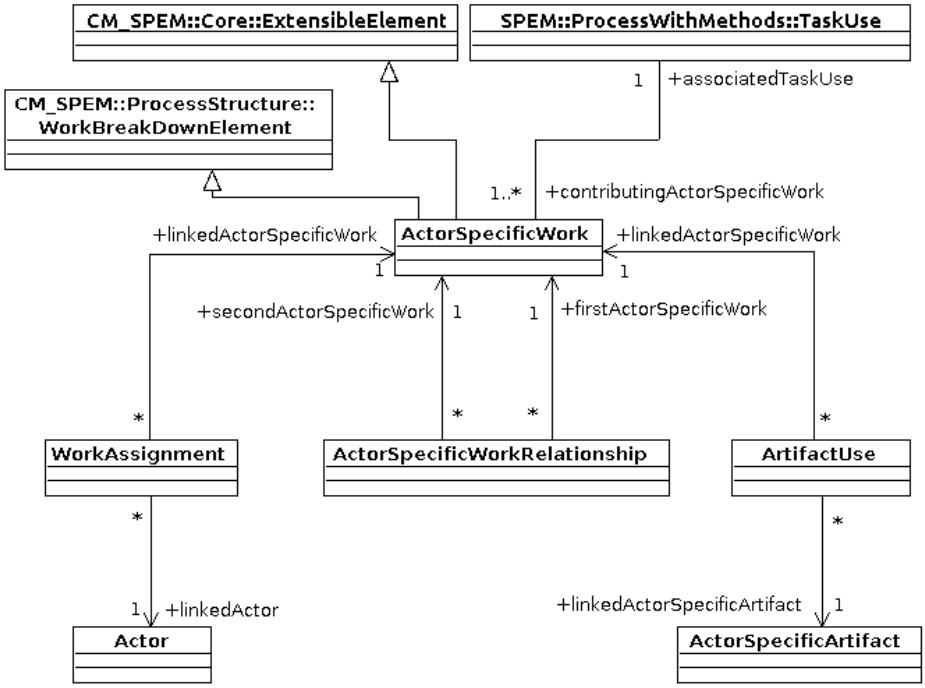

FIGURE 4.3 – Le concept d’ActorSpecificWork et ses relations en CMSPEM

4.3.3.3 ActorSpecificArtifact

Super Classes

ExtensibleElement (SPEM::Core)

Sémantique

ActorSpecificArtifact représente une copie de travail d’un WorkProductUse propre à un Actor (figure 4.4).

Exemple

Dans une équipe de testeurs, le module de reporting (présentation des résultats de test)

peut être modifié par n'importe quel participant. Une copie du code source de ce module est disponible dans l'espace de travail de chaque participant, qui peut le modifier, et partager ses modifications avec les autres (selon l'organisation de gestion de version utilisée). Chaque copie de travail est un ActorSpecificArtifact. Le WorkProductUse correspondant est le module tel que disponible dans le dépôt officiel du projet.

Les ActorSpecificArtifact représentant un même WorkProductUse ne sont donc pas dans le même état au même moment. Les conventions de synchronisation entre ces ActorSpecificArtifact et le WorkProductUse forment l'organisation de gestion de version du projet.

Notons qu'un ActorSpecificArtifact, tout comme un WorkProductUse, ne correspond pas forcément à un seul fichier physique, mais peut désigner tout un groupe de fichiers. Le choix de la granularité se base sur la pertinence du WorkProductUse en tant qu'unité d'information ayant son cycle de vie, ses dépendances, et ses règles de validation propres.

Attributs

isPartialCopy **:** Boolean Cet attribut, quand il a la valeur true indique que cet ActorSpecificArtifact est une copie partielle du WorkProductUse correspondant. Autrement dit, le contenu du ActorSpecificArtifact est un sous-ensemble de celui du WorkProductUse correspondant.

Le cas se présente quand un WorkProductUse peut être décomposé en des parties suffisamment indépendantes pour confier leur manipulation à des participants différents. Il peut s'agir par exemple de la décomposition d'un code source en modules, ou d'un document en chapitres. Chaque partie devient donc un ActorSpecificArtifact, avec isPartialCopy mis à true.

Propriétés d'association

associatedWorkProductUse **:** WorkProductUse Cette association lie un ActorSpecificArtifact au WorkProductUse dont il est une copie.

4.3.4 Relations de collaboration

Les relations de collaboration lient les concepts CMSPEM entre eux, et spécifient comment ils interagissent ou dépendent l'un de l'autre. Ces relations peuvent représenter des affectations de ressources ou des décisions d'organisation de travail.

4.3.4.1 WorkAssignment

Super Classes

ExtensibleElement (SPEM::Core)
BreakdowElement (SPEM::ProcessStructure)
ProcessPerformer (SPEM::ProcessStructure)

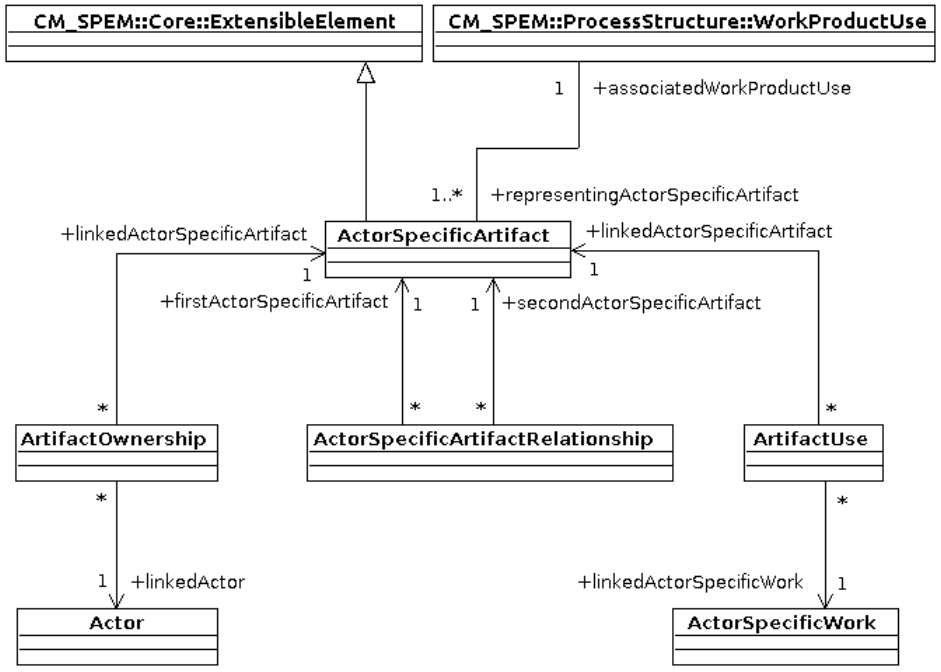

FIGURE 4.4 – Le concept d'ActorSpecificArtifact et ses relations en CMSPEM

Sémantique

WorkAssignment représente l'affectation d'un ActorSpecificWork à un Actor (figure 4.2). WorkAssignment peut être considéré comme une version raffinée [5] de ProcessPerformer de SPEM (dans la mesure où ActorSpecificWork et Actor sont des raffinements de TaskUse et RoleUse).

Un WorkAssignment est une précision utile lorsqu'une instance de TaskUse correspond à plus d'une instance de ActorSpecificWork. Même dans le cas où un WorkAssignment correspond à un seul ActorSpecificWork, le WorkAssignment spécifie quel Actor est affecté au ActorSpecificWork (autrement dit, au TaskUse), ce qui ne peut pas être précisé avec seulement des concepts SPEM comme WorkDefinitionPerformer (SPEM::MethodContent) ou ProcessPerformer (SPEM::ProcessStructure).

Exemple

Lorsqu'on affecte l'écriture des tests unitaires pour un ensemble de modules donné (ActorSpecificWork), à un participant donné (Actor), l'affectation est matérialisée par un WorkAssignment. La mise à jour des WorkAssignment en cours de projet représente donc des réaffectations de tâches, tout en conservant la structure de ces tâches. Ces réaffectations peuvent intervenir par exemple parce qu'un participant est temporairement indisponible.

Attributs

5. Ni UML, ni SPEM, ne définit de concept formel de raffinement. Le concept est utilisé ici pour indiquer un passage d'un concept de granularité relativement moins fine à un concept de granularité relativement plus fine. Par exemple, le passage de TaskUse à Actor est, dans ce sens, un raffinement, vu qu'un TaskUse correspond à un groupe d'Actors.

linkedActor : Actor Le participant auquel est affectée l'unité de travail.

linkedActorSpecificWork : ActorSpecificWork L'unité de travail affectée au participant.

Propriétés d'association

Contraintes

- Si un ActorSpecificWork et un Actor sont liés par un WorkAssignment, les TaskUse et RoleUse correspondants doivent être liés par un ProcessPerformer (défini dans SPEM::ProcessStructure).

4.3.4.2 ArtifactUse

Super Classes

ExtensibleElement (SPEM::Core)
BreakdowElement (SPEM::ProcessStructure)
ProcessParameter (SPEM::ProcessStructure)

Sémantique

ArtifactUse représente l'utilisation d'un ActorSpecificArtifact dans un ActorSpecificWork (figure 4.4). ArtifactUse peut être considéré comme une version raffinée de ProcessParameter de SPEM (dans la mesure où ActorSpecificArtifact et ActorSpecificWork sont des raffinements de WorkProductUse et TaskUse).

Il faut noter que la présence d'un ArtifactUse implique que l'Actor auquel l'ActorSpecificWork est affecté est lié à l'ActorSpecificArtifact par un ArtifactOwnership (l'inverse n'est pas toujours vrai).

ArtifactUse diffère des concepts de WorkDefinitionParameter (SPEM::Core), ProcessParameter (SPEM::ProcessStructure), et DefaultTaskDefinitionParameter (SPEM::MethodContent), en ceci qu'il est plus précis, étant relatif, non pas à un WorkProductUse, mais à une de ses représentations physiques (autrement dit, à un ActorSpecificArtifact).

Exemple

Un ArtifactUse décrit l'organisation locale du travail dans l'espace de travail d'un participant. Un participant (Actor) écrivant des tests d'intégration (ActorSpecificWork) pour un ensemble de modules (ActorSpecificArtifact) pourra utiliser des ArtifactUse pour définir le code de test, les interfaces des modules testés, et la documentation éventuelle des modules, comme utilisés dans cette tâche.

Une telle information peut être exploitée par une interface orientée-tâches [Goth 09] (MyLyn sous Eclipse par exemple), localement, pour reconfigurer les vues disponibles dans l'IDE lors du travail sur l'ActorSpecificWork correspondant.

Attributs

linkedActorSpecificArtifact : ActorSpecificArtifact La copie de travail manipulée

linkedActorSpecificWork : ActorSpecificWork L'unité de travail dans laquelle est manipulée la copie de travail.

Propriétés d'association

Contraintes

– Si un ActorSpecificWork et un ActorSpecificArtifact sont liés par un ArtifactUse, les TaskUse et WorkProductUse correspondants doivent être liés par un ProcessParameter (défini dans SPEM::ProcessStructure).

4.3.4.3 ArtifactOwnership

Super Classes

ExtensibleElement (SPEM::Core)
BreakdowElement (SPEM::ProcessStructure)
ProcessResponsibilityAssignment (SPEM::ProcessStructure)

Sémantique

ArtifactOwnership représente la possession d'un ActorSpecificArtifact par un Actor (figure 4.4). ArtifactOwnership peut être considéré comme une version raffinée de ProcessResponsibilityAssignment de SPEM (dans la mesure où ActorSpecificArtifact et Actor sont des raffinements de WorkProductUse et RoleUse).

Il faut noter qu'un ArtifactOwnership n'indique rien sur le ActorSpecificWork dans lequel un ActorSpecificArtifact est manipulé, dans la mesure où un Actor peut être responsable de plus d'un ActorSpecificWork.

ArtifactOwnership diffère de ProcessResponsibilityAssignment (SPEM::ProcessStructure) et DefaultResponsibilityAssignment (SPEM::MethodContent) par son niveau de granularité, étant utilisé pour lier un Actor (et non un RoleUse) à un ActorSpecificArtifact (et non un WorkProductUse).

Exemple

Si un testeur implémente des tests unitaires pour un module donné, la copie de travail du code de ces tests constitue un ActorSpecificArtifact. Dans le modèle de processus, un ArtifactOwnership sera alors utilisé pour indiquer que cet ActorSpecificArtifact est produit par le testeur (Actor) en question.

Attributs

linkedActorSpecificArtifact : ActorSpecificArtifact La copie de travail possédée.

linkedActor : Actor Le participant qui possède la copie de travail.

Propriétés d'association

Contraintes

– Si un Actor et un ActorSpecificArtifact sont liés par un ArtifactOwnership, les RoleUse et WorkProductUse correspondants doivent être liés par un ProcessResponsibilityAssignment (défini dans SPEM::ProcessStructure).

4.3.4.4 ActorSpecificArtifactRelationship

Super Classes

ExtensibleElement (SPEM::Core) WorkProductUseRelationship (SPEM::ProcessStructure)

Sémantique

ActorSpecificArtifactRelationship représente une relation entre deux ActorSpecificArtifact (figure 4.4). ActorSpecificArtifactRelationship peut être considéré comme une version raffinée de WorkProductUseRelationship (SPEM::ProcessStructure) ou WorkProductDefinitionRelationship (SPEM::MethodContent), dans la mesure où ActorSpecificArtifact est une précision de WorkProductUse.

La sémantique de la relation entre les deux ActorSpecificArtifact peut être précisée en associant à ActorSpecificArtifactRelationship une sous-classe de ActorSpecificArtifactRelationshipKind.

Un ActorSpecificArtifactRelationship doit être utilisé seulement pour des relations spécifiques à deux ActorSpecificArtifact. Si la relation est applicable à tout couple de ActorSpecificArtifact représentant le couple de WorkProductUse associé, il est préférable d'utiliser un WorkProductUseRelationship (SPEM::ProcessStructure) ou un WorkProductDefinitionRelationship (SPEM::MethodContent).

Exemple

Un ActorSpecificArtifactRelationship peut être utilisé pour décrire une organisation de gestion de version. En effet, si trois participants (Actor) modifient un même ensemble de tests unitaires, chacune des copies de travail est un ActorSpecificArtifact. Un participant peut être chargé de l'intégration des différentes modifications, avant l'envoi dans le dépôt officiel du projet. Dans ce cas, un ActorSpecificArtifactRelationship est utilisé pour spécifier qu'un ActorSpecificArtifact est la version de référence d'un autre ActorSpecificArtifact.

Attributs

firstActorSpecificArtifact : ActorSpecificArtifact Le premier ActorSpecificArtifact qui est impliqué dans la relation.

secondActorSpecificArtifact : ActorSpecificArtifact Le second ActorSpecificArtifact qui est impliqué dans la relation.

Propriétés d'association

4.3.4.5 ActorRelationship

Super Classes

ExtensibleElement (SPEM::Core)

Sémantique

ActorRelationship représente une relation entre deux Actor (figure 4.2).

La sémantique de la relation entre les deux Actor est précisée en associant une sous-classe de ActorRelationshipKind à l'instance de ActorRelationship.

Exemple

L'utilisation la plus simple d'un ActorRelationship est la spécification des relations de travail entre participants. Ainsi, un ActorRelationship peut définir le contact principal d'un testeur dans l'équipe d'assurance qualité. De même, un ensemble de ActorRelationship peut être utilisé pour définir le responsable de l'équipe d'assurance qualité.

Attributs

firstActor : Actor Le premier Actor impliqué dans la relation.

secondActor : Actor Le second Actor impliqué dans la relation.

Propriétés d'association

4.3.4.6 ActorSpecificWorkRelationship

Super Classes

ExtensibleElement (SPEM::Core)

Sémantique

ActorSpecificWorkRelationship représente une relation entre deux ActorSpecificWork (figure 4.3).

WorkSequence de SPEM peut être considéré comme un cas particulier de ActorSpecificWorkRelationship (qui lie deux instances de WorkBreakDownElement), qui ne concerne que les relations de précédence.

La sémantique de la relation entre les deux ActorSpecificWork est précisée en associant une sous-classe de ActorSpecificWorkRelationshipKind à l'instance de ActorSpecificWorkRelationship.

Exemple

Un ActorSpecificWorkRelationship peut être utilisé pour définir le séquencement entre deux ActorSpecificWork, tout comme un WorkSequence est utilisé pour spécifier le séquencement entre deux TaskUse. Dans ce cas, le ActorSpecificWorkRelationship peut spécifier l'ordre dans lequel les différents ActorSpecificWork contribuant à un même TaskUse sont réalisés.

Une autre utilisation possible d'un ActorSpecificWorkRelationship est la mise en relation d'une tâche et de la tâche de validation correspondante. Par exemple, un ActorSpecificWork "Valider la version pour le passage en production", effectué par un membre de l'équipe d'assurance qualité, sera lié aux différents ActorSpecificWork qui matérialisent la création des différents tests d'intégration. L'information donnée par le ActorSpecificWorkRelationship peut alors être exploitée pour lancer des actions automatiques sur la base de la décision de l'ingénieur qualité (marquer les tâches de création de tests d'intégration comme "à revoir" par exemple).

Attributs

firstActorSpecificWork : ActorSpecificWork Le premier ActorSpecificWork impliqué dans la relation.

secondActorSpecificWork : ActorSpecificWork Le second ActorSpecificWork impliqué dans la relation.

Propriétés d'association

4.3.5 Mécanisme d'extension

Dans SPEM, "SPEM 2.0 Base Plug-in" est un "Method Plug-in" prédéfini, qui fournit des instances de concepts SPEM couramment utilisés en ingénierie logicielle. De même, dans CMSPEM, nous avons défini le "CMSPEM Base Plug-in", qui fournit des instances couramment utilisées pour la modélisation de la collaboration, et qui servent de point de départ pour la modélisation de processus collaboratifs.

"CMSPEM Base Plug-in" définit trois sous-classes (ActorRelationshipKind, ActorSpecificArtifactRelationshipKind, et ActorSpecificWorkRelationshipKind) de SPEM::Core::Kind, qui servent à qualifier les relations définies dans CMSPEM (figure 4.5).

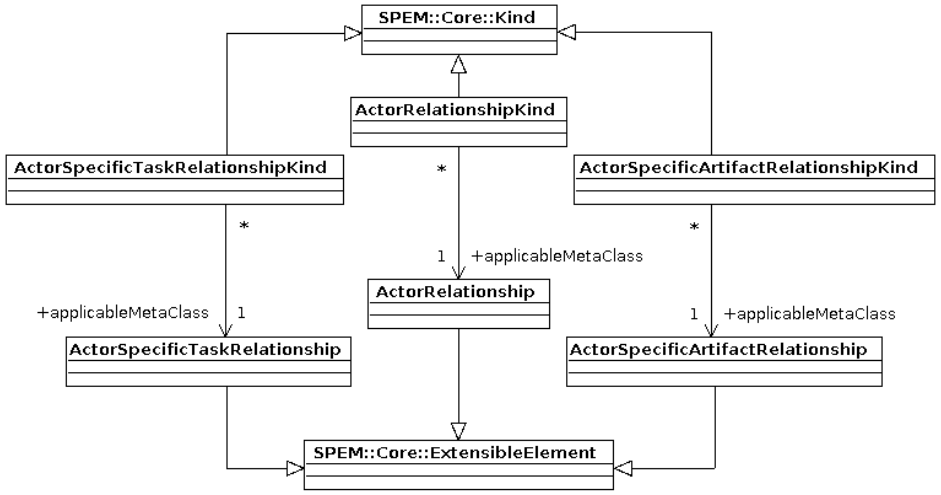

FIGURE 4.5 – Le plugin de base du méta-modèle CMSPEM

4.3.5.1 ActorRelationshipKind

ActorRelationshipKind est une sous-classe de SPEM::Core::Kind, dont la propriété d'association applicableMetaClass la relie à un ActorRelationship (figure 4.5). Ses instances sont utilisées pour qualifier des relations entre deux instances de la métaclasse Actor (voir figure 4.2).

Les instances suivantes de ActorRelationshipKind sont disponibles par défaut :

DefaultPushDestination
Spécifie que, par défaut, les contributions du participant désigné par firstActor sont envoyées au participant désigné par secondActor pour une instruction de publication de contribution dans un outil de gestion de version ("push" dans Git par exemple).

DefaultPullSource
Spécifie que, par défaut, le participant désigné par firstActor met à jour son espace de travail en récupérant les changements depuis l'espace de travail du participant désigné par secondActor (l'opération correspondante étant un "pull" dans Git par exemple).

4.3.5.2 ActorSpecificWorkRelationshipKind

ActorSpecificWorkRelationshipKind est une sous-classe de SPEM::Core::Kind, dont la propriété d'association applicableMetaClass la relie à un ActorSpecificWork (figure 4.5). Ses instances sont utilisées pour qualifier des relations entre deux instances de la métaclasse ActorSpecificWork (voir figure 4.3).

Les instances suivantes de ActorSpecificWorkRelationshipKind sont disponibles par défaut :

Impacts
Spécifie que tout changement dans l'unité de travail désignée par firstActorSpecificWork a de potentielles répercussions sur l'unité de travail désignée par secondActorSpecificWork.

4.3.5.3 ActorSpecificArtifactRelationshipKind

ActorSpecificArtifactRelationshipKind est une sous-classe de SPEM::Core::Kind, dont la propriété d'association applicableMetaClass la relie à ActorSpecificArtifact (figure 4.5). Ses instances sont utilisées pour qualifier des relations entre deux instances de la métaclasse ActorSpecificArtifact (voir figure 4.4).

Les instances suivantes de ActorSpecificArtifactRelationshipKind sont disponibles par défaut :

Supersedes
Spécifie que la copie de travail désignée par firstActorSpecificArtifact est plus récente, ou plus généralement, est recommandée par rapport à celle désignée par secondActorSpecificArtifact. Ce type de relation peut être utilisé par le concepteur d'une API par exemple pour marquer le remplacement d'une classe par une autre.

4.4 Conclusion

Ce chapitre a décrit les aspects statiques du méta-modèle CMSPEM, en motivant sa définition par les manques de SPEM pour la description de la collaboration, puis en conceptualisant et en décrivant des solutions à ces manques, sous forme de nouveaux concepts et relations.

Les concepts introduits dans CMSPEM ont été motivés par leur capacité à représenter des informations additionnelles qui sont nécessaires pour une prise en compte effective de la collaboration. Ces nouveaux concepts, ainsi que les relations associées, forment avec SPEM, le méta-modèle CMSPEM.

CMSPEM permet de représenter les données décrivant un processus collaboratif. L'évolution de ces données dans le temps est modélisée par des évènements, décrits au chapitre 5. Les données et les évènements de processus forment l'information de processus, qui est gérée par le gestionnaire de préoccupation dont l'implémentation fera l'objet du chapitre 6. Ce chapitre d'implémentation reviendra sur les données de processus, pour montrer comment le concept de "deep-link", introduit au chapitre 3 peut s'y appliquer.

Chapitre 5

Formalisation des aspects dynamiques de la collaboration

5.1 Introduction

Le chapitre 4 a présenté le méta-modèle CMSPEM, une extension de SPEM adaptée à la description des processus collaboratifs. Dans le présent chapitre, nous nous intéressons à la manière dont les modèles de processus CMPSEM évoluent dans le temps, et comment cette évolution peut être formellement représentée [Kedj 12c].

La norme SPEM [OMG 07] ne définit pas de modèle comportemental[1], mais propose un mécanisme pour faire le lien entre un modèle structurel SPEM, et un modèle comportemental *externe* comme BPMN [OMG 09].

Des options possibles de modèle comportemental externe sont les diagrammes d'activité UML, ou les machines à état UML [OMG 05]. Cependant, ces deux formalismes sont prescriptifs, par opposition à l'approche descriptive de CMSPEM. Par exemple, les concepts de JoinNode, MergeNode, et DecisionNode des diagrammes d'activité UML spécifient des comportements ou alternatives autorisées. De même, les machines à états spécifient les transitions d'états autorisés, en réponse à des événements. Cependant, le but de CMSPEM est de relâcher les contraintes sur les comportements dans un cadre collaboratifs, tout en donnant le moyen d'annoncer les changements de ces comportements et d'y réagir.

Il existe dans la littérature diverses propositions d'extension de SPEM avec une formalisation comportementale, dont xSPEM [Bend 07] qui utilise les réseaux de Petri et eSPEM [Elln 10] qui propose une manière alternative d'interfacer SPEM avec les modèles comportementaux UML. Cependant, ces propositions ont pour objectif principal de rendre SPEM exécutable, et requièrent bien plus de planification que nécessaire pour répondre à notre cas d'utilisation de prise en compte d'événements de processus.

Le modèle de base utilisé pour représenter l'évolution des processus CMSPEM est le modèle

1. Voir la section 10 "Process Behavior" de la norme SPEM 2.0, en page 69.

événementiel de la publication et de l'abonnement. Chaque évolution possible d'un modèle CMSPEM correspond à un événement bien défini. Ce chapitre décrit le mécanisme de prise en compte de tels événements, et la définition de réactions grâce à un mécanisme de souscription.

L'évolution des modèles de processus CMSPEM est déterminée par, et perçue comme, une série d'événements, ordonnés dans le temps. Notre approche de modélisation de la collaboration est donc temporelle et performative[Polt 09]. Autrement dit, on décrit le comportement observé, plutôt que celui prévu.

Dans le cadre du modèle conceptuel présenté au chapitre 3, et qui inclus les événements de préoccupation (section 3.3.5), nous identifions un catalogue d'événements décrivant la préoccupation de "gestion de processus de développement".

Le chapitre est structuré comme suit. La section 5.2 montre l'utilisation des événements dans le support du développement collaboratif, et la section 5.3 traite de la pertinence des événements pour modéliser les processus collaboratifs. La section 5.4 formalise les notions d'événement, de source d'événement, de gestionnaire d'événement, et de souscription. Le mécanisme de réaction aux événements est présenté dans la section 5.5. Enfin, la section 5.6 décrit le catalogue d'événements disponibles dans CMSPEM ainsi que leur utilité pour le support du développement collaboratif. Le chapitre 6 décrit l'application de la conceptualisation faite dans le présent chapitre au serveur de processus CMSPEM.

5.2 Les événements en développement logiciel

Dans chaque préoccupation de développement logiciel peuvent se produire un certain nombre d'occurrences significatives ou événements. Ces événements, rendent compte de l'évolution de la préoccupation, et peuvent potentiellement être exploitées par des outils de développement logiciel.

Les systèmes de gestion de versions, de gestion de défauts logiciels, d'intégration continue, de mailing-list, de salons de discussion, etc., peuvent tous générer des événements.

Dans un système de gestion de versions, un événement peut correspondre à un commit (nouvelle révision), la création d'une branche de développement, la fusion de deux branches, l'envoi ou la réception d'une série de révisions dans un dépôt, etc.

Dans un système de gestion de défauts logiciels, le cycle de vie d'un rapport de bug peut être naturellement décrit comme une suite d'événements, qui correspondent aux transitions entre les différents états du rapport. Des exemples sont la création ou la vérification de rapport de bug, l'assignation d'un rapport de bug à un développeur, le marquage d'un rapport de bug en tant que duplication d'un autre rapport, la proposition ou la confirmation d'une solution, la fermeture ou la réouverture de rapport de bug, etc. De plus, l'ajout de commentaire sur un rapport de bug peut aussi être considéré comme un événement.

Les listes de discussion peuvent aussi générer des événements, comme l'envoi d'un nouveau message dans un thread [2] existant, ou la création d'un nouveau thread. Les mailing-lists

2. Un thread est une suite de messages de mailing-list portant sur le même sujet, identifié en général par le titre du premier message.

ont aussi la particularité d'être utilisées pour annoncer des événements relatifs à d'autres préoccupations de développement logiciel. Par exemple, dans les projets comme le Kernel Linux qui utilisent des mailing-lists pour l'échange de patchs [3], un message peut correspondre à une nouvelle série de patchs, ou à un nouveau patch dans une série existante. De même, sur une mailing-list consacrée à l'intégration continue, un message peut correspondre à une réussite ou un échec de compilation ou de déploiement.

La disponibilité des événements rend possible l'intégration entre différents outils de support à la collaboration, grâce au mécanisme de réaction aux événements. La formalisation (faite dans ce chapitre) des événements de processus et leur exposition aux autres outils (chapitre 6) permet à tous les outils de support à la collaboration d'exploiter l'information de processus. Une telle intégration rend la modélisation et la mise en ouvre de processus véritablement utile pour le travail collaboratif.

5.3 Les événements dans les modèles de processus

Il est nécessaire de prendre en compte certaines informations qui deviennent disponibles dans un projet au cours du développement. Un événement peut être considéré comme un moyen de signaler une information nouvelle. Sans la prise en compte de l'information nouvelle, les modèles de processus deviennent très vite des photographies du passé : remarquables pour rendre compte du monde tel qu'il était, ou tel qu'il était imaginé, mais inutiles pour rendre compte de son évolution réelle, présente.

Pour prendre en compte l'évolution de la réalité, la vision du *"modèle de processus en tant que plan d'exécution"* doit être reléguée au second plan, pour que prenne le dessus la vision du *"modèle de processus en tant qu'une abstraction de la succession de faits dans la réalité"*. Ces faits sont des changements, qui peuvent avoir été prévus ou être inattendus. Dans notre approche, la modélisation comportementale est essentiellement de la modélisation du changement. Les événements sont un formalisme idéal pour rendre compte du changement.

Pourquoi modéliser le changement ? Dans toute activité humaine, le modèle mental que l'individu a du produit en cours d'élaboration guide ses actions, et est donc important pour le succès de la tâche. C'est grâce à ce modèle mental qu'il peut anticiper les conséquences de ses actions, ou comprendre l'état actuel du produit, afin de déterminer les actions à entreprendre pour le mener à l'état final désiré. En situation de collaboration, ce modèle mental doit prendre en compte les actions des autres participants au projet, ce qui complique sa construction. Conceptualiser le changement et sa transmission à chaque participant est donc nécessaire pour aider un développeur seul derrière son écran à prendre en compte ce qui se passe ailleurs dans le projet.

Dans les formalismes de processus existants comme SPEM, la modélisation du comportement est prévue, même si elle n'est pas toujours incluse par défaut. SPEM par exemple prévoit de greffer un formalisme de modélisation comportemental au choix. Cependant, une faiblesse essentielle de ces approches est qu'elles essayent de décrire le comportement avec des concepts

3. Un patch est un fichier contenant la différence entre deux versions du code source d'un système logiciel.

très abstraits. Les changements qu'ils décrivent sont beaucoup trop généraux pour être d'une utilité pratique.

