PMS+: Un outil pour les processus de production de logiciels

Amen Souissi, Cedric Dumoulin, Pierre Boulet

PMS+ : Un outil pour les processus de production de logiciels

Amen SOUISSI, Cedric DUMOULIN and Pierre BOULET

[1] Ecreall, France
amensouissi@ecreall.com
[2] LIFL, France.
cedric.dumoulin@lifl.fr
[3] LIFL, France.
pierre.boulet@lifl.fr

Résumé

Toutes approche IDM nécessite l'enchaînement de plusieurs transformations. Cet enchaînement peut être vu comme un processus de production de logiciels. Dans le plus simple des cas, cet enchaînement est linéaire : les transformations s'exécutent les unes après les autres. Cependant, il arrive souvent que l'on ait besoin de contrôler l'enchaînement lors de l'exécution : par exemple, on peut vouloir choisir la transformation suivante en fonction du résultat des transformations précédentes. Dans cet article, nous décrivons l'outil **PMS+** permettant de spécifier l'enchaînement de transformations en tant que processus de production de logiciels.

1 Notre problématique

Sur un plan technique, un développeur a besoin d'un cahier des charges complet décrivant le système à implémenter et un ensemble de connaissances sur un ensemble de langages lui permettant d'implémenter le système. Cet ensemble de connaissances se matérialise par la maîtrise du langage (grammaire, syntaxe...) et l'ensemble de librairies lui permettant de ne pas réinventer la roue. Le développeur a donc comme rôle de transformer le cahier des charges écrit en langage humain en un langage compréhensible par la machine. Cela en se basant sur les connaissances acquises par le développeur (grammaire, syntaxe , librairies...). Pour passer du domaine de l'humain au domaine de la machine, le cahier des charges passe par un processus de production jusqu'à l'application finale. Ce processus est décrit par les différentes activités à appliquer sur le cahier des charges ainsi que le flux d'exécution de ces activités orienté selon différentes contraintes, par exemple le choix des librairies les plus adaptées ou le type de la plate-forme cible.

Dans le cadre d'une approche IDM [9] ce processus est généralement automatisé et est centré sur le modèle. Par conséquent, le modèle est considéré comme un objet auquel nous pouvons poser des questions (c'est-à-dire faire des opérations) et non comme un simple fichier. Les activités sont principalement des transformations de modèles, de la génération de code ou des boites noires. Les contraintes permettant d'orienter le flux d'exécution, quant à elles, portent sur les modèles manipulés par le processus. Ce qui constitue une spécificité pour les processus de production IDM par rapport aux workflow [5] qui présentent une vue automatisable des processus métier.

D'une manière générale, un processus de production de logiciels prend un ensemble de modèles décrivant l'application à concevoir et produit l'ensemble de code source correspondant. Au cours de l'exécution du processus, le concepteur est amené à faire des choix, comme par exemple spécifier l'emplacement des modèles d'entrées ou la destination du code à générer. Ces

choix peuvent-être fixés à l'avance, mais il peut être intéressant de rendre certains d'entre eux interactifs : le système demande alors les valeurs lors de l'exécution du processus.

Il arrive souvent que des processus de production de logiciels aient des parties similaires. Il est intéressant de pouvoir capitaliser sur ces parties similaires, en ayant la possibilité de les réutiliser dans différents processus de production. Une approche intéressante est de voir le processus de production comme étant une activité elle-même réutilisable dans un autre processus.

Dans un contexte où les technologies de transformation de modèles et de génération de code sont devenues de plus en plus nombreuses, un processus de production doit accepter l'hétérogénéité des technologies et donc permettre d'enchaîner différentes technologies dans un même processus.

En résumé, un processus de production IDM doit respecter les points suivants :

- Le processus doit décrire un enchaînement d'activités avec un flux d'exécution orienté selon des contraintes (c'est-à-dire, une execution conditionelle).
- Dans un processus les modèles doivent être manipulables en tant qu'objets et non uniquement comme des simples fichiers.
- Un processus peut être interactif. Nous devons avoir, dans ce cas, une gestion simple des interactions (avec l'utilisateur ainsi qu'avec le système de fichier).
- Dans un processus, les activités doivent être réutilisables afin de simplifier la modélisation.
- Le processus peut être hétérogène et indépendants des téchnologies utilisées par les activités. Nous devons avoir, dans ce cas, la possibilité de rajouter facilement des activités de différentes technologies.

