Lignes de produits logiciels et usines logicielles

Article *in* L Objet · September 2008

DOI: 10.3166/obj.14.3.15-31 · Source: DBLP

5 authors, including:

Nicolas Anquetil
University of Lille Nord de France
139 PUBLICATIONS **2,676** CITATIONS
SEE PROFILE

Hugo Arboleda
ICESI University
67 PUBLICATIONS **223** CITATIONS
SEE PROFILE

Angel Nunez
National Commission for Scientific and Technological Research
13 PUBLICATIONS **113** CITATIONS
SEE PROFILE

Lignes de produits logiciels et usines logicielles

Nicolas Anquetil, Hugo Arboleda, Fabricio de Alexandria Fernandes, Angel Nuñez, Jean-Claude Royer

ASCOLA, EMN-INRIA
École des Mines de Nantes,
4 Rue A Kaslter, BP 20722
F-44307 Nantes Cedex 3

(Nicolas.Anquetil,Hugo.Arboleda,Fabricio.Fernandes,Angel.Nunez,Jean-Claude.Royer)@emn.fr

RÉSUMÉ. *L'industrie du logiciel tente de produire des systèmes à un rythme toujours plus rapide tout en restant prévisible dans les temps de production et la qualité des systèmes produits. Pour ce faire, beaucoup de nouvelles approches ont été proposées, telles que les lignes de produits logiciels, le développement génératif ou les usines logicielles. Dans le but d'aider les néophytes à s'y retrouver, nous avons regroupé dans cet article quelques notions de bases. Nous présenterons les trois concepts avec leurs principales caractéristiques. Nous les comparerons aussi les un aux autres pour mieux comprendre leurs spécificités. Finalement, nous citerons les outils et nous proposerons quelques références actuelles pour aider les lecteurs intéressés à rechercher plus d'information sur le sujet.*

ABSTRACT. *The software industry is trieving to keep producing systems at an ever increasing pace while maintaining a high level of predictability and quality in these systems. In this struggle, many new approaches are proposed such as software factories, generative software development, and software product lines. To help new comers understand the basics of these concepts, we have collected some initial informations in this article. We will present the three concepts with their main characteristics. We compare them one to the other to hep understand each one specificities. Finally we propose some references and tools to help interested readers to get more information on the topic.*

MOTS-CLÉS : *Dévelopment génératif, ingéniérie des modèles, ligne de produits, usine logicielle*

KEYWORDS: *Generative Software Development, Model Driven Ingeneering, Software Product Line, Software Factory*

1. Introduction

Pour pouvoir produire toujours plus rapidement et avec plus de qualité les nouveaux systèmes logiciels que la société demande, de nouvelles approches doivent être inventées. Dans cet article d'introduction, nous présentons et définissons les concepts de développement génératif, d'usine logicielle et de ligne de produits logiciels. La section 2 donne une vue plus précise sur le processus de développement d'une ligne et la section 3 présente deux activités importantes et spécifiques. Une brève description de quelques outils est faite dans la section 4. Des références additionnelles sur les projets et ouvrages traitant du sujet sont proposées en fin d'article.

1.1. *Développement génératif de logiciel*

L'objectif du développement génératif est de libérer le programmeur humain du travail d'écriture de code. Une tache qui est typiquement répétitive, longue et sujette à erreurs. Ces techniques s'inscrivent donc dans la lignée des travaux qui ont conduit aux compilateurs et autres générateurs de code. Le principe est d'écrire les fonctionnalités requises dans un langage de haut niveau d'abstraction (modèles de type UML, langages dédiés : DSL[1]) et d'utiliser un programme pour transformer automatiquement cette description en code source. Des applications concrètes existent pour la construction d'interface graphique, les mécanismes de persistance de données, interface avec les banques de données, langages dédiés ou macros. Les générateurs de code permettent aux programmeurs et configurateurs de systèmes de se concentrer sur la description de concepts spécifiques au domaine abordé sans se préoccuper des détails de programmation et de la plate-forme technique cible.