En effet, lorsque des modifications interviennent dans un modèle de processus, cela se passe dans l'immédiat au niveau le plus bas : une personne tombe malade, il faut rajouter telle classe utilitaire, etc. Si l'on ne permet pas d'exprimer (simplement) ces modifications de tous les jours, les participants sont obligés de traduire *dans leur tête* ces changements en des impacts sur des concepts plus abstraits comme rôle, produit, etc. Il est normal qu'ils déterminent régulièrement qu'il n'y a pas de changement significatif, puisque le changement se remarque de moins en moins en montant en abstraction. Par exemple, la fin d'une tâche spécifique à un acteur est un changement qui n'a pas d'impact automatique et prévisible sur la tâche parente. On ne peut donc représenter adéquatement un tel changement rien qu'avec la notion de tâche. Cette situation encourage à prendre en compte les changements seulement quand ils sont significatifs, ce qui peut impliquer malheureusement qu'on les prend en compte trop tard. Cette erreur est essentiellement due à l'inadéquation du langage disponible. Ceci réduit considérablement l'assistance qui peut être fournie aux participants.

Nous avons indiqué plus haut que la collaboration complique le modèle mental que l'individu a d'un produit en cours d'élaboration, parce qu'il n'est pas le seul à le manipuler. Les événements permettent de saisir les occurrences significatives qui rendent compte de la participation des entités autres que soi dans le projet. C'est une vue simplifiée – donc une abstraction – du monde extérieur : à défaut d'observer son évolution de près, dans les détails, on garde un œil sur les événements majeurs qui rendent compte de cette évolution. Par exemple, énormément de choses peuvent se produire dans l'existence d'un autre participant au projet. Parmi ces événements, certains sont significatifs, autrement dit influencent le projet. Il s'agit par exemple du fait qu'il ait commencé une tâche qu'on lui a assigné, qu'il l'ait abandonnée temporairement pour répondre à un imprévu, qu'il soit temporairement indisponible (malade pas exemple), ou même qu'il ait besoin d'une information particulière pour progresser dans sa tâche.

Les événements permettent, d'une part, de rendre compte des changements qui interviennent sur un élément participant au projet, et qui affectent potentiellement le travail des autres. D'autre part, le mécanisme de souscription associé aux événements permet à n'importe quel participant de ne surveiller que les types d'événements qui sont pertinents pour le modèle mental qu'il a du projet. Un participant n'est par exemple pas concerné par une indisponibilité d'un membre d'une équipe qui travaille sur une tache indépendante de la sienne, mais le chef de projet l'est.

5.4 Conceptualisation des événements

Dans CMSPEM, le comportement des modèles de processus lors de la mise en œuvre est représenté par une approche de type "action-réaction" : un événement se produit, et les abonnés sont avertis afin qu'ils puissent réagir. Les éléments de modèle qui peuvent produire des événements sont appelés "sources d'événement", et ceux qui écoutent des événements afin d'y réagir sont appelés "gestionnaires d'événement". Les gestionnaires d'événements sont avertis seulement des événements auxquels ils se sont abonnés, et cet abonnement est représenté par

le concept de “souscription”.

Les événements indiquent des changements relatifs au processus, c’est-à-dire des modifications faites sur le modèle de processus. Par exemple, une tache spécifique à un acteur (ActorSpecificTask) peut être ajoutée au modèle, de même qu’une relation créée entre deux participants (ActorRelationship).

Les événements servent à signaler des changements. Une fois un changement survenu, et signalé par un événement, des actions peuvent être exécutées en réaction. On peut, par exemple, ajouter un nouveau participant à un projet pour compenser l’absence d’un autre, ou notifier un participant d’une tâche qui vient de lui être assignée.

Les entités capables d’exécuter des actions, en réaction à des événements, sont nommées “EventHandler” ; celles qui peuvent générer des événements sont nommées “EventSource”. Les événements eux-mêmes sont regroupés en classes d’événements, qui permettent de faire abstraction des caractéristiques concrètes d’un événement particulier.

Pour qu’un EventHandler puisse exécuter des actions en réaction à des événements, il faut au préalable qu’une souscription aie été faite au type d’événement concerné. Une souscription est un triplet (E, S, H), où E est une classe d’événements, S une source d’événements (EventSource), et H un traiteur d’événements (EventHandler). Une telle souscription, nommée EventSubscription, spécifie que le traiteur H doit être activé lorsqu’un événement de classe E est généré par la source S.

Le comportement d’un modèle de processus CMSPEM est complètement spécifié par l’ensemble des classes d’événements, des sources d’événements, des gestionnaires d’événements, et des souscriptions. Cette spécification ne suffit cependant pas pour déterminer tous les états futurs du modèle de processus. Pour ce faire, il faut coupler à cette spécification, la série effective d’événements qui se produisent lors du déroulement du projet collaboratif.

5.4.1 Event

Un Event est une occurrence remarquable, qui peut provoquer une réaction, si un EventSubscription (section 5.4.4) a été défini. Les Events ont des paramètres, et des valeurs de paramètre qui décrivent l’événement. Les événements sont produits par des EventSources (section 5.4.2), et reçu par des EventHandlers (section 5.4.3), du moment qu’un EventSubscription approprié existe avant l’occurrence de l’événement.

La fin d’un `ActorSpecificWork` par exemple constitue un événement. La source de l’événement est l’`ActorSpecificWork` qui se termine. Un paramètre associé est la date de fin de l’événement. Ainsi, lors d’une prise en charge asynchrone de l’événement, la date de fin effective, distincte de la date de traitement de l’événement, reste disponible.

5.4.2 EventSource

Un EventSource est un élément de modèle qui peut générer des événements. Un EventSource déclenche un événement pour informer les gestionnaires d’un changement intervenu sur son état interne.

La plupart des éléments de modèle CMSPEM peuvent générer des événements (voir le catalogue d'événements, groupés par élément, dans la section 5.6). Par exemple, un `Actor` est une source d'événement, pouvant générer, entre autres, des événements ActorRemoval (l'acteur correspondant a quitté le projet) et ActorUnavailable (l'acteur correspondant est temporairement indisponible).

5.4.3 EventHandler

Un EventHandler est un élément de modèle qui peut réagir à des événements. Un EventHandler peut recevoir des notifications d'événement, et y réagir. Pour pouvoir recevoir des événements, l'EventHandler doit avoir souscrit à l'événement avant son occurrence, via un EventSubscription. Seuls les événements du type correspondant à la souscription sont envoyés à l'EventHandler.

Concrètement, un EventHandler représente un outil tiers, intéressé par des événements de processus. Par exemple, un tableau de bord dynamique de projet sera intéressé, entre autres, par les événements de début et de fin d'`ActorSpecificWork`. Le tableau de bord est donc un EventHandler, pour lequel un ou plusieurs EventSubscription (voir section 5.4.4 pour la composition de la souscription) seront définis, pour spécifier les `ActorSpecificWork` que le tableau de bord veut surveiller. Un notification, dont la nature précise est dépendante de l'implémentation [4], sera envoyée au tableau de bord chaque fois qu'un des événements concernés se produit.

5.4.4 EventSubscription

Un EventSubscription est la conceptualisation de la souscription faite, par un EventHandler, à un type d'événement, sur un EventSource. Les événements du type spécifié, qui se produisent après la souscription, sont notifiés au EventHandler, si, et seulement si, ils ont été générés par le EventSource spécifié. Pour chaque combinaison de type d'événement et de source d'événement, plusieurs souscriptions d'événement peuvent être définies.

Dans l'exemple de la section 5.4.3, une souscription définira le tableau de bord comme EventHandler, ActorSpecificWorkStart (voir le catalogue d'événements dans la section 5.6) comme Event, et un `ActorSpecificWork` particulier comme EventSource. Dans ce cas particulier, le tableau de bord est probablement intéressé par les événements ActorSpecificWorkStart sur tous les `ActorSpecificWork`, présents comme futurs. Il n'est donc pas pratique de définir un EventSubscription pour chaque `ActorSpecificWork`, et le concept de hiérarchie de parenté d'éléments sera exploité pour ne définir qu'une seule souscription, utilisant l'élément racine du modèle comme EventSource (voir section 5.5.1).

5.5 Mécanisme de réaction aux événements

Le mécanisme de réaction aux événements est le processus qui commence à la génération d'un événement, et se termine quand l'exécution de tous les gestionnaires d'événement est

4. Dans l'implémentation décrite au chapitre 6 par exemple, cette notification est faite avec une requête HTTP de type POST.

terminée. Afin d'expliquer ce mécanisme, nous introduisons les concepts de hiérarchie de parenté d'éléments et de remontée d'événement.

5.5.1 Hiérarchie de parenté d'éléments

Un lien de parenté est défini entre éléments d'un modèle CMSPEM, selon les règles suivantes :

- Chaque élément de modèle a au plus un élément "parent".
- Les éléments qui n'ont pas de parent sont appelés "éléments de premier niveau".
- Si un élément A est le parent d'un élément B, B est appelé "élément fils" de A.
- Un élément A est appelé "ancêtre" d'un élément B si, et seulement si, A est l'élément parent de B, ou s'il existe un élément C, tel que C est le parent de B et A est ancêtre de C.
- Chaque modèle a un unique "élément racine", qui est élément "ancêtre" de tout autre élément [5].

Les règles de parenté dans CMSPEM sont comme suit :

- Tout `RoleUse` est parent des `Actor` associés.
- Tout `WorkProductUse` est parent des `ActorSpecificArtifact` associés.
- Tout `TaskUse` est parent des `ActorSpecificWork` associés.
- Toute relation `CMSPEM` (`ActorRelationship`, `ActorSpecificArtifactRelationship`, `ActorSpecificWorkRelationship`, `WorkAssignment`, `ArtifactOwnership`, ou `ArtifactUse`) est élément fils de chacun des concepts qu'il lie.

5.5.2 Remontée d'événement

Un événement peut être traité, non seulement par les gestionnaires définis sur la source de l'événement, mais aussi par des gestionnaires définis sur n'importe quel ancêtre de la source de l'événement. En effet, un événement généré par un élément est aussi généré par tous ces ancêtres. On dit que l'événement "remonte" vers le haut. Ce mécanisme est inspiré du modèle DOM (Document Object Model) utilisé dans les navigateurs internet.

La remontée d'événement est nécessaire pour rendre compte de la nature évolutive des modèles de processus logiciel, qui peuvent être enrichis à tout moment par de nouveaux éléments de modèle. Pour pouvoir écouter un événement, il faut spécifier l'élément qui génère cet événement. Il est donc impossible, à priori, de spécifier qu'un écouteur est intéressé par les événements qui peuvent être émis par un élément de modèle qui n'est pas présent dans le modèle lors de la souscription. Par exemple, un gestionnaire peut demander à être notifié chaque fois qu'un attribut est modifié sur un ActorSpecificWork qui est *contenu* dans un

5. Cet élément peut être défini librement par chaque implémentation. Dans le prototype décrit dans le chapitre 6, cet élément est nommé "Model", et a comme type `ecore :EClass`.

certain TaskUse. Sans la remontée d'événement, ceci peut être fait seulement pour les ActorSpecificWorks déjà présents dans le modèle. Avec la remontée d'événement, le gestionnaire d'événement peut tout simplement écouter l'événement sur le TaskUse. Tous les événements [6] générés par un ActorSpecificWork présent ou futur, ayant le TaskUse comment parent, atteindrons le TaskUse.

Quand un événement remonte vers les ancêtres de l'élément source, un ancêtre sur lequel aucun gestionnaire n'a été défini ne réagira tout simplement pas.

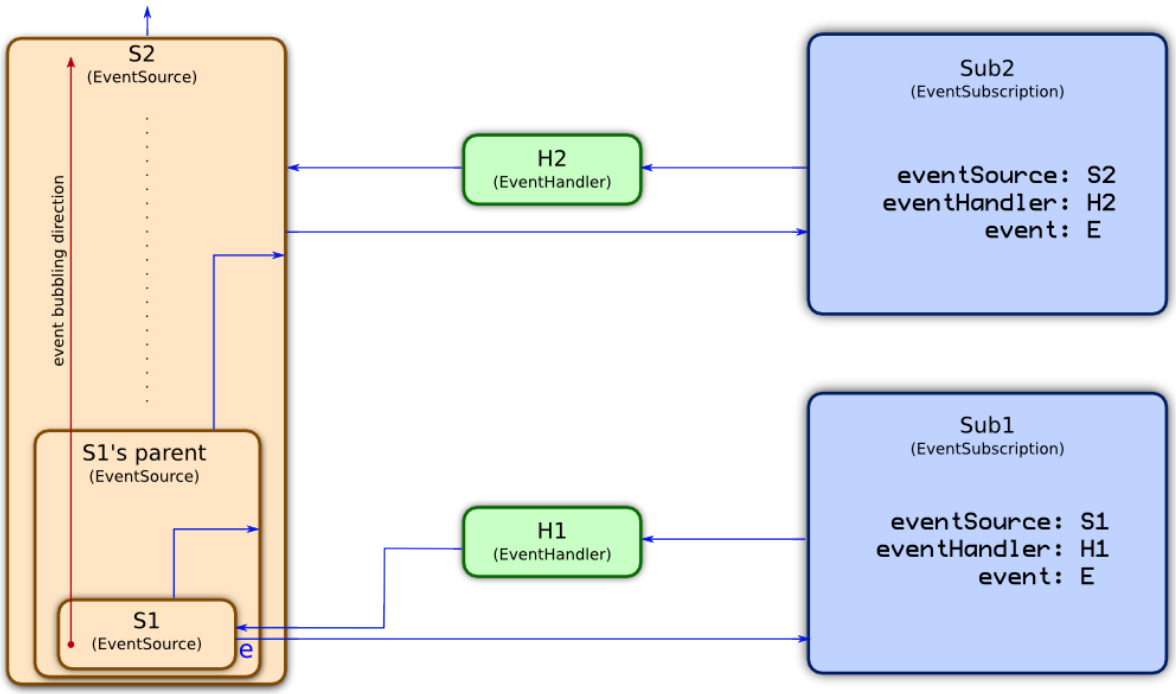

FIGURE 5.1 – Exemple de prise en compte d'événement

La figure 5.1 montre un exemple des différentes étapes de prise en compte d'un événement. E est un type d'événement, et e un événement spécifique de type E. H1 et H2 sont des gestionnaires d'événement. S1 et S2 sont des sources d'événement, et S2 est l'ancêtre de S1. Sub1 et Sub2 sont des souscriptions d'événement. Les étapes, depuis la souscription jusqu'à la prise en compte de l'événement e sont les suivantes :

1. Sub1 est défini comme une souscription sur la source S1, pour les événements de type E, et H1 est spécifié comme gestionnaire. De même, Sub2 est défini comme une souscription sur la source S2, pour les événements de type E, et H2 est spécifié comme gestionnaire. L'ordre dans lequel les souscriptions Sub1 et Sub2 sont définies n'a aucune importance. La seule contrainte est que les souscriptions Sub1 et Sub2 soient définies avant que l'événement e ne se produise.
2. L'événement e (de type E) se produit, et S1 est sa source.

6. Du type approprié, spécifié lors de la souscription.

3. Les souscriptions d'événement ayant S1 comme source sont consultées. Pour chacune de ces souscriptions, dont le type d'événement associé est E, le gestionnaire associé est invoqué, avec les paramètres propres à l'événement e. Dans le cas de la figure 5.1, H1 est le seul gestionnaire correspondant. Si plusieurs gestionnaires étaient définis, ils seraient invoqués l'un après l'autre, dans l'ordre de définition.
4. Tous les gestionnaires invoqués dans l'étape précédente terminent leur exécution. Dans ce cas, l'exécution de H1 se termine.
5. Si le gestionnaire H1 a mis fin à la remontée de l'événement e, la prise en compte de l'événement s'arrête ici. Sinon, l'événement e remonte.
6. L'événement e est envoyé à l'élément parent de S1. La procédure de l'étape 3 est répétée pour toutes les souscriptions dont la source est le parent de S1. Quand la prise en compte est terminée sur le parent de S1, l'événement remonte de nouveau.
7. L'événement finit par atteindre l'ancêtre S2 de S1. Les souscriptions d'événement dont la source est S2 sont consultées. Pour chacune de ces souscriptions dont le type d'événement associé est E, le gestionnaire associé (H2 dans ce cas) est invoqué. Chaque gestionnaire est invoqué avec les paramètres de l'événement e.

Les événements peuvent avoir des paramètres arbitraires, selon leur type. Cependant, pour les besoins de la remonté d'événement et la réaction aux événements, deux paramètres standards sont toujours transmis aux gestionnaires :

source
: L'élément (EventSource) sur lequel le gestionnaire a été défini. Quand l'événement remonte, ce paramètre est continuellement mis à jour. Sa valeur correspond toujours à l'élément sur lequel le gestionnaire en cours d'exécution a été défini.

originalSource
: L'élément sur lequel l'événement s'est produit à l'origine, avant toute remontée. Pour chaque événement, la valeur de ce paramètre reste la même, même quand l'événement est traité par des gestionnaires définis sur les ancêtres, lors de la remontée d'événement.

Quand un événement est généré initialement par une source, source et originalSource sont identiques. Ils diffèrent seulement lors de la remontée d'événement, quand l'événement est traité par des gestionnaires définis sur des ancêtres de sa source originale.

Dans l'exemple de tableau de bord introduit dans les sections 5.4.3 et 5.4.4, la remontée d'événement entre en jeux. En effet, une seule souscription S à l'événement ActorSpecificWorkStart est définie, et l'élément racine du modèle est spécifiée comme source. Le tableau de bord TB est défini comme EventHandler. A chaque début d'`ActorSpecificWork` (événement e de type ActorSpecificWorkStart, associé à un `ActorSpecificWork` ASW), une souscription spécifiant ASW comme source est recherchée. Dans ce cas, il n'en existe pas. L'événement e remonte donc, l'un après l'autre, les éléments parents de ASW. Éventuellement, l'élément racine est atteint. Lors de la recherche de souscriptions ayant pour source cet élément racine, la souscription S est trouvée. Le type d'événement de e (ActorSpecificWorkStart) correspondant à celui de la souscription S, l'EventHandler associé (TB, le tableau de bord) est notifié.

5.6 Catalogue d'événements

Les modèles de processus CMSPEM peuvent être continuellement mis à jour, pour refléter l'organisation changeante du travail collaboratif. De nouvelles personnes peuvent rejoindre ou quitter l'équipe, des tâches peuvent commencer ou se terminer, etc. Notons que l'ajout et la suppression sont des événements génériques disponibles pour chaque élément de modèle.

Dans la présente section, chaque événement est présenté avec sa description, et les éventuels paramètres qui l'accompagnent.

5.6.1 Événements relatifs aux acteurs

5.6.1.1 NewActor

Signale l'ajout au modèle d'un nouvel acteur.

5.6.1.2 ActorRemoval

Signale la suppression d'un acteur du modèle.

5.6.1.3 ActorAvailable

Signale qu'un acteur présent dans le modèle est maintenant disponible.

5.6.1.4 ActorUnavailable

Signale qu'un acteur présent dans le modèle est maintenant indisponible. Un acteur peut devenir indisponible parce qu'il est en congé maladie par exemple.

5.6.1.5 ActorRoleAssignment

Signale qu'un nouveau `RoleUse` a été assigné à l'acteur.

Paramètres additionnels :

roleUse : `RoleUse`
Le `RoleUse` assigné à l'acteur.

5.6.1.6 ActorRoleUnassignment

Signale qu'une assignation existante de `RoleUse` a été annulée pour un acteur.

Paramètres additionnels :

roleUse : `RoleUse`
Le `RoleUse` dont l'assignation est annulée.

5.6.2 Événements relatifs aux tâches spécifiques à un acteur

5.6.2.1 NewActorSpecificWork

Signale l'ajout au modèle d'une tâche spécifique à un acteur.

5.6.2.2 ActorSpecificWorkRemoval

Signale la suppression d'une tâche spécifique à un acteur du modèle.

5.6.2.3 ActorSpecificWorkStart

Signale le début de la réalisation d'une tâche spécifique à un acteur.

Paramètres additionnels :

date : Date
La date de début de la tâche.

5.6.2.4 ActorSpecificWorkEnd

Signale la fin de l'exécution d'une tâche spécifique à un acteur.

Paramètres additionnels :

date : Date
La date de fin de la tâche.

5.6.3 Événements relatifs aux artéfacts

5.6.3.1 NewActorSpecificArtifact

Signale l'ajout au modèle d'un nouvel artéfact.

5.6.3.2 ActorSpecificArtifactRemoval

Signale la suppression d'un artéfact du modèle.

5.6.3.3 ActorSpecificArtifactChange

Signale la modification d'un artéfact. Ceci peut correspondre simplement à la modification d'un fichier.

5.6.4 Événements relatifs aux relations de collaboration

5.6.4.1 NewRelationship

Signale l'ajout d'une nouvelle relation au modèle.

5.6.4.2 RelationshipRemoval

Signale la suppression d'une relation du modèle.

5.6.4.3 RelationshipDisabled

Signale la désactivation d'une relation. Une relation désactivée et considérée temporairement comme non présente par les divers outils qui exploitent le modèle.

5.6.4.4 RelationshipEnabled

Signale l'activation d'une relation présente dans le modèle, mais désactivée.

5.6.5 Séquence d'événements

Le concepts de *séquence d'événement* n'est pas explicitement couvert par CMSPEM. Cependant, il faut noter que certaines suites d'événement forment un tout cohérent, qui correspond à une seule modification logique du modèle de processus.

En effet, une modification décidée par un chef de projet peut correspondre à une série de modifications élémentaires, chacune générant un événement distinct. Il est de la responsabilité des outils traitant les événements de détecter ces séquences, et d'y réagir spécialement si nécessaire. Un exemple de séquence d'événements est le suivant, correspondant à l'introduction d'un participant, Pierre, dans une équipe de développement après l'indisponibilité d'un participant, Paul, due à un congé maladie :

– *ActorUnavailable* avec pour source l'`Actor` Paul.
– *NewActor* avec pour source l'`Actor` Pierre.

5.7 Conclusion

Ce chapitre a présenté la conceptualisation de l'évolution des modèles de processus CMSPEM. Cette évolution est représentée par des événements, qui peuvent être générés par des sources, et traités via des souscriptions qui spécifient des gestionnaires d'événement à notifier.

Un mécanisme de réaction aux événements a été défini, et clarifie les grandes étapes du traitement d'un événement, depuis la génération de l'événement par une source d'événement, jusqu'à la notification de tous les gestionnaires d'événements associés à l'événement. Ce mécanisme a été enrichi par un concept de parenté d'élément de modèle, qui, d'une part, simplifie

considérablement la définition des souscriptions pour des événements pouvant être générés par un grand nombre de sources. D'autre part, le concept de parenté permet à une souscription de couvrir des événements générés par des sources absentes du modèle au moment de la définition de la souscription, du moment que ces sources ont comme ancêtre l'élément associé à la souscription. Ceci permet aux souscriptions d'être robustes par rapport à l'évolution du modèle de processus.

Le chapitre 6 décrit l'implémentation concrète du mécanisme de souscription et de notification d'événement, qui, dans le serveur CMSPEM, se base sur des requêtes HTTP et des identifiants XMI.

Le catalogue d'événements de processus présenté dans la section 5.6 montre la richesse des modèles de processus en événements utiles pour le support à la collaboration. Cette utilité repose en grande partie sur le contexte que ces événements offrent aux autres préoccupations de développements. Ainsi, un commit et un rapport de bug peuvent être placés dans le contexte de l'évolution d'une tâche définie au niveau du modèle de processus. Le chapitre 7 contient des exemples d'exploitation de ces possibilités nouvelles.

Chapitre 6

Implémentation

6.1 Introduction

Le chapitre 3 propose un modèle conceptuel du support au développement collaboratif, et explique comment appliquer ce modèle aux processus de développement. Les chapitres 4 et 5 définissent le méta-modèle CMSPEM, qui formalise, dans le cadre du précédent modèle conceptuel, les données et événements de processus logiciels. Le présent chapitre décrit l'implémentation des différents outils de manipulation de modèles de processus CMSPEM.

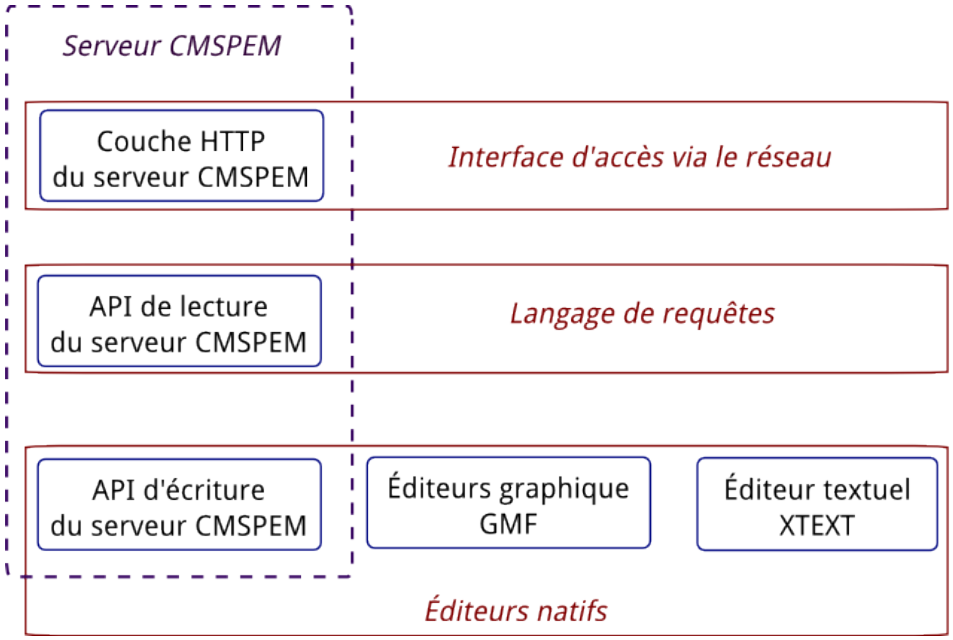

FIGURE 6.1 – Vue d'ensemble de l'implémentation du gestionnaire de préoccupation

D'une part, divers outils ont été développés pour permettre de créer des modèles de processus CMSPEM. Il s'agit d'éditeurs réalisés sur la plate-forme Eclipse. Pour chacun de ces

éditeurs, nous décrivons les différentes raisons ayant motivé leur conception, l'utilité des fonctionnalités qu'ils offrent, et les étapes de leur implémentation.

D'autre part, un serveur de processus CMSPEM a été développé, afin de rendre possible l'exploitation des modèles de processus via les concepts de "deep-link" et de "hook". Nous établissons un cahier des charges des fonctionnalités d'un tel serveur, spécifions ses fonctionnalités, décrivons les étapes de son implémentation, et discutons les compromis pratiques réalisés au cours de son implémentation.

Le but principal des discussions techniques faites dans ce chapitre est de montrer comment chacun des choix techniques effectués contribue à un ou plusieurs aspects du modèle conceptuel de support au développement collaboratif. Des références sont donc fréquemment faites aux sous-sections correspondantes de la section 3.3.

La correspondance entre les outils développés et le modèle conceptuel de la section 3.3 est schématisée dans la figure 6.1. Le serveur CMSPEM offre les fonctionnalités d'éditeur natif (sous-section 3.3.7), de langage de requête (sous-section 3.3.6), et d'interface d'accès au réseau (sous-section 3.3.8). Les éditeurs Eclipse, pour leur part, servent essentiellement d'éditeurs natifs pour la préoccupation de gestion des processus. Ces outils sont utilisés, avec des utilitaires additionnels, présentés au chapitre 7 pour valider, sur des cas pratiques, l'approche proposée.

6.2 Éditeurs de modèle

Les éditeurs de modèle jouent aussi bien le rôle d'éditeur natif (sous-section 3.3.7) que celui de langage de requête (sous-section 3.3.6). Ces rôles ont été définis dans le modèle conceptuel de support au développement collaboratif (section 3.3). En effet, d'une part, les éditeurs sont utilisés pour créer et mettre à jour des modèles de processus CMSPEM. D'autre part, la représentation du modèle utilisée par chaque éditeur (graphique ou textuelle), ainsi que les diverses fonctionnalités de parcours de cette représentation (défilement, recherche, zoom, etc.) constituent des moyens de consulter l'information disponible dans le modèle de processus.

6.2.1 Éditeur graphique GMF

6.2.1.1 Description

L'éditeur graphique permet de manipuler visuellement et de consulter un modèle de processus. Le fonctionnement général est assez intuitif pour permettre une prise en main rapide. L'éditeur reprend l'interface générale d'Eclipse (figure 6.2) :

1. Un canevas contenant la représentation graphique du modèle de processus.
2. Un explorateur de projet donnant accès aux divers modèles disponibles dans l'espace de travail.
3. Une palette d'outils de création des divers concepts et liens de CMSPEM.