Un processus de production de logiciels est avant tout un procédé logiciel [8], citons par exemple SPEM [11] ou SPEM4MDE [15], dont les activités sont orientées vers le développement de l'application et non la gestion des équipes de développement. En effet, les procédés logiciels sont des approches généralistes centrées sur l'organisation du travail au sein d'une équipe dont le but est de produire des logiciels.

Le problème d'enchainement des transformations, dans une approche IDM, a été traité dans plusieurs approches. Nous trouvons, par exemple, les approches classiques qui se base des langes de script comme ANT [10]. Ces approches ne sont pas dédiées aux approches IDM et reste difficile à adapter. Dans [1], les auteurs présentent l'environnement MCC permettant la description textuelle de l'enchainement des transformations de modèle. Cette approche permet la réutilisation des transformations ainsi que l'exécution conditionnelle de leur enchainement. Tout en adoptant une description textuelle des processus, le moteur MWE [3] permet l'enchainement simple (non conditionnel) des activités et ne se limite pas seulement aux transformations de modèle. D'autres approches ont adopté une description à base de modèle ce qui a permis une utilisation plus simple. Nous citons par exemple le moteur Acceleo [2] qui permet l'enchaînement simple des templates de génération de code Acceleo ou l'outil ATLFlow [14] permettant de décrire un enchainement conditionnel des transformations ATL. Ces outils sont dépendants de la technologie utilisée pour les activités. Le besoin d'enchainer des activités avec différentes technologies a été évoqué dans plusieurs approches. Nous citons, par exemple, celle de transML [6] qui permet un enchainement simple des transformations ou le travail fait dans [7] permettant un enchainement conditionnel de différentes activités. Dans [16] les auteurs présentent l'outil UniTI permettant un enchainement simple des processus de production. Cet outil se focalise sur la réutilisation des activités de différentes technologies. Les technologies, dans ce cas, sont décrites dans le métamodèle utilisé par cette approche. Ce qui la rend dépendante des technologies utilisées pour les activités.

Ces approches montrent des manques au niveau de l'interactivité avec l'utilisateur ainsi qu'avec le système de fichier. La pluspart de ces approches sont dépendantes des technolo-

gies des activités, par exemple le moteur Acceleo ou l'outil ATLFlow. Les approches revendiquant l'hétérogénéité des technologies présentent une dépendance aux technologies au niveau du métamodèle, par exemple UniTI. Ce qui rend l'ajout d'une nouvelle technologie difficile. Aucune de ces approches ne considère le modèle comme un objet manipulable. Les conditions permettant d'orienter le flux d'exécution portent généralement sur des expressions manipulant des variables primitives. Enfin ces approches restent difficiles à adapter pour respecter les points cités plus haut.

Dans cet article, nous présentons l'outil **PMS+** (**P**rocess **M**anufacturing **S**oftware and +) qui permet de décrire et d'exécuter des processus de production de logiciels. Nous commençons par une description générale du fonctionnement du moteur d'exécution puis nous décrivons les concepts de base et les particularités techniques, enfin nous exposons un cas d'utilisation sur lequel nous avons validé notre moteur.

2 Principe générale du fonctionnement du moteur

Nous allons illustrer le fonctionnement de notre moteur sur un exemple dans lequel deux transformations de modèles sont enchaînées (voir figure 1). Cet exemple de processus permet de transformer un modèle UML *mymodel*, chargé à l'exécution, en un modèle Python. Le résultat est enregistré sur le système de fichiers dans le dossier *myfolder*. Le nom du fichier résultat est génère automatiquement par le moteur.