La source de génération de code est souvent un modèle qui donne une description abstraite ou de haut niveau du système ou de la fonctionnalité à implémenter. Dans ce cas, l'approche est connue comme développement de logiciel basé sur les modèles : MDD ou génie logiciel basé sur les modèles : MDE. La technologie inclut différents outils pour répondre aux besoins : définition et implémentation de DSLs (langages dédiés), construction, édition et validation des modèles, définition de transformation d'un modèle à un autre, génération de code, etc.

Cette approche particulière est détaillée un peu plus dans la prochaine section. Puis nous nous intéressons à une autre approche voisine : les usines logicielles.

1.2. *Développement dirigé par les modèles*

Le développement de logiciel dirigé par les modèles (MDD) est une forme de développement génératif où la description initiale de la solution est faite avec des modèles. Bien que l'idée ne soit pas neuve, elle a reçu un intérêt croissant avec l'apparition

1. Les termes techniques anglais sont regroupés et traduit dans un lexique en section 5.3.

du langage unifié de modélisation, UML, qui a donné à la communauté de génie logiciel un langage de modélisation connu et accepté « universellement ». Dans cette approche, on peut générer une partie du code source à partir des modèles et compléter ce code à la main, ou l'on peut inclure un grand niveau de détail (semblable aux détails du code source) dans les modèles et ne pas avoir à modifier le code généré. L'approche de génie logiciel dirigé par les modèles (MDE) cible plus un niveau architectural et d'abstraction plus élevé que pour le MDD.

Finalement, l'approche architecture dirigée par les modèles (MDA) est un cadre de travail proposé par l'OMG. Son objectif est de permettre aux développeurs de spécifier les applications indépendamment d'une plate-forme technologique. Dans le MDA, une application est construite en créant d'abord un modèle indépendant de la plate-forme (PIM), celui-ci est transformé automatiquement en modèle spécifique à une plate-forme (PSM). Ce dernier est, finalement, automatiquement transformé en code exécutable pour la plate-forme technologique choisie. Un analyste ou programmeur humain peut modifier et raffiner les modèles aux trois niveaux : définition du PIM, raffinement du PSM, modification ou ajouts dans le code source.

1.3. *Usine logicielle*

Une usine logicielle est un processus de gestion qui permet de développer des produits par la combinaison d'un ensemble d'artefacts de base. Cette approche promet d'améliorer la qualité des produits et la productivité. Cependant pour fonctionner elle requiert une gestion des processus rigoureuse. Microsoft est souvent cité comme un pionnier des usines logicielles avec l'usine EFx `http://msdn.microsoft.com/en-us/library/bb977473.aspx`, une combinaison de développement dirigé par les modèles et d'environnement d'exécution intégré pour produire des applications orientées services. Les usines logicielles sont souvent associées à l'ingénierie des modèles, mais d'autres méthodes génératives peuvent être utilisées.

In fine, une usine logicielle est conçue pour produire une famille de systèmes. Les différences avec les lignes de produits logiciels (voir section suivante) sont que : les usines logicielles ne font pas explicitement une étude systématique du domaine d'application ni une analyse précise de la variabilité (voir section 3.2).

2. Lignes de produits logiciels

L'idée des lignes de produits logiciels vient de la perception que dans beaucoup de domaines, les applications ne sont pas des systèmes isolés, mais partagent entre elles des besoins, fonctionnalités et des propriétés. L'idée de base des lignes de produits logiciels est de profiter de ces points communs pour définir une architecture de base à partir de laquelle de nouvelles applications pourront être construites plus facilement, plus rapidement et avec une meilleure qualité.

Dans cette section, nous verrons un rapide historique des lignes de produits logiciels (section 2.1), puis une introduction plus formelle du processus (section 2.2).

2.1. *Historique*

Une courte introduction aux lignes de produits logiciels est proposée dans (Sugumaran *et al.,* 2006), incluant une partie de son histoire. La présente section est partiellement basée sur cet article.

