Vers une vérification d'un procédé de développement modélisé en SPEM

B. Combemale, X. Crégut
IRIT - LYRE
2, rue Charles Camichel
31071 Toulouse Cedex 7

A. Caplain, B. Coulette
GRIMM - ISYCOM
5, allées Antonio Machado
31058 Toulouse Cedex 9

A. Garcia
Tectosages
Parc technologique Delta Sud
09120 Varilhes

Résumé

L'OMG a proposé le méta-modèle SPEM pour décrire les procédés de développement. Ce langage semi-formel peut être complété par des contraintes OCL pour en préciser formellement la sémantique. Notre objectif est de vérifier un tel procédé. Nous présentons d'abord une approche qui utilise la méthode B, en particulier son raffinement et expliquons pourquoi elle n'a pas été concluante. Nous indiquons ensuite la voie actuellement suivie qui consiste à rester sur les standards proposés par l'OMG, en particulier OCL.

MOTS-CLÉS : Ingénierie des procédés, Ingénierie des modèles, SPEM, Spécification formelle, Méthode B, OCL.

1 Introduction

Nos travaux portent depuis une dizaine d'années sur la modélisation des procédés de développement. Nous avons en particulier défini un langage de description de procédé et un environnement d'exécution [8]. Un défaut de cette approche est que la sémantique du procédé (la manière de l'exécuter) est directement codée dans l'environnement d'exécution et peut donc difficilement être adaptée. Avec l'apparition du méta-modèle SPEM [19] proposé par l'OMG pour modéliser les procédés [3], il nous semble important d'exprimer cette sémantique directement dans le méta-modèle et les modèles de procédés. Elle pourra ensuite être vérifiée avant mais aussi pendant l'exécution du procédé. Si une vérification *a priori* est intéressante pour détecter et corriger au plus tôt les erreurs, il est souvent impossible, dans ce domaine, de le faire systématiquement car le procédé est souvent adapté pendant son déroulement. Les contraintes sémantiques

peuvent alors être utilisées pour contrôler le déroulement et détecter des incohérences qui devront être corrigées.

Dans cet article, nous présentons le méta-modèle SPEM (section 2) et nous l'appliquons à un exemple inspiré de RUP[1] sur lequel nous énonçons quelques propriétés (section 2.2). Nous exposons ensuite la prise en compte de ces contraintes sémantiques au niveau de la modélisation du procédé (section 3), les travaux réalisés avec la méthode B (section 4) et ceux actuellement menés sur OCL (section 5).

2 Présentation de SPEM

2.1 SPEM, langage de modélisation

L'ingénierie des modèles tente de répondre à la problématique des procédés de développement. L'OMG a pour cela proposé le méta-modèle SPEM [19] qui permet la spécification de procédés de développement en utilisant la notation UML. SPEM s'inscrit doublement dans l'organisation de la pile de modélisation du MDA [13] : en tant que méta-modèle conforme au MOF et en tant qu'extension d'UML sous la forme d'un profil.

Bien que l'approche consistant à utiliser SPEM comme langage de modélisation pour la spécification des procédés de développement soit naturelle et s'intègre dans l'approche MDA, elle n'est pas pour autant triviale. En effet, la version actuelle du méta-modèle SPEM proposé par l'OMG se veut très généraliste et par conséquent n'offre pas de directive sur le découpage d'un procédé ni sur l'utilisation de certains stéréotypes propres à ce langage pour lequel la sémantique est principalement exprimée en langage naturel. Des travaux annexes à ceux présentés dans cet article ont consisté à apporter des précisions sur les concepts de ce langage [7]. Nous nous appuierons sur ces travaux afin de préciser le sens de certains stéréotypes.

SPEM reprend une grande partie des diagrammes UML (packages, cas d'utilisation, classes, activités, séquences, états-transitions) mais exclut certains concepts (node, component, ...) [19, §11.1] et ne permet donc pas, par exemple, l'établissement de diagrammes d'implémentations[2]. De nombreux stéréotypes ont par ailleurs été ajoutés pour affiner la sémantique de certains éléments UML [19, §11.4].

Conceptuellement, SPEM repose sur l'idée (fig. 1) qu'un procédé de développement logiciel est une collaboration entre des entités actives et abstraites appelées *rôles* qui exécutent des *activités* sur des entités concrètes et réelles, les *produits*. Les différents rôles agissent les uns sur les autres ou collaborent en échangeant des produits et en déclenchant l'exécution d'activités. L'objectif du procédé est de mener un ensemble de produits à un état bien défini.