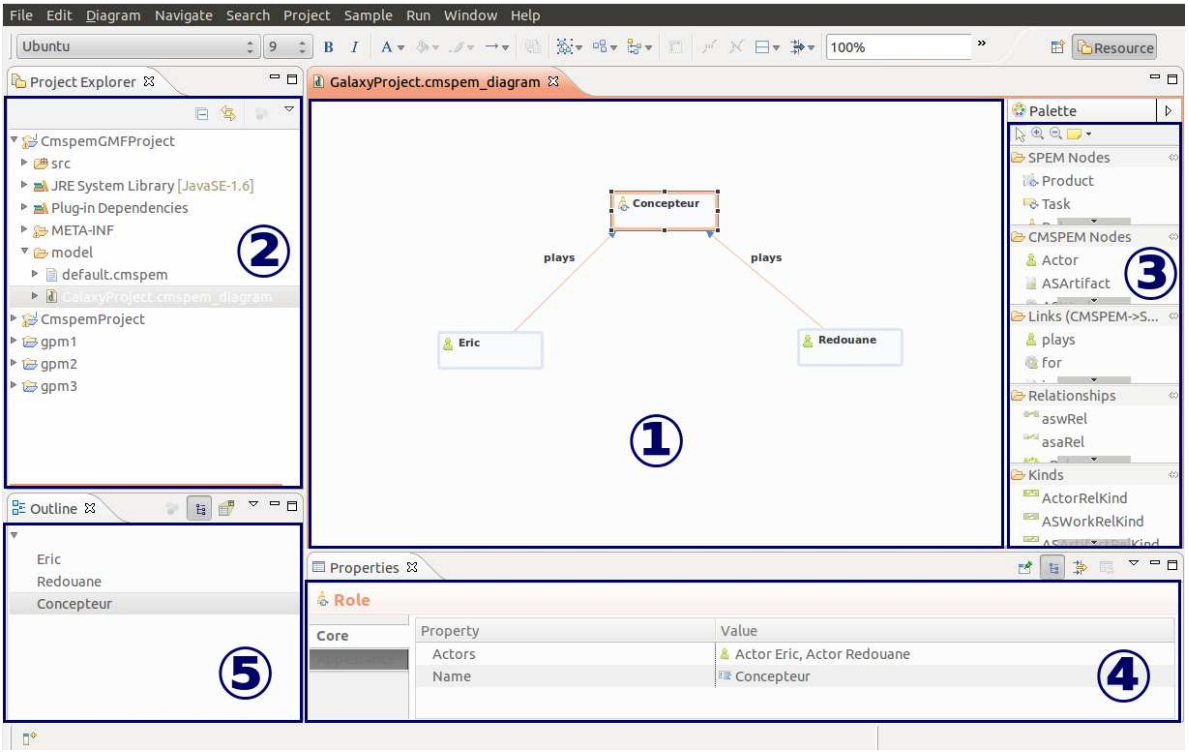

FIGURE 6.2 – Vue d'ensemble de l'éditeur GMF

4. Une vue permettant de renseigner diverses propriétés des éléments du modèle, en particulier celles non représentées sur le graphique.
5. Une aperçu listant tous les éléments de modèle, et permettant d'accéder directement à chaque élément sur le canevas en le sélectionnant dans l'aperçu.

La fonction de modification de modèle de l'éditeur (voir section 3.3.7) est exposée, en grande partie, via la palette d'outils (zone 3, figure 6.2) pour la création des concepts et des relations, et dans une moindre mesure via la vue de propriétés (zone 4, figure 6.2) pour la spécification des propriétés de concept.

La palette d'outils de création (figure 6.3) de l'éditeur GMF conserve l'organisation logique des éléments CMSPEM. Ceci a pour objectif de rendre intuitif le processus de création des modèles CMSPEM. En effet, la mise en place d'un modèle CMSPEM commence par les concepts de grosse granularité SPEM (`RoleUse`, `TaskUse`, `WorkProductUse`). Ensuite, les concepts CMSPEM de base qui les raffinent (`Actor`, `ActorSpecificWork`, `ActorSpecificArtifact`) sont ajoutés, et les liens avec les concepts SPEM sont établis. Enfin, les diverses relations entre concepts CMSPEM de base (`ArtifactUse`, `WorkAssignment`, `ArtifactOwnership`, `ActorRelationship`, `ActorSpecificWorkRelationship`, `ActorSpecificArtifactRelationship`) sont introduites, ainsi que les différents `Kinds` qui qualifient ces relations. Cette structuration de la palette est aussi reprise dans la différenciation de couleur entre les représentations graphiques sur le canevas. La palette contient les groupes d'outils suivants :

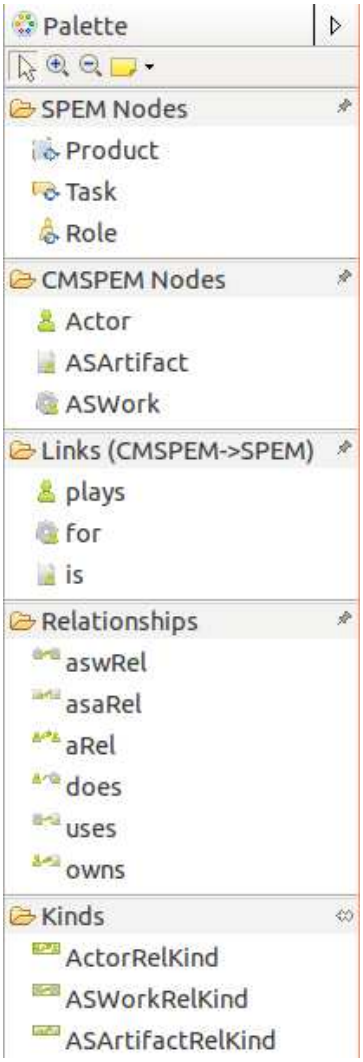

FIGURE 6.3 – Palette d'outils de création de l'éditeur GMF

- Outils de création de concepts propres à SPEM (`RoleUse`, `TaskUse`, `WorkProductUse`).
- Outils de création de concepts CMSPEM (`Actor`, `ActorSpecificWork`, `ActorSpecificArtifact`).
- Outils de création de liens entre concepts CMSPEM et concepts SPEM.
- Outils de création de relations CMSPEM (`ArtifactUse`, `ArtifactOwnership`, `WorkAssignment`, `ActorRelationship`, `ActorSpecificWorkRelationship`, `ActorSpecificArtifactRelationship`).
- Outils de création de kinds CMSPEM (`ActorRelationshipKind`, `ActorSpecificWorkRelationshipKind`, `ActorSpecificArtifactRelationshipKind`).

La fonction de consultation de modèle (voir section 3.3.6) est assurée par le canevas (zone 1, figure 6.2) et la vue d'aperçu (zone 5, figure 6.2). Cependant, la facilité à retrouver des

éléments de modèle est limitée. En effet, dans le canevas, on ne dispose que des fonctions natives de zoom et de cadrage d'Eclipse. Dans la vue d'aperçu, l'ensemble des éléments de modèle est listé linéairement, ce qui implique une procédure séquentielle dans la recherche d'information. De plus, ces fonctions étant disponibles dans une interface graphique, elles ne sont pas pratiques à automatiser par un outil tiers désirant accéder à des éléments de modèle spécifiques. L'éditeur graphique, de par la métaphore visuelle utilisée, offre donc un langage de requêtes limité pour les modèles CMSPEM.

6.2.1.2 Implémentation

L'éditeur graphique a été implémenté sous GMF (Graphical Modeling Framework). Le framework GMF [Gron 09] est un projet Eclipse pour la génération d'éditeurs graphiques, qui s'appuie sur GEF [Modi 09] (Graphical Editing Framework), une librairie de construction d'éditeurs graphiques et EMF [Stei 08] (Eclipse Modeling Framework), un framework de manipulation de modèles.

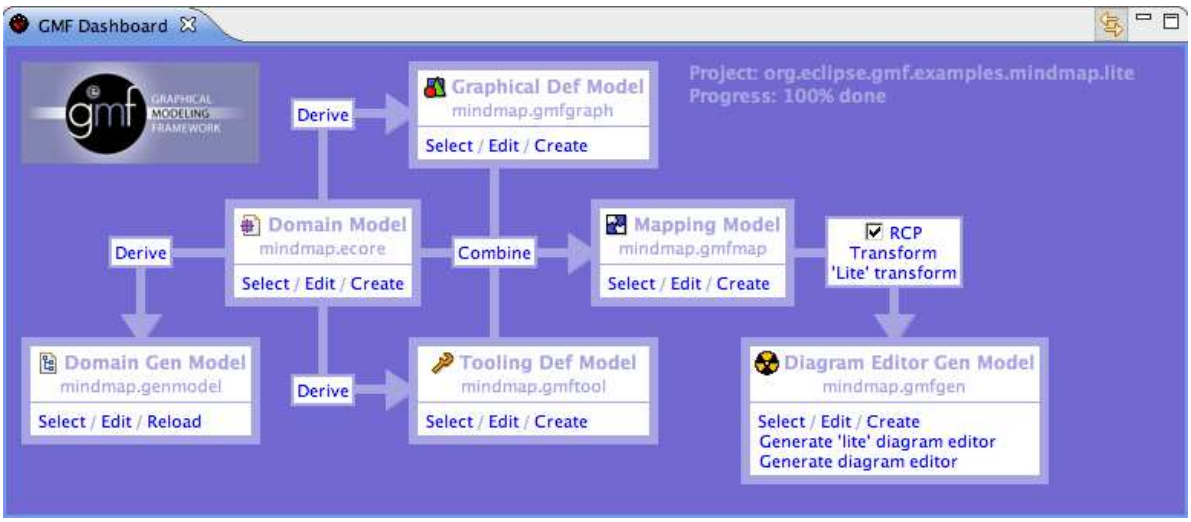

FIGURE 6.4 – Relations entre les divers modèles d'entrée GMF

GMF prend en entrée un certain nombre de modèles décrivant un éditeur graphique, et génère le code de l'éditeur en tant que plugin Eclipse ou application indépendante. Les modèles en entrée décrivent chacun un aspect de l'éditeur :

- Un modèle de domaine (ECORE). Ce modèle correspond au méta-modèle sur lequel est basé le futur éditeur, CMSPEM dans ce cas.
- Un modèle de génération (GENMODEL). Il contient les paramètres de génération de code permettant de produire :
  - Les classes JAVA représentant les concepts du méta-modèle ("metamodel code").
  - Les utilitaires de manipulation de la représentation XMI (XML Model Interchange) des modèles ("edit code").

  - L'éditeur arborescent de modèle. Il s'agit d'un éditeur basique, distinct de l'éditeur GMF, et non décrit ici.

- Un modèle graphique (GMFGRAPH). Il définit les symboles graphiques à utiliser, dans le canevas du futur éditeur, pour représenter les instances des différents concepts et relations du méta-modèle.
- Un modèle d'outils (GMFTOOL). Il décrit la palette d'outils de création disponibles dans le futur éditeur, ainsi que les icônes et labels associés.
- Un modèle d'association (GMFMAP). Il fait le lien entre le modèle de domaine, le modèle graphique, et le modèle d'outils. Il associe par exemple au concept `Actor` du modèle de domaine un outil `ActorNode` du modèle d'outils, qui crée, sur le canevas, un instance du concept `Actor`, avec la représentation visuelle `ActorFigure` du modèle graphique.
- Un modèle de génération de l'éditeur graphique (GMFGEN). Il spécifie divers paramètres de la génération de code comme l'environnement Eclipse d'exécution du plugin, les conventions de codage, les noms des packages générés, etc.

Le lien entre ces divers modèles d'entrée GMF est résumé dans la figure 6.4, extraite de la documentation GMF [1].

La manipulation des divers modèles d'entrée GMF pouvant être fastidieuse, l'outil "GMF Simple Map Editor" [2], qui permet de créer des modèles "SIMPLEMAPPINGS", a été utilisé. Un modèle SIMPLEMAPPINGS décrit complètement, et de manière succincte, un éditeur GMF. Il est utilisé par l'outil "GMF Simple Map Editor" pour générer les modèles GMFTOOL, GMFGRAPH, GMFMAP, et GMFGEN. Le modèle GMFGEN est ensuite utilisé par GMF pour générer le code de l'éditeur graphique.

Le modèle SIMPLEMAPPINGS créé pour CMSPEM est montré sur la figure 6.5. Les rectangles correspondent à des concepts, et les cercles à des relations. Sur la figure, les cercles (relations) sont placés manuellement entre les rectangles (concepts) qu'ils lient, afin de faciliter la compréhension du modèle [3]. Les deux dernières lignes en bas de la figure correspondent aux `Kinds` (rectangles) et aux liens (cercles) utilisés pour les associer aux relations qu'ils qualifient.

6.2.2 Éditeur textuel XTEXT

6.2.2.1 Motivation

La motivation essentielle pour le développement de l'éditeur textuel est la création de modèles CMSPEM par des outils tiers. En effet, il n'est pas pratique pour des outils tiers de comprendre le format XMI utilisé par EMF pour ces (bouts de) modèles CMSPEM. Une représentation textuelle a l'avantage d'être très facile à générer, ce qui est crucial pour une adoption aisée des outils CMSPEM.

1. http://wiki.eclipse.org/index.php/GMF_Tutorial_Part_4
2. https://github.com/aamattos/GMF-Tooling-Visual-Editor
3. Les liens entre les concepts et les relations étant spécifiés avec la vue de propriétés, ils ne sont pas visibles sur la figure. L'organisation spatiale des figures a donc été utilisée pour mettre ces liens en évidence.

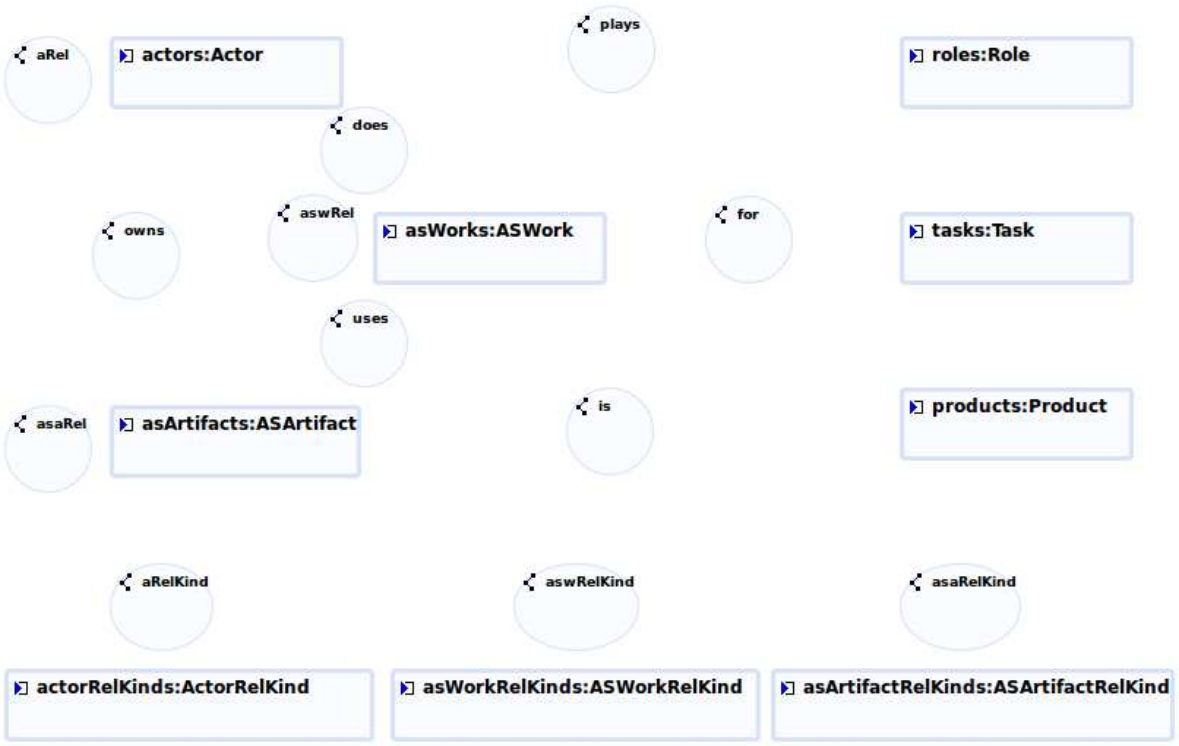

FIGURE 6.5 – Modèle SIMPLEMAPPINGS pour l'éditeur GMF

Une organisation souhaitant créer des modèles de processus CMSPEM dispose certainement déjà d'une liste de participants au projet, sous format électronique, typiquement dans une base de données SQL ou dans un annuaire LDAP. Vu qu'il n'y a aucun moyen simple d'automatiser l'entrée de données dans l'éditeur GMF, l'effort supplémentaire nécessaire pour entrer manuellement toutes les données aurait créé une barrière non négligeable à l'adoption. Par contre, il est très simple de générer automatiquement la représentation textuelle du listing 6.6 qui représente un `Actor` dans la notation textuelle CMSPEM, à partir de la source de donnée disponible.

```
actor Komlan {
  fullName: "Komlan Akpédjé KEDJI"
  email: "komlan.kedji@univ-tlse2.fr"
}
```

FIGURE 6.6 – Représentation d'un acteur dans la notation textuelle de CMSPEM.

6.2.2.2 Description

Un syntaxe textuelle (voir l'annexe 10.3) a été définie pour CMSPEM, et est supportée par un éditeur fonctionnant sous Éclipse.

L'éditeur textuel est équipé d'un ensemble de fonctionnalités habituelles d'un IDE pour langage de programmation, afin de permettre la création simple et efficace de modèles CM-

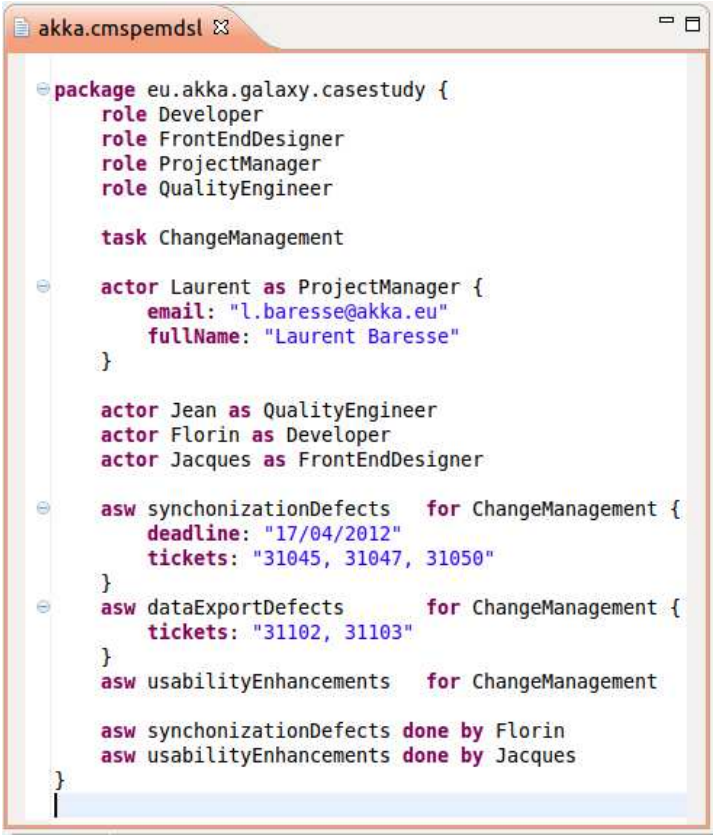

FIGURE 6.7 – Exemple de modèle décrit avec la syntaxe textuelle CMSPEM

SPEM. Il supporte l'indentation automatique, la coloration syntaxique, ainsi que la complétion automatique. Ces fonctionnalités d'édition correspondent à la fonction d'éditeur natif tel que défini dans le modèle conceptuel du développement collaboratif (section 3.3.7). Un exemple de modèle CMSPEM défini avec la syntaxe textuelle est illustré par la figure 6.7.

L'éditeur dispose aussi de fonctionnalités de recherche et de navigation de code qui utilisent les vues habituelles de l'IDE Eclipse. Les fonctionnalités de navigation de code correspondent à une forme limitée du langage de requêtes décrit dans la section 3.3.6.

Bien que la représentation textuelle soit facile à créer, elle ne se prête pas toujours à la visualisation intuitive des relations entre les éléments d'un modèle. Une vue graphique a donc été développée, afin d'offrir une visualisation constamment mise à jour du modèle. Cette vue dispose de son propre langage qui peut être utilisé pour personnaliser la visualisation, et mettre l'accent sur seulement certains aspects du modèle. Dans la figure 6.8 par exemple, les acteurs et leurs relations sont mises en évidence sur le panneau de droite. La visualisation est mise à jour dynamiquement, au fur et à mesure que la représentation textuelle du modèle CMSPEM est modifiée.

Lors de la saisie d'un modèle CMSPEM avec l'éditeur textuel, des vérifications syntaxiques et sémantiques sont faites à la volée. Les vérifications syntaxiques portent sur la syntaxe tex-

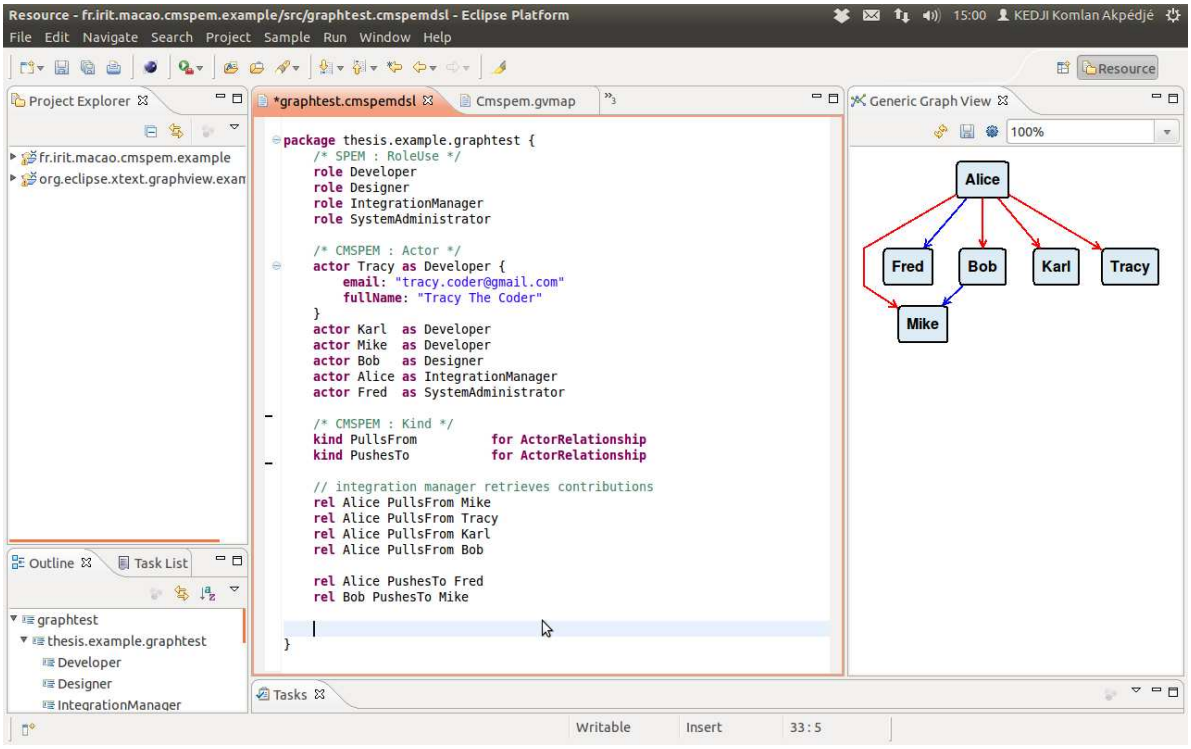

FIGURE 6.8 – Exemple de visualisation générée à partir d'un modèle CMSPEM

tuelle CMSPEM telle que définie dans XTEXT. La vérification sémantique s'appuie sur les règles OCL (voir l'annexe 10.2) définies pour le méta-modèle CMSPEM. Des infos-bulles sont ainsi offertes au fur et à mesure de la saisie de modèle pour signaler les incohérences, comme dans la figure 6.9.

6.2.2.3 Implémentation

L'éditeur textuel CMSPEM est implémenté avec le framework XTEXT [Efft 06]. XTEXT est un framework de développement de langages de programmation et de DSLs (Domain Specific Language). Il fournit une infrastructure complète de langage, depuis les analyseurs syntaxiques jusqu'aux éditeurs intégrés dans la plate-forme Eclipse. XTEXT s'appuie sur le framework EMF pour la manipulation de modèles.

Le point de départ de la création de l'éditeur a été la définition de la grammaire de la représentation textuelle XTEXT. Un grammaire XTEXT est un document texte pseudo-EBNF (Extended Backus-Naur Form [Wirt 96]), décrivant la syntaxe du langage. La grammaire contient des références vers le méta-modèle ECORE de CMSPEM, afin de spécifier les concepts correspondant à chaque forme textuelle. Ceci permet à XTEXT de valider au fur à mesure la grammaire, en s'assurant qu'elle respecte la structure des concepts telle que définie dans le méta-modèle. La grammaire XTEXT utilisée est disponible dans l'annexe 10.3.

L'éditeur généré par XTEXT à partir de la grammaire sauvegarde, non seulement la représentation textuelle manipulée, mais aussi une représentation XMI du modèle. Pour les

FIGURE 6.9 – Vérification de contrainte OCL lors de la saisie dans l'éditeur XTEXT

besoins du serveur CMSPEM (section 6.3), cette représentation a besoin d'être enrichie. En particulier, il est nécessaire de s'assurer que chaque élément de modèle possède un identifiant unique. La fonctionnalité de sauvegarde de l'éditeur généré a donc été modifiée pour prendre ces contraintes en compte.

Une piste explorée lors de l'implémentation de l'éditeur textuel est la possibilité de spécifier les gestionnaires d'événement avec du code JAVA, directement à l'intérieur de la représentation textuelle. Cependant, cette option résulte en une mauvaise séparation entre les données du modèle, et les traitements sur ces données (réactions aux événements). La spécification des gestionnaires d'événement a donc été réservée au serveur CMSPEM, décrit dans la section 6.3.

6.3 Le serveur CMSPEM

Le serveur CMSPEM expose une simple API de manipulation de modèle de processus via HTTP. Le serveur a été développé en Java, en utilisant le framework Play, les librairies de manipulation de modèles d'Eclipse EMF, et la librairie EMFJSON [4] pour la conversion entre les formats EMF et JSON [5].

4. https://github.com/ghillairet/emfjson
5. http://json.org

6.3.1 Fonctionnalités

La finalité principale du développement du serveur CMSPEM est de permettre à des outils tiers d'exploiter les données et évènements de processus, en utilisant des liens profonds et des crochets. Pour ce faire, le serveur implante les fonctionnalités suivantes, classées selon les éléments du modèle conceptuel de la section 3.3.

Le serveur de processus sert d'éditeur natif (section 3.3.7) pour les modèles de processus CMSPEM. Pour ce faire, il :

- Permet de créer des modèles de processus CMSPEM, en envoyant au serveur une requête contenant le contenu initial du modèle.
- Stocke les modèles de processus CMSPEM, et toute donnée utile associée (date de création, de dernière modification, etc.)
- Permet de mettre à jour des modèles de processus CMSPEM. La mise à jour doit pouvoir être chirurgicale, dans le sens où l'on doit pouvoir, en une requête modifier la valeur d'un attribut précis sur un élément de modèle précis. Par exemple, il doit être possible de mettre à jour le nom d'une tâche, sans avoir à envoyer au serveur une copie complète du modèle de processus modifié.

Le serveur met à disposition des outils tiers un langage de requête (sections 3.3.4 et 3.3.6) sur le modèle de processus, en permettant de :

- Exporter le modèle de processus pour consommation par des outils tiers, sous divers formats.
- Lire une partie du modèle de processus, suite à une requête suivant des conventions bien définies pour spécifier l'élément recherché. Il doit donc être possible, par exemple, de demander des informations sur une tâche particulière, sans pour autant devoir obtenir et traiter l'ensemble du modèle de processus.
- Représenter une requête de lecture sous un format facile à partager, et réutilisable.

Dans le cadre de la gestion des événements de processus (section 3.3.5), le serveur a la capacité de :

- Recevoir et enregistrer les abonnements d'outils tiers à des évènements de processus. Les évènements de processus (la modification d'une tâche par exemple) ont chacun des types préalablement définis, qui sont précisés lors de la souscription (voir section 5.6.
- Envoyer des notifications d'événement de processus à des outils tiers, suite à leur occurrence. Les événements sont générés par le serveur suite à une modification du processus, et les outils ayant des souscriptions correspondantes sont notifiés.

Le serveur garde aussi la trace des participants au projet (acteurs) responsables de chaque action sur le serveur de processus. Ceci demande de mettre en place des comptes utilisateur, et que chaque requête soit associée à un compte existant. Les utilisateurs pouvant envoyer des requêtes au serveur CMSPEM peuvent être rangés en deux grandes catégories :

– Les développeurs, qui sont des participants normaux à un projet.
– Les chefs de projet ou ingénieurs qualité, qui supervisent le projet et ont des droits étendus sur le modèle de processus.

Suivant le type d'utilisateur et les besoins, on distingue divers scénarios d'utilisation. Ces scénarios sont décrits dans l'annexe 10.4, en utilisant les conventions de description de fonctionnalités des méthodes de développement agiles.

6.3.2 Conception

Chaque agent logiciel qui fait une requête sur le serveur CMSPEM le fait avec une clé d'API, qui est envoyée avec chaque requête, pour l'authentifier. Une clé d'API spéciale est utilisée par l'administrateur du serveur, qui seul peut faire des requêtes de gestion d'utilisateur (ajouter et supprimer des clés d'API). L'architecture interne du serveur (voir figure 6.10) est présentée ci-après.

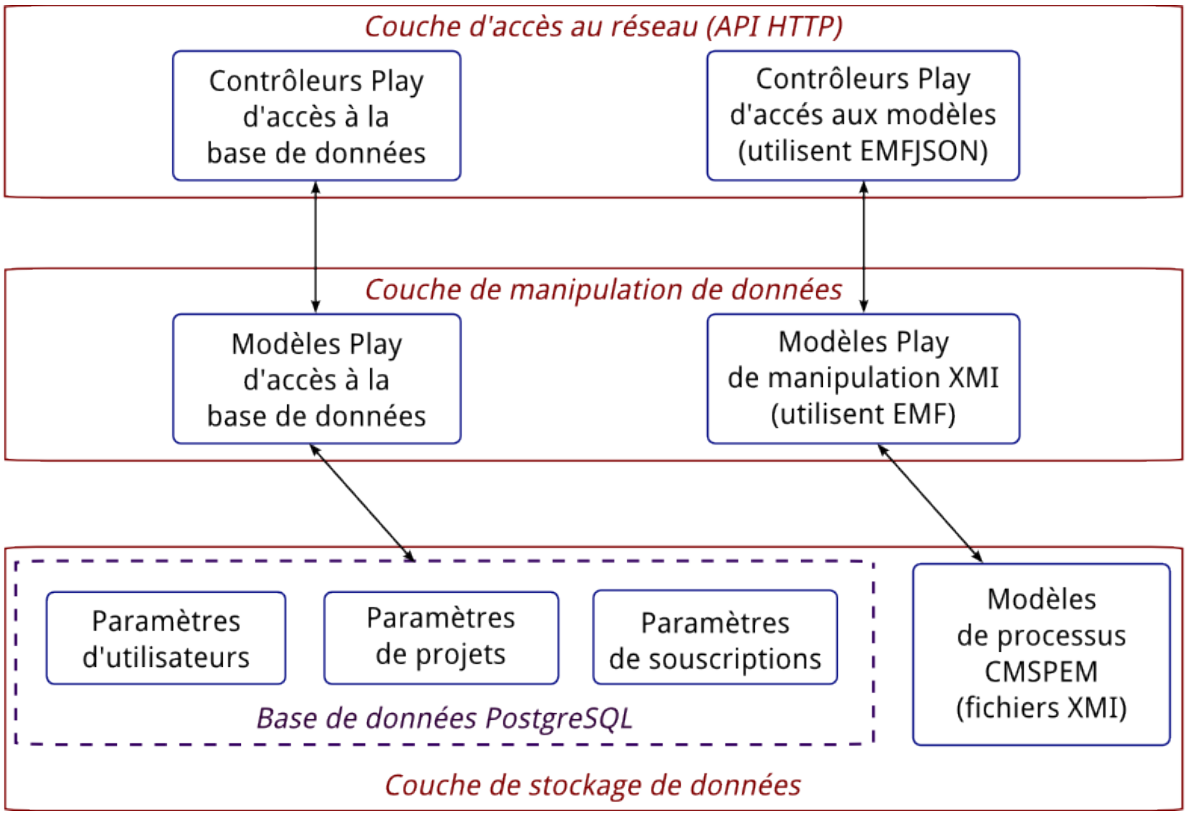

FIGURE 6.10 – Architecture interne du serveur CMSPEM

6.3.2.1 Stockage de données

Les modèles manipulés par le serveur sont sauvegardés sur disque comme des fichiers XMI. Une base de données conserve les métadonnées associées aux projets (créateur du projet, dates de création et de dernière modification, etc.), les données sur les utilisateurs, les

souscriptions d'événements, et toute autre donnée relative au modèle comme l'historique de modifications.