L'idée des lignes de produits logiciels vient de deux domaines de recherche : réutilisation et analyse de domaine. L'article de Mc Ilroy (Ilroy, 1989) est habituellement considéré comme la source du mouvement de réutilisation, un mouvement qui visait à construire des logiciels comme sont construits les produits matériels, à partir d'un ensemble de composants interchangeables, au lieu de monter à la main chaque nouveau système en programmant chaque ligne de code.

Très tôt, l'idée fut amplifiée, quand on compris que l'on aurait plus de bénéfices à réutiliser des composants plus abstraits au lieu de combiner (en opposition à programmer) des composants logiciels (de code source). Ces composants plus abstraits sont créés plus tôt dans le processus de développement, tels que la spécification des besoins ou l'architecture. Les lignes de produits logiciels suivent cette idée en spécifiant les besoins dans le domaine d'application choisi et en définissant une architecture générique pour répondre à ces besoins.

Un autre prédécesseur des lignes de produits logiciels peut-être trouvé dans le travail de (Neighbors, 1989) sur l'analyse de domaine. L'analyse de domaine est le processus d'identification, collecte, organisation et représentation des informations pertinentes d'un domaine, en se basant sur l'analyse de systèmes existants, leur évolution, des connaissances extraites des experts du domaine, les théories sous-jacentes au domaine, et les technologies utilisées. Bien que la motivation initiale soit d'aider la rétro-ingénierie des systèmes d'un domaine, l'objectif était de recenser et représenter les informations pertinentes de ce domaine d'application. Un concept-clé des lignes de produits logiciels est la compréhension que les systèmes d'un domaine d'application donné doivent partager beaucoup de points communs imposés par ce domaine.

Le concept de lignes de produits logiciels a ses fondements dans les deux idées suivantes :

1) tout système dans un domaine d'application donné partage des points communs avec les autres systèmes de ce domaine comme, par exemple des besoins, des architectures logicielles ou des composants logiciels similaires ;

2) au travers d'une analyse de domaine, il est possible d'identifier ces points communs. Ceci fait, il est possible de définir une structure de base des systèmes qui permet de construire de nouveaux systèmes (dans le même domaine) plus rapidement et avec une meilleure qualité. Le processus de développement devient donc un processus de

dérivation d'une nouvelle application à partir de la structure de base et en ajoutant les fonctions spécifiques nécessaires à l'application.

2.2. *Définition*

Les lignes de produits logiciels se basent sur les idées de personnalisation de masse (*mass customization*) telles qu'appliquées à la production de biens matériels, c'est-à-dire la production à grande échelle de biens conçus pour des marchés spécifiques ou des individus. La personnalisation de masse est appliquée dans plusieurs marchés comme les voitures, les appareils photos, les téléphones portables, etc. Elle fonctionne à partir d'une base commune à tous les produits qui autorise l'addition ou la modification de petites variations permettant de différencier les produits. Par exemple, dans le cas de l'industrie automobile, la base peut-être le châssis, la suspension et la transmission. A partir de cette base, on peut construire des voitures très différentes selon la carrosserie et le moteur que l'on adapte. Normalement, cette plate-forme de base est la partie la plus chère de l'ensemble ce qui offre de meilleurs bénéfices quand on peut la réutiliser dans de nombreux produits.

Une bonne ligne de produits doit avoir une bonne plate-forme de base, adaptée à de nombreux produits à un moment donné, mais aussi dans le futur. Pour cela, on doit avoir une bonne vision des besoins actuels du marché et des possibles évolutions. Une activité, préliminaire au développement technique, et importante pour la pérennité de la ligne de produit est de faire une analyse précise du marché et de l'évolution des besoins dans le domaine concerné. Cet aspect n'est pas du ressort des développeurs ni des architectes mais doit être confié à des spécialistes connaissant bien le marché (consultants, commerciaux, etc.). Une ligne de produit raisonnable doit reposer sur une bonne étude commerciale et fonctionnelle. A partir de cela, les analyses doivent se concentrer sur les *points communs* entre tous les produits désirés et les *différences* prévues. Les points communs vont permettre de définir la plate-forme de base partagée par tous les produits ; les différences vont permettre de rendre cette plate-forme plus flexible.