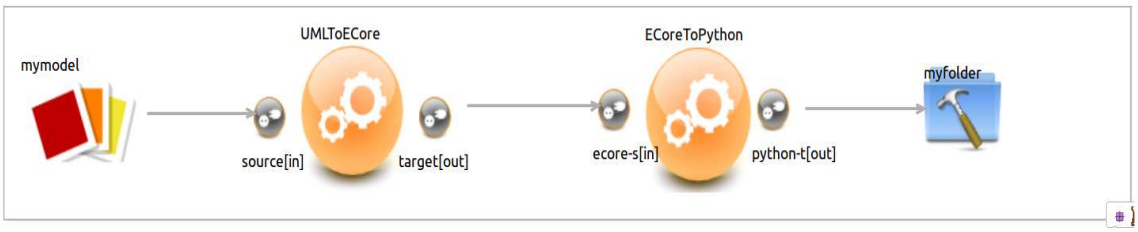

FIGURE 1 – Exemple d'un processus avec deux transformations de modèles

Dans notre approche, les transformations sont considérées comme des activités à exécuter par le moteur de **PMS+**. La transformation *UMLToEcore* permet de transformer un modèle UML en un modèle Ecore. Elle a un paramètre d'entrée *source* de type *métamodèle UML* et un paramètre de sortie *target* de type *métamodèle Ecore*. La transformation *EcoreToPython*, quant à elle, permet de transformer le modèle *Ecore* en un modèle *Python*. Cette seconde transformation a un paramètre d'entrée *ecore-s* de type *métamodèle Ecore* et un paramètre de sortie *python-t* de type *métamodèle Python*. Le paramètre *source* est connecté à la donnée *mymodel* qui représente le modèle d'entrée UML. Cette donnée permet de charger le modèle qui est utilisé comme un objet et non comme un fichier. Le paramètre *ecore-s* est connecté au paramètre de sortie *target* et enfin le paramètre *python-t* est connecté à la donnée *myfolder* qui présente le dossier dans lequel nous voulons enregistrer le résultat du processus. Cette donnée permet de référencer un dossier existant sur le système de fichiers.

Notre processus est complet. Nous pouvons à présent l'exécuter. Le moteur de **PMS+** commence par identifier les transformations dont tous les paramètres d'entrées sont présents. Dans notre cas, le moteur commence par exécuter la transformation *UMLToEcore* car son paramètre d'entrée *source* est présent et prend la valeur de la donnée *mymodel*. Le paramètre d'entrée

ecore-s de la transformation *EcoreToPthon* n'est pas encore présent puisque celui-ci est conditionné par la présence du paramètre *target*. La donnée *mymodel* transite sur la connexion pour atteindre le paramètre *source* ensuite la transformation *UMLToEcore* est déclenchée. Celle-ci produit un modèle de sortie sur le paramètre de sortie *target*. Cette donnée transite sur la connexion pour atteindre le paramètre *ecore-s*. De ce fait, les paramètres d'entrées de la transformation *EcoreToPython* sont présents ce qui permettra de la déclencher. Cette transformation produit un modèle de sortie sur le paramètre *python-t*. Cette donnée sera ensuite envoyée sur la donnée *myfolder* qui se charge de sauvegarder le modèle résultat sur le système de fichier dans le dossier référencé.

Dans la section suivante, nous présentons plus en détail les principaux concepts de **PMS+**.

3 Les principaux concepts

Les principaux concepts de notre approche sont les **activités**, les **données** et les **flux de données**. Le processus de production est décrit comme un graphe orienté dont les nœuds sont des **activités** ou des **données** et les arcs sont des **flux de données**. Ces derniers permettent de connecter les données aux paramètres des activités, considérés comme les point de connexion des activités, ainsi que les paramètres entre eux.

Dans **PMS+** les **activités** sont les unités à exécuter. Elles peuvent être une *transformation de modèle*, une *génération de code*, un *moteur d'exécution*, une *boîte noire* ou une *référence vers un autre processus de production*. Les activités de type *moteur d'exécution* permettent d'exécuter dynamiquement d'autres activités. Ces activités prennent en général une activité comme paramètre d'entrée ainsi qu'un modèle décrivant les données à fournir en entrée pour l'activité à exécuter. Les boites noires permettent de réaliser des tâches complexes, par exemple faire appel à un service ou une application externes. Dans **PMS+** un processus de production est lui-même considéré comme une activité, permettant ainsi la réutilisation des processus dans d'autres processus.

Une **activité** prend des paramètres d'entrée et/ou de sortie. Ces paramètres sont typés et ont une multiplicité. Les paramètres représentent ce que l'activité attend ou produit comme données. Les **données** représentent les objets véhiculés entre les activités. nous pouvons spécifier des données primitives ou complexes. Pour les données complexes, nous pouvons spécifier des ressources permettant de référencer des éléments présents sur le système de fichiers, par exemple un fichier ou un dossier. Pour les données primitives, nous pouvons spécifier des entiers ou des booléens.