6.3.2.2 Manipulation de données

Une couche d'abstraction au dessus des API EMF permet de réaliser les opérations courantes comme ajouter un acteur à un modèle, ou masquer une relation existante dans un modèle de processus. Cette couche manipule aussi les données hors du modèle comme les métadonnées sur les utilisateurs.

Un utilisateur peut modifier les métadonnées qui lui sont associés, les modèles qu'il a créés, et les souscriptions dont il est l'auteur.

6.3.2.3 API HTTP

L'API HTTP reçoit les requêtes sur le serveur, et les traduit en appels à la couche de manipulation de données. Cette couche valide aussi les paramètres en entrée, et fait les contrôles d'authentification et d'autorisation nécessaires.

6.3.2.4 Outils tiers

Tout outil capable de faire des requêtes HTTP peut communiquer avec le serveur CMSPEM. Typiquement, il s'agit des autres outils de collaboration comme les gestionnaires de versions ou les solutions d'intégration continue.

Les outils de modélisation sont une catégorie spéciale d'outil tiers. Les opérations qu'ils réalisent sur le serveur se limitent à la création et à la mise à jour de modèles de processus.

6.3.3 Implémentation

Le serveur de processus (voir figure 6.10) a été implémenté comme une application web, en Java, avec le framework Play[6]. Play est un framework Java/Scala pour la création d'applications web, avec le modèle MVC. Le modèle MVC (Modèle-Vue-Contrôleur) permet de séparer méthodiquement, dans une application, les aspects métier (modèle), des aspects de présentation (vue), et des liens entre les actions utilisateurs et les traitements métier (contrôleur).

Un ensemble de modèles Play ont été créés pour représenter les trois concepts clés mis à disposition par l'API, et conservés sur le serveur : les utilisateurs, les projets (modèles de processus), et les souscriptions. Les souscriptions et utilisateurs sont stockés dans une base de données PostgreSQL. Les métadonnées de projet (créateur, date de création et de modification, etc.) sont conservées dans la base de données, et les modèles de processus eux-mêmes sont conservés avec des fichiers contenant une représentation XMI. Des contrôleurs Play ont été créés pour chacun des concepts CMSPEM, et correspondent aux URLs décrites dans la documentation de l'API (voir l'annexe 10.5).

6. http://www.playframework.com/

L'API HTTP mise à disposition par le serveur CMSPEM est conçue selon l'architecture REST [Fiel 00] (REpresentational State Transfer). REST est un style d'architecture pour les systèmes hypermédia distribués comme le web. Son choix se justifie d'une part par le fait qu'il impose à chaque ressource exposée par le serveur d'avoir sa propre URL, ce qui correspond exactement au besoin de liens profonds exprimés dans la section 3.3.4. D'autre part, l'interface uniforme REST formée par les méthodes HTTP (voir l'annexe 10.5) permet de définir une simple interface d'accès au serveur par les outils tiers.

Le serveur communique avec les outils tiers au format JSON, afin de s'intégrer facilement avec un grand nombre de systèmes tiers, surtout ceux qui possèdent une interface HTTP comme la plupart des outils de développement modernes. En effet, JSON (JavaScript Object Notation) reprend la notation d'objet du langage Javascript, et possède des librairies dans la plupart des langages de programmation. La conversion entre les données de processus au format XMI et la représentation JSON est faite avec la librairie EMFJSON [7].

Les souscriptions sont reçues via les URLs prévues dans l'API (voir l'annexe 10.5). Afin d'être contactées par le serveur lors d'une occurrence d'événement, les outils tiers qui enregistrent des souscriptions spécifient une adresse web, sur laquelle une requête HTTP de type POST est envoyée, pour signaler une occurrence d'événement.

L'API CMSPEM possède un ensemble de tests écrits en Scala [8], qui garantissent que le serveur renvoie les codes de retour HTTP prévus, selon la requête reçue. Le serveur CMSPEM a été déployé en ligne sur la plate-forme Heroku [9], est disponible à l'adresse `https://cmspem.herokuapp.com/`.

6.4 Scenarios d'utilisation et exemples

Cette section décrit quelques cas pratiques d'utilisation du serveur CMSPEM, ainsi que les différentes actions correspondantes sur le serveur.

6.4.1 Flux de données et de contrôle

6.4.1.1 Lecture de modèle

1. Un outil externe fait une requête de lecture à l'API CMSPEM.
2. Le serveur authentifie la requête, et vérifie les autorisations.
3. La couche d'accès aux données charge éventuellement le modèle XMI en mémoire, et extrait les données demandées.
4. L'API CMSPEM convertit les données extraites dans le format de sortie (JSON) et les renvoie à l'outil externe.

7. `https://github.com/ghillairet/emfjson`
8. Scala est un langage de programmation qui peut être compilé vers du bytecode Java. Voir `http://www.scala-lang.org/`.
9. Heroku est une plateforme applicative dans le Cloud. Voir `https://www.heroku.com/`.

6.4.1.2 Modification de modèle

1. Un outil externe fait une requête de modification de modèle à l'API CMSPEM.
2. Le serveur authentifie la requête, et vérifie les autorisations.
3. La couche d'accès aux données charge éventuellement le modèle XMI en mémoire.
4. Les données sont modifiées en mémoire.
5. La couche d'accès aux données sauvegarde éventuellement les données modifiées sur le disque.
6. L'API CMSPEM convertit éventuellement les données associées à l'élément modifié vers le format de sortie (JSON), et le renvoie à l'outil externe.
7. Le serveur déclenche les événements liés à la modification, et notifie les éventuels gestionnaires d'événement définis.

6.4.1.3 Souscription d'événement par un outil tiers

1. Un outil tiers lit éventuellement les éléments disponibles dans le modèle, et les événements disponibles sur ces éléments.
2. L'outil tiers envoie une requête de souscription à l'API CMSPEM, précisant, entre autres, l'adresse de notification.
3. L'API CMSPEM authentifie et autorise éventuellement la requête.
4. La couche d'accès aux données charge le modèle XMI en mémoire.
5. La couche d'accès aux données enregistre la souscription dans la base de données.
6. Le serveur CMSPEM confirme la souscription à l'outil tiers.

6.4.1.4 Notification d'événement par le serveur CMSPEM

1. Il se produit sur le modèle un événement pour lequel il existe des souscriptions (modification de modèle).
2. Pour chacune des souscriptions existantes, le serveur CMSPEM :
   (a) Rassemble les paramètres de l'événement, et les convertit au format d'échange (JSON).
   (b) Lance une requête HTTP de type POST sur l'URL spécifiée lors de la souscription, avec les paramètres de l'événement.

6.4.2 Exemples de requêtes sur le serveur CMSPEM

Le serveur CMSPEM implémente trois principales familles de cas d'utilisation [10] :

10. Les exemples sont donnés en ligne de commande, avec un utilitaire d'envoi de requêtes HTTP nommé `http` (https://github.com/jkbr/httpie). De plus, la variable `CMSPEM_KEY` contient la clé d'API de l'utilisateur effectuant la requête.

– L'obtention d'informations sur le modèle de processus. Par exemple, une requête sur `/projects/XXX/asworks/YYY` renvoie des données (attributs et valeurs, ansi que des références vers d'autres éléments de modèle) au sujet de l'`ActorSpecificWork` identifié par YYY, dans le projet identifié par XXX (Figure 6.11).

```
$ http GET                                                                    \
  http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA      \
  X-API-KEY:$CMSPEM_KEY
{
  "item": {
    "deadline": "25/11/2012",
    "name": "Create Wireframe",
    "task": {
      "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA"
    },
    "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
  },
  "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
}
```

FIGURE 6.11 – Exemple d'extraction d'information de processus à partir d'un modèle CMSPEM.

– La modification des informations du modèle de processus. Une requête de type POST sur `/projects/XXX/tasks/YYY` par exemple peut être utilisée pour modifier la deadline d'un `ActorSpecificWork` (Figure 6.12).

```
$ http -f POST                                                                \
  http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA      \
  deadline="05/07/2013"                                                       \
  X-API-KEY:$CMSPEM_KEY
{
  "item": {
    "deadline": "05/07/2013",
    "name": "Create Wireframe",
    "task": {
      "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA"
    },
    "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
  },
  "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
}
```

FIGURE 6.12 – Exemple de requête de mise à jour d'un modèle CMSPEM.

– Le traitement des souscriptions à des évènements de processus, et l'envoi de notifications quand ces évènements se produisent. Une requête sur `/projects/XXX/tasks/YYY/subscriptions` peut être utilisée pour demander à être notifié quand la tâche YYY se termine par exemple (Figure 6.13). Quand l'évènement concerné se produit, le serveur CMSPEM envoie une requête au gestionnaire (l'adresse URL spécifiée en paramètre lors de la souscription).

```
$ http -f POST                                                                    \
  http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions  \
  handler="http://cmspem-helper.herokuapp.com/task/new"                           \
  event=end                                                                       \
  X-API-KEY:$CMSPEM_KEY

{
  "item": {
    "correlationId": null,
    "created": 1348660352188,
    "event": "end",
    "eventsource": "/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions",
    "eventsourceId": "_EmwHoAc_EeK1qtFkCWkJBA",
    "handler": "http://cmspem-helper.herokuapp.com/task/new",
    "id": "21",
    "updated": 1348660352279,
    "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions/21"
  },
  "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions/21"
}
```

FIGURE 6.13 – Exemple de souscription à un évènement de processus.

6.5 Conclusion

Le présent chapitre a présenté les divers outils de support au méta-modèle CMSPEM, réalisés dans le cadre du travail de thèse. D'une part, des éditeurs de modèle ont été réalisés sous la plate-forme Eclipse. D'autre part, un serveur de modèles de processus, communiquant via le protocole HTTP, et gérant les données et événements de processus, a été développé.

Les fonctionnalités de consultation de modèle (langage de requête) sont présentes dans une moindre mesure dans les éditeurs graphiques et textuels développés. Cependant, c'est le serveur CMSPEM qui offre véritablement un langage de requête pour extraire de manière précise l'information de processus, grâce à son API HTTP/REST. L'association faite entre les URLs du serveur et les éléments de modèle permet d'avoir un accès direct à chacun de ces éléments, ce qui permet au serveur de mettre à disposition des liens profonds vers l'information de processus. De plus, la spécification du format de données utilisé par le serveur, et l'utilisation systématique des liens profonds pour désigner les références entre concepts, permet à tout outil tiers de naviguer facilement à travers le modèle de processus [11].

La modification de modèle est implémentée via un éditeur graphique implémenté sous GMF, et un éditeur textuel implémenté sous XTEXT. L'éditeur graphique est optimisé pour la manipulation intuitive, alors que l'éditeur textuel ouvre la possibilité de générer certaines parties d'un modèle de processus CMSPEM à partir de sources de données externes. L'éditeur textuel a aussi été muni d'une fonctionnalité d'extraction de visualisations personnalisées, permettant de mettre en valeur un aspect particulier du modèle de processus.

11. Ceci correspond à l'acronyme HATEOS (Hypertexte As The Engine Of Application State), utilisé pour décrire les systèmes REST. En effet, cette conception permet à un outil automatisé de parcourir et manipuler le modèle de processus en suivant les liens inclus dans les représentations renvoyées par le serveur.

Une des responsabilités principales du serveur de processus CMSPEM développé est de permettre la réaction aux événements de processus, grâce aux crochets. Une API de souscription aux événements a ainsi été mise en place, et la technique des web-hooks a été utilisée pour notifier les outils tiers des occurrences d'événement.

Les outils décrits dans le présent chapitre constituent une instanciation du modèle conceptuel défini dans le chapitre 3. Le chapitre 7 exploite ces outils sur un ensemble d'exemples, afin d'illustrer l'utilisation des données et événements de processus dans le support du développement collaboratif.

Chapitre 7

Validation de l'approche

7.1 Introduction

Le chapitre 3 a présenté un modèle conceptuel du support au développement collaboratif, qui repose sur la mise à disposition des données (chapitre 4) et événements (chapitre 5) des diverses préoccupations de développement collaboratif, dont les processus logiciels. Le chapitre 6 a, pour sa part, décrit l'implémentation des différentes fonctionnalités prévues par ce modèle conceptuel. Dans le présent chapitre, nous illustrons le méta-modèle sur un exemple, et présentons un ensemble de validations de l'approche proposée.

En premier lieu, nous décrivons les résultats d'une étude des données de listes de discussion d'un ensemble de 219 projets open source [Kedj 13]. Le but de cette étude est d'illustrer comment les concepts de "lien profond" et de "crochet" structurent l'intégration des outils de support au développement collaboratif, afin de valider l'approche conceptuelle du chapitre 3.

En second lieu, nous montrons, sur des cas pratiques, comment le serveur CMSPEM dont l'implémentation a été décrite au chapitre 6, grâce au support des "liens profonds" et des "crochets", peut être exploité dans le support automatisé au développement collaboratif [Kedj 12a, Kedj 13]. Pour ce faire, nous présentons quelques utilitaires dont certains ont été développés et démontrés sur des exemples.

7.2 Illustration et validation du modèle conceptuel

Pour valider la caractérisation des outils de développement existants faite au chapitre 3, une étude a été menée sur des projets open source. Le but de l'étude est de mettre en évidence le rôle central des concepts de "lien profond" et de "crochet" dans l'utilisation des outils de support au développement collaboratif.

La mise en évidence du concept de "lien profond" s'est faite sur des messages de listes de discussion. En effet, l'essentiel des discussions autour des projets open source se fait sur les listes de discussion, ce qui en fait l'endroit idéal pour retrouver des références à tous les autres aspects du projet, et donc aux outils utilisés pour gérer ces aspects.

Le concept de “crochet” est illustré sur des exemples de communication entre outils de développement, sur la base d’évènements. Ici encore, les pratiques de projets open source existants sont utilisées, ainsi que des exemples d’offres commerciales d’outils de développement basés sur les “crochets”.

7.2.1 Liens profonds

7.2.1.1 Récolte des données

Afin de récolter les données nécessaires pour l’étude du concept de “lien profond”, des centaines de listes de discussion de projets open source ont été consultées. La sélection initiale s’est faite, d’une part, sur la base de la disponibilité des archives de listes de discussion. En effet, tous les projets ne mettent pas leur liste de discussion en consultation ouverte sur internet. D’autre part, afin de ne pas surcharger les serveurs hébergeant les archives, une restriction supplémentaire a été faite aux projets dont les archives sont déjà groupées au moins par mois, réduisant ainsi considérablement le nombre de requêtes sur les serveurs. Enfin, une dernière réduction a été faite sur la base du nombre de messages et de l’âge du projet, résultant en une liste de 219 projets.

Des scripts shell ont été développés pour télécharger les archives de la liste de discussion à partir d’Internet. Les fichiers téléchargés sont au format “mbox”, un format standard utilisé par les logiciels de gestion de messages électroniques. Certains projets ont des archives complètes en un seul fichier, d’autres ont des archives par mois, qui peuvent être disponibles sous forme compressée ou pas, et mises en téléchargement via HTTP ou FTP. Des scripts distincts de téléchargement ont été écrits pour chacun de ces cas, afin d’automatiser les 15 587 requêtes nécessaires pour récupérer les données.

La sélection finale contient des projets open source majeurs comme PostgreSQL, QEMU, Emacs, Python, le Kernel Linux pour ARM, Mythtv, FreeBSD, et LLVM. Ces projets ont chacun des centaines de contributeurs, et sont donc appropriés pour une analyse de la collaboration. De plus, 135 projets dans la sélection finale ont de 1000 à 24000 messages sur leur liste de discussion et 204 projets ont de 30 mois à 17 ans d’âge.

7.2.1.2 Analyse générale des données

Le contenu d’un message de liste de discussion est du texte simple [1] contenant des références vers d’autres messages de liste de discussion et d’autres informations sur le projet, stockés dans des outils externes à la liste de discussion. Ces liens sont sous forme d’URLs inclus dans les messages.

7.2.1.2.1 Procédé d’analyse Un ensemble de scripts ont été développés pour analyser les archives de listes de discussion téléchargées, et en extraire des statistiques. Le traitement effectué est décomposé en étapes bien définies. Pour chaque étape, et pour chaque projet, le

1. On trouve aussi, bien que ce soit rare dans les projets open source, des messages au format HTML. Dans de tels cas, une version sous forme de texte simple est automatiquement extraite avant d’être analysée.

Indicateur	Minimum	Maximum	Moyenne
Nombre de mois	5	207	93
Nombre de liens	0	329 840	11 895
Nombre de liens profonds	0	283 665	8 803
Nombre de liens uniques	0	48 356	2 448
Nombre de messages	15	247 489	11 292
Nombre de messages par mois	0,30	2711,18	107,78
Nombre de liens par message	0	16,86	1,21
Fréquence des liens profonds	0,25	1	0,74

TABLE 7.1 – Principaux indicateurs sur les projets open sources étudiés

script shell correspondant à la réalisation de l'étape est généré à partir d'un gabarit et enregistré sur le disque, avant d'être exécuté. Ceci permet de faire des adaptations manuelles selon la nature du projet, tout en conservant la possibilité de piloter l'exécution de chaque étape de manière uniforme pour tous les projets. Les résultats de chaque étape sont aussi enregistrés afin de pouvoir être réutilisés. Sur chacun des projets retenus, après le téléchargement des archives, les traitements suivants sont effectués :

- Extraire tous les liens des archives téléchargées, et les enregistrer dans un fichier texte, avec un lien par ligne.
- Générer un rapport, à partir de la liste de liens, qui montre pour chaque lien le nombre d'occurrences. Ce rapport est trié des liens les moins fréquents aux plus fréquents.
- Déterminer, pour chaque lien, si c'est un "lien profond" ou pas, puis compter le nombre de liens profonds.
- Calculer l'âge de la liste de discussion, on considérant le message le plus ancien et le message le plus récent.
- Compter le nombre de messages échangés sur la liste de discussion depuis sa création.
- Calculer divers indicateurs comme le nombre moyen de messages par mois, le nombre moyen de liens par message, la proportion de lien profonds, etc., à partir des totaux précédents.

7.2.1.2.2 Résultats d'analyse L'étude précédente a permis de constater que l'utilisation des liens est courante sur les listes de discussion. Sur les 219 projets étudiés, il y a en moyenne 1,21 liens par message (pouvant aller jusqu'à 16 liens par message). 74% des liens sont des liens profonds dans le sens où ils pointent vers une information précise dans un système externe (et non pas juste vers une page d'accueil). Le tableau 7.1 résume les résultats obtenus.

7.2.1.3 Étude de la nature des liens dans le projet PostgreSQL

Les liens contenus dans les messages pointent vers une grande variété de ressources. Une analyse supplémentaire sur la nature de ces ressources a été faite pour la plus grande liste de discussion analysée, pg-hackers. Cette analyse a pour but de mettre en évidence les besoins auxquels répond l'utilisation des liens profonds.

7.2.1.3.1 Aperçu général La liste pg-hackers est la liste de discussion principale consacrée au développement du projet PostgreSQL, un système de gestion de bases de données. L'archive pg-hackers téléchargée représente 247 489 messages échangés sur plus de 14 années (entre Janvier 1997 et Mai 2012), contenant 120 325 liens dont 17 873 sont uniques (soit environ 0,48 liens par message). Les liens présents pointent, par exemple, vers les types de ressources suivants :

- Sites FTP ou BitTorrent de téléchargement de PostgreSQL, de son code source, ou de la documentation :
  ftp://ftp2.uk.postgresql.org
  http://bt.postgresql.org
- Sites ftp de téléchargement de produits tiers :
  ftp://ftp.debian.org
- Documentation sur les standards pertinents pour le projet :
  ftp://sqlstandards.org
- Pages personnelles de développeurs :
  http://developer.postgresql.org/~adunstan/
- Données de gestion de version internes au projet :
  http://anoncvs.postgresql.org
- Données de gestion de projets tiers :
  http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git
- Archives de listes de discussion internes au projet :
  http://archives.postgresql.org
- Publications sur des blogs :
  http://blogs.sun.com/bonwick/entry/zfs_end_to_end_data
- Fiches de bug externes au projet :
  http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=139389
- Serveur d'intégration continue :
  http://buildfarm.postgresql.org/cgi-bin/show_log.pl?nm=crake&dt=2011-07-12%2022:32:02
- Articles de recherche :
  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.7100

7.2.1.3.2 Étude détaillée et résultats Le contenu de certains liens peut être inféré à partir de leur structure et des mots clés présents dans leurs URLs. Des heuristiques ont donc été développées pour déterminer la nature de l'information vers laquelle pointe chacun de liens. Par exemple, un lien pointant vers une page du domaine archives.postgresql.org parle d'un échange sur une liste de discussion. Sur le sous-ensemble de 22 623 liens pour lesquels cette inférence peut être réalisée, on obtient la répartition de liens du tableau 7.2.

Cette répartition montre la prépondérance des références à la documentation et aux discussions précédentes. Cette utilisation n'est possible que parce que les systèmes logiciels (archive de liste de discussion, site web de documentation) qui contiennent l'information permettent de la référencer de manière relativement précise par une URL. Dans le cas contraire,

les références se limiteraient à des "indications" pour retrouver l'information, comme "voir la discussion que j'ai eu avec <participant> il y a 3 mois au sujet de <thème>" ou "voir la section de la documentation qui parle des problèmes avec <thème> sous <système d'exploitation>". Cependant, cette manière de communiquer ralentit toute l'équipe, parce qu'il faut un travail de recherche chaque fois qu'une telle référence est donnée.

Cette utilité des liens profonds est encore plus manifeste quand on considère que les liens vers les données de gestion de versions sont des plus fréquents. En effet, l'information de gestion de version est capitale, entre autres, dans la résolution des bugs. Elle permet d'identifier les changements qui ont introduit les bugs, de retrouver les justifications avancées pour ces changements, et d'évaluer les sous-systèmes impactés en analysant l'historique du projet. Le système de gestion de versions met à disposition des liens profonds vers des informations précises comme les fichiers modifiés dans un commit particulier. Ceci facilite l'analyse collaborative d'un bug, en mettant l'information utile à portée de clic.

Cet exemple montre que le système de gestion de version ne se contente pas simplement de gérer l'évolution des informations de version. Il fournit, d'une part, des concepts structurant cette information (commit, branche, auteur de commit, merge, etc.). D'autre part, il associe à ces informations des URLs qui forment un langage de requête sur les données de gestion de version. Ceci permet lors des échanges entre développeurs d'utiliser facilement cette information pour construire de la connaissance partagée, ce qui est crucial dans toute situation de collaboration [Rosc 94]. En l'absence d'une telle fonctionnalité, les développeurs sont réduits à décrire à leurs pairs les étapes nécessaires pour retrouver l'information, ce qui introduit des lourdeurs, et donc des frictions [Booc 03] dans la collaboration. Autrement dit, l'information qui contribue à la collaboration ne doit pas seulement être utile, mais aussi facilement accessible.

La mise à disposition de l'information de processus à travers des liens profonds permet d'exploiter cette information (description d'une tâche, deadlines, disponibilité des participants, état d'avancement des tâches, etc.) dans les outils de discussion. Ainsi, l'information de processus ne sera plus, de part la conception des outils qui la gèrent, détachée et difficile à relier aux préoccupations journalières des participants à un projet logiciel.

7.2.2 Crochets

Les crochets (hooks) sont très utilisés pour l'intégration ad hoc de services. Les web-hooks en particuliers sont très utilisés dans les services populaires de plates-formes de gestion de projet logiciel (Github, Bitbucket, Assembla, Google Code) ainsi que dans divers services disponibles sur internet (Facebook, Paypal, ZenDesk, MailChimp, Amazon, etc.). Les web-hooks sont mis en œuvre en envoyant une requête POST sur une URL configurée à l'avance, afin de notifier un système tiers d'un évènement auquel il a souscrit.

Les participants à un projet logiciel consultent l'information liée à une préoccupation de développement collaboratif, puis réagissent à l'information consultée. Cependant, certaines des réactions peuvent ne pas nécessiter une intervention humaine, et peuvent donc être automatisées. Dans ces cas, les hooks sont utilisés, dans la mesure où ils permettent de recevoir une information dès qu'elle devient disponible, et d'exécuter une action en réaction.

Type de lien	Nombre de liens	Nombre de liens uniques	Fréquence de répétition de liens
Standards industriels	25	14	1.78
Sous projets PostgreSQL	64	36	1.77
Gestionnaires de bug tiers	73	5	14.60
Bouts de code partagés en ligne	102	25	4.08
Données de gestion de version de projets tiers	106	70	1.51
Messages sur des forums internet	194	128	1.51
Pages personnelles de développeurs	292	93	3.13
Articles de recherche académique	339	93	3.64
Liste de discussion de projets tiers	407	254	1.60
Bugs PostgreSQL	515	267	1.92
Intégration continue PostgreSQL	729	292	2.49
Encyclopédies, dictionnaires, etc.	733	328	2.23
Téléchargement de logiciels PostgreSQL	734	270	2.71
Documentation de projets tiers	876	441	1.98
Téléchargement de logiciels tiers	1306	616	2.12
Données de gestion de version PostgreSQL	1651	586	2.81
Autres listes de discussion PostgreSQL	3002	1396	2.15
Mailing-list principale PostgreSQL	5728	2398	2.38
Documentation PostgreSQL	7074	1870	3.78

TABLE 7.2 – Distribution de liens sur la liste de discussion principale du projet PostgreSQL

La plupart des projets étudiés ont des listes de discussion dédiées aux commits [2]. Les messages dans ces listes sont envoyés par des crochets qui sont activés chaque fois qu'un commit est effectué (cas des SCM centralisés) ou une mise à jour envoyée sur le dépôt officiel du projet (cas des SCM décentralisés). Les développeurs intéressés par les nouvelles contributions à un sous-projet particulier ont donc juste besoin de s'abonner à la liste associée. Cette automatisation de l'accès à l'information n'est possible que parce que le système de gestion de version permet de définir des crochets pour l'événement "commit" (système centralisé) ou "post-receive" (système décentralisé).

Le projet PostgreSQL par exemple a une liste de discussion spéciale à laquelle les notifications de commit (dans CVS et GIT) sont envoyées [3]. Chaque message de notification contient une courte description, le nom de la branche, un résumé des changements, et un lien profond vers plus de détails sur le commit, mis à disposition par l'outil de gestion de version. La figure 7.1 montre une exemple de notification sur la liste de discussion pgsql-commiters [4].

```
Fix syslogger so that log_truncate_on_rotation works in the first rotation.

In the original coding of the log rotation stuff, we did not bother to
[snipped]

Branch
------
REL9_2_STABLE

Details
-------
http://git.postgresql.org/pg/commitdiff/63aba79c7f1f06422b22e2b44fdcb563bbc3f7a5

Modified Files
--------------
src/backend/postmaster/postmaster.c |    6 ++++-
src/backend/postmaster/syslogger.c  |   43 ++++++++++++++++++++++++----------
2 files changed, 35 insertions(+), 14 deletions(-)
```

FIGURE 7.1 – Exemple de notification de commit sur une liste de discussion (PostgreSQL)

Les crochets permettent aussi d'automatiser l'enrichissement de rapports avec des informations de contexte. On peut trouver une tel exemple dans Launchpad, l'outil de gestion de projet de la fondation Ubuntu. Des crochets sont responsables de la mise à jour automatique de rapports de bug suite à un commit ou un release qui les référence. De plus, dans le cadre d'opérations régulières de maintenance comme la détection de duplication de rapports de bug, un programme nommé "Launchpad janitor" écoute les événements de création de bug, et analyse les descriptions afin de trouver des duplications.

En mettant à disposition les événements de processus, et en gérant les crochets associés, il devient possible d'utiliser les événements des processus logiciels dans cette stratégie d'auto-

2. Ces listes sont souvent nommées "<nom du projet>-cvscommits" ou "<nom du projet>-commits"
3. http://archives.postgresql.org/pgsql-committers/
4. http://archives.postgresql.org/pgsql-committers/2012-07/msg00291.php

matisation de certaines actions routinières. Ces actions routinières s'assurent que des préoccupations périphériques au projet ou dues à la multiplicité d'intervenants ne compromettent pas les efforts de collaboration.

7.3 Support du développement logiciel avec le serveur CMSPEM

Cette section a pour but de montrer l'intérêt de l'intégration des données et événements de processus dans les outils de support au développement logiciel. Pour ce faire, nous considérons un ensemble de problématiques courantes dans un projet de développement logiciel collaboratif, et montrons comment la mise à disposition des données et événements de processus peut contribuer aux solutions de ces problématiques.

Les démonstrations sont faites avec le serveur CMSPEM, dont l'implémentation est décrite dans la section 6.3. De plus, pour chaque cas de figure, nous décrivons des utilitaires additionnels requis pour implémenter la solution proposée. La section est introduite et illustrée par un exemple de développement collaboratif, dont certains aspects sont modélisés en CMSPEM.