En génie logiciel, les principes sont les mêmes. Au lieu de développer les applications individuellement, on définit une plate-forme commune – l'architecture du système – qui sera réutilisée dans toutes les applications du domaine. Plusieurs conditions sont nécessaires pour réussir dans la création d'une bonne ligne de produits :

– une profonde connaissance du domaine d'application est absolument nécessaire ;

– il faut aussi un processus de développement mature, d'abord parce que le développement de la plate-forme de base est un projet en soi, difficile et risqué ; ensuite pour pouvoir profiter de tous les avantages de meilleure prévisibilité que la ligne de produits propose ;

– la gestion de configuration est aussi un domaine qui gagne une importance renouvelée avec l'existence de plusieurs applications, dérivées d'une même plate-forme, à différents moments de son existence ;

– finalement, il faut disposer des technologies adéquates telles que la liaison tardive (*late binding*) qui permet de définir la plate-forme commune et de ne choisir et connecter certains des éléments nécessaires seulement à l'exécution d'une nouvelle application. Une autre technologie importante est celle de composants logiciels qui permet que différents composants soient développés de manière indépendante pour être ensuite composés en une application. Les techniques génératives vont évidemment s'imposer comme des outils essentiels pour produire rapidement des composants corrects à partir d'un modèle ou d'un moule qui sera configuré suivant les spécificités de la ligne de produits.

La gestion d'une ligne de produits logiciels est typiquement divisée en deux phases et deux perspectives (voir figure 1) :

– dans la phase de *génie du domaine* (*domain engineering*) la plate-forme de base de la ligne de produits est définie ;

– dans la phase de génie des applications (*application engineering*) sont développées (ou dérivées) les applications à partir de la plate-forme commune ;

– dans la *perspective du problème* (*problem space*) on se concentre sur la définition des besoins, de la plate-forme commune ou de chaque application ;

– dans la *perspective de la solution* (*solution space*) on se concentre sur l'implémentation des composants logiciels pour satisfaire ces besoins.

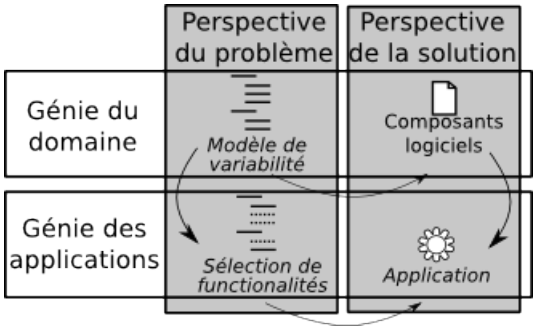

Figure 1. *Représentation traditionnelle de la gestion d'une ligne de produits logiciels*

Dans les diverses activités du développement les différents acteurs vont utiliser et créer des modèles, des besoins fonctionnels, des classes, des codes etc. Habituellement ces différents items ou objets sont banalisés sous le vocable d'*assets* ou d'*artefacts*. Une activité non négligeable est de gérer le catalogue de ces assets et de permettre leur réutilisation dans les différents produits ou applications issus de la ligne.

3. Fonctionnalités et variabilité

L'approche des lignes de produits introduit une plus grande complexité du processus de développement. Deux activités propres à la ligne de produit sont la description des fonctionnalités et la gestion de la variabilité.

3.1. *La description des fonctionnalités*

L'analyse du domaine est très souvent concrétisée par un document appelé *modèle des fonctionnalités* (*feature model*). Celui-ci décrit les fonctionnalités communes aux applications du domaine, les variations prévues et leurs dépendances. L'analyse de domaine orienté par les caractéristiques (*Feature-Oriented Domain Analysis – FODA*) a été introduite par (Kang *et al.,* 1990) comme une méthode d'analyse de domaine pour identifier des caractéristiques ou fonctionnalités d'un système. En FODA, le modèle des caractéristiques présente les concepts communs et variables et les propriétés des structures dans le domaine d'intérêt. La description des grands systèmes est normalement faite en utilisant un ensemble de diagrammes plutôt qu'un seul modelé, pour faire simple nous considérons un unique modèle ici. Une *configuration* est la description d'un produit individuel obtenu en sélectionnant un sous-ensemble des caractéristiques du modèle.