En fait le méta-modèle SPEM est plus compliqué. Un extrait est donné figure 2. SPEM définit la notion de *WorkDefinition* qui peut être décomposée en *WorkDefinition*. Outre les activités, il

[1] Rational Unified Process est une méthode proposée par IBM, cf. [11]

[2] Le concept de noeud (*Node*) étant exclu, les diagrammes de déploiement ne peuvent pas être établis. Il en est de même avec le concept de composant (*Component*) et le diagramme de composant. Notons que le concept de composant de procédé présent dans SPEM (*ProcessComponent*) est une spécialisation du concept de paquetage (*Package*) et apparaitra donc dans un diagramme de packages.

existe d'autres spécialisations (non présente sur la figure 2) : *LifeCycle* qui est une séquence de *Phases* et *Iteration* pour définir un travail composite. Une *activité* peut être décomposée en *étapes* (*Step*). Une étape n'est pas une définition de travail et n'est donc pas liée à des produits.

Chaque *WorkDefinition* est sous la responsabilité d'un unique rôle (*ProcessPerformer*). Un ensemble d'autres rôles (*ProcessRole*) peuvent également assister le rôle principal dans la réalisation d'activités. Dans la spécification de SPEM, un rôle est une entité abstraite qui correspond à un ensemble de compétences. Un acteur concret (pas forcément humain) d'un procédé pourra donc jouer un ou plusieurs rôles en fonction de ses compétences. Réciproquement, un rôle peut être joué par un ou plusieurs acteurs.

SPEM définit également les notions de *procédé* et de *discipline*. Un procédé (*Process*) correspond à la racine du modèle de procédé à partir duquel un outil peut faire la fermeture transitive du procédé complet. Une *Discipline* permet au sein d'un procédé de partitionner les activités selon un thème commun. Les produits de sortie de chacune des activités d'une discipline doivent être catégorisées sous ce même thème.

2.2 Exemple de modélisation en SPEM

Nous proposons comme exemple la modélisation simplifiée et de granularité large du procédé de développement RUP [11]. RUP est un procédé de génie logiciel développé par Rational Software. Il s'agit d'un procédé de développement basé sur l'utilisation du standard UML, itératif et incrémental. C'est également un processus basé sur une approche disciplinée afin de bien maîtriser l'assignation des tâches et la responsabilisation des différents acteurs tout au long du cycle de développement. RUP a comme objectif principal de faire appliquer les bonnes pratiques de développement aux entreprises, ce qui confère au produit final une meilleure qualité. La représentation informelle du cycle de vie complet est proposée figure 3. Celle-ci exprime pour chaque discipline l'effort fourni dans chaque phase/itération.

Nous avons exprimé cette spécification du cycle de vie à l'aide du langage SPEM. Nous avons pour cela utilisé le logiciel Enterprise Architect de la société Sparx Systems[3] et son profil SPEM. La figure 4 est la modélisation en SPEM du cycle de vie proposé sur la figure 3 (sans

[3] `http://www.sparxsystems.com.au/`.