7.3.1 Exemple de modèle de développement collaboratif CMSPEM

Considérons un exemple fictif, mais réaliste, de système complexe de réservation de tickets pour événements sportifs, désigné ci-après par "projet SCRT". Le projet développe une application web de réservation, avec des interfaces utilisateur additionnelles sous forme d'applications natives. Une stratégie de déploiement continu est utilisée. En effet, de nouvelles fonctionnalités sont régulièrement ajoutées à la demande du client et sont implémentées par l'équipe, puis envoyées en production. L'équipe travaillant sur le projet SCRT est constituée comme suit :

- Bob, le concepteur. Bob travaille sur des modèles de description d'architecture logicielle. Ces modèles sont utilisés pour générer des interfaces et des classes de conversion de données.
- Alice, la responsable de l'intégration. Le rôle d'Alice est de décider qu'un ensemble de fonctionnalités et de correctifs sont prêts à aller en production, puis de fusionner ces éléments tout en s'assurant que l'ensemble est fonctionnel et raisonnablement sans défauts.
- Fred, le responsable du déploiement. Fred est responsable de l'envoi du système développé en production, du suivi de son exécution, et de la notification des problèmes rencontrés aux développeurs.
- Karl et Mike, des développeurs. Ils implémentent la majorité du système, et développent les intégrations nécessaires avec des systèmes tiers.
- Tracy, un développeur. Elle développe des plans de test, et implémente les tests d'intégration.

La plupart des fonctionnalités implémentées sont originales, d'où la nécessité de faire des expérimentations et de les valider rapidement par les retours recueillis auprès des utilisateurs

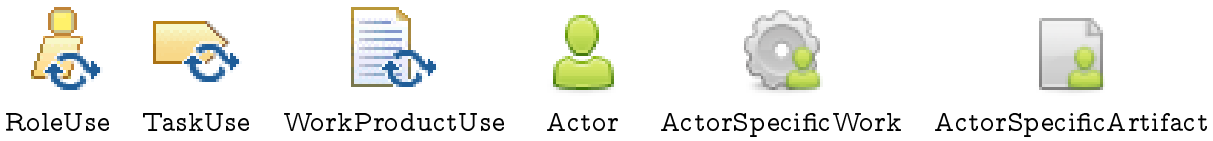

TABLE 7.3 – Icônes utilisées dans la description du projet SCRT

finaux. L'équipe est géographiquement distribuée, et utilise un système de gestion de versions distribué, afin d'exploiter les fonctionnalités de branches et de manipulation locale d'historique nécessaires dans une telle configuration.

Les participants au projet et leurs rôles sont les suivants : Bob (Architecte), Alice (Responsable de l'intégration), Fred (Responsable du déploiement), Karl (Développeur), Mike (Développeur), Tracy (Développeur). Les tâches à effectuer sont affectées comme suit : "Concevoir l'architecture" (Bob), "Implémenter le système" (Karl et Mike), "Intégrer et déployer" (Alice), et "Écrire les tests" (Tracy).

Dans les figures illustrant le modèle de l'exemple, les icônes de la table 7.3 sont utilisées. Afin d'améliorer la lisibilité de l'exemple, lorsque les acteurs sont représentés, les rôles qui leur sont associés sont placés en dessous de leurs noms, entre crochets. La figure 7.2 montre le modèle global, avec uniquement les éléments de modèle du niveau SPEM.

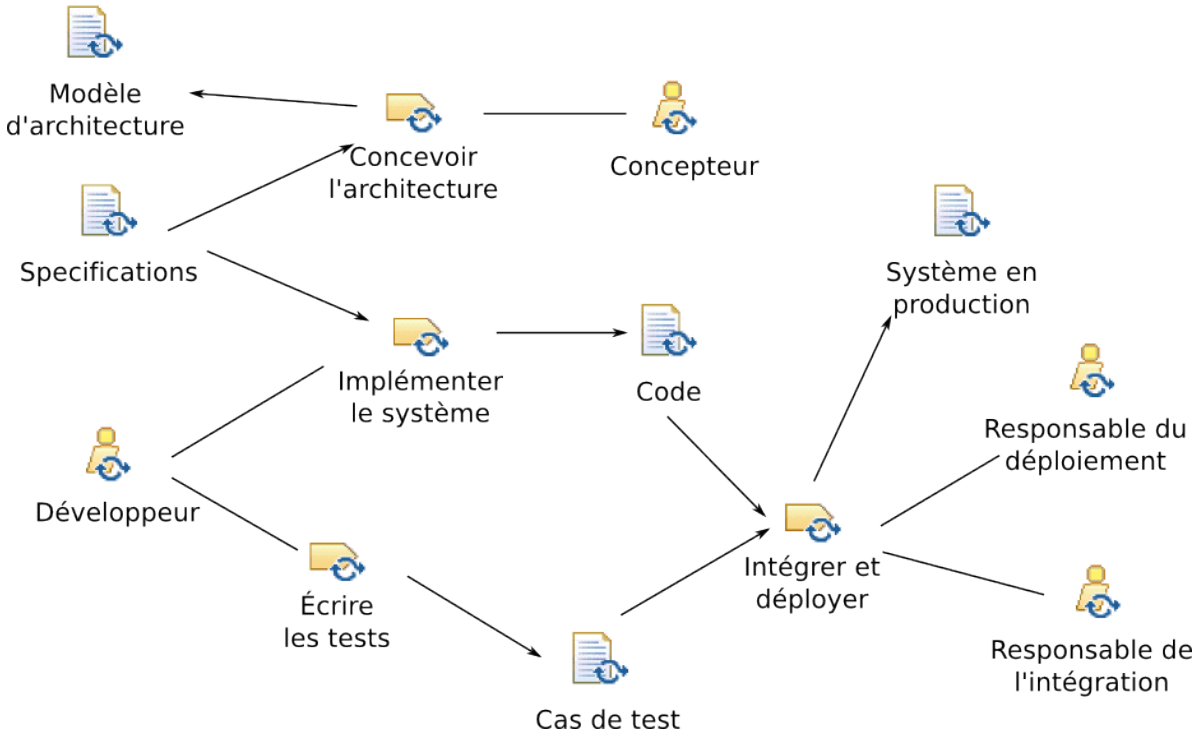

FIGURE 7.2 – Vue globale du processus du projet SCRT

Les participants au projet sont chacun modélisé avec le concept `Actor`, et leur rôle avec le concept `RoleUse` de SPEM. Pour chaque `TaskUse`, il est possible d'expliciter la participation

de chaque personne qui y contribue à l'aide de `ActorSpecificWorks`. Dans le cas du `TaskUse` "Implémenter le système" par exemple, deux `ActorSpecificWorks`, "Implémenter le backend" et "Implémenter le frontend" sont créés, et chacun affecté à un acteur différent (voir figure 7.3.

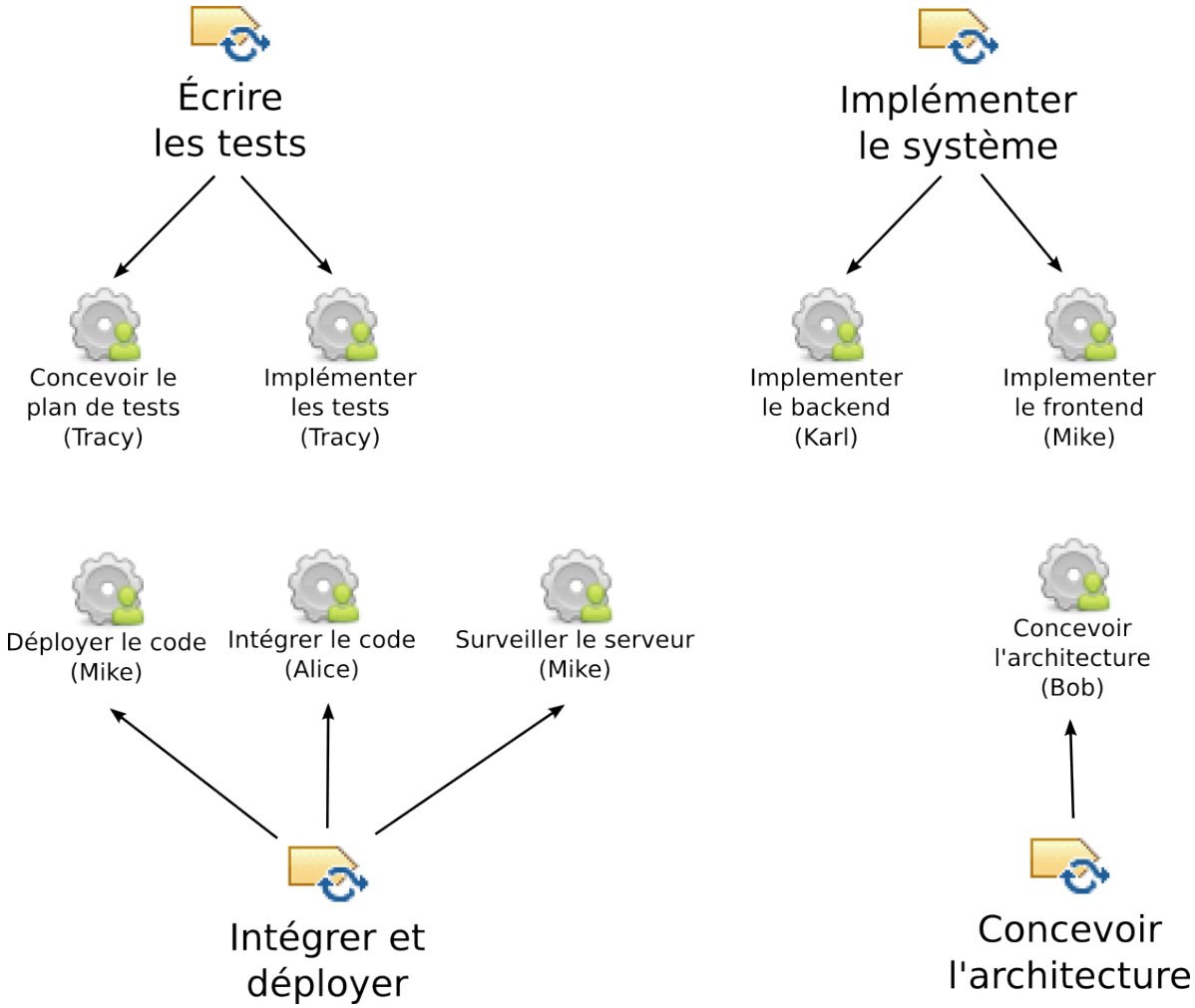

FIGURE 7.3 – Affectations des tâches spécifiques aux acteurs. Les flèches représentent la contribution d'un `ActorSpecificWork` à un `TaskUse` (référence `contributingActorSpecificWork` de `RoleUse`). Le nom de l'acteur affecté à l'`ActorSpecificWork` est marqué entre parenthèses.

Dans le projet SCRT, chaque fonctionnalité est d'abord prototypée dans le dépôt local d'un développeur, puis récupérée dans le dépôt du responsable de l'intégration, avant d'être envoyée dans le dépôt officiel du projet, qui est utilisé en production. Les développeurs mettent à jour leur dépôt local en récupérant les modifications récentes à partir du dépôt officiel. Un ensemble initial de relations entre acteurs peut être introduit, (figure 7.4), afin de refléter cette organisation :

– "PullsFrom", qui spécifie qu'un acteur a1 met à jour son dépôt local à partir du dépôt

d'un acteur a2 par défaut.

– "PushesTo" qui spécifie qu'un acteur a1 envoie par défaut ses contributions vers le dépôt de l'acteur a2.

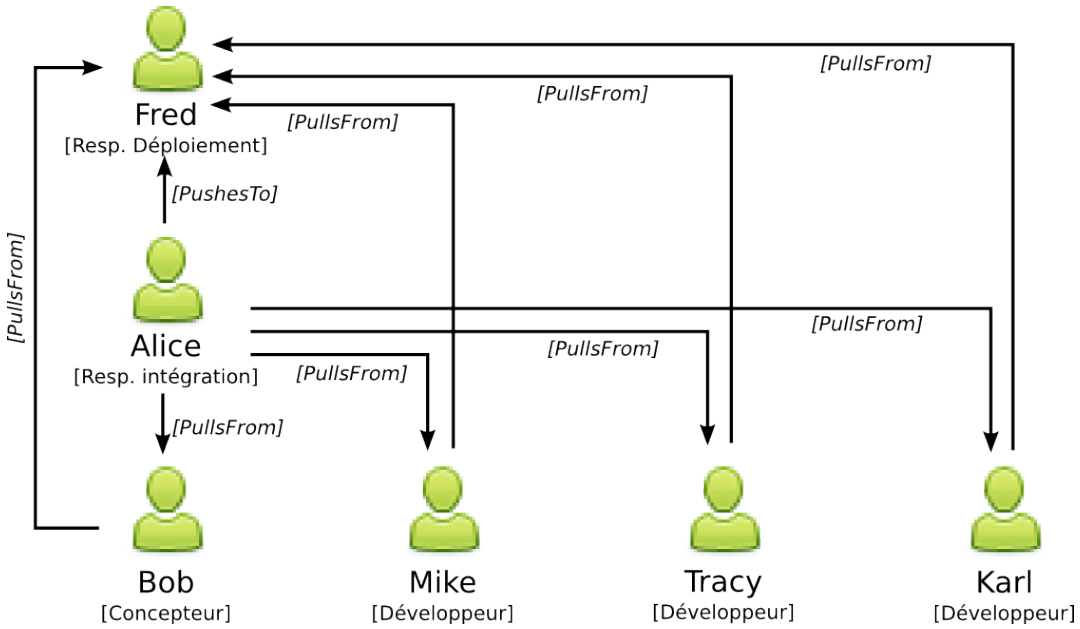

FIGURE 7.4 – Relations entre acteurs dans le modèle initial

Ce modèle initial peut subir des évolutions selon la progression du projet. Par exemple, l'équipe peut se rendre compte que le code de validation des entrées est répétitif, et décider de convertir ce code en un modèle de validation, supporté par des générateurs de code. Une nouvelle participante, Sue, qui joue le rôle de concepteur, est ajoutée à l'équipe, afin d'assister Bob dans cette tâche. Sue a pour instruction d'envoyer ses contributions à Bob, qui agit maintenant comme un intégrateur de sous-système. La conséquence pour le modèle est que la relation "PullsFrom", qui aurait dû être créée entre Alice et Sue, est créée entre Bob et Sue (figure 7.5).

Une autre évolution de modèle est déclenchée par la découverte d'un bug, grâce à des tests d'intégration implémentés par Tracy. Le bug est bloquant, et demande une collaboration étroite entre Bob (le concepteur), et tous les développeurs (Karl, Mike, et Tracy).

Karl s'occupe de la coordination des efforts sur cette investigation de bug particulière. L'équipe réalise très vite qu'une organisation différente est utilisée pour les besoins de l'analyse et de la résolution du bug. L'effort prenant plusieurs jours, l'équipe décide d'introduire la nouvelle organisation temporaire dans le modèle de processus, afin de bénéficier pleinement de l'assistance des outils qui exploitent le modèle de processus.

Une nouvelle branche est créée dans l'outil de gestion de versions pour la résolution du bug. Karl joue le rôle d'intégrateur sur cette branche, par rapport aux autres développeurs, jusqu'à ce que le problème soit résolu, ce qui résulte en l'organisation de la figure 7.6.

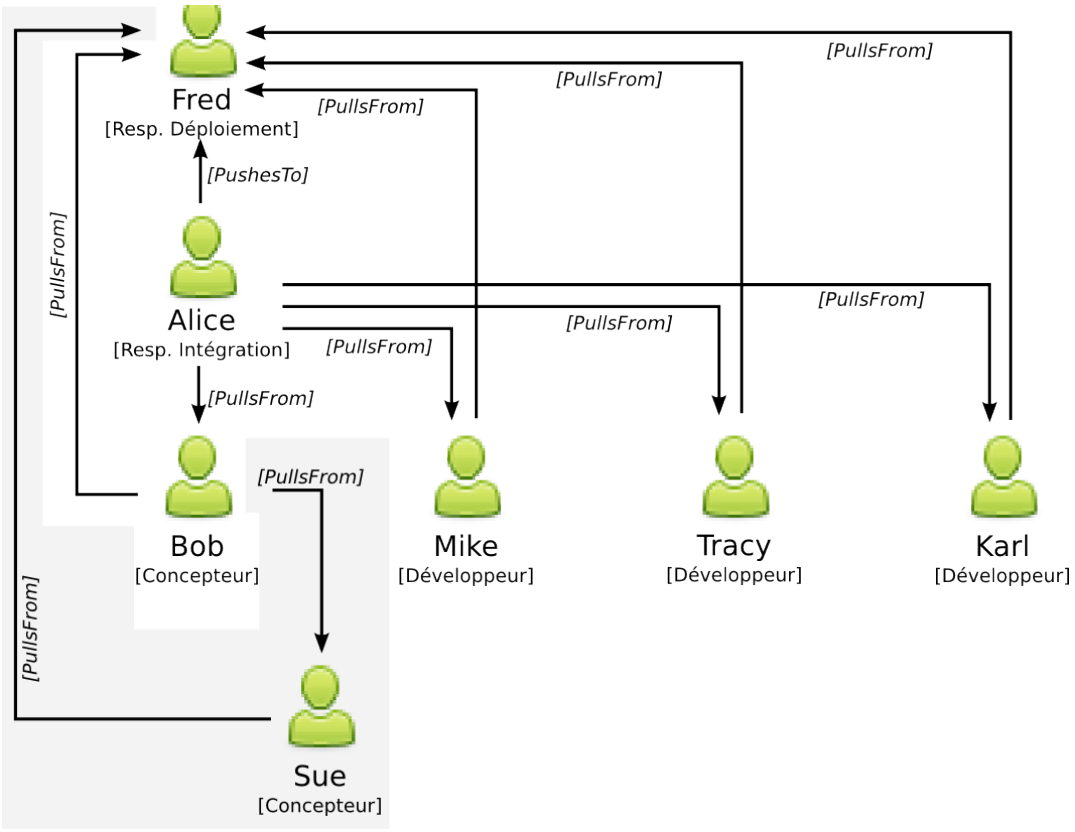

FIGURE 7.5 – Relations entre acteurs après l'intégration de Sue dans l'équipe

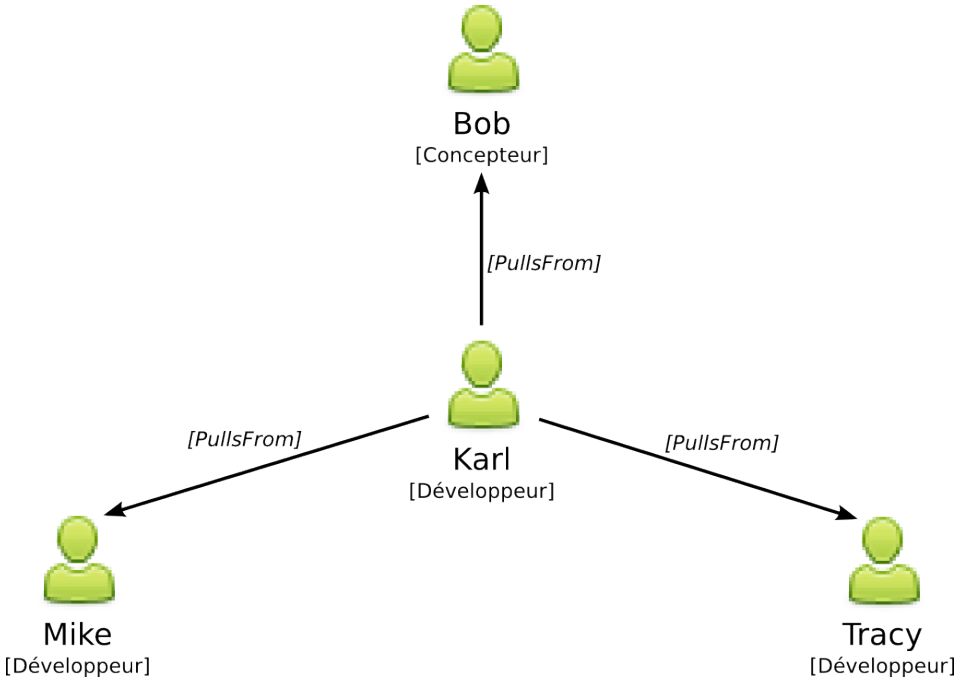

FIGURE 7.6 – Relations entre acteurs pour la résolution d'un bug

7.3.2 Ajout d'informations contextuelles dans les notifications

Lors de la création d'un ticket de bug ou lorsqu'il y a un échec de déploiement, des notifications sont typiquement envoyées aux participants d'un projet logiciel. Dans le modèle de l'exemple précédent, il peut s'agir d'un cas de test qui échoue reporté par Tracy, ou d'un problème en production signalé par Fred. A la réception d'une telle notification, un développeur doit rassembler des informations de contexte avant d'agir. Ces informations peuvent porter sur les autres développeurs concernés par la notification et leur disponibilité, les tâches dépendantes ou liées et leur statut, etc. Une partie de cette information contextuelle est disponible dans le modèle de processus, et peut être facilement récupérée sur le serveur de processus CMSPEM.

Notre but ici est de mettre en place un utilitaire qui évite au développeur d'aller chercher cette information, en la mettant automatiquement à disposition. Dans le cas d'une notification de création d'un rapport de bug par exemple, la description du rapport peut contenir des références à des participants au projet, ou à des tâches. La mission de l'utilitaire est d'analyser automatiquement cette description, de rechercher des informations sur les tâches et acteurs référencés, puis d'ajouter un commentaire au rapport de bug, contenant l'information rassemblée. Les principales étapes dans le fonctionnement d'un tel utilitaire sont les suivantes :

- Souscrire à l'évènement de création de bug dans un système de gestion de défauts logiciels, en utilisant son API. Cette opération est réalisée une fois pour toutes, et dépend du service tiers utilisé.
- Chaque fois qu'une notification de création de bug est reçue, envoyer une requête à l'outil de gestion de bug pour en récupérer la description. Cette description est analysée pour en extraire des références à des tâches et acteurs. Ceci suppose qu'il existe une manière conventionnelle de faire référence à ces éléments de processus dans du texte. Pour les besoins de la démonstration, nous considérons "asw#XXX" comme une référence à l'`ActorSpecificWork` d'identifiant XXX, et "a#YYY" comme une référence à l'`Actor` d'identifiant YYY.
- Si des références sont trouvées, envoyer autant de requêtes de lecture que nécessaire sur le serveur CMSPEM, afin de récupérer des informations d'état sur les `ActorSpecificWorks` et `Actors` identifiés à l'étape précédente.
- Construire un commentaire présentant les informations recueillies, et utiliser l'API du gestionnaire de bugs pour l'ajouter au rapport de bug concerné.

Une implémentation d'un tel utilitaire, utilisant l'API de gestion de bugs de l'outil collaboratif en ligne Gihub [5], a été réalisée. La souscription à l'événement de création de rapport de bug avec l'API Gihub se fait avec une simple requête HTTP qui précise l'adresse à laquelle les notifications de création de rapport de bug seront envoyées. Dans notre cas, cette adresse est une URL mise à disposition par l'utilitaire développé.

Une exécution de l'utilitaire génère le commentaire automatique de la figure 7.7. Un rapport de bug a été créé, et référence de concepts du modèle de processus tels que des partici-

5. https://github.com

pants et des tâches. Des informations de contexte sur ces tâches (deadline, etc.) et ces participants sont automatiquement mises à disposition des développeurs, leur permettant d'exploiter l'information de processus pertinente au rapport de bug. De plus, le commentaire contient des liens profonds vers les éléments de modèle pertinents, afin que les développeurs puissent avoir plus d'information en suivant le lien.

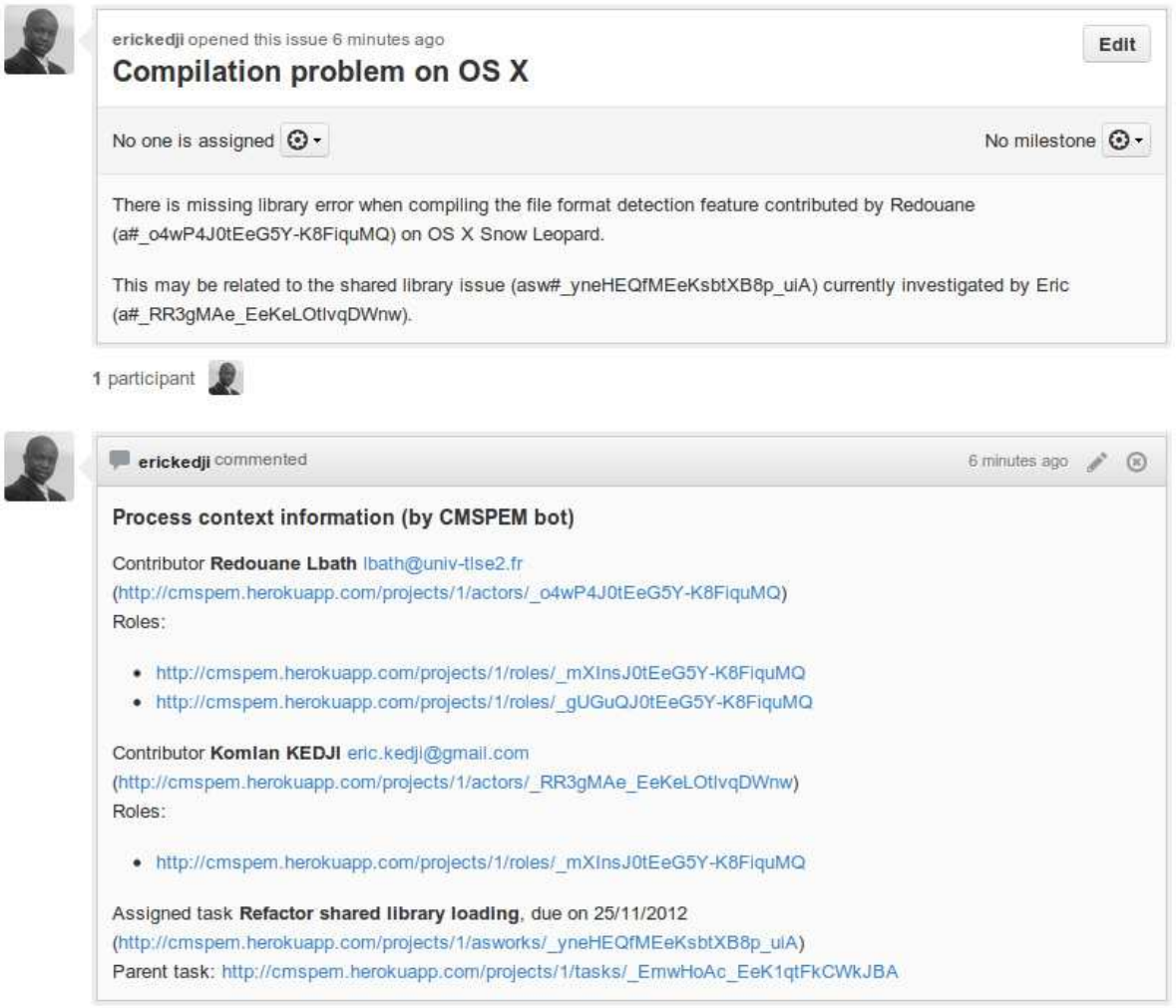

FIGURE 7.7 – Un commentaire sur un ticket de bug offrant des informations de contexte.

Les informations d'état sur une tâche et sur un acteur étant liées au temps, les commentaires laissés par l'utilitaire finiront par devenir obsolètes. Cependant, l'exposition des événements de processus par le serveur CMSPEM permet d'être alerté des changements, et d'y réagir. Il suffit donc de surveiller ces événements, et d'y réagir en mettant à jour le commentaire laissé, ou en en ajoutant un autre.

Dans l'utilitaire implémenté, après la création du commentaire initial avec l'information de contexte, le suivi des événements associés aux éléments de modèle référencés est configuré. Les étapes de l'exécution sont les suivantes :

- Souscrire aux événements clés de modification sur chacun des concepts référencés. Dans le cadre de cette démonstration, il s'agit des événements de disponibilité et d'indisponibilité pour les `Actors`, et de ceux de début, de fin, et de mise à jour pour les `ActorSpecificWorks`. Dans la souscription, l'utilitaire précise une URL mise à disposition pour traiter les futures notifications d'événement en provenance du serveur CMSPEM.
- Lorsqu'un événement de l'étape précédente survient sur le serveur CMSPEM, une notification est envoyée à l'adresse fournie par l'utilitaire.
- L'utilitaire prépare un message de mise à jour avec l'information relative à l'événement (indisponibilité d'un `Actor` par exemple), et poste le message en tant que commentaire sur le rapport de bug initial.

Enfin, l'utilitaire peut souscrire à l'événement de suppression de bug sur l'API Github, afin d'avoir l'opportunité d'annuler les souscriptions effectuées sur le serveur CMSPEM en relation avec ce rapport de bug particulier.

7.3.3 Génération de rapports

Les participants à un projet informatique ont souvent besoin de rendre des rapports hebdomadaires d'activité, surtout dans l'industrie. De tels rapports listent les tâches planifiées et réalisées, les tâches exceptionnelles, et le travail restant à faire sur les tâches assignées à l'individu.

L'information nécessaire pour produire de tels rapports est disponible dans les outils de gestion de bugs, dans les outils de contrôle de version, etc. Cependant, cette information doit être placée dans le contexte des tâches planifiées dans le modèle de processus. Les concepts introduits dans CMSPEM sont exactement ceux nécessaire pour réaliser une telle classification. Par exemple, le code écrit par un `Actor` dans un système de gestion de version est signé avec son adresse email. Les instances d'acteur dans un modèle CMSPEM ayant aussi un attribut "email", il est possible de retracer les contributions de code faites par chaque acteur.