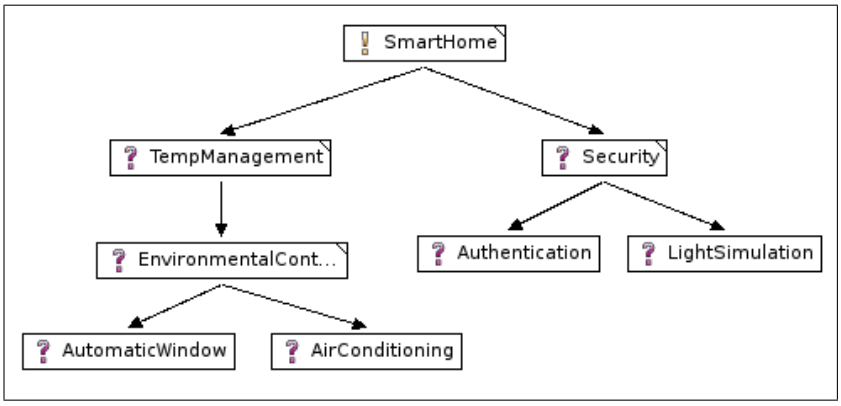

Figure 2. *Un exemple de modèle et de configuration avec pure::variants*

La figure 2 montre un modèle des caractéristiques pour une application de type « smart home ». Les notations sont celles de l'outil pure::variants, voir section 3.1. Le `!` correspond à une caractéristique de type obligatoire et le `?` représente une caractéristique optionnelle (de cardinalité $[0..n]$). Les caractéristiques `TempManagement` et `Security` sont optionnelles pour l'application `SmartHome`. Il est possible aussi de représenter des caractéristiques alternatives (cardinalité 1) et de type « ou » (cardinalité n).

Les autres approches de description des caractéristiques sont principalement basées sur FODA. (Trigaux *et al.,* 1990) ont identifié les formalismes suivants :

– la méthode de réutilisation orientée caractéristique (*Feature Oriented Reuse Method – FORM*) (Kang *et al.,* 1998) ;

– la notation de Jan Bosch (Bosch *et al.,* 2002) ;

– la notation de Matthias Riebisch (Riebisch *et al.,* 2002).

FORM ajoute au modèle originel FODA des couches de caractéristiques pour considérer les détails liés aux services, environnement, domaine et implémentation. En outre, il y a trois relations possibles entre les caractéristiques : généralisation/spécialisation, composé par et implémenté par Jan Bosch *et al.* ont ajouté une notation pour décrire les caractéristiques externes et Riebisch *et al.* ont ajouté au modèle la notion de cardinalité des caractéristiques.

(Czarnecki *et al.,* 2000) ont introduit les caractéristiques de type « ou » qui sont similaires aux caractéristiques alternatives définies en FODA, mais au lieu de sélectionner exactement une caractéristique, n'importe quel nombre de l'alternative, plus grand ou égal à un, peut être sélectionné. Ils ont fait une extension des cardinalités et introduit les groupes et cardinalité de groupe, les attributs et référence aux attributs (Czarnecki *et al.,* 2002). Les cardinalités peuvent être utilisées pour définir combien de sous-caractéristiques un type particulier doit avoir. Les caractéristiques utilisent les attributs comme paramètres. La référence aux attributs peut être utilisée pour référencer d'autres caractéristiques dans la hiérarchie. Groupes et cardinalité de groupe sont des mécanismes pour exprimer des restrictions sur le nombre de sous-caractéristiques groupées que l'utilisateur peut sélectionner. (Czarnecki *et al.,* 2004) définissent les cardinalités comme une séquence d'intervalles sous la forme $[n_1..n'_1]...[n_l..n'_l]$, le formalisme est très similaire à celui d'UML. Le dernier intervalle dans une cardinalité peut être noté par l'étoile de Kleene $(*)$ pour exprimer un intervalle non borné.