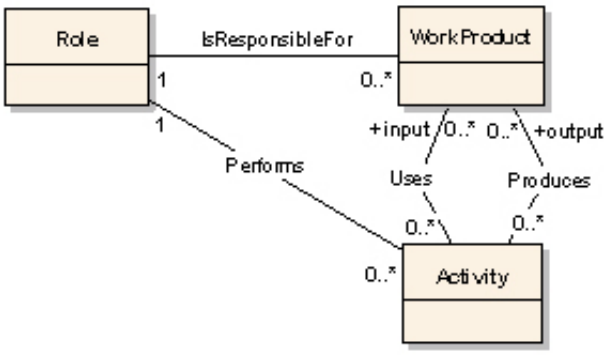

FIG. 1 – Modèle conceptuel de SPEM

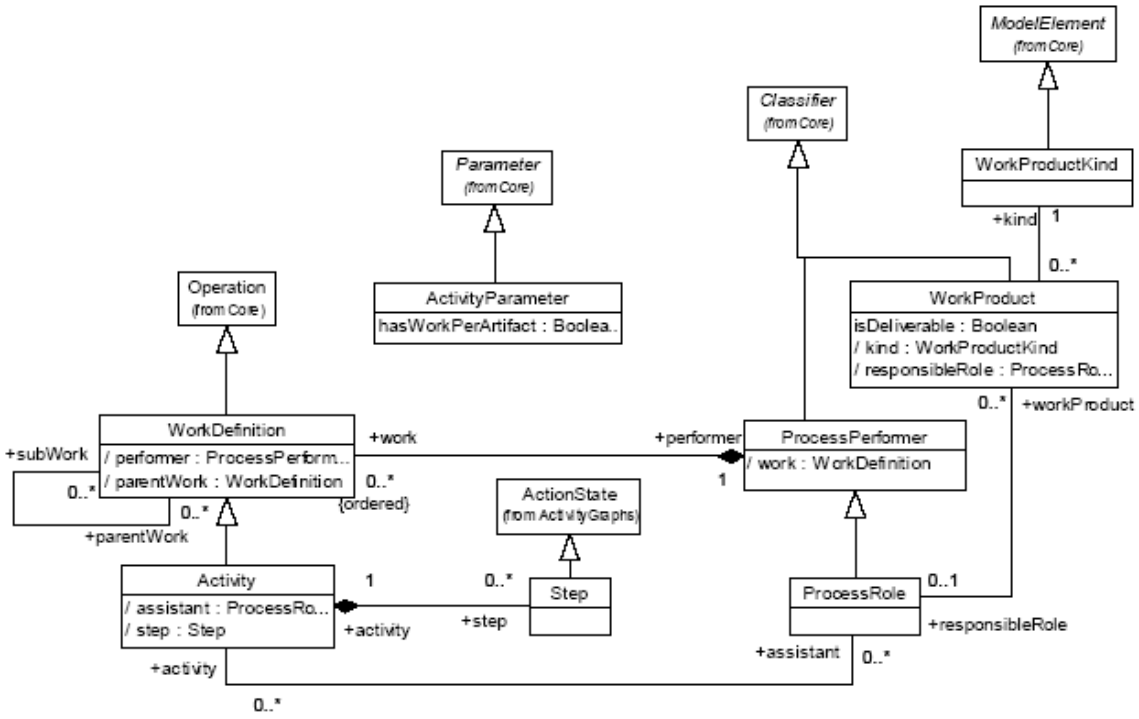

FIG. 2 – Structure d'un procédé SPEM

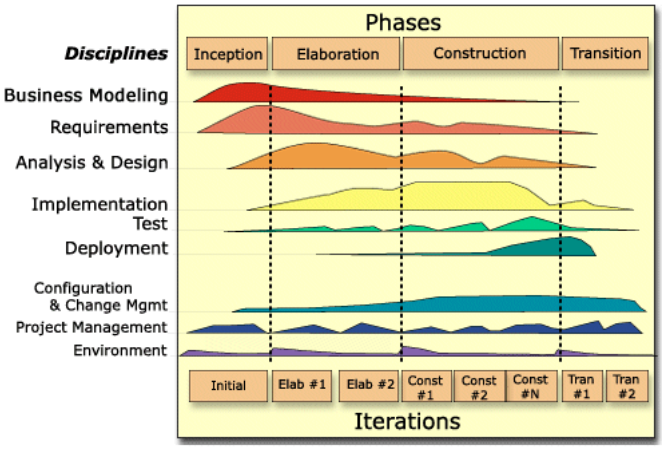

FIG. 3 – Cycle de vie du procédé de développement RUP

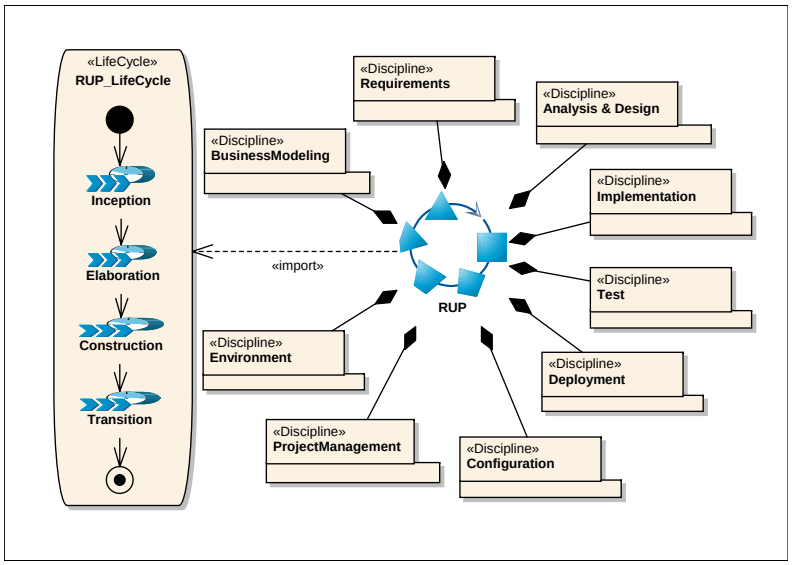

FIG. 4 – Modélisation en SPEM du RUP

les itérations). Au sein de ce modèle, le procédé de développement RUP (*Process*) importe le paquetage du cycle de vie complet (*Lifecycle*) dans lequel sont ordonnées (à l'aide du diagramme d'état) les quatre *Phases* du RUP. Le procédé est également composé de *Disciplines* permettant de classifier les activités de l'ensemble des phases selon des thèmes communs.