Dans un processus de production, il arrive souvent que le processus ait besoin de récupérer des données en fonction des résultats des activités exécutées. Pour cela, nous avons le concept d'*expression* permettant, à l'aide d'une expression OCL, d'exprimer une requête ou un filtre sur les données. Les expressions sont calculées dynamiquement à l'exécution du processus. *L'expression* permet entre autres de renvoyer le résultat d'une requête sur un modèle fourni par une activité. Par exemple, elle permet d'obtenir tous les éléments dont le type est "Class".

PMS+ propose un concept de *donnée utilisateur*. Ce concept permet de spécifier des interactions avec l'utilisateur, comme la saisie de donnée. Ce concept est une donnée, et est donc représenté comme un nœud dans le graphe du processus. Ce nœud est relié à un ou plusieurs paramètres d'une ou plusieurs activités. Lorsqu'une de ces activités est prête à être exécutée (sans tenir compte des paramètres reliés à la donnée utilisateur), **PMS+** propose une boîte de dialogue permettant la saisie des valeurs pour tous les paramètres reliés à la *donnée utilisateur*. Le moteur génère dynamiquement la boîte de dialogue en se servant des métadonnées des paramètres comme par exemple son type, son nom ou sa description.

Les flux de données sont orientés. La direction du flux indique le sens de circulation des données. La source et la destination d'un flux doivent avoir le même type. Par exemple, un modèle peut être connecté à un paramètre d'entrées de type métamodèle d'une activité. Cela indique que les données correspondant au modèle iront du nœud modèle vers le paramètre. Un **flux de données** implique une dépendance de données entre la source et la destination (on dit alors que la destination dépend de la source). Dans le cas où la source et la destination sont des paramètres, le flux de données implique une dépendance entre les activités de ces paramètres. L'activité source est alors exécutée avant l'activité destination. Les flux de données permettent au moteur de **PMS+** d'ordonnancer les différentes activités du processus. D'une manière générale, l'ordonnancement des activités est calculé selon les dépendances des données existant entre les paramètres. Dans **PMS+**, la notion de dépendance des données est déduite des flux de contrôle. Il arrive parfois que deux activités aient une dépendance entre elles qui n'est pas matérialisée par une dépendance de données. Pour résoudre ces cas, **PMS+** permet d'exprimer explicitement une dépendance entre deux activités. Une fois l'ordonnancement fait, le moteur commence par exécuter les activités dont tous les paramètres d'entrées sont présents. Les activités suivantes sont exécutées dès que leurs paramètres d'entrée sont à leur tour présents.

Les *flux de données* peuvent être contrôlés par des *gardes* à l'aide d'expressions booléennes indiquant si la donnée peut circuler sur le lien. Si l'expression est évaluée à vrai, la donnée peut aller de la source à la destination. Si l'expression est évaluée à faux la donnée est bloquée et par conséquent l'activité suivante ne sera pas activée. Ceci nous permet d'exprimer des contraintes bien définies sur le flux d'exécution.

Dans la section suivante, nous parlons des aspects techniques de **PMS+**, comme l'ajout d'une technologie, et sa conception basée sur la réutilisation.

4 Les points techniques particuliers

PMS+ est implémenté par un ensemble de plugins Eclipse. L'outil est développé suivant une démarche basée sur la réutilisation. Ainsi, la plupart des fonctionnalités de **PMS+** sont décrites sous forme de *processus de production* qui sont exécutés par le moteur d'exécution. L'ajout de nouvelles fonctionnalités peut se faire par ajout de nouveaux processus. Cette approche donne la possibilité d'exécuter des processus de production à la volée. Cette technique basée sur la réutilisation nous permet d'maintenir ainsi que faire évoluer **PMS+** plus facilement.

Les activités dans **PMS+** peuvent utiliser des différentes technologies : QVT [1], ATL [2], Java, etc. Si une technologie n'est pas supportée, **PMS+** permet de l'ajouter : il faut pour cela fournir le moteur d'exécution correspondant sous la forme de classes Java implémentant certaines interfaces. Le système d'ajout est basé sur le concept de point d'extension d'Eclipse [3]. Actuellement, **PMS+** supporte les moteurs d'exécution de QVT, Acceleo, Java et celui des *processus de production*.