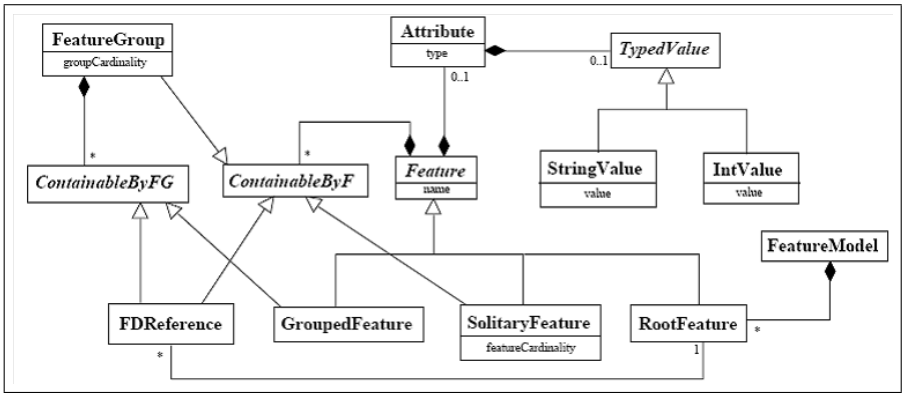

Figure 3. *Le métamodèle de caractéristique de Czarnecki*

La figure 3 présente une partie du métamodèle de caractéristique de (Czarnecki *et al.,* 2004). Un `GroupFeature` exprime le choix à partir d'un ensemble de `GroupedFeatures` dans le groupe et sa cardinalité définit la restriction sur le nombre de choix. Par exemple, la cardinalité $[1..2][4..5]$ pour un `GroupFeature` signifie que entre un et deux et entre quatre et cinq de ses `GroupFeatures` peuvent être choisis. Il faut remarquer que `GroupFeature` n'a pas de cardinalité. Un `SolitaryFeature` est une caractéristique qui n'est pas groupée dans un `GroupFeature`. La cardinalité de `SolitaryFeature` définit le nombre maximum de fois que cette fonctionnalité peut apparaître dans la configuration finale.

La figure 4 montre le même type d'application que précédemment mais en utilisant les notations plus avancées de Czarnecki Il faut remarquer que la cardinalité de `SolitaryFeatures` est $[0..1]$ ce qui veut dire que le nombre maximum de fois que cette caractéristique peut apparaître dans la configuration finale est un. Deux caractéristiques solitaires `tempManagement` et `security` sont définies au-dessous du niveau de base. La caractéristique `tempManagement` a comme caractéristique `environmentalControl` qui a les caractéristiques groupées `airConditioning` et `automaticWindow`. La caractéristique `security` a les caractéristiques unitaires `authentication` et `lightSimulation`.

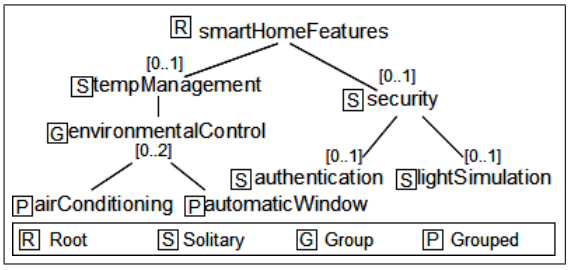

Figure 4. *Modèle de caractéristiques avec groupes et cardinalités*

3.2. *La gestion de la variabilité*

La variabilité est au cœur des lignes de produits logiciels. C'est elle qui permet de créer différentes applications qui répondent aux besoins spécifiques des différents utilisateurs. Une ligne de produits doit définir ce qui est commun à tous les produits de la même famille, mais aussi ce qui peut varier. Cela revêt une grande importance car, si il est (relativement) simple et sûr de créer une nouvelle application à l'intérieur des limites fixées par la variabilité de la ligne de produits, essayer d'introduire une fonctionnalité non prévue est aussi coûteux, risqué et sujet aux mêmes erreurs que la maintenance de systèmes traditionnels. Il est donc important d'identifier toutes les variations possibles et comment elles peuvent apparaître.