Malgré l'intérêt de l'abstraction offerte par ce type de langage graphique, les modèles établis ne permettent pas une vérification et une validation formelle. En effet, la sémantique de cette notation est principalement exprimée en langage naturel et conserve donc une certaine ambiguïté. La validation des modèles reste pourtant indispensable afin d'assurer leur cohérence vis-à-vis du méta-modèle et des spécifications métiers que l'on souhaite transcrire. L'utilisation des méthodes formelles apparaît alors indispensable lorsque l'on souhaite apporter une sémantique précise aux éléments de nos modèles.

D'autre part, RUP est un procédé qui peut être configuré à convenance pour répondre aux besoins spécifiques des organisations. Ainsi, à une telle spécification générique, il est souhaitable d'ajouter un certain nombre de propriétés. Celles-ci peuvent être exprimées soit au sein du méta-modèle pour apporter une sémantique générale aux concepts du langage soit au sein des modèles pour préciser la sémantique des procedés. Nous donnons ci-dessous un classement de ces propriétés.

– Les propriétés *structurelles* permettent de structurer sémantiquement le procédé. Voici l'exemple d'une telle propriété :

  (1) « La réalisation d'une activité ne peut pas être assistée par le rôle qui en a la responsabilité. »

– Les propriétés *de causalité* (qualitative) expriment la temporalité et donc la dynamique sans toutefois quantifier le temps.

  (2) « Pour chaque itération, les activités classées dans la discipline *Business Modeling* doivent être réalisées avant les activités classées dans la discipline

Requirements. »

– Les propriétés *temps réel* (quantitative) qui offrent une temporalité quantifiée au sein du procédé. Voici un exemple tiré des bonnes pratiques de la méthode XP (eXtreme Programming) [1] :

  (3) « Une itération ne doit pas excéder quatre semaines. »

– Les propriétés *opérationnelles* permettent l'expression de la sémantique opérationnelle au sein du modèle de procédé et offrent ainsi la possibilité de l'exécuter.

  (4) « Au démarrage d'une phase, chacune des (instances des) activités doit être affectée à (une instance d')un rôle. »

Notre problématique est d'essayer d'exprimer et de vérifier ces propriétés. Les seules possibilités offertes par SPEM sont d'attacher de manière informelle une *note* (ou *Guidance*) exprimée en langage naturel ou de manière formelle par une contrainte en OCL.

3 Vérification d'un procédé

SPEM propose une description semi-formelle d'un procédé de développement sous la forme d'un méta-modèle instance du langage de méta-modélisation MOF[16] ou d'un profil UML. Nous souhaitons la compléter pour exprimer formellement des propriétés telles que celles listées section 2.2. On peut distinguer les propriétés communes à tout procédé de développement (propriété 1) de celles qui sont liées à un procédé particulier (les autres). Cette distinction modèle / méta-modèle est orthogonale à la classification proposée en 2.2. Il est alors naturel d'exprimer les premières au niveau des concepts du méta-modèle SPEM pour qu'elles s'appliquent automatiquement à toutes ses instances (donc à tous les procédés) et d'exprimer les secondes sur le modèle de procédé concerné.

Une fois la sémantique d'un procédé exprimée, nous pouvons l'exploiter pour vérifier la cohérence du procédé avant et pendant son exécution. Il est bien entendu préférable de faire des vérifications avant exécution. Malheureusement, elles sont généralement difficiles voire impossibles à réaliser . C'est par exemple le cas de la quatrième contrainte qui dit qu'une itération ne doit pas dépasser quatre semaines. Comment le vérifier avant l'exécution d'un procédé ? Notons que le domaine des procédés de développement a ceci de particulier qu'un procédé ne peut être complètement décrit *a priori*. Si les grandes étapes du développement sont généralement connues, le détail de leur réalisation dépend des difficultés rencontrées lors de cette réalisation. Il pourra alors être nécessaire de renforcer une équipe, d'envisager d'autres solutions ou d'autres approches. La vérification du bon déroulement du procédé lors de son exécution devient donc une nécessité. Par exemple, si une itération dépasse la durée maximale, une alerte doit être émise. De la même façon, il doit être interdit de démarrer une activité si les produits qu'elle utilise ne sont pas encore disponibles.