Dans ce qui suit nous montrons comment ajouter une activité ainsi qu'une technologie dans **PMS+**.

4.1 Comment ajouter une description d'une activité ?

L'ajout d'une description est le fait de la sauvegarder dans le registre des descriptions de **PMS+**. Ce registre permet de garder une référence sur le comportement ainsi que le modèle le

1. http ://www.omg.org/spec/QVT
2. http ://www.eclipse.org/atl/
3. http ://www.eclipse.org/resources/resource.php ?id=495

décrivant. Dans **PMS+**, nous faisons la différence entre installer une description et l'enregistrer dans **PMS+**. En effet, l'installation implique que la description est validée par le concepteur et peut être intègrée dans un projet (par exemple, une application Eclipse). L'installation peut être faite dans le but de l'exporter définitivement dans un projet. Dans ce cas, elle est faite en packageant le comportement dans un plugin Eclipse. Ce plugin doit avoir un point d'extension permettant de déclarer le comportement en tant que description d'activité dans **PMS+**. Le point d'extension demande un modèle décrivant le comportement (le modèle de description de l'activité). Selon ce modèle le comportement est installé en tant que description de transformation de modèle, de génération de code ou d'exécuteur... Le type du comportement ne peut pas être déduit automatiquement d'où la nécessité d'avoir le modèle de l'activité.

Dans **PMS+** la mise en package d'un comportement peut être faite automatiquement à travers une fonctionnalité permettant de générer le plugin avec le modèle de description de l'activité correspondant. Cette fonctionnalité prend en paramètres le type de la description désirée ainsi que le fichier contenant le comportement. Elle explore le fichier afin de déterminer les métadonnées associées au comportement, par exemple le nom les paramètres. Actuellement, la génération du modèle de la description dépend de la technologie du comportement. En effet, certaines technologies, par exemple Java, sont difficiles à explorer automatiquement.

PMS+ propose, aussi, l'installation au niveau du runtime. Dans ce cas l'activité est prise en compte, automatiquement, à chaque démarrage de l'application. Ce type d'installation est fait en enregistrant la position du comportement dans le système de fichier. De ce fait, le changement de la position du comportement rend l'installation obsolète.

L'enregistrement, quant à lui, permet de déclarer les comportements dynamiquement au niveau du runtime dans le but de les tester et les valider. L'enregistrement est perdu lors du prochain démarrage de l'application. Dans ce cas, le comportement n'a pas besoin d'être packagé.

4.2 Comment ajouter une technologie ?

Afin d'ajouter une technologie dans **PMS+** nous nous basons sur la notion des points d'extension d'Eclipse. Dans **PMS+**, une technologie est une classe Java implémentant l'interface *IExecutor*. Celle-ci décrit l'ensemble des fonctionnalités qu'un exécuteur doit respecter, par exemple la fonction *execute* qui permet d'exécuter l'activité. Le point d'extension quant à lui permet de capter les métadonnées de cet exécuteur afin que **PMS+** le prenne en compte et le propose comme une technologie possible au niveau du modèle de la description de l'activité. Une fois l'exécuteur enregistré, il sera pris en compte par le moteur d'execution de **PMS+**.

4.3 Comment exécuter un processus de production ?

Une fois le processus de production modélisé, il faut pouvoir l'exécuter. Si le processus nécessite des paramètres en entrée, il faut un mécanisme permettant de saisir ou de spécifier la valeur de ces paramètres. **PMS+** propose deux approches pour exécuter un processus : une approche généraliste et une approche personnalisée.

Dans l'approche généraliste, l'utilisateur déclenche le processus en sélectionnant l'action *Execute Process* du menu de **PMS+**. **PMS+** propose alors une boîte de dialogue dans laquelle l'utilisateur renseigne le modèle du processus à exécuter, ainsi qu'un modèle contenant les valeurs des paramètres nécessaires à l'exécution.