Des exemples de telles remontées d'information sont les suivants :

- Auteur d'un commit $\mapsto$ Actor (en utilisant l'adresse email de l'auteur du commit, et la propriété "email" du concept Actor).
- Commit $\mapsto$ ActorSpecificWork (en utilisant une référence à l'ActorSpecificWork dans le message de commit).
- Commit $\mapsto$ ActorSpecificArtifact (en utilisant les noms des fichiers modifiés dans le commit)
- Auteur d'un rapport de bug $\mapsto$ Actor (en utilisant l'email de l'auteur du rapport de bug, et la propriété "email" du concept Actor).
- Rapport de bug $\mapsto$ ActorSpecificWork (en utilisant une référence à l'ActorSpecificWork dans la description du bug)

Ces correspondances entre les concepts de processus logiciel d'une part, et les concepts d'autres préoccupations de collaboration d'autre part, permettent d'obtenir une vue d'ensemble des contributions de chaque individu, vue d'ensemble qui peut être synthétisée en un rapport d'activité.

7.3.4 Activités de nettoyage

De manière générale, l'information disponible dans les systèmes de contrôle de version et ceux de gestion de bugs est plus détaillée que celle contenue dans les modèles de processus. La conséquence immédiate est qu'une action sur un seul élément de processus affecte tout un ensemble d'entités dans un système tiers. Ceci présente une opportunité d'automatisation, qui peut être implémentée en écoutant les évènements de processus, et en exécutant automatiquement les actions requises lorsque ces évènements se produisent.

Un cas typique est celui des actions de nettoyage. Par exemple, quand un membre d'une équipe de développement est temporairement indisponible ou remplacé (actions sur le modèle de processus), les rapports de bug qui lui sont assignés peuvent être automatiquement mis à jour, complétés avec un commentaire, etc. Le serveur CMSPEM rend possible l'implémentation de tels utilitaires, en exposant des évènements de processus tels que la disponibilité d'un acteur et la fin d'une tâche.

7.4 Conclusion

Le présent chapitre a illustré et validé l'approche conceptuelle du travail de thèse, ainsi que son implémentation, sur une série d'exemples.

En effet, nous avons montré la régularité avec laquelle les concepts de "lien profond" et de "crochet" reviennent dans le support au développement collaboratif. Pour ce faire, nous avons extrait des liens profonds et mis en évidence des réactions à des événements, en analysant les données de listes de discussion de 219 projets open source. Ceci montre que les développeurs sont familiers avec cette manière de concevoir le support au développement collaboratif. Un outil de support de processus logiciels qui embrasse cette approche pose donc peu de soucis d'adoption dus à la familiarité. De plus, un tel outil s'intègre facilement à l'environnement existant, comme le montre l'inclusion simple de liens vers des informations de projet dans des discussions de listes de discussion.

De plus, nous avons illustré la réaction aux événements de processus, telle que rendue possible par le serveur de processus CMSPEM. Pour ce faire, des intégrations du serveur CMSPEM avec l'outil de gestion de défauts logiciels du service en ligne Github ont été réalisées. Ces intégrations montrent comment la surveillance des événements de processus peut contribuer au support du développement collaboratif. Enfin, des pistes additionnelles (génération de rapport, activités de nettoyage) pour l'exploitation des données et événements de processus mis à disposition par le serveur CMSPEM ont été offertes.

Chapitre 8

Conclusion

Le présent travail de thèse s'est intéressé à la modélisation et à la mise en œuvre des processus collaboratifs ad hoc. Le travail s'est basé sur une conceptualisation du support au développement collaboratif, que nous avons appliqué aux processus logiciels. Nous avons proposé CMSPEM, un méta-modèle pour la modélisation des processus logiciels collaboratifs ad hoc, ainsi qu'un ensemble d'outils permettant d'exploiter les modèles de processus CMSPEM. Une série d'études et d'illustrations ont été développées, pour démontrer l'application et la pertinence de l'approche proposée.

Ce chapitre rappelle le contexte de la thèse et les enjeux de la problématique traitée. Il présente notre contribution et repositionne les différents apports dans la démarche globale de la thèse. Il analyse aussi la contribution, afin de relever ses limites et les nuances à considérer lors de l'exploitation des résultats. Enfin, il propose un ensemble de directions de recherche possibles, afin de compléter ou étendre la présente contribution.

8.1 Contributions

Le thème principal de cette thèse est le support au développement collaboratif à l'aide des processus logiciels. Nous nous sommes intéressés particulièrement à la collaboration ad hoc, dont l'étude de la littérature a montré la prépondérance en génie logiciel.

8.1.1 Conceptualisation du support au développement collaboratif

La réflexion a été construite autour d'une analyse préliminaire du support au développement collaboratif, en dehors du cadre des processus. Nous avons étudié les systèmes de gestion de versions et ceux de gestion de défauts logiciels, deux des outils de support à la collaboration les plus utilisés. Cette étude a permis d'identifier des éléments structurant la conception de ces systèmes, éléments que nous avons consolidés dans un modèle conceptuel du support au développement collaboratif.

Le modèle conceptuel définit un outil de support au développement collaboratif comme un produit logiciel résolvant un problème de développement collaboratif. Un tel problème met en

jeu un ensemble de préoccupations de développement collaboratif, qui sont des aspects de développement pouvant être décrits et suivis de manière indépendante. La description et le suivi d'une préoccupation est implémentée par un gestionnaire de préoccupation. Le gestionnaire offre un langage de requêtes pour consulter les données de la préoccupation, et des éditeurs natifs pour les manipuler. De plus, le gestionnaire permet à tout outil de développement d'observer l'évolution de la préoccupation, en exposant les événements clés qui rendent compte de cette évolution, et en permettant aux outils d'observer ces événements et d'y réagir.

Plusieurs réalités du développement collaboratif sont mises en relief par le modèle conceptuel. D'une part, la collaboration est un ensemble de préoccupations corrélées, qui, même si elles peuvent être traitées séparément quand il s'agit de stocker des données et de les modifier, doivent être interconnectées quand il s'agit d'être exploitées par un utilisateur final. En effet, un développeur logiciel ne passe pas d'une tâche de gestion de version à une tâche de gestion de défauts logiciels, puis à une tâche de gestion de processus logiciel. Un développeur a plutôt des tâches définies selon les besoins métier, dont chacune met en jeu différents aspects. La gestion séparée d'une préoccupation de développement collaboratif, qui se justifie par le principe de "séparation des préoccupations", doit donc être complétée par des mécanismes permettant de lier les préoccupations entre elles.

Une première nécessité pour lier des préoccupations de développement entre elles est que chaque préoccupation soit structurée par des concepts pouvant être simplement rattachés aux concepts des autres préoccupations. Par exemple, des liens peuvent être établis entre l'auteur d'une modification dans un système de gestion de versions, et les intervenants sur un bug dans un système de gestion de défauts logiciels. Une préoccupation dont le vocabulaire est à peine lié à celui des autres préoccupations va se retrouver naturellement isolée, dans la mesure où un changement significatif de contexte est nécessaire pour la prendre en compte. Autrement dit, la conceptualisation d'une préoccupation doit utiliser une sémantique proche de celle des discussions entre participants au projet. En complément de cette contrainte sur le choix des concepts, nous avons identifié deux mécanismes, qui sont centraux dans les gestionnaires de préoccupation étudiés, et qui permettent de lier les préoccupations entre elles.

D'une part, un gestionnaire de préoccupation doit permettre d'associer à chaque donnée de la préoccupation, susceptible d'être utilisée par d'autres outils, une adresse qui peut être stockée et transmise en dehors de la préoccupation. Ceci permet à des systèmes externes de référencer ces données ou de les utiliser à tout moment en déréférençant l'adresse associée. Cette approche permet de remplacer des instructions sur "comment retrouver une information" à l'intérieur d'une préoccupation, par une notation, comprise par le gestionnaire de la préoccupation, et facile à transmettre, qui *représente* l'information. Au delà d'une économie de notation, ceci permet de faire référence à des informations externes sans changement perturbateur de contexte, ce qui facilite la construction d'une connaissance partagée globale sur la collaboration.

D'autre part, le gestionnaire de préoccupation doit permettre de suivre l'évolution de la préoccupation "de l'extérieur", en exposant les événements significatifs qui structurent l'évolution des données de la préoccupation, et en permettant de réagir à ces événements. En effet, la collaboration ad hoc faisant l'objet de perpétuels changements, le support de la collaboration peut être vu comme une réaction à ces changements. Cette approche permet d'avoir un

intérêt sélectif dans une préoccupation de collaboration, sans avoir besoin de la suivre dans sa globalité. Il est de la responsabilité du gestionnaire de préoccupation de faciliter ce suivi, en définissant un catalogue d'événements et en permettant de souscrire aux événements et d'y réagir avec des "crochets".

8.1.2 Modélisation de processus logiciels ad hoc

Afin d'appliquer notre modèle conceptuel de support au développement collaboratif aux processus logiciels, nous avons défini le méta-modèle CMSPEM, qui étend la norme SPEM.

En effet, d'une part, nous nous sommes interrogés sur la pertinence des concepts existants de modélisation de processus pour décrire la collaboration ad hoc en génie logiciel. Nous avons noté comment la focalisation sur les problématiques de planification prive les approches existantes, dont le standard SPEM en particulier, de la capacité à fournir les concepts nécessaires pour comprendre et représenter les interactions intrinsèques à la collaboration ad hoc. Afin de remédier à ce manque, nous avons proposé des extensions à la norme SPEM, avec un double but.

Nous avons introduit en premier lieu le concept de participant au projet, ainsi que les concepts liés de tâche spécifique à un acteur, et d'artéfact spécifique à un acteur. Ces concepts répondent au besoin de rapprocher le vocabulaire des processus de celui utilisé dans les préoccupations courantes de collaboration. De plus, ces concepts sont liés par des relations qui précisent les rapports de collaboration entre participants, entre leurs tâches, ou encore entre les artéfacts qu'ils modifient dans leur travail.

D'autre part, nous avons noté que les instances des nouveaux concepts introduits dans CMSPEM et leurs relations peuvent changer fréquemment, et que ce changement rend compte des dynamiques de collaboration au sein d'un projet. Afin de suivre ces changements, nous avons proposé de représenter l'évolution de la configuration d'un projet par des événements, avec un mécanisme associé de réaction à ces événements. Nous avons équipé ce mécanisme de concepts de parenté d'élément de modèle et de remontée d'événement, afin de faciliter l'utilisation pratique des événements de processus par des outils tiers.

8.1.3 Apport des processus logiciels dans le support du développement collaboratif

En conformité avec le modèle conceptuel de support au développement collaboratif proposé, nous avons développé un ensemble d'outils de manipulation de processus logiciels CMSPEM, formant un gestionnaire de la préoccupation des processus logiciels.

En effet, d'une part, nous avons développé divers moyens de consulter les modèles de processus CMSPEM et d'en extraire de l'information. Il s'agit, premièrement, d'un éditeur graphique et d'un éditeur textuel équipé de vues graphiques dynamiques et personnalisables. Deuxièmement, un serveur de processus a été développé, dont les différentes URLs exposées (liens profonds) forment un langage de requêtes sur le modèle de processus.

D'autre part, les éditeurs graphique et textuel développés permettent de modifier globalement un modèle de processus CMSPEM. De plus, le serveur CMSPEM permet de faire des

mises à jour du processus, à un niveau de granularité extrêmement fin, afin de s'assurer qu'il reflète constamment la réalité du développement.

Enfin, le serveur CMSPEM implémente un mécanisme de souscription aux événements de processus, basé sur les web-hooks. Ce mécanisme permet à tout outil de support au développement collaboratif d'écouter des événements de processus et d'y réagir. Ainsi, nous réduisons les barrières à l'intégration des processus logiciels dans les préoccupations courantes des développeurs.

8.2 Discussion

Le but fixé pour ce travail de thèse était d'étudier la contribution des modèles de processus logiciels au support du développement collaboratif. Nous avons, pour cela, d'une part, proposé un modèle conceptuel du support au développement collaboratif. D'autre part, nous avons appliqué ce modèle conceptuel aux processus logiciels, en développant un méta-modèle et des outils associés.

Une première prise de position de ce travail porte sur la pertinence de guider la conception des outils basés sur les processus avec une étude de la conception des outils relatifs aux autres préoccupations de développement. En effet, les processus ont traditionnellement étés outillés par des systèmes supposés orchestrer les autres préoccupations de développement. Ceci a encouragé une approche de conception où la communication entre les outils de support de processus et les autres outils est à sens unique, l'information remontant toujours des autres outils vers l'outil de support au processus, et le flux de contrôle prenant le sens inverse. Cette approche contribue à l'isolement des préoccupations de processus, tout en rendant leurs coûts d'adoption prohibitifs.

Un autre argument défendu dans cette thèse est celui de la nécessité de relâcher certaines contraintes pour pouvoir prendre en compte l'évolution dynamique de la collaboration. En effet, nous nous concentrons, non plus sur la planification de comment un processus doit évoluer, mais sur la prise en compte de comment il évolue. Ce changement de perspective a des avantages évidents en termes d'adaptation à des comportements dynamiques. De plus, ce dynamisme est limité, dans la mesure où il s'applique principalement aux détails du processus, les éléments de granularité plus grande (rôle, produit, activité) pouvant rester inchangés. Cependant, il n'est pas tout à fait clair si l'impact de ces changements sur l'ensemble du processus est lui aussi limité. Par exemple, l'influence de l'indisponibilité d'un participant sur le processus peut varier largement selon l'implication de l'acteur dans les tâches auxquelles il participe, les relations de dépendance entre ces tâches et les autres tâches du processus, les contraintes de temps associées à la complétion de ces tâches, etc. Il n'y a actuellement aucun moyen d'évaluer l'étendue d'un tel impact.

L'outillage des processus, tel que nous le préconisons dans ce travail, repose sur le fait que les préoccupations de développement collaboratif sont liées entre elles, et doivent être utilisées de concert dans les outils de support à la collaboration. En particulier, nous défendons l'utilité des données et événements de processus en tant qu'information de contexte pour d'autres préoccupations de développement. Ceci justifie le fait de permettre, par conception, un accès

facile et uniforme à ces informations, puisqu'il n'est pas possible d'anticiper quel genre d'outil aura besoin d'exploiter l'information de processus.

Le serveur de processus développé communique sur le réseau via une interface HTTP, et s'intègre à des outils possédant une interface HTTP. Ce choix pragmatique est dû au fait que ce protocole est omniprésent, pose peu de problème avec les pare-feux d'entreprise, et surtout, a déjà montré son adéquation, dans le cadre du web, à l'intégration de systèmes distribués faiblement couplés. Cependant, tous les outils de développement ne possèdent pas une interface de communication HTTP, le plus souvent parce qu'ils ne communiquent pas du tout sur le réseau. L'utilisation du protocole HTTP n'élimine donc pas le besoin d'adaptateur reproché aux approches existantes. Cependant, ceci réduit considérablement ce besoin, en épousant une "norme de fait" d'intégration d'outils de développement collaboratif.

Les mises à jour du modèle de processus sont destructives. En effet, une fois qu'un élément de modèle est modifié ou supprimé sur le serveur, il n'est pas possible, dans l'implémentation réalisée, de retrouver l'information perdue. Cependant, cette information peut être importante pour des besoins d'audit par exemple. Il ne s'agit cependant pas tout simplement d'enregistrer une nouvelle version du modèle après chaque modification. En effet, l'information sur les anciennes versions n'est utile que si elle peut être facilement exploitée. Par exemple, on peut s'interroger sur les changements intervenus sur le modèle de processus entre deux dates arbitraires. Le serveur devrait pouvoir répondre avec un rapport de modifications de "haut niveau", groupant les changements par concepts. De plus, un tel rapport pourrait reconstituer les modifications logiques. En effet, une suppression suivie d'un ajout pourra être présentée comme une substitution, si les liens mettant en jeu l'élément supprimé et l'élément ajouté permettent de faire cette inférence.

8.3 Perspectives

Ce travail de thèse a proposé une manière de modéliser les processus collaboratifs ad hoc, et de suivre leur évolution à l'aide d'un serveur de processus.

Une des fonctionnalités du serveur de processus est la mise à disposition de liens profonds, donnant accès à des informations précises sur des éléments du modèle de processus. Dans l'implémentation présentée, ces données sont renvoyées par le serveur au format JSON, qui est approprié pour leur utilisation par des systèmes logiciels tiers. Cependant, comme l'a montré l'étude de validation réalisée sur les listes de discussion (voir section 7.2), les données d'une préoccupation peuvent être consultées directement par des agents humains. Dans ce cas, des formats plus appropriés sont possibles. Il serait donc pertinent d'étudier des représentations visuelles appropriées, sous forme de pages web, mises à disposition par le serveur de processus, et navigables avec des liens internes. Ce sera l'occasion d'étudier l'impact des métaphores visuelles [1] sur la compréhension du modèle de processus, et donc son exploitation dans le développement collaboratif.

1. Une métaphore visuelle est une correspondance établie entre des éléments graphiques d'une part, et d'autre part, des concepts et les relations qu'elles entretiennent. Par exemple, on peut représenter la disponibilité d'un participant, sur une période donnée, par une succession de bandes claires (correspondant à des absences) et de bandes sombres (correspondant à des présences) dont les longueurs sont proportionnelles aux durées.

Les modifications sur le modèle de processus correspondant à une série de requêtes sur le serveur CMSPEM, la consultation de l'historique des modifications donne une image globale de l'évolution d'un modèle de processus dans le temps. L'analyse d'un tel historique (attributs modifiés, dates des modifications, auteurs des modifications, etc.) a le potentiel de révéler de précieuses informations sur la dynamique des rapports de collaboration. En particulier, il serait pertinent de détecter des enchainements de modifications récurrents, et d'analyser ces enchainements pour mettre en évidence des mises à jour typiques faites à un projet logiciel. Ces mises à jour peuvent ensuite être réifiées en des changements nativement implémentés, qui peuvent être appliqués à un modèle de processus en une seule opération.

Le serveur de processus met aussi à disposition des événements de processus. Ces événements sont traités l'un après l'autre par les systèmes qui y souscrivent. Cependant, certaines séquences d'événements peuvent correspondre à une action logique sur le serveur de processus. Une étude plus approfondie permettrait d'identifier ces séquences, afin d'y associer des macro événements, simplifiant ainsi la tâche des outils tiers qui écoutent ces événements.

Les organisations de travail décrites par les modèles de processus CMSPEM sont chacune appropriée pour une situation de collaboration particulière. Ces situations se répètent probablement, avec des variations mineures, dans un projet, ou d'un projet à l'autre. Dans de telles circonstances, la partie correspondante du modèle de processus existant peut être copiée, et modifiée. Cependant, cette opération pourrait être systématisée, en la formalisant et en l'outillant. En effet, l'analyse de plusieurs situations de collaboration similaires devrait permettre d'identifier les points les plus fréquents de variations. Ces variations peuvent ensuite être conceptualisées, et leur application outillée sous forme d'opérateurs de modification de modèle. Ainsi, il devient possible pour une équipe de développement de disposer d'une collection de "patrons de collaboration", avec des opérateurs pour les appliquer à un nouveau projet ou à un projet en cours de déroulement.

Chapitre 9

Publications

9.1 Revues

– Komlan Akpédjé Kedji, Redouane Lbath, Bernard Coulette, and Mahmoud Nassar. Supporting collaborative development using process models : A tooled integration-focused approach (accepté). *Journal of Software : Evolution and process*, July 2013.

Cet article présente une architecture de support du développement collaboratif exploitant les informations de processus. L'approche est validée par une étude portant sur 219 projets open source, et illustrée par des utilitaires du support à la collaboration qui communiquent avec un serveur de processus.

– Komlan Akpédjé Kedji, Redouane Lbath, Bernard Coulette, and Mahmoud Nassar. Exploiting process events for the integration of collaborative software development tools. *Journal of Software Engineering*, November 2012.

Cet article présente une formalisation de la gestion d'évènement pour processus collaboratifs, et montre comment l'intégration des évènements aux modèles de processus facilite leur exploitation pour le support au développement.

9.2 Conférences

– Komlan Akpédjé Kedji, Redouane Lbath, Bernard Coulette, Mahmoud Nassar, Laurent Baresse, and Florin Racaru. Supporting collaborative development using process models : an integration-focused approach (regular paper). In *ICSSP -> International Conference on Software and System Process (co-located with ICSE), Zürich, 02/06/2012-03/06/2012*, http ://ieeexplore.ieee.org/, June 2012. IEEE.

Cet article analyse les considérations habituelles qui dirigent la création des outils de collaboration en génie logiciel, et les applique aux outils de gestion de processus logiciel. C'est ainsi que l'on identifie l'adressage et les points d'extension comme étant des mécanismes efficaces pour l'intégration des outils de support à la collaboration. L'article s'appuie sur ce constat pour proposer une architecture de support de processus,

qui puisse corriger les problèmes d'intégration des PSEE (environnements de développement centrés processus).

– Komlan Akpédjé Kedji, Bernard Coulette, Redouane Lbath, and Mahmoud Nassar. Modeling ad-hoc collaboration for automated process support. In *Software Quality Days 2012, January 17-19, 2011*. Springer, January 2012.

  Cet article s'appuie sur le constat selon lequel la majorité des situations collaboratives en ingénierie logicielle ne sont pas planifiées, pour proposer une manière d'offrir un support à ce genre de collaboration. Pour ce faire, les spécificités de la collaboration ad hoc sont conceptualisés dans un méta-modèle, CMSPEM, qui permet de décrire ce genre de situation de collaboration.

– Komlan Akpédjé Kedji, Bernard Coulette, Minh Tu Ton That, Redouane Lbath, Mahmoud Nassar, and Hanh Nhi Tran. Towards a tool-supported approach for collaborative process modeling and enactment. In *The Eighteenth Asia-Pacific Software Engineering Conference (APSEC 2011), Hochiminh, December 05-08, 2011*. IEEE, December 2011.

  Cette contribution décrit une démarche de description et de mise en œuvre de processus collaboratifs, et présente des outils qui ont été réalisés pour supporter l'approche (un éditeur de processus, et un outil de génération de plans de projets à partir du modèle de processus).

– Komlan Akpédjé Kedji, Bernard Coulette, Mahmoud Nassar, Redouane Lbath, and Minh Tu Thon That. Collaborative processes in the real world : Embracing their essential nature (regular paper). In *International Symposium on Model Driven Engineering : Software & Data Integration, Process Based Approaches and Tools. Colocated with ECMFA 2011 conference, Birmingham, June 06-07, 2011*, June 2011.

  Dans ce papier, nous identifions les caractéristiques essentielles des processus collaboratifs que les professionnels rencontrent sur le terrain. Nous démontrons que ces caractéristiques (formalisation progressive, processus discuté en terme de personnes réelles qui participent au projet, décomposition plus fine des tâches, etc.) ne sont pas prises en compte par le standard de modélisation de processus, SPEM. L'article décrit ensuite comment l'on peut prendre en compte ces manques, en rajoutant des extensions au méta-modèle SPEM.

– Komlan Akpédjé Kedji, Bernard Coulette, Mahmoud Nassar, and Redouane Lbath. Vers un modèle pour la description des processus IDM collaboratifs (regular paper). In *Journées sur l'Ingénierie Dirigée par les Modèles (IDM), Lille, June 07-08, 2011*, http ://www.editions-hermes.fr/, June 2011. Hermès.

  Cet article récapitule les motivations derrière la modélisation des processus collaboratifs IDM, et notre approche d'ensemble pour proposer un méta-modèle permettant de réaliser cette modélisation.

9.3 Rapports internes de laboratoire

– Komlan Akpédjé Kedji and Bernard Coulette. Definition of collaborative processes for model driven engineering - state of the art. Rapport de recherche IRIT/RR-2011-3-EN, Institut de Recherche en Informatique de Toulouse (IRIT), February 2011.

9.4 Rapports de projet

– Komlan Akpédjé Kedji, Bernard Coulette, and Redouane Lbath. Tailored MDCD process assistance definition. June 2012.

– Komlan Akpédjé Kedji, Bernard Coulette, Redouane Lbath, and Sophie Ebersold. Collaborative MDE process assistance definition. June 2011.

– Komlan Akpédjé Kedji, Bernard Coulette, Redouane Lbath, Sophie Ebersold, and Hanh Nhi Tran. Collaborative MDE process modeling. February 2011.

– Komlan Akpédjé Kedji, Bernard Coulette, Redouane Lbath, Sophie Ebersold, Jacques Robin, Wolfgang Kling, Patrick Vlaeminck, Florin Racaru, and Yves Bernard. Survey of academic research works and industrial approaches to model driven collaborative development. August 2010.

– Komlan Akpédjé Kedji, Bernard Coulette, Redouane Lbath, Sophie Ebersold, Hanh Nhi Tran, Jacques Robin, Wolfgang Kling, Florin Racaru, and Yves Bernard. Galaxy glossary. April 2010.

Chapitre 10

Annexes

10.1 Description des concepts de base de SPEM

Les trois concepts clés de la norme SPEM [OMG 07] réutilisés dans le méta-modèle CM-SPEM sont `RoleUse` (section 13.13 de la norme), `TaskUse`(section 13.14 de la norme), et `WorkProductUse`(section 9.11 de la norme). Nous reprenons ici les passages de la norme relatives à ces trois concepts, pour référence.

10.1.1 RoleUse

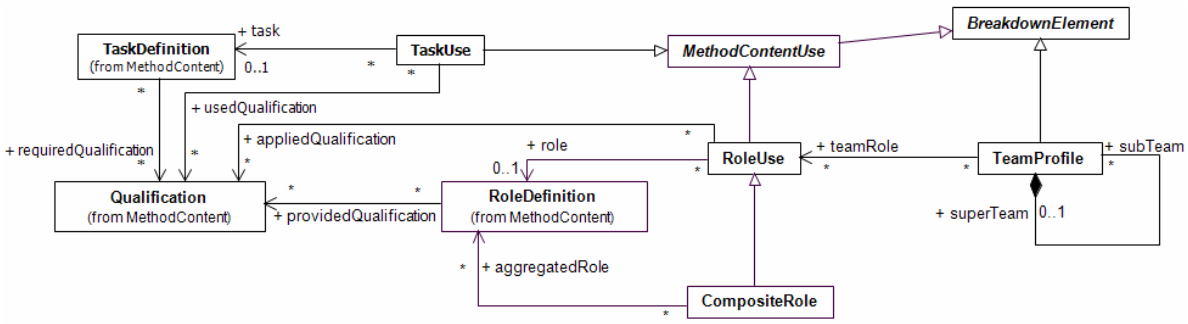

FIGURE 10.1 – Le concept `RoleUse` et ses concepts associés

A Role Use represents a Role in the context of one specific Activity. Every breakdown structure can define different relationships of Role Uses to Task Uses and Work Product Uses. Therefore, one role can be represented by many Role Uses, each within the context of an Activity with its own set of relationships.

10.1.2 TaskUse

A Task Use is a Method Content Use and Work Breakdown Element that represents a proxy for a Task Definition in the context of one specific Activity. Every breakdown structure can define different relationships of Task Uses to Work Product Uses and Role Uses. Therefore,

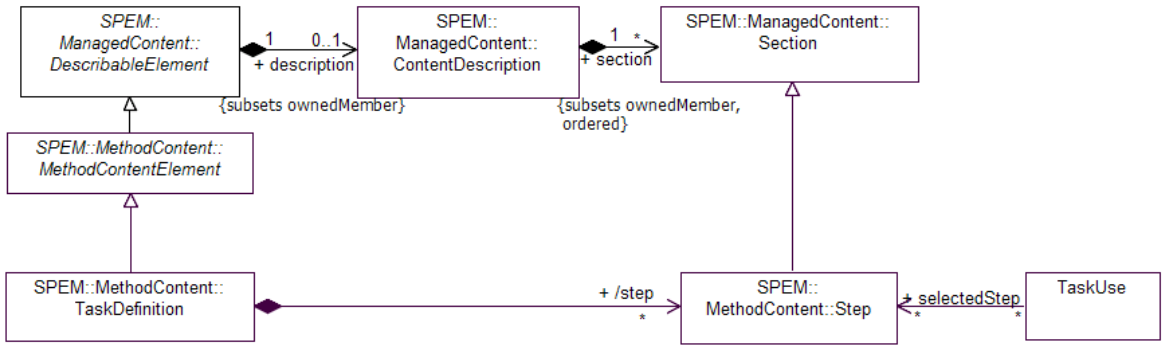

FIGURE 10.2 – Le concept TaskUse et ses concepts associés

one Task Definition can be represented by many Task Uses each within the context of an Activity with its own set of relationships.

A key difference between Method Content and Process is that a Method Content Element, such as Task Definition, describes all aspects of doing work defined around this task. This description is managed in steps, which are modeled as Sections of the Task Definitions' Content Descriptions. When applying a Task Definition in a Process' Activity with a Task Use, a Process Engineer needs to indicate that at that particular point in time in the Process definition for which the Task Use has been created, only a subset of steps shall be performed.

10.1.3 WorkProductUse

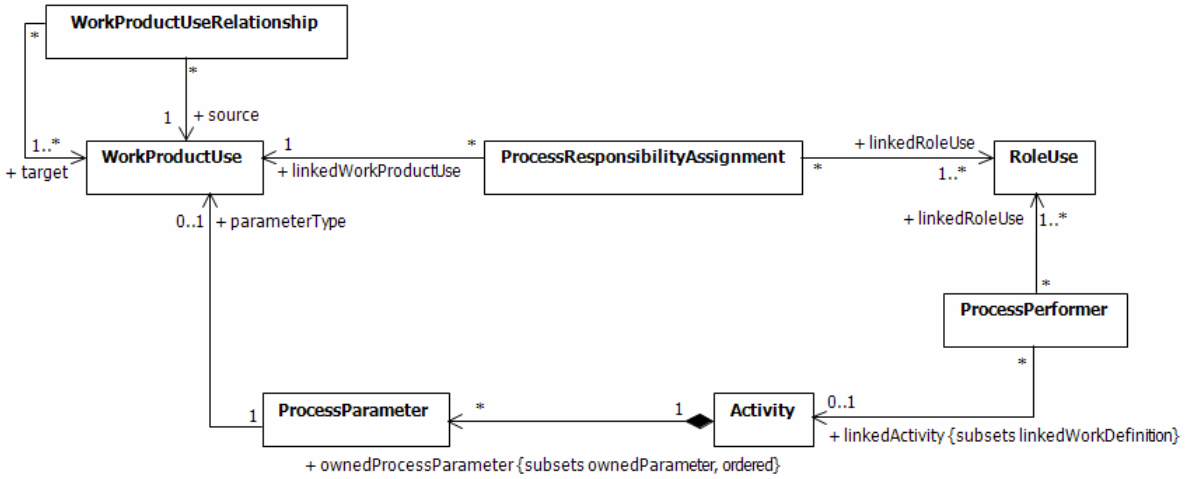

FIGURE 10.3 – Le concept WorkProductUse et ses concepts associés

A Work Product Use is a special Breakdown Element that either represents an input and/or output type for an Activity or represents a general participant of the Activity. If it is an input/output, then the Work Product Use needs to be related to the Activity via the Process Parameter class. If it is a participant, then the Work Product Use is stored in the nestedBreak-

downElement composition of the Activity and might be used by one of the sub-activities as an input/output and/or be related to a Role Use via a Process Responsibility Assignment. Role Use instances are only valid within and specific to the context of an Activity and not to be reused across activities.