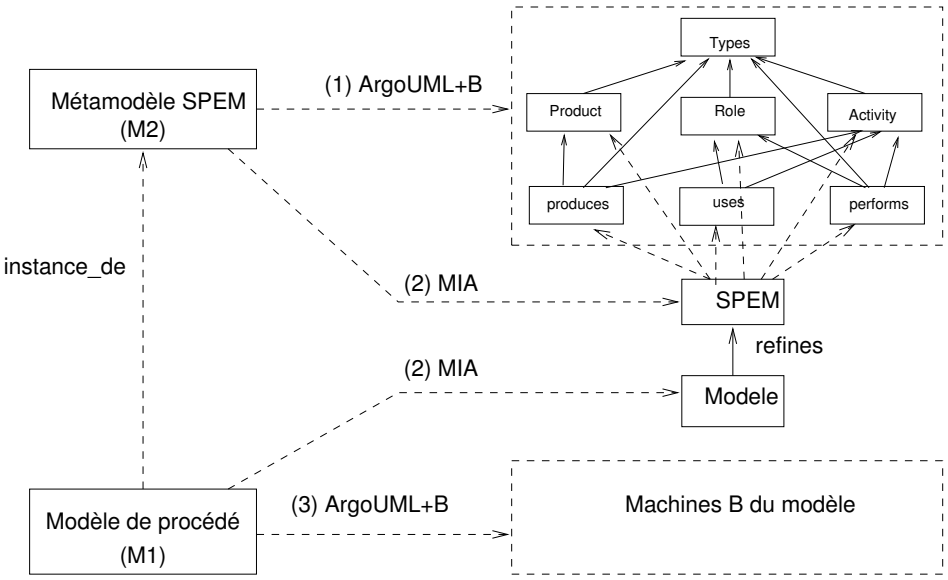

FIG. 5 – Transformations SPEM vers B

4 Vérification avec B

La méthode B est basée sur le principe de « machine abstraite ». Une machine abstraite décrit des constantes, des variables et des opérations à l'aide de concepts mathématiques tels que ensembles, relations, fonctions, séquences, etc. Les propriétés des données et des opérations sont décrites à l'aide de la logique des prédicats et de la logique des propositions. Une machine abstraite est une spécification qui peut être raffinée plusieurs fois jusqu'à l'obtention d'un code exécutable. La cohérence de chaque raffinement peut être vérifiée par l'Atelier B qui engendre les obligations de preuves correspondantes et fournit un démonstrateur automatique ou interactif.

Notre démarche a été de décomposer la vérification des procédés en trois étapes (figure 5). Nous l'avons appliquée sur la version simplifiée de SPEM présentée figure 1 et à un procédé très simple donné figure 6 comme une instance du métamodèle SPEM. Ce procédé décrit l'activité d'écriture d'une spécification détaillée à partir d'une spécification générale.

L'*étape 1* consiste à transformer le méta-modèle SPEM en un ensemble de machines B dont nous vérifions la cohérence. Cette transformation a été réalisée en utilisant les travaux menés au Loria à Nancy [12, 9] sur l'outil ArgoUML+B. Nous avons utilisé la partie transformation du diagramme de classe. Chaque classe et relation donne une machine B. Les propriétés du méta-modèle sont ajoutées manuellement sur les machines B. ArgoUML+B devrait à terme prendre en compte les contraintes OCL exprimées sur un modèle UML.

Dans l'*étape 2*, nous vérifions que le modèle de procédé est conforme à son méta-modèle (SPEM). Nous nous sommes inspirés des travaux de Mamoun Filali [20]. Le principe est de définir une machine B qui correspond au méta-modèle de SPEM (machine SPEM sur la figure 5 qui étend les machines B du méta-modèle de l'étape 1) et d'exprimer le modèle de procédé comme un rafinement de la machine SPEM. Il s'agit d'initialiser les variables de la machine SPEM avec les éléments du modèle. L'atelier B engendre alors les obligations de preuve qui

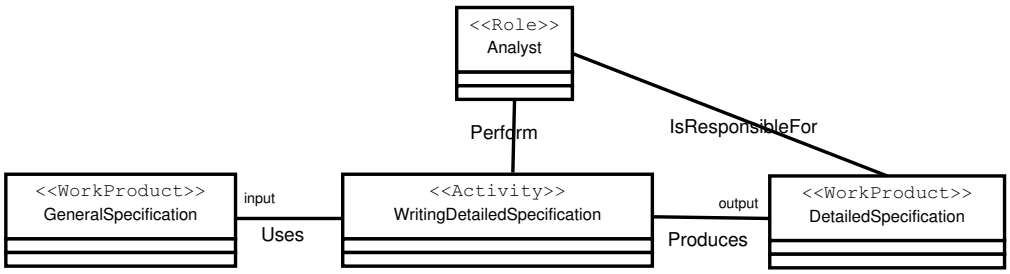

FIG. 6 – Modèle SPEM de l'activité « Écriture de la spécification détaillée »

garantissent que le modèle est conforme à son métamodèle.