Dans l'approche personnalisée, l'utilisateur peut ajouter des actions personnalisées dans la barre d'actions d'Eclipse. Chaque action permet alors de déclencher le processus de production

qui lui est associé. Les actions peuvent être installées dynamiquement, sans avoir besoin de relancer **PMS+** ou Eclipse. Pour installer une action personnalisée, il faut choisir la fonctionnalité *Create Process Action*. **PMS+** propose alors une boîte de dialogue dans laquelle l'utilisateur renseigne le modèle du processus à installer ainsi que le menu dans lequel ce processus doit apparaître. Le menu est décrit par une chaîne de caractères sous la forme *sousMenu1 : :... : :sousMenuN : :action*. L'utilisateur a ainsi un accès rapide personnalisé à ses processus. Cette seconde méthode, quoi que plus longue à mettre en œuvre, s'avère plus intéressante lorsque l'utilisateur est amené à exécuter souvent les mêmes processus de production.

Il est à noter que les actions *Execute Process*, *Create Process Action*, ainsi que la plupart des actions du menu de **PMS+**, sont elles-même décrites par des actions personnalisées associées à des processus de production. L'utilisateur peut ainsi très facilement étendre les actions de **PMS+** en ajoutant des actions associées à des processus de production.

5 Exemple

Nous utilisons PMS+ afin de décrire et d'exécuter notre processus de production de logiciels. Ce processus permet de générer des portails collaboratifs opérationnels à partir d'omegsi [4][12],qui est une modélisation centrée sur les processus métier.

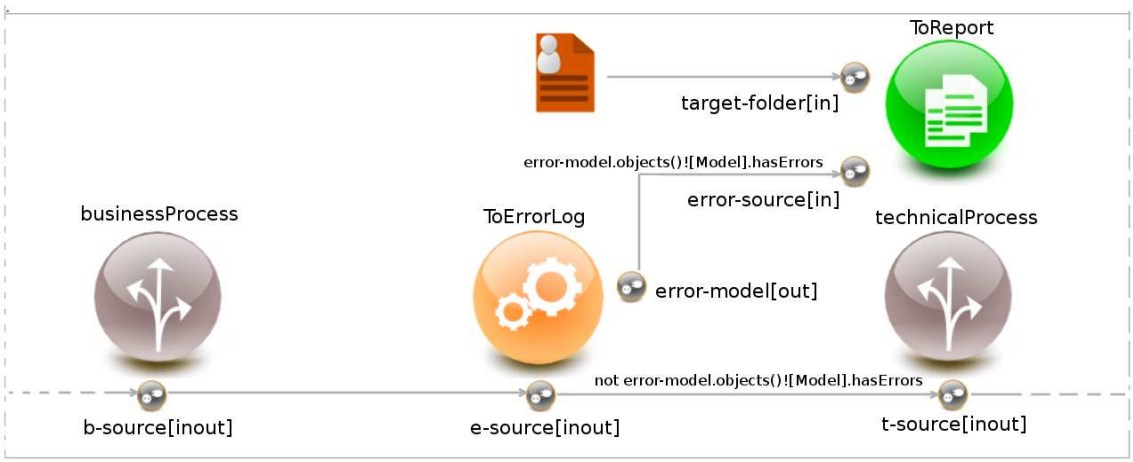

FIGURE 2 – Exemple d'un processus de production

Notre processus de production comprend près d'une vingtaine d'activités, essentiellement des transformations de modèles, et bien sûr de la génération de code. Nous avons découpé notre processus en trois sous-processus : le premier traite les modèles de haut niveau, le second les modèles intermédiaires, et le troisième les modèles techniques permettant de générer le code. Chaque processus comprend des *activités*, des *données*, et des *flux de données*. La figure 2 montre une partie de l'enchaînement des trois processus principaux. Chacun de ces processus est lui-même composé d'activités. Le processus *businessProcess* prend en entrée un modèle, le raffine (processus métier non détaillé ici) et fournit en sortie un modèle raffiné. Ce modèle est transféré à la transformation *ToErrorLog* chargée de détecter les erreurs métier de l'utilisateur, par exemple la détection des inter-blocages entre les processus métier. Cette transformation permet de produire un modèle décrivant les différentes erreurs ainsi qu'une proposition pour les résoudre. La transformations met à vrai un booléen (*hasError*) placé dans le modèle. La

transformation *ToErrorLog* comprend deux paramètres en sortie (*e-source* et *error-model*). De ces deux paramètres partent deux flots, chacun comportant une garde exprimée en OCL (`error-model.objects()![Model].hasErrors`) Ces gardes permettent d'interroger le booléen *hasError*. Si il est vrai, la donnée ira vers *ToReport*, qui pourra se déclencher et générer un log de rapport. Si il est faux, la donnée déclenchera l'activité *TechnicalProcess*.