10.2 Règles de bonne formation OCL du méta-modèle CMPSEM

10.2.1 Description informelle

- Le nom d'un Actor doit être unique dans tout le modèle de processus.
- L'adresse email d'un Actor doit être unique dans tout le modèle de processus.
- Le nom d'un ActorSpecificWork doit être unique parmi tous les ActorSpecificWorks liés à un TaskUse donné.
- Le nom d'un ActorSpecificArtifact doit être unique parmi tous les ActorSpecificArtifacts liés à un WorkProductUse donné.
- Si une ActorSpecificArtifact est une copie partielle du WorkProductUse qu'il représente, alors il existe au moins un ActorSpecificArtifact, distinct du premier, copie partielle du WorkProductUse en question.
- Si une TaskUse T est reliée à un seul ActorSpecificWork asw, et un RoleUse R est relié à un seul Actor a, et qu'un ProcessPerformer (ou Performer) connecte T et R, alors il doit exister un WorkAssignment entre asw et a.
- Si un WorkProductUse WP est relié à un seul ActorSpecificArtifact asa, et un TaskUse TU est relié à un seul ActorSpecificWork asw, et qu'un WorkDefinitionParameter (ou DefaultTaskDefinitionParameter) connecte WP et TU, alors il doit exister un ArtifactUse entre asa et asw.
- Si un RoleUse RU est relié à un seul Actor a, et un WorkProductUse WP est relié à un seul ActorSpecificArtifact asa, et qu'un ProcessResponsibilityAssignment (ou DefaultResponsabilityAssignment) connecte RU à WP, alors il doit exister un ArtifactOwnership entre a et asa.
- Si un WorkAssignment lie un Actor et un ActorSpecificWork, le RoleUse et le TaskUse correspondants doivent être aussi liés par un ProcessPerformer.
- Si un ArtifactUse lie un ActorSpecificWork et un ActorSpecificArtifact, le TaskUse et le WorkProductUse correspondant doivent être liés par un WorkDefinitionParameter (ou DefaultTaskDefinitionParameter).
- Si un ArtifactOwnership lie un Actor et un ActorSpecificArtifact, le RoleUse et le WorkProductUse correspondant doivent être liés par un ProcessResponsibilityAssignment (ou DefaultResponsabilityAssignment).
- Les relations entre deux Actor (resp. ActorSpecificWork, ActorSpecificArtifact) ne doivent être définies que si, en utilisant des concepts provenant de SPEM, des liens entre RoleUse (resp. TaskUse, WorkProductUse) ne peuvent pas être utilisés (parce que ne s'appliquant pas à chaque couple d'instance).

10.2.2 Description formelle

```
import 'http://www.irit.fr/macao/cmspem'

package cmspem

-- 1) Le nom d'un acteur doit être unique dans tout le modèle de procédé
context Actor
inv ActorNamesAreUnique('Duplicate actor name «' + name + '»') :
  Actor.allInstances()->forAll(
    a | a <> self implies a.name <> self.name
  )

-- 2) L'email d'un acteur doit être unique dans tout le modèle de procédé
context Actor
inv ActorEmailsAreUnique('Duplicate actor email «' + email + '»') :
  Actor.allInstances()->forAll(
    a | a <> self implies  a.email <> self.email
  )

-- 3) Le nom d'une ActorSpecificWork doit être unique parmi
-- tous les ActorSpecificWork liées à un TaskUse donné.
context ActorSpecificWork
inv ActorSpecificWorkNamesAreUniqueWithinTaskUse(
    'TaskUse «' + associatedTaskUse.name
    + '»: duplicate ActorSpecificWork name «' + name + '»'
  ):
  associatedTaskUse->isEmpty()
  or
  associatedTaskUse.contributingActorSpecificWorks->forAll(
    asw | asw <> self implies asw.name <> self.name
  )


-- 4) Le nom d'un ActorSpecificArtifact doit être unique parmi
-- tous les ActorSpecificArtifact liés à un WorkProductUse donné.
context ActorSpecificArtifact
inv ActorSpecificArtifactNamesAreUniqueWithinWorkProductUse(
    'WorkProductUse «' + associatedWorkProductUse.name
    + '»: duplicate ActorSpecificArtifact name «' + name + '»'
  ):
  associatedWorkProductUse->isEmpty()
  or
  associatedWorkProductUse.representingActorSpecificArtifacts->forAll(
    asa | asa <> self implies asa.name <> self.name
  )
```

```
-- 5) Si un ActorSpecificArtifact est une copie partielle du WorkProductUse
-- qu'il représente, alors il existe au moins un ActorSpecificArtifact,
-- distinct du premier, copie partielle du WorkProductUse en question.
context ActorSpecificArtifact inv :
  if isPartialCopy = true then
    associatedWorkProductUse.representingActorSpecificArtifacts->exists(
        asa | asa <> self and asa.isPartialCopy
    )
  else
    true
  endif

-- [TEMPLATE 1] : Toute relation inter-concept de niveau CM_SPEM
-- doit avoir un correspondant de niveau SPEM.


-- 6) Si un TaskAssignment lie un Actor a et un ActorSpecificWork asw,
-- et que l'Actor a est lié à un RoleUse R, l'ActorSpecificWork asw lié
-- à un TaskUse T, et le TaskUse T lié à un TaskDefinition TD
-- alors un ProcessPerformer doit lier R et TD.
-- [TEMPLATE 1]
context TaskAssignment inv :
  let  a   : Actor = linkedActor,
    asw : ActorSpecificWork  = linkedActorSpecificWork,
    R   : RoleUse = a.associatedRoleUse,
    T   : TaskUse = asw.associatedTaskUse,
    TD   : TaskDefinition = T.task
  in
    ProcessPerformer.allInstances()->exists(

      pp |
        pp.linkedRoleUse->includes(R)
        and
        pp.linkedActivity->includes(TD)

    )

-- 7) Si un ArtifactOwnership lie un ActorSpecificArtifact asa et un Actor a,
-- et que l'Actor a est lié à un RoleUse R, l'ActorSpecificArtifact lié
-- à un WorkProductUse WP, alors un ProcessResponsibilityAssignment doit
-- lier R et WP.
-- [TEMPLATE 1]
context ArtifactOwnership inv :
  let a   : Actor = linkedActor,
    asa : ActorSpecificArtifact = linkedActorSpecificArtifact,
```

```
    R   : RoleUse = a.associatedRoleUse,
    WP   : WorkProductUse = asa.associatedWorkProductUse
  in
    ProcessResponsibilityAssignment.allInstances()->exists(

      pra |
        pra.linkedRoleUse->includes(R)
        and
        pra.linkedWorkProductUse->includes(WP)

    )

-- 8) Si un ArtifactUse lie un ActorSpecificArtifact asa et un
-- ActorSpecificWork asw, et que l'ActorSpecificArtifact asa est lié à
-- un WorkProductUse WP, l'ActorSpecificWork asw lié à un TaskUse T,
-- et le TaskUse T lié à un TaskDefinition TD, alors un ProcessParameter
-- doit lier TD et WP.
-- [TEMPLATE 1]
context ArtifactUse inv :
  let asa : ActorSpecificArtifact = linkedActorSpecificArtifact,
    asw : ActorSpecificWork = linkedActorSpecificWork,
    WP   : WorkProductUse = asa.associatedWorkProductUse,
    T   : TaskUse = asw.associatedTaskUse,
    TD   : TaskDefinition = T.task
  in
    ProcessParameter.allInstances()->exists(

      pp |
        pp.parameterType->includes(WP)
        and
        TD.ownedParameter->includes(pp)

    )

-- [TEMPLATE 2] : L'existence d'une relation entre deux concepts de niveau SPEM,
-- implique l'existence d'au moins une relation inter-concepts de niveau
-- CM_SPEM qui la réalise

-- 9) Si un RoleUse R est relié à un ensemble d'Actor a_set, qu'un
-- TaskUse T est relié à ensemble d'ActorSpecificWork asw_set,
-- et que le TaskUse T est lié à un TaskDefinition TD, et qu'au moins un
-- ProcessPerformer connecte TD et R, alors il doit exister au moins
-- un TaskAssignment entre un élément de a_set et un élément de asw_set.
-- [TEMPLATE 2]
context ProcessPerformer inv:
  let TD     : TaskDefinition = linkedActivity,
```

```
    R     : RoleUse = linkedRoleUse,
    a_set  : Set(Actor) = R.affectedActor
  in
    TaskUse.allInstances()->select(task = TD)->forAll(

      T |
      let asw_set : Set(ActorSpecificWork) =
        ActorSpecificWork.allInstances()->select(associatedTaskUse = T)
      in
        TaskAssignment.allInstances()->exists(
          ta |
          a_set->exists(a | ta.linkedActor = a)
          and
          asw_set->exists(asw | ta.linkedActorSpecificWork = asw)
        )

    )

-- 10) Si un RoleUse R est relié à un ensemble d'Actor a_set, qu'un
-- WorkProductUse WP est relié à un ensemble d'ActorSpecificArtifact asa_set,
-- et qu'au moins un ProcessResponsibilityAssignment PRA connecte WP et R,
-- alors il doit exister au moins un ArtifactOwnership entre
-- un élément de a_set et un élément de asa_set.
-- [TEMPLATE 2]
context ProcessResponsibilityAssignment inv :
  let R     : RoleUse = linkedRoleUse,
    WP     : WorkProductUse = linkedWorkProductUse,
    a_set  : Set(Actor) = R.affectedActor
  in
    let asa_set : Set(ActorSpecificArtifact) =
      ActorSpecificArtifact.allInstances()->select(
        linkedWorkProductUse = WP
      )
    in
      ArtifactOwnership.allInstances()->exists(
        ao |
        a_set->exists(a | ao.linkedActor = a)
        and
        asa_set->exists(asa | ao.linkedActorSpecificArtifact = asa)
      )


-- 11) Si un WorkProductUse WP est relié à un ensemble de
-- ActorSpecificArtifact asa_set, qu'un TaskUse T est relié à un ensemble de
-- ActorSpecificWork asw_set, que le TaskUse T est lié à un
-- TaskDefinition TD, et qu'au moins un ProcessParameter connecte TD et WP,
```

```
-- alors il doit exister au moins un ArtifactUse entre un élément de asw_set
-- et un élément de asa_set.
-- [TEMPLATE 2]

context ActorSpecificWork inv :
  let T     : TaskUse = associatedTaskUse,
    TD     : TaskDefinition = T.task,
    asw_set : Set(ActorSpecificWork) =
      ActorSpecificWork.allInstances()->select(
        asw | asw.associatedTaskUse = T
      )
  in
    if
      TD.ownedParameter->size() = 0
    then
      true
    else
      TD.ownedParameter.parameterType->asSet()->forAll(

        WP : WorkProductUse |
          let
            asa_set : ActorSpecificArtifact =
              ActorSpecificArtifact.allInstances()->select(
                associatedWorkProductUse = WP
              )
          in
            ArtifactUse.allInstances()->exists(
              au |
              asw_set->exists(
                asw | au.linkedActorSpecificWork = asw
              )
              and
              asa_set->exists(
                asa | au.linkedActorSpecificArtifact = asa
              )
            )

      )
    endif

endpackage
```

10.3 Représentation textuelle CMSPEM

10.3.1 Description de la syntaxe

La syntaxe textuelle de CMSPEM est conçue pour être succincte, afin que les modèles CMSPEM qui l'utilisent soient faciles à saisir manuellement et à générer automatiquement.

Dans un fichier contenant un modèle textuel CMSPEM, des commentaires peuvent être insérés partout, entre les délimiteurs `/*` et `*/`. Les espaces blancs ne sont pas significatifs, et peuvent être utilisés librement pour aérer le fichier ou permettre de saisir facilement sa structure. Les instructions ne sont ni délimitées, ni terminées par des points-virgules ; un simple retour à la ligne est suffisant.

La notation textuelle possède un mécanisme de package qui permet de décrire les modèles de manière modulaire. Un bout de modèle peut être décrit dans un package à l'intérieur d'un fichier, et importé pour utilisation dans un autre fichier. Le mécanisme est similaire à celui des packages Java, et permet de faire référence aux éléments de modèle CMSPEM avec leur identifiant, ou leur référence complète (le nom du package, un point, puis l'identifiant).

Un package est déclaré comme suit :

```
package fr.irit.cmspem.projets.foobar {
  role Designer
  /* autres éléments de modèle CMSPEM */
}
```

Dans l'exemple précédent, l'élément Designer peut être utilisé dans un autre fichier, soit en important le package qui le contient, soit en utilisant sa référence complète :

```
/* référence complète */
actor Alice as fr.irit.cmspem.projects.foobar.Designer

/*
  importer dans le fichier courant tous les éléments
  définis dans le package fr.irit.cmspem.projects.foobar

  un unique élément peut être importé avec:

      import fr.irit.cmspem.projects.foobar.Designer
*/
import fr.irit.cmspem.projects.foobar.*

/*
  Designer peut maintenant être utilisé comme s'il était définit
  localement
```

```
*/
actor Tracy as Designer
```

Les éléments de modèle sont introduits par un mot clé associé, suivi de leur identifiant, comme dans `actor Komlan`, qui introduit un `Actor` identifié par “Komlan”. Des attributs supplémentaires peuvent être précisés entre accolades (`{` et `}`), sous forme d’une liste de couples (clé, valeur), chacun étant noté `<clé> : <valeur>`. Pour préciser le nom complet et l’adresse email de l’acteur précédent par exemple, on écrira :

```
actor Komlan {
  fullName: "Komlan Akpédjé KEDJI"
  email: "eric.kedji@gmail.com"
}
```

Les références entre concepts sont aussi introduites par des mots clés réservés, placés généralement après le concept jouant le rôle de source pour la référence. Par exemple, pour spécifier qu’un acteur, “Karl”, joue le rôle “Développeur”, on utilisera la notation suivante (“`as`” est le mot clé) :

```
role Developer

actor Karl as Developer
```

La section suivante présente la grammaire complète de la notation textuelle, dans le formalisme pseudo-EBNF de XTEXT, et précise les mots clés utilisés pour chaque concept, relation, et référence entre concepts.

10.3.2 Grammaire XTEXT

```
grammar fr.irit.macao.cmspem.dsl.CmspemDsl with org.eclipse.xtext.xbase.Xbase

import "http://www.irit.fr/macao/cmspem/Cmspem"

CmspemModel:
  elements += AbstractElement*
;

AbstractElement:
  PackageDeclaration   | Import | Classifier
;

Classifier:
  RoleUse | TaskUse           | WorkProductUse         |
```

```
  Actor   | ActorSpecificWork  | ActorSpecificArtifact  |
  ArtifactOwnership | ArtifactUse | WorkAssignment |
  ActorRelationship |
  ActorRelationshipKind |
  ActorSpecificWorkRelationship |
  ActorSpecificWorkRelationshipKind |
  ActorSpecificArtifactRelationship   |
  ActorSpecificArtifactRelationshipKind |
  Event  | EventHandler | EventSubscription
;

PackageDeclaration:
  'package' name=QualifiedName '{'
      elements+=AbstractElement*
  '}';

Import:
  'import' importedNamespace=QualifiedNameWithWildCard;

QualifiedNameWithWildCard :
  QualifiedName  ('.' '*')?;

RoleUse:
  'role' name = QualifiedName
;

TaskUse:
  'task' name = QualifiedName
;

WorkProductUse:
  'product' name = QualifiedName
;

Actor:
  'actor' name = QualifiedName (
    ('as'|'for'|'plays')
    associatedRoleUses += [RoleUse|QualifiedName]
    (',' associatedRoleUses += [RoleUse|QualifiedName])*
  )?
  ('{'
    (
      ('email' ':' email = STRING)?
      &
      ('fullName' ':' fullName = STRING)?
    )
```

```
  '}')?
;

ActorSpecificWork:
  ('ast'|'asw') name = QualifiedName
  (('as'|'for') associatedTaskUse = [TaskUse|QualifiedName])?
  ('{'
    (
      ('tickets' ':' refs = STRING)?
      &
      ('deadline' ':' deadline = STRING)?
    )
  '}')?
;

ActorSpecificArtifact:
  (isPartialCopy ?= 'partial')?
  'asa' name = QualifiedName (('as'|'for')
  associatedWorkProductUse = [WorkProductUse|QualifiedName])?
;

WorkAssignment:
  ('ast'|'asw') associatedActorSpecificWork = [ActorSpecificWork|QualifiedName]
  'done' 'by' associatedActor = [Actor|QualifiedName]
;

ArtifactOwnership:
  'asa' associatedActorSpecificArtifact = [ActorSpecificArtifact|QualifiedName]
  'owned' 'by' associatedActor = [Actor|QualifiedName]
;

ArtifactUse:
  'asa' associatedActorSpecificArtifact = [ActorSpecificArtifact|QualifiedName]
  'used' 'in' associatedActorSpecificWork = [ActorSpecificWork|QualifiedName]
;

ActorRelationshipKind:
  'kind' name = QualifiedName 'for' ('ActorRelationship' | 'actorRelationship')
;

ActorSpecificWorkRelationshipKind:
  'kind' name = QualifiedName
  'for' ('ActorSpecificWorkRelationship' | 'astRelationship')
;

ActorSpecificArtifactRelationshipKind:
```

```
  'kind' name = QualifiedName
  'for' ('ActorSpecificArtifactRelationship' | 'asaRelationship')
;

ActorRelationship:
  ('actor')? 'rel'
  firstActor = [Actor|QualifiedName]
  kind = [ActorRelationshipKind|QualifiedName]
  secondActor = [Actor|QualifiedName]
;

ActorSpecificWorkRelationship:
  ('task' 'rel' | 'work' 'rel' | 'trel' | 'wrel')
  firstActorSpecificWork = [ActorSpecificWork|QualifiedName]
  kind = [ActorSpecificWorkRelationshipKind|QualifiedName]
  secondActorSpecificWork = [ActorSpecificWork|QualifiedName]
;

ActorSpecificArtifactRelationship:
  ('artifact' 'rel' | 'arel')
  firstActorSpecificArtifact = [ActorSpecificArtifact|QualifiedName]
  kind = [ActorSpecificArtifactRelationshipKind|QualifiedName]
  secondActorSpecificArtifact = [ActorSpecificArtifact|QualifiedName]
;

Event:
  'event' name = QualifiedName
;

EventSubscription:
  'on' event = [Event|QualifiedName]
  'from' source = [EventSource|QualifiedName]
  ':' handler = [EventHandler|QualifiedName]
;

EventHandler:
  'handler' name = QualifiedName
;
```

10.4 Récits utilisateur pour le serveur CMSPEM

Le cahier des charges du serveur CMSPEM est décrit ici sous forme de récits utilisateur. Les récits utilisateur sont utilisés en développement agile pour décrire de manière simple des fonctionnalités à développer [Cohn 04].

- En tant que chef de projet / ingénieur qualité, je voudrais pouvoir :
  - créer un projet sur le serveur
  - envoyer sur le serveur un modèle initial de projet, que sera utilisé pour coordonner les aspects processus du développement.
  - mettre à jour un modèle CMSPEM
- En tant que chef de projet / ingénieur qualité, je voudrais pouvoir :
  - modifier sur le serveur le modèle d'un projet, afin de réfléter les changements intervenus dans la réalité (ajout, suppression, modification d'éléments de processus).
  - remplacer le modèle de processus associé à un projet
  - souscrire à un événement (externe ou de processus)
- En tant que chef de projet / ingénieur qualité, je voudrais pouvoir être informé :
  - de la complétion d'une tâche
  - du dépassement d'un deadline
- En tant que chef de projet, je voudrais pouvoir :
  - consulter l'avancement global du projet
  - consulter les actions menées par chacun des participants à un projet
  - exporter des rapports sur le déroulement global du projet
  - m'assurer du respect des deadlines
  - détecter les occurrences de régressions
  - localiser des défauts (activité, participant, etc.)
- En tant que développeur, je voudrais pouvoir :
  - spécifier à l'avance mon intérêt pour des événements de processus spécifiques, sur des éléments de processus spécifiques.
  - Notifier le serveur CMSPEM d'un événement externe
- En tant que développeur, je voudrais que le serveur soit notifié, sans mon intervention :
  - du changement d'état d'un bug dans le bug tracker
  - de la complétion d'une tâche telle que spécifiée dans un message de commit
- En tant que développeur, je voudrais pouvoir notifier le serveur :
  - de mon indisponibilité temporaire pour réaliser une tâche
  - d'imprévus survenus dans la réalisation d'une tâche
- En tant que développeur, je voudrais pouvoir être notifié :
  - de changements spécifiques intervenus sur une tâche à laquelle je participe
  - des événements de processus générés par les membres de mon équipe
- En tant que développeur, je voudrais pouvoir :
  - consulter un modèle de processus
  - consulter mes tâches en attente de complétion, ainsi que les deadlines associés
  - consulter le progrès de mon équipe

10.5 Documentation de l'API CMSPEM

10.5.1 Généralités

L'API CMSPEM est un ensemble d'URLs, de méthodes HTTP disponibles sur ces URLs, ainsi que de formats de données utilisés lors des échanges avec le serveur.

Chaque URL exposée par l'API est soit un conteneur, soit un unique élément (projet, utilisateur, élément de modèle, ou souscription) génériquement désigné par "ressource". Les conteneurs sont utilisés pour grouper des ressources. A l'intérieur d'un conteneur, chaque ressource a un identifiant qui permet de la distinguer des autres ressources dans le même conteneur. Dans la suite, cet identifiant sera noté `<uuid>` [1] Par exemple, `/users` est un conteneur qui désigne tous les utilisateurs connus du serveur, et `/users/<uuid>` désigne un utilisateur particulier, avec l'identifiant `<uuid>`.

De manière générale, les requêtes HTTP suivantes sont disponibles sur les conteneurs (`/users` par exemple) :

- POST : Créer une nouvelle ressource dans ce conteneur.
- GET : Lister toutes les ressources dans ce conteneur (`/users/<uuid>` par exemple).
- GET : Obtenir les données associées à la ressource.
- POST : Modifier la ressource.
- PUT : Remplacer la ressource.
- DELETE : Supprimer la ressource.

Les requêtes renvoient les codes HTTP suivants :

- 200 : Réponse normale, suite à une lecture (GET).
- 201 : Réponse normale, suite à une création de ressource, comme par exemple un nouvel utilisateur (POST).
- 204 : Réponse normale, suite à une action effectuée avec succès, pour laquelle on ne souhaite pas donner plus de détails, comme la suppression d'un utilisateur (DELETE/-POST).
- 400 : Mauvaise requête. Des paramètres en entrée manquent ou sont incomplets. L'erreur précise se trouve dans le corps (body) de la requête (POST/GET/DELETE/PUT).
- 403 : L'utilisateur n'est pas autorisé à effectuer cette requête. Le corps de la réponse explique pourquoi (POST/GET/DELETE/PUT). Il s'agit le plus souvent d'une clé d'API erronnée.
- 404 : La ressource demandée ou manipulée est introuvable (POST/GET/DELETE/PUT).

1. Les identifiants sur le serveur CMSPEM sont effectivement des UUID (Universal Unique Identifier).

– 500 : Une erreur interne au serveur est survenue lors du traitement de la requête.

– 501 : L'opération demandée n'est pas encore implémentée.

Chaque réponse renvoyée par le serveur est un objet JSON, avec une clé "url" qui rappelle l'URL correspondant à la ressource sur le serveur. Les réponses de lecture de données (GET sur une ressource ou un conteneur), de modification qui renvoient la nouvelle version(POST et PUT sur une ressource), et de création de ressource (POST sur un conteneur) renvoient des données avec le format suivant [2] :

– Pour une ressource unique :

```
{
  "item": {
    "name": "<nom>",
    "email": "user@example.com",
    "url": "http://localhost/users/<uuid>"
  }
}
```

– Pour une liste de ressources :

```
{
  "items": [
    {
      "name": "<nom1>",
      "email": "user1@example.com",
      "url": "http://localhost/users/<uuid1>"
    },
    {
      "name": "<nom2>",
      "email": "user2@example.com",
      "url": "http://localhost/users/<uuid2>"
    }
  ],
  "url": "http://localhost/users"
}
```

Si un objet contient une référence vers un autre, et que le serveur renvoie l'objet parent dans une réponse, l'objet fils est représenté par un objet ayant une unique clé "url", qui est l'addresse de l'objet fils. Pour un `Actor` par exemple, on a :

2. On suppose que le serveur est disponible à l'adresse http://localhost/.

```
{
  "item": {
    "name": "Bob",
    "associatedRoleUses": [
      {
        "url": "http://localhost/projects/xxx/roles/<uuid1>"
      },
      {
        "url": "http://localhost/projects/xxx/roles/<uuid2>"
      }
    ],
    "url": "http://localhost/projects/xxx/actors/<uuid3>"
  }
}
```

10.5.2 Adresses des ressources exposées

10.5.2.1 Ressources régulières

Les ressources régulières correspondent aux utilisateurs, aux projets, et aux éléments de modèle de processus.

`/`
: Racine de l'API, renvoie les informations de version.

`/users`
`/users/<uuid>`
: Informations sur les participants aux projet, qui peuvent envoyer des requêtes sur l'API. Chaque utilisateur créé se voit associer une clé d'API, qu'un outil tiers peut utiliser pour faire des requêtes au nom de l'utilisateur.

`/projects`
`/projects/<uuid>`
: Projets de développement logiciel gérés sur ce serveur. Chaque projet correspond à un modèle de processus CMSPEM. Chaque projet a un créateur.

`/projects/<uuid>/tasks`
`/projects/<uuid>/tasks/<uuid>`
: TaskUses d'un projet CMSPEM.

`/projects/<uuid>/roles`
`/projects/<uuid>/roles/<uuid>`
: RoleUses d'un projet CMSPEM.

`/projects/<uuid>/products`
`/projects/<uuid>/products/<uuid>`
: WorkProductUses d'un projet CMSPEM.

`/projects/<uuid>/actors`
`/projects/<uuid>/actors/<uuid>`
: Actors d'un projet CMSPEM.

/projects/<uuid>/as-artifacts
/projects/<uuid>/as-artifacts/<uuid>
: ActorSpecificArtifacts d'un projet CMSPEM.

/projects/<uuid>/as-works
/projects/<uuid>/as-works/<uuid>
: ActorSpecificWorks d'un projet CMSPEM.

/projects/<uuid>/a-rels
/projects/<uuid>/a-rels/<uuid>
: ActorRelationships d'un projet CMSPEM.

/projects/<uuid>/as-artifact-rels
/projects/<uuid>/as-artifact-rels/<uuid>
: ActorSpecificArtifactRelationships d'un projet CMSPEM.

/projects/<uuid>/as-work-rels
/projects/<uuid>/as-work-rels/<uuid>
: ActorSpecificWorkRelationships d'un projet CMSPEM.

/projects/<uuid>/wa-rels
/projects/<uuid>/wa-rels/<uuid>
: WorkAssignments d'un projet CMSPEM.

/projects/<uuid>/ao-rels
/projects/<uuid>/ao-rels/<uuid>
: ArtifactOwnerships d'un projet CMSPEM.

/projects/<uuid>/au-rels
/projects/<uuid>/au-rels/<uuid>
: ArtifactUses d'un projet CMSPEM.

10.5.2.2 Ressources pour la gestion d'événement

Pour chaque url RESOURCE_URL désignant une ressource, RESOURCE_URL/subscriptions désigne les souscriptions d'événements sur cette ressource, et une souscription particulière est désignée par RESOURCE_URL/subscriptions/xxx.

Les requêtes possibles sur RESOURCE_URL/subscriptions sont les suivantes :

GET
: Lister toutes les souscriptions d'événements sur la ressource associée (RESOURCE_URL). Si l'auteur de la requête est un utilisateur normal, et non un administrateur, restreindre la liste aux souscriptions faites par cet utilisateur.

  Exemple de réponse :

```
{
  "items": [
    {
      "event": "start",
      "handler": "http://github.com/...",
      "url": "RESOURCE_URL/subscriptions/<uuid1>"
```

```
    },
    {
      "event": "end",
      "handler": "http://bugzilla.com/...",
      "url": "RESOURCE_URL/subscriptions/<uuid2>"
    }
  ]
}
```

POST
Souscrire à un événement (créer une nouvelle souscription)
Paramètres POST :

event
Nom de l'événement.

handler
URL sur laquelle une requête de type POST sera envoyée, pour notifier que l'événement s'est produit. Il s'agit d'une adresse HTTP/HTTPS. Dans l'industrie, on décrit cette utilisation d'adresses web pour la gestion d'événements par le terme "webhook".

Les requêtes possibles sur `RESOURCE_URL/subscriptions/<uuid>` sont les suivantes :

GET
Consulter une souscription (événement associé, URL associée, date de création, etc.).

POST
Modifier une souscription (changer l'URL qui sert de gestionnaire).

DELETE
Annuler une souscription (aucune notification d'événement ne sera plus envoyée au gestionnaire).

Les souscriptions respectent la notion de remontée d'événement définie dans la section 5.5.2. Des exemples d'URLs de souscription, et les événements associés sont les suivants :

`/projects/<uuid>/subscriptions`
Souscription à des événements sur l'ensemble du projet, comme la suppression ou la mise à jour de tout élément de modèle.

`/projects/<uuid>/actors`
Souscription à des événements sur n'importe quel acteur (ajout, mise à jour, indiponibilité, etc.).

`/projects/<uuid>/actors/xxx`
Souscription à des événements sur un acteur en particulier (mise à jour, indisponibilité, etc.).

La remontée d'évènements se fait conformément à la hiérarchie de parenté définie dans la section 5.5.1. Supprimer un acteur par exemple notifie les gestionnaires d'événement définis sur les éléments suivants :

- L'acteur supprimé, `/projects/<uuid>/actors/<actor-uuid>`.
- La liste des acteurs du projet concerné, `/projects/<uuid>/actors`.
- Le projet qui contient l'acteur concerné, `/projects/<uuid>`.