L'*étape 3* consiste à transformer le modèle du procédé en un ensemble de machines B sur le même principe que pour le méta-modèle. Sur ces machines nous ajoutons les propriétés propres au modèle. Notons que ces machines ne peuvent pas être considérées comme un raffinement des machines du méta-modèle car nous souhaitons étudier et vérifier les interactions entre ces machines et pas seulement entre leurs abstractions.

Enfin, pour pouvoir exécuter un procédé, nous ajoutons des opérations sur les machines B telles que « commencer » et « terminer » sur les activités, « construire » sur un produit, etc. La sémantique de ces opérations est d'abord définie au niveau du méta-modèle SPEM (e.g. on ne peut commencer une activité que si les activités précédentes sont commencées). Dans les machines du modèle, cette sémantique est affinée pour expliquer le traitement spécifique. Ainsi, en utilisant l'animateur de l'atelier B nous pouvons suivre et contrôler le développement réel.

Lors de cette utilisation de B, nous avons été confrontés à plusieurs difficultés. La première concerne la difficulté à prouver les obligations de preuve, en particulier dans la vérification du raffinement, et ceci malgré la simplicité du méta-modèle SPEM et du procédé utilisés.

La seconde difficulté concerne l'impossibilité que nous avons eue à faire cohabiter dans les machines B les « méta-entités », les « entités » et leurs instances. Par exemple, les propriétés communes à toutes les activités doivent être exprimées au sein de la notion d'activité de SPEM, les propriétés d'une activité particulière, par exemple l'activité de test, doivent être exprimées sur la machine test qui elle-même pourra avoir plusieurs instances lors de l'exécution du procédé. Pour exprimer les propriétés opérationnelles nous sommes alors obligés de les dupliquer dans toutes les machines des « entités ».

Au total, au vu des difficultés rencontrées et de la lourdeur des transformations nécessaires, nous avons décidé d'explorer une autre voie, celle qui consiste à rester dans le domaine des standards de l'OMG.

5 Formalisation avec OCL

La communauté d'où est issue l'Ingénierie Des Modèles (IDM) a proposé dans un premier temps des langages de spécification graphique permettant dans de nombreux domaines la spécification abstraite par les modèles (logiciels, bases de données, procédés, etc.). Il a toutefois été montré que ces langages sont limités dans leur pouvoir d'expressivité et dans la définition

formelle de leur sémantique [21]. Les manques et les ambiguïtés que l'on peut trouver au sein des modèles ne permettent donc pas l'interprétation ni la vérification formelle nécessaires pour considérer ces modèles comme « exécutables ».

5.1 Présentation du langage OCL

Afin de formaliser la sémantique des modèles, Jos Warmer a proposé OCL, langage formel pour l'expression de contraintes et de requêtes appliquées à des diagrammes UML. OCL a été intégré à UML 1.1 en 1999 [15].

Au sein de l'IDM, le langage OCL sert aussi bien à la définition des standards proposés par l'OMG qu'à la modélisation de l'application. Dans les deux cas, l'objectif est le même : il s'agit d'attacher à un modèle un certain nombre de propriétés qui devront être respectées dans l'ensemble des instances de ce modèle.

Au niveau du méta-modèle, OCL 2.0 apporte un certain nombre de modifications afin de mieux s'intégrer dans l'approche MDA de l'OMG. Les concepts ont pour cela été en grande partie intégrés au MOF afin de rendre son intégration à UML et aux autres méta-modèles plus naturelle. Cela permet sont utilisation au sein de différents domaines sémantiques[10].

Concrètement, on retrouve deux définitions de la syntaxe au sein de la spécification. La première utilise une approche basée sur le méta-méta-modèle MOF (règles *wellformedness*), c'est *la syntaxe abstraite* [17, §8]. La deuxième s'appuie sur l'utilisation d'un ensemble de théories et propose une grammaire, c'est *la syntaxe concrète* [17, §9].

La syntaxe OCL étant limitée sur un certain nombre de points [6], il fût logique de voir très rapidement apparaître de nombreuses extensions à ce langage. Les premières et les plus nombreuses intègrent la logique temporelle. Citons par exemple les extensions TOCL (i.e. *Temporal OCL*) [22] et OCL+ [2] offrant des opérateurs temporels afin de spécifier des contraintes par rapport à un ou plusieurs enchaînements d'états souhaités ou interdits.

5.2 OCL pour la formalisation des modèles de procédé

L'approche que nous proposons consiste à utiliser le langage OCL pour compléter les modèles de procédés de développement exprimés en SPEM. Cette démarche s'intègre dans celle de l'OMG qui s'efforce de fournir des langages graphiques semi-formels pour chaque domaine et propose OCL comme langage formel commun pour l'expression de contraintes.