6 Conclusion

Dans cet article, nous avons présenté notre outil **PMS+** de modélisation et d'exécution des processus de production de logiciels dans une démarche IDM. Cet outil permet de décrire des processus complexes centrés sur le modèle. Les processus de production permettent l'enchaînement des activités suivant un flux d'exécution orienté selon des contraintes relatives aux données manipulées par ces activités. Ces dernières peuvent être des transformations de modèle, des générations de code ou des boites noires... Dans PMS+ nous avons simplifié la notion d'interaction avec l'utilisateur et d'interaction avec le système de fichiers. Cela permet, par exemple, de charger ou d'enregistrer des données selon les choix de l'utilisateur. Pour cela notre outil génère automatiquement des interfaces utilisateurs basées sur les formulaires afin de saisir les données et les injecter dans le processus.

PMS+ est totalement indépendant des technologies utilisées par les activités. Le concepteur peut étendre **PMS+** en ajoutant ses technologies à l'aide d'un mécanisme basé sur les points d'extension d'Eclipse. Actuellement seul les technologies QVT, Acceleo, Java et le moteur de processus sont supportés par **PMS+**. Les fonctionnalités de **PMS+** sont elles-mêmes décrites sous forme de processus réutilisables. Cela permet de les maintenir facilement. **PMS+** est un logiciel open source disponible ici [13].

Références

[1] Kleppe Anneke. Mcc : A model transformation environment. In Warmer Jos Rensink Arend, editor, *Model Driven Architecture – Foundations and Applications*, number 4066 in Lecture Notes in Computer Science, pages 173–187. Springer Berlin Heidelberg, jan 2006.

[2] Eclipse. Acceleo. `http://wiki.eclipse.org/Acceleo`.

[3] Eclipse. Mwe. `http://wiki.eclipse.org/MWE/oaw4.3_doc`.

[4] Ecréall. Omegsi. `http://omegsi.ecreall.com/`.

[5] Layna Fischer. *2010 BPM and Workflow Handbook, Spotlight on Business Intelligence*. Future Strategies Inc, jun 2010.

[6] Kolovos D. S. Paige R. F. dos Santos O. M. Guerra E., de Lara J. Engineering model transformations with transml. *Software and Systems Modeling*, page 1–23, 2011.

[7] Nguyen Viet Hoa. Automatisation de l'enchaînement des tâches exécutées sur des modèles. Rapport de fin d'étude, 2009.

[8] Sonia JAMAL. Environnement de procédé extensible pour l'orchestration application aux services web, 2005.

[9] Jean-Marc Jézéquel, Benoît Combemale, and Didier Vojtisek. *Ingénierie Dirigée par les Modèles : des concepts a la pratique*. Ellipses Marketing, February 2012.

[10] Matthew Moodie. *Pro Apache Ant*. Apress, 1 edition, jan 2012.

[11] OMG. Spem2.0. `http://www.omg.org/spec/SPEM`.

[12] Amen Souissi Cedric Dumoulin Pierre Boulet, Michael Launay. Modèlisation centrèe sur les processus métier pour la génération compléte de portails collaboratifs. Rapport de recherche, 2012.

[13] PMS+. Pms+. https://gforge.inria.fr/frs/?group_id=3616.

[14] López-romero F. Bautist J. Vallecillo A. Rivera J. E., Ruiz-gonzález D. Orchestrating atl model transformations. In *In Jouault [7]*.

[15] Vinh Le Thai-Bernard Coulette Samba Diaw, Redouane Lbath. Spem4mde : a metamodel for mde software processes modeling and enactment. *3rd Workshop on Model-Driven Tool & Process Integration - Associated to EC-MFA*, 2010.

[16] Baelen Stefan Van-Joosen Wouter Berbers E. Vanhooff Bert, Ayed Dhouha. Uniti : A unified transformation infrastructure. In *In ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems (MoDELS 2007)*, 2007.