Références

[Abra 02] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. "Agile software development methods : Review and analysis". *Oulu, Finland : VTT Publications*, 2002.

[Alei 11] F. A. Aleixo, M. A. Freire, W. C. Santos, and U. Kulesza. "Automating the variability management, customization and deployment of software processes : A model-driven approach". *Enterprise Information Systems*, pp. 372–387, 2011.

[Allo 96] I. Alloui. *PEACE+ : a formalism and a system for cooperation in PSEEs*. PhD thesis, University of Grenoble II, France, 1996.

[Ambr 97] V. Ambriola, R. Conradi, and A. Fuggetta. "Assessing process-centered software engineering environments". *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 6, No. 3, pp. 283–328, 1997.

[Ande 04] D. J. Anderson. *Agile management for software engineering : Applying the theory of constraints for business results*. Prentice Hall, 2004.

[Ashr 03] N. Ashrafi. "The impact of software process improvement on quality : in theory and practice". *Information & Management*, Vol. 40, No. 7, pp. 677 – 690, 2003.

[Aust 75] J. L. Austin. *How to do things with words*. Vol. 88, Harvard University Press, 1975.

[Avri 07] A. Avritzer, W. Hasling, and D. Paulish. "Process investigations for the global studio project version 3.0". In : *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pp. 247–251, IEEE, 2007.

[Avri 10] A. Avritzer and D. Paulish. "A comparison of commonly used processes for multi-site software development". *Collaborative Software Engineering*, pp. 285–302, 2010.

[Bake 03] W. Baker, R. Cross, and M. Wooten. "Positive organizational network analysis and energizing relationships". *Positive organizational scholarship : Foundations of a new discipline*, pp. 328–342, 2003.

[Band 94] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. "SPADE : An environment for software process analysis, design, and enactment". *Software process modelling and technology*, pp. 223–247, 1994.

[Barg 94] N. Barghouti. “Separating process model enactment from process execution in Provence”. In : *Proceedings of the ninth International Software Process Workshop*, pp. 70–73, IEEE, 1994.

[Bart 03] P. Barthelmess. “Collaboration and coordination in process-centered software development environments : a review of the literature”. *Information and Software Technology*, Vol. 45, No. 13, pp. 911–928, 2003.

[Beck 99] K. Beck. “Extreme programming”. In : *tools*, p. 411, 1999.

[Bego 03] J. B. Begole, J. C. Tang, and R. Hill. “Rhythm modeling, visualizations and applications”. In : *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pp. 11–20, ACM, 2003.

[Ben 94] I. Ben-Shaul and G. Kaiser. “A paradigm for decentralized process modeling and its realization in the oz environment”. In : *Proceedings of the 16th international conference on Software engineering*, pp. 179–188, IEEE Computer Society Press, 1994.

[Bend 07] R. Bendraou, B. Combemale, X. Cregut, and M.-P. Gervais. “Definition of an Executable SPEM 2.0”. In : *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*, pp. 390–397, IEEE, 2007.

[Bend 09] R. Bendraou, J.-M. Jézéquel, and F. Fleurey. “Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution”. In : Q. Wang, V. Garousi, R. J. Madachy, and D. Pfahl, Eds., *Trustworthy Software Development Processes, International Conference on Software Process, ICSP 2009 Vancouver, Canada, May 16-17, 2009 Proceedings*, pp. 148–160, Springer, 2009.

[Bezi 04] J. Bézivin and E. Breton. “Applying the basic principles of model engineering to the field of process engineering”. *Novatica N° 171, Software Process Technologies*, 2004. http ://www.ati.es/novatica/infonovatica.html.

[Bezi 05] J. Bézivin. “On the unification power of models”. In : *Software and System Modeling*, May 2005.

[Boeh 02] B. Boehm. “Get ready for agile methods, with care”. *COMPUTER*, pp. 64–69, 2002.

[Boeh 88] B. Boehm. “A spiral model of software development and enhancement”. *Computer*, Vol. 21, No. 5, pp. 61–72, 1988.

[Boeh 89] B. W. Boehm and R. Ross. “Theory-W software project management principles and examples”. *Software Engineering, IEEE Transactions on*, Vol. 15, No. 7, pp. 902–916, 1989.

[Boeh 94] B. Boehm and P. Bose. “A Collaborative Spiral Software Process Model Based on Theory W”. In : *Proceedings, 3rd International Conference on the Software Process, Applying the Software Process, IEEE*, pp. 59–68, 1994.

[Bolc 96] G. Bolcer and R. Taylor. “Endeavors : A process system integration infrastructure”. In : *Software Process, 1996. Proceedings., Fourth International Conference on the*, pp. 76–89, 1996.

[Booc 03] G. Booch and A. Brown. “Collaborative development environments”. *Advances in Computers*, Vol. 59, pp. 1–27, 2003.

[Booc 04] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic. “An MDA manifesto”. *MDA Journal*, Vol. 5, pp. 2–9, 2004.

[Boss 10] P. Van den Bossche, W. Gijselaers, M. Segers, G. Woltjer, and P. Kirschner. “Team learning : building shared mental models”. 2010.

[Brag 07] J. Bragge, H. Merisalo-Rantanen, A. Nurmi, and L. Tanner. “A repeatable e-collaboration process based on ThinkLets for multi-organization strategy development”. *Group Decision and Negotiation*, Vol. 16, No. 4, pp. 363–379, 2007.

[Brag 08] J. Bragge and H. Merisalo-Rantanen. “Engineering E-Collaboration Processes to Obtain Innovative End-User Feedback on Advanced Web-Based Information Systems”. *Journal of the Association for Information Systems*, Vol. 10, No. 3, 2008.

[Brig 06] R. Briggs, G. Kolfschoten, G. de Vreede, and D. Dean. “Defining key concepts for collaboration engineering”. In : *Proceedings of the 12th Americas Conference on Information Systems, Garcia I and Trejo R (eds), Acapulco, Mexico*, pp. 121–128, 2006.

[Broo 87] F. Brooks. “No silver bullet : Essence and accidents of software engineering”. *IEEE computer*, Vol. 20, No. 4, pp. 10–19, 1987.

[Buxt 70] J. N. Buxton and B. Randell. *Software Engineering Techniques : Report on a Conference Sponsored by the NATO Science Committee*. NATO Science Committee ; available from Scientific Affairs Division, NATO, 1970.

[Card 83] S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction*. CRC, 1983.

[Carr 97] J. M. Carroll. “Human-computer interaction : psychology as a science of design”. *Annual review of psychology*, Vol. 48, No. 1, pp. 61–83, 1997.

[Cars 99] P. H. Carstensen and K. Schmidt. “Computer supported cooperative work : New challenges to systems design”. In : *In K. Itoh (Ed.), Handbook of Human Factors*, Citeseer, 1999.

[Caus 10] P. Causes. “Why Can’t We Manage Large Projects ?”. *CROSSTALK*, 2010.

[Char 03] K. Charmaz. “Grounded theory”. *Strategies of qualitative inquiry*, Vol. 2, p. 249, 2003.

[Cher 08] S. Cherry and P. Robillard. “The social side of software engineering–A real ad hoc collaboration network”. *International journal of human-computer studies*, Vol. 66, No. 7, pp. 495–505, 2008.

[Cicc 08] A. Cicchetti, D. Di Ruscio, and A. Pierantonio. “Managing model conflicts in distributed development”. In : *Model Driven Engineering Languages and Systems*, pp. 311–325, Springer, 2008.

[Cock 01] A. Cockburn. “Agile Software Development”. 2001.

[Cock 05] A. Cockburn. *Crystal clear : a human-powered methodology for small teams*. Addison-Wesley Professional, 2005.

[Cohn 04] M. Cohn. *User stories applied : For agile software development*. Addison-Wesley Professional, 2004.

[Colo 02] M. Colombetti and M. Verdicchio. “An analysis of agent speech acts as institutional actions”. In : *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 3*, pp. 1157–1164, ACM, 2002.

[Conr 94] R. Conradi, C. Fernström, and A. Fuggetta. “Concepts for evolving software processes”. *Software Process Modelling and Technology*, pp. 9–31, 1994.

[Conw 68] M. Conway. “How do committees invent”. *Datamation*, Vol. 14, No. 4, pp. 28–31, 1968.

[Cook 03] C. Cook, N. Churcher, and N. Christchurch. “An extensible framework for collaborative software engineering”. In : *Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC’03), IEEE*, pp. 290–301, Citeseer, 2003.

[Cook 07] C. Cook. *Towards Computer-Supported Collaborative Software Engineering*. PhD thesis, 2007.

[Cram 01] C. Cramton. “The mutual knowledge problem and its consequences for dispersed collaboration”. *Organization science*, pp. 346–371, 2001.

[Cros 04] R. L. Cross and A. Parker. *The hidden power of social networks : Understanding how work really gets done in organizations*. Harvard Business Press, 2004.

[Cugo 99] G. Cugola and C. Ghezzi. “Design and Implementation of PROSYT : A Distributed Process Support System”. In : *Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, p. 39, IEEE Computer Society, 1999.

[Dami 98] S. Dami, J. Estublier, and M. Amiour. “Apel : A graphical yet executable formalism for process modeling”. *Automated Software Engineering*, Vol. 5, No. 1, pp. 61–96, 1998.

[Dewa 10] P. Dewan. “Towards and beyond being there in collaborative software development”. In : *Collaborative Software Engineering*, pp. 135–151, Springer, 2010.

[Dewa 93] P. Dewan and J. Riedl. “Toward computer-supported concurrent software engineering”. *Computer*, Vol. 26, No. 1, pp. 17–27, 1993.

[Diaw 10] S. Diaw, R. Lbath, and B. Coulette. “SPEM4MDE : un métamodèle basé sur SPEM 2 pour la spécification des procédés MDE”. In : *Workshop on Model-Driven Tool & Process Integration - Associated to ECMFA*, 2010.

[Diaw 11] S. Diaw, R. Lbath, and B. Coulette. “Specification and Implementation of SPEM4MDE, a metamodel for MDE software processes (regular paper)”. In : *International Conference on Software Engineering and Knowledge Engineering (SEKE), Miami - USA, 07/07/11-09/07/11*, pp. 646–653, Knowledge Systems Institute, http ://www.ksi.edu, July 2011.

[Dijk 82] E. Dijkstra. “A Personal Perspective. On the role of scientific thought. Selected Writings on Computing”. 1982.

[Efft 06] S. Efftinge and M. Volter. “oAW xText : A framework for textual DSLs”. In : *Workshop on Modeling Symposium at Eclipse Summit*, 2006.

[Elli 04] G. Elliott. *Global business information technology : an integrated systems approach*. Addison-Wesley, 2004.

[Elln 10] R. Ellner, S. Al-Hilank, J. Drexler, M. Jung, D. Kips, and M. Philippsen. “eSPEM–a SPEM extension for enactable behavior modeling”. In : *Modelling Foundations and Applications*, pp. 116–131, Springer, 2010.

[Fiel 00] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[Fish 04] D. Fisher and P. Dourish. “Social and temporal structures in everyday collaboration”. In : *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 551–558, ACM, 2004.

[Glas 67] B. Glaser and A. Strauss. *The discovery of grounded theory : Strategies for qualitative research*. Aldine, 1967.

[Gold 02] A. Goldberg. “Collaborative software engineering”. *Journal of Object Technology*, Vol. 1, No. 1, pp. 1–19, 2002.

[Gonz 07a] C. Gonzalez-Perez. “Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach”. *Situational Method Engineering : Fundamentals and Experiences*, pp. 7–18, 2007.

[Gonz 07b] C. Gonzalez-Perez. “Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach”. In : *Situational Method Engineering : Fundamentals and Experiences*, pp. 7–18, Springer, 2007.

[Gord 95] V. S. Gordon and J. M. Bieman. “Rapid prototyping : lessons learned”. *Software, IEEE*, Vol. 12, No. 1, pp. 85–95, 1995.

[Goth 09] G. Goth. “The Task-Based Interface : Not Your Father’s Desktop”. *Software, IEEE*, Vol. 26, No. 6, pp. 88–91, 2009.

[Gron 09] R. C. Gronback. *Eclipse Modeling Project : A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[Grud 01] J. Grudin, R. McCall, J. Ostwald, and F. Shipman. "Seeding, Evolutionary Growth, and Reseeding : The Incremental Development of Collaborative Design Environments". *Coordination theory and collaboration technology*, p. 447, 2001.

[Gruh 02a] V. Gruhn. "Process-centered software engineering environments, a brief history and future challenges". *Annals of Software Engineering*, Vol. 14, No. 1, pp. 363–382, 2002.

[Gruh 02b] V. Gruhn. "Process-centered software engineering environments, a brief history and future challenges". *Annals of Software Engineering*, Vol. 14, No. 1, pp. 363–382, 2002.

[Hart 00] D. E. Harter and S. A. Slaughter. "Process maturity and software quality : a field study". In : *Proceedings of the twenty first international conference on Information systems*, pp. 407–411, Association for Information Systems, Atlanta, GA, USA, 2000.

[Hatt 10] L. Hattori and M. Lanza. "Syde : A Tool for Collaborative Software Development". 2010.

[Hawr 05] I. Hawryszkiewycz. "A metamodel for modeling collaborative systems". *Journal of Computer Information Systems*, Vol. 45, No. 3, pp. 63–72, 2005.

[Hend 06] B. Henderson-Sellers. "Method engineering : Theory and practice". *Information Systems Technology and its Applications*, pp. 13–23, 2006.

[Hend 08] B. Henderson-Sellers and C. Gonzalez-Perez. "Standardizing methodology metamodelling and notation : An ISO exemplar". *Information Systems and e-Business Technologies*, pp. 1–12, 2008.

[Herb 07] J. Herbsleb. "Global software engineering : The future of socio-technical coordination". In : *2007 Future of Software Engineering*, pp. 188–198, IEEE Computer Society, 2007.

[Huds 02] J. M. Hudson, J. Christensen, W. A. Kellogg, and T. Erickson. "I'd be overwhelmed, but it's just one more thing to do : Availability and interruption in research management". In : *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pp. 97–104, ACM, 2002.

[ISO 08] ISO. "ISO 9001". http://www.iso.org/iso/home/standards/management-standards/iso_9000.htm, 2008.

[IST 04] IST. "The Modelware IST project.". http://www.modelware-ist.org, 2004.

[IST 06] IST. "The Modelplex IST project.". http://www.modelplex.org, 2006.

[Jaco 99] I. Jacobson, G. Booch, and J. Rumbaugh. "The Unified Software Development Process–The complete guide to the Unified Process from the original designers". *Rational Software Corporation, US*, 1999.

[Jian 06] T. Jiang, J. Ying, M. Wu, and M. Fang. "An Architecture of Process-centered Context-aware Software Development Environment". In : *Computer Supported Cooperative Work in Design, 2006. CSCWD'06. 10th International Conference on*, pp. 1–5, IEEE, 2006.

[Jime 09] M. Jiménez, M. Piattini, and A. Vizcaíno. "Challenges and improvements in distributed software development : a systematic review". *Advances in Software Engineering*, Vol. 2009, p. 3, 2009.

[Junk 95] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. *MERLIN : Supporting cooperation in software development through a knowledge based environment*. Citeseer, 1995.

[Kabb 08] M. Kabbaj, R. Lbath, and B. Coulette. "A deviation management system for handling software process enactment evolution". In : *Proceedings of the International COnference on Software Process*, pp. 186–197, Springer-Verlag, 2008.

[Kais 90] G. Kaiser, N. Barghouti, and M. Sokolsky. "Preliminary Experience with Process Modeling in the Marvel Software Development Kernel". In : *Proceeding of the 23rd International Conference on System Sciences*, 1990.

[Kais 97] G. Kaiser, S. Dossick, W. Jiang, and J. Yang. "An architecture for WWW-based hypercode environments". 1997.

[Kedj 11a] K. A. Kedji, B. Coulette, M. Nassar, R. Lbath, and M. T. Thon That. "Collaborative Processes in the Real World : Embracing their Essential Nature (regular paper)". In : *International Symposium on Model Driven Engineering : Software & Data Integration, Process Based Approaches and Tools. Colocated with ECMFA 2011 conference, Birmingham*, June 2011.

[Kedj 11b] K. A. Kedji, B. Coulette, M. T. Ton That, R. Lbath, M. Nassar, and H. N. Tran. "Towards a tool-supported approach for collaborative process modeling and enactment". In : *The Eighteenth Asia-Pacific Software Engineering Conference (APSEC 2011), Hochiminh, December 05-08, 2011*, IEEE, December 2011.

[Kedj 12a] K. Kedji, R. Lbathd, B. Coulette, M. Nassar, L. Baresse, and F. Racaru. "Supporting collaborative development using process models : An integration-focused approach". In : *Software and System Process (ICSSP), 2012 International Conference on*, pp. 120 –129, june 2012.

[Kedj 12b] K. A. Kedji, B. Coulette, R. Lbath, and M. Nassar. "Modeling Ad-hoc Collaboration for Automated Process Support". In : *Software Quality Days 2012*, Springer, January 2012.

[Kedj 12c] K. A. Kedji, R. Lbath, B. Coulette, and M. Nassar. "Exploiting process events for the integration of collaborative software development tools.". *Journal of Software Engineering*, November 2012.

[Kedj 13] K. A. Kedji, R. Lbath, B. Coulette, and M. Nassar. "Supporting Collaborative Development Using Process Models : A Tooled Integration-Focused Approach (accepté)". *Journal of Software : Evolution and process*, July 2013.

[Kier 04] D. E. Kieras and T. P. Santoro. “Computational GOMS modeling of a complex team task : Lessons learned”. In : *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 97–104, ACM, 2004.

[Kill 10] P. Killisperger, M. Stumptner, G. Peters, G. Grossmann, and T. Stückl. “A Framework for the Flexible Instantiation of Large Scale Software Process Tailoring”. *New Modeling Concepts for Today’s Software Processes*, pp. 100–111, 2010.

[Kipe 87] J. Kiper. “The Integration of Software Development Tools”. Tech. Rep., Technical Report 87-001, Miami University, 1987.

[Kobi 04] H. Kobialka. “Supporting the Software Process in A Process-centered Software Engineering Environment”. *European Journal for the Informatics Professional*, 2004.

[Kolf 04] G. Kolfschoten, R. Briggs, J. Appelman, and G. de Vreede. “ThinkLets as building blocks for collaboration processes : a further conceptualization”. *Lecture Notes in Computer Science*, pp. 137–152, 2004.

[Kolf 06] G. Kolfschoten, R. Briggs, G. De Vreede, P. Jacobs, and J. Appelman. “A conceptual foundation of the thinkLet concept for Collaboration Engineering”. *International Journal of Human-Computer Studies*, Vol. 64, No. 7, pp. 611–621, 2006.

[Kont 98] J. Kontio. “A Software Process Engineering Framework”. In : M. V. Zelkowitz, Ed., , pp. 35 – 108, Elsevier, 1998.

[Kris 93] B. Krishnamurthy and N. Barghouti. “Provence : A process visualization and enactment environment”. *Software Engineering - ESEC’93*, pp. 451–465, 1993.

[Kurt 02] I. Kurtev, J. Bézivin, and M. Aksit. “Technological spaces : An initial appraisal”. In : *CoopIS, DOA’2002 Federated Conferences, Industrial track*, 2002.

[Lanu 10] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaíno. “Collaboration tools for global software engineering”. *Software, IEEE*, Vol. 27, No. 2, pp. 52–55, 2010.

[Lonc 90] J. Lonchamp, K. Benali, C. Godart, and J. Derniame. “Modeling and enacting software processes : an analysis”. In : *Computer Software and Applications Conference, 1990. COMPSAC 90. Proceedings., Fourteenth Annual International*, pp. 727–736, IEEE, 1990.

[Lonc 91] J. Lonchamp, K. Benali, J. Derniame, and C. Godart. “Towards assisted software engineering environments”. *Information and Software Technology*, Vol. 33, No. 8, pp. 581–593, 1991.

[Lonc 94] J. Lonchamp. “An assessment exercise”. *Software Process Modelling and Technology*, pp. 335–356, 1994.

[Lonc 96] J. Lonchamp and F. Seguin. “Issue-based collaborative process modeling”. In : *Proceedings of the Second International Conference on the Design of Cooperative Systems*, pp. 547–565, Citeseer, 1996.

[Maci 09] R. S. P. Maciel, B. C. da Silva, P. F. Magalhães, and N. S. Rosa. "An integrated approach for model driven process modeling and enactment". In : *Software Engineering, 2009. SBES'09. XXIII Brazilian Symposium on*, pp. 104–114, IEEE, 2009.

[Malo 03] T. Malone, K. Crowston, and G. Herman. *Organizing business knowledge : the MIT process handbook*. The MIT Press, 2003.

[Malo 94] T. W. Malone and K. Crowston. "The interdisciplinary study of coordination". *ACM Computing Surveys (CSUR)*, Vol. 26, No. 1, pp. 87–119, 1994.

[Mati 12] R. Matinnejad and R. Ramsin. "An Analytical Review of Process-Centered Software Engineering Environments". In : *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, pp. 64–73, IEEE, 2012.

[Mehr 05] A. Mehra, J. Grundy, and J. Hosking. "A generic approach to supporting diagram differencing and merging for collaborative design". In : *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, p. 213, ACM, 2005.

[Mist 10] I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead. *Collaborative Software Engineering : Challenges and Prospects*. Springer, 2010.

[Modi 09] T. Modica, E. Biermann, and C. Ermel. "An eclipse framework for rapid development of rich-featured gef editors based on emf models". *Proceedings of Informatik*, pp. 2972–2985, 2009.

[OASI] OASIS. "Web Services Business Process Execution Language Version 2.0". http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[Odel 94] J. Odell. "Power types". *JOOP*, Vol. 7, No. 2, pp. 8–12, 1994.

[Olso 90] J. R. Olson and G. M. Olson. "The growth of cognitive modeling in human-computer interaction since GOMS". *Human–computer interaction*, Vol. 5, No. 2-3, pp. 221–265, 1990.

[OMG 02] OMG. "Software Process Engineering Metamodel, version 1.0". http://www.omg.org/cgi-bin/doc?formal/02-11-14, 2002.

[OMG 05] OMG. "Unified Modeling Language, version 2.0". http://www.omg.org/spec/UML/2.0/, 2005.

[OMG 07] OMG. "Software Process Engineering Metamodel, version 2.0". http://www.omg.org/spec/SPEM/2.0/, 2007.

[OMG 09] OMG. "Business Process Model And Notation, version 1.2". http://www.omg.org/spec/BPMN/1.2/, 2009.

[Omor 10] I. Omoronyia, J. Ferguson, M. Roper, and M. Wood. "A review of awareness in distributed collaborative software engineering". *Software : Practice and Experience*, Vol. 40, No. 12, pp. 1107–1133, 2010.

[Ossh 00] H. Ossher, W. Harrison, and P. Tarr. “Software engineering tools and environments : a roadmap”. In : *Proceedings of the Conference on the Future of Software Engineering*, pp. 261–277, ACM, 2000.

[Oste 87] L. Osterweil. “Software processes are software too”. In : *Proceedings of the 9th international conference on Software Engineering*, pp. 2–13, IEEE Computer Society Press, 1987.

[Part 09] T. G. Partners. “Model Driven Collaborative Development of Complex Systems - Proposal”. 2009.

[Pene 91] M. H. Penedo. “Making process-based environments viable”. In : *Software Process Workshop, 1991. Communication and Coordination in the Software Process., Proceedings of the 7th International*, pp. 106–110, IEEE, 1991.

[Perr 94] D. Perry, N. Staudenmayer, and L. Votta. “People, Organizations, and Process Improvement”. *IEEE Software*, Vol. 11, No. 4, p. 45, 1994.

[Polt 09] S. Poltrock and M. Handel. “Modeling collaborative behavior : Foundations for collaboration technologies”. In : *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pp. 1–10, IEEE, 2009.

[Popp 03] M. Poppendieck and T. Poppendieck. *Lean Software Development*. Addison-Wesley, 2003.

[Porr 06] I. Porres and M. Valiente. “Process definition and project tracking in model driven engineering”. *Product-Focused Software Process Improvement*, pp. 127–141, 2006.

[Quin 05] R. W. Quinn and J. E. Dutton. “Coordination As Energy-in-Conversation.”. *Academy of Management Review*, Vol. 30, No. 1, pp. 36–57, 2005.

[Redd 02] M. Reddy and P. Dourish. “A finger on the pulse : temporal rhythms and information seeking in medical work”. In : *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 344–353, ACM, 2002.

[Rich 10] I. Richardson, V. Casey, J. Burton, and F. McCaffery. “Global software engineering : A software process approach”. *Collaborative Software Engineering*, pp. 35–56, 2010.

[Risi 00] L. Rising and N. Janoff. “The Scrum software development process for small teams”. *Software, IEEE*, Vol. 17, No. 4, pp. 26–32, 2000.

[Robi 00] P. Robillard and M. Robillard. “Types of collaborative work in software engineering”. *Journal of Systems and Software*, Vol. 53, No. 3, pp. 219–224, 2000.

[Robi 10] H. Robinson and H. Sharp. “Collaboration, Communication and Coordination in Agile Software Development Practice”. In : *Collaborative Software Engineering*, pp. 93–108, Springer, 2010.

[Robi 91] M. Robinson and L. Bannon. “Questioning representations”. In : *Proceedings of the Second European Conference on Computer-Supported Cooperative Work ECSCW*, pp. 219–233, Springer, 1991.

[Rosc 94] J. Roschelle and S. Teasley. "The construction of shared knowledge in collaborative problem solving". *NATO ASI Series F Computer and Systems Sciences*, Vol. 128, pp. 69–69, 1994.

[Royc 89] W. Royce. "Managing the Development of Large Software Systems : Concepts and Techniques, 1970 WESCON Technical Papers, Western Electronic Show and Convention, Los Angeles, August 1970, pp". Proceedings of the 11th International Conference on Software Engineering, Pittsburgh, 1989.

[Rubi 93] H. A. Rubin. "Software process maturity : measuring its impact on productivity and quality". In : *Proceedings of the 15th international conference on Software Engineering*, pp. 468–476, IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.

[Sari 91] S. Sarin, K. Abbott, and D. McCarthy. "A process model and system for supporting collaborative work". In : *Proceedings of the conference on Organizational computing systems*, p. 224, ACM, 1991.

[Scac 10] W. Scacchi. "Collaboration practices and affordances in free/open source software development". In : *Collaborative software engineering*, pp. 307–327, Springer, 2010.

[Scho 01] M. Schoop. "An introduction to the language-action perspective". *ACM SIGGROUP Bulletin*, Vol. 22, No. 2, pp. 3–8, 2001.

[Scho 99] M. Schoop. "An empirical study of multidisciplinary communication in healthcare using a language-action perspective". In : *Proceedings of the Fourth International Workshop on the Language Action Perspective on Communication Modelling (LAP 99)*, pp. 59–72, Citeseer, 1999.

[Sear 85] J. R. Searle and D. Vanderveken. *Foundations of illocutionary logic*. Cambridge University Press, 1985.

[Srip 08] P. Sriplakich, X. Blanc, and M. Gervals. "Collaborative software engineering on large-scale models : requirements and experience in modelbus". In : *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 674–681, ACM, 2008.

[Stei 08] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF : eclipse modeling framework*. Addison-Wesley Professional, 2008.

[Such 87] L. A. Suchman. *Plans and situated actions : the problem of human-machine communication*. Cambridge university press, 1987.

[Sutt 91] S. Sutton Jr. "APPL/A : a prototype language for software-process programming". 1991.

[Swam 08] K. S. Swaminathan. "Not your father's collaboration". http://www.accenture.com/Global/Research_and_Insights/Outlook/By_Issue/Y2008/fatherscollaboration.htm, 2008.

[Taen 10] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. “Conflict Detection for Model Versioning Based on Graph Modifications”. *Graph Transformations*, pp. 171–186, 2010.

[Tolv 04] J. Tolvanen. “Making model-based code generation work”. *Embedded Systems Europe*, Vol. 8, No. 60, pp. 36–38, 2004.

[Wang 03] Y. Wang, D. DeWitt, and J. Cai. “X-Diff : An effective change detection algorithm for XML documents”. In : *19th International Conference on Data Engineering, 2003. Proceedings*, pp. 519–530, 2003.

[Wass 90] A. Wasserman. “Tool integration in software engineering environments”. In : *Software Engineering Environments*, pp. 137–149, Springer, 1990.

[Weig 06] T. Weigert and F. Weil. “Practical experiences in using model-driven engineering to develop trustworthy computing systems”. In : *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC’06)*, pp. 208–217, 2006.

[Weis 05] A. Weiss. “The power of collective intelligence”. *Networker*, Vol. 9, No. 3, p. 23, 2005.

[Whit 07] J. Whitehead. “Collaboration in Software Engineering : a Roadmap”. In : *FOSE 07 : 2007 Future of Software Engineering*, pp. 214–225, IEEE Computer Society, Washington, DC, USA, 2007.

[Wick 04] M. Wicks. “Tool integration in software engineering : The state of the art in 2004”. 2004.

[Wick 07] M. Wicks and R. Dewar. “A new research agenda for tool integration”. *Journal of Systems and Software*, Vol. 80, No. 9, pp. 1569–1585, 2007.

[Wiki] Wikipedia. “Revision Control”. http://en.wikipedia.org/wiki/Revision_control.

[Wino 86] T. Winograd. “A language/action perspective on the design of cooperative work”. In : *Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, pp. 203–220, ACM, 1986.

[Wirt 96] N. Wirth. “Extended Backus-Naur Form (EBNF)”. *ISO/IEC*, Vol. 14977, p. 2996, 1996.

[Wits 10] H. Witschel, B. Hu, U. Riss, B. Thönssen, R. Brun, A. Martin, and K. Hinkelmann. “A Collaborative Approach to Maturing Process-Related Knowledge”. *Business Process Management*, pp. 343–358, 2010.

[Youn 10] J. Young Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. Rama, and S. Padmanabhuni. “CoDesign–A Highly Extensible Collaborative Software Modeling Framework”. 2010.

[Zaml 05] K. Z. Zamli, N. A. M. Isa, and N. Khamis. “The Design and Implementation of the VRPML Support Environments”. *Malaysian Journal of Computer Science*, Vol. 18, No. 1, pp. 57–69, 2005.

[Zeru 85] E. Zerubavel. *Hidden Rhythms : schedules and calendars in social life*. Univ of California Press, 1985.