Cette approche pose toutefois un certain nombre de problèmes. En effet, OCL est très lié au méta-modèle UML car au niveau de sa syntaxe abstraite il est relié aux notions d'*objet* et d'*instance* du méta-modèle UML. On ne peut donc appliquer des règles OCL sur des modèles SPEM que si SPEM garde une définition en tant que profil UML et hérite ainsi des concepts nécessaires. Cette application d'OCL doit d'ailleurs être traitée dans les réponses à l'*appel à proposition* lancé par l'OMG pour la future version 2.0 de SPEM [18]. Notons toutefois que les contraintes apparaissent dans la définition de la structure du fichier XMI associé à une modélisation SPEM.

D'autre part, il doit être impossible d'instaurer des contraintes ne respectant pas les méta-modèles ou contredisant le diagramme (par exemple les multiplicités). Les contraintes doivent

être respectées au cours des exécutions possibles du modèle (validation dynamique) sinon elles révèlent un défaut qui devra être corrigé (évolution dynamique du procédé).

5.3 Exemples de formalisation en OCL

Sur la base de l'exemple présenté dans la section 2.2, OCL permet de compléter les modèles décrits en SPEM afin d'exprimer des contraintes qui n'ont pas pu l'être de manière graphique. Les contraintes sont exprimées au sein du méta-modèle ou des modèles en prenant une classe ou un de ses membres comme contexte[4].

La propriété (1) peut être directement exprimée en OCL sur la méta-classe *Activity* du méta-modèle SPEM de la manière suivante :

context Activity *inv* :
self.assistant→ *excludes* (self.performer)

Le pouvoir expressif d'OCL reste toutefois encore limité et ne permet pas d'exprimer toutes les contraintes nécessaires à une spécification complète et rigoureuse d'un modèle de procédé [6]. Les principales limites sont le manque d'effet de bord, l'absence d'état courant et la nécessité de recourir à des extensions pour l'expression de contraintes temporelles.

Par ailleurs, le langage OCL permet d'exprimer des contraintes sans toutefois offrir la possibilité d'accéder aux niveaux d'abstraction supérieur. Certaines contraintes pourraient être exprimées au niveau du méta-modèle mais ceci n'est pas nécessairement souhaitable afin de garder les propriétés d'un procédé au sein de son modèle. Dans le cadre de l'exemple présenté dans la section 2.2, nous rencontrons le problème dans la propriété (2). Dans ce cas, la solution la plus facile (mais la moins pertinente) consisterait à exprimer sur chaque itération le fait que les activités de la première discipline doivent être terminées avant les activités de la deuxième discipline. Cette solution pose toutefois une première limite car il serait nécessaire, pour l'expression de la contrainte, d'ajouter un certain nombre de propriétés au concept d'« activité » (e.g. date de début, durée, etc.). Il faudrait pour cela modifier la description de la méta-classe « Activity » du méta-modèle SPEM. La deuxième solution, permettant une abstraction supplémentaire de l'expression de la propriété, consisterait à généraliser la contrainte à toutes les itérations et ainsi de ne l'écrire qu'une seule fois. Cette solution nécessiterait de spécialiser la sémantique du concept d'itération en prenant comme contexte la méta-classe « Iteration » tout en exprimant la contrainte au sein du modèle pour garder le détail des différentes disciplines (i.e. les différentes instanciations de la méta-classe « Discipline »). L'expression de cette contrainte nécessite donc de pouvoir accéder aux éléments de modèles situés à différents niveaux d'abstraction : le modèle et le méta-modèle.

Notons que ce principe consistant à modifier la description ou la sémantique d'une méta-classe est différent de la notion de profil qui entraînerait la spécialisation du méta-modèle pour la spécification de chaque procédé de développement. La réalisation d'un profil doit être strictement réservée à l'établissement d'un langage spécifique à un domaine (DSL) permettant l'instanciation de divers modèles.

La propriété (3) proposée dans l'exemple de la section 2.2 peut pour sa part être exprimée en OCL en prenant comme contexte la méta-classe « Iteration » et en ajoutant une propriété à

[4]Le contexte définit le type auquel la contrainte se rapporte

la description de la méta-classe. Elle ne pourra toutefois être vérifiée qu'à l'exécution car il est indispensable pour cela d'exécuter le procédé de développement et de vérifier ainsi que la durée d'une itération n'excède pas quatre semaines.

Enfin, la propriété (4) ne peut pas être exprimée à l'aide du langage OCL et nécessiterait l'utilisation d'un langage de sémantique opérationnelle.

6 Conclusion

Dans cet article, nous avons utilisé le méta-modèle SPEM complété par des contraintes OCL exprimées soit au niveau du méta-modèle, soit au niveau d'un procédé de développement pour les propriétés spécifiques. En nous inspirant des travaux réalisés à l'IRIT (utilisation du raffinement B pour montrer la conformité d'un modèle par rapport à un méta-modèle) et au Loria (transformation UML vers B), nous avons proposé un schéma de traduction de SPEM vers B pour vérifier la conformité entre un modèle de procédé et son méta-modèle ainsi que les propriétés propres au modèle et au méta-modèle. Malheureusement, cette voie s'est révélée infructueuse en raison des difficultés à prouver les obligations de preuves engendrées par les outils B et l'impossibilité de faire cohabiter sous forme de machines B les notions issues du méta-modèle, du modèle de procédés et de ses instances. Pour ces raisons, nous avons décidé de nous concentrer sur SPEM et OCL. Si OCL nous permet d'exprimer les propriétés structurelles, il ne nous permet pas d'exprimer le comportement à l'exécution d'un procédé pourtant nécessaire pour animer un procédé et contrôler le développement réel. En conséquence, nous étudions actuellement la possibilité de définir une sémantique opérationnelle [5] pour SPEM et étudions pour cela les approches existantes telles que KerMeta[14], xOCL [4]...

Références

[1] Kent Beck. *eXtreme Programming*. Génie logiciel. Paris, CampusPress Référence edition, août 2002. 223 p.

[2] Jean-Paul Bodeveix, Thierry Millan, Christian Percebois, Christophe Le Camus, Pierre Bazex, and Louis Feraud. Extending OCL for verifying UML models consistency. In *UML 2002*, pages 75–90, 2002.

[3] Alain Caplain. Checking software development processes. In *IFIP Student Forum*, pages 113–122, 2004.

[4] Tony Clark, Andy Evans, Paul Sammut, and James Willans. Applied metamodelling - a foundation for language driven development. version 0.1, 2004.

[5] Benoit Combemale, Sylvain Rougemaille, Xavier Crégut, Frédéric Migeon, Marc Pantel, Christine Maurel, and Bernard Coulette. Towards a rigorous metamodeling. In Springer-Verlag, editor, *2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2006)*, 2006.

[6] Benoît Combemale. Spécification et vérification de modèles de procédés de développement. Master's thesis, Université Toulouse II - INPT ENSEEIHT, Master SLCP, juin 2005.

[7] Benoît Combemale, Xavier Crégut, Alain Caplain, and Bernard Coulette. Modélisation rigoureuse en SPEM de procédé de développement. In Hermès Sciences / Lavoisier, editor, *12ième conférence sur les Langages et Modèles à Objets (LMO 2006)*, pages 135–150, 2006.

[8] Xavier Crégut and Bernard Coulette. PBOOL : an object-oriented language for definition and reuse of enactable processes. *Software Concepts & Tools*, 18(2), June 1997.

[9] H.Ledang and J.Sousquières. Integrating UML and B specification techniques. In *The Informatik2001*, Vienna (Austria), septembre 2001. Workshop on Integrating Diagrammatic and Formal Specification Techniques.

[10] Heinrich Hussmann and Steffen Zschaler. The Object Constraint Language for UML 2.0 - overview and assessment. *UPGRADE*, (2) :25–28, avril 2004.

[11] Per Kroll and Philippe Kruchten. *Guide pratique du RUP*. PEARSON Education France, 2003.

[12] Eric Meyer and Thomas Santen. Behavioral conformance verification in an integrated approach using UML and B. *IFM 00*, Novembre 2000.

[13] Joaquin Miller and Jishnu Mukerji. *Model Driven Architecture 1.0.1 Guide*. OMG, Inc., 2003.

[14] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In S. Kent L. Briand, editor, *LNCS*, Montego Bay, Jamaica, october 2005. MODELS/UML'2005, Springer.

[15] OMG, Inc. *Object Constraint Language 1.1 Specification*, 1997.

[16] OMG, Inc. *Meta Object Facility 2.0 Core Specification*, 2003.

[17] OMG, Inc. *UML Object Constraint Language 2.0 Specification*, 2003.

[18] OMG, Inc. *Software Process Engineering Metamodel 2.0 RFP*, 2004.

[19] OMG, Inc. *Software Process Engineering Metamodel 1.1*, 2005.

[20] M. Rached. Spécification de la sémantique statique de HRT-HOOD. Master's thesis, DEA-PS Université Paul Sabatier Toulouse, 2002.

[21] Jos Warmer and Anneke Kleppe. OCL : The constraint language of the UML. *Journal of Object-Oriented Programming*, mai 1999.

[22] Paul Ziemann and Martin Gogolla. An extension of OCL with temporal logic. *UML'02 workshop*, pages 53–62, 2002.