Les deux questions principales sont : *quoi* peut varier et *comment* ? De plus, il peut aussi être important de savoir le *pourquoi* de cette variation. Le quoi est appelé *point*

de variation et le comment sont les *variantes*. Un point de variation est un point dans l'architecture de la famille de produits où plusieurs options sont disponibles, incluant l'option de ne rien avoir quand il s'agit d'un point de variation optionnel. Construire une application impose de prendre une décision pour chaque point de variation. Les options disponibles pour un point de variation sont les variantes. Tout point de variation doit avoir au moins une variante.

On distingue deux types d'expression de la variabilité : positive ou négative. Dans la variabilité positive, on commence une application avec un système minimal auquel sont rajoutées les parties correspondant aux fonctionnalités nécessaires dans le futur produit. A l'inverse, la variabilité négative part d'un système complet maximal et lui soustrait les parties qui correspondent aux fonctionnalités non requises. Cette dernière est plus couteuse en taille et en maintenance, mais cette approche est utilisée en pratique quand on manque de bons moyens pour gérer la variabilité positivement. Par la suite nous considérons uniquement la variabilité positive qui propose une meilleure maitrise de la configuration des produits.

A part le quoi et le comment, on peut aussi s'intéresser au pourquoi. La variabilité peut avoir deux sources : interne ou externe. La variabilité externe dépend de décision des intéressés (clients, utilisateurs). Typiquement, c'est une option offerte au client comme, par exemple, avoir ou non un registre de toutes les transactions effectuées dans l'application (*logging*). La variabilité interne est normalement liée à des décisions plus techniques en conséquence d'autres décisions de variabilité. Par exemple un point de décision interne pourrait être de décider comment le registre des transactions sera effectué (simple fichier texte, banque de données, etc.). Cette décision dépend du fait que le client ait choisi d'avoir ce registre de transactions (variabilité externe) et peut-être résolue directement par les analystes en fonction de critères trop techniques pour le client.

La variabilité existe à tous les niveaux d'abstraction du processus de développement : analyse des besoins, définition de l'architecture, conception du projet, programmation du code source. Pour une famille de produits donnée il peut y avoir des centaines ou des milliers de points de variation. Le choix des variantes pour une application peut être facilité par la définition de règles et contraintes sur les points de variation et les variantes. Par exemple, deux variantes ou points de variations peuvent être mutuellement exclusifs ou la décision pour l'un peut imposer ou restreindre la décision pour l'autre.

Pour toutes ces raisons, le modèle de variabilité est une pièce fondamentale de la définition d'une ligne de produits logiciels. Il documente tous les points de variation et leurs variantes ainsi que leurs relations (règles et contraintes). Une avancée importante dans ce domaine est l'introduction du modèle de variabilité par (Pohl *et al.,* 2005). Dans la même ligne le langage VML (Loughran *et al.,* 2008), issu du projet AMPLE, a pour objectif d'exprimer la variabilité dans les architectures.

4. Les outils supports

Nous proposons ici quelques exemples d'outils commerciaux pour les lignes de produits logiciels.

4.1. *Pure : :variants*

Pure : :variants est un produit commercial offert par Pure Systems `http://www.pure-systems.com/`. Il gère toutes les phases du développement d'application des besoins jusqu'à l'implémentation, les tests et la maintenance. Pure : :variants est basé sur la notion de fonctionnalité (*feature*) qu'une application aura ou non. Ces fonctionnalités permettent de définir la plate-forme de base (fonctionnalités obligatoires) comme la variabilité (fonctionnalités optionnelles). Pure : :variants offre une vue « modèle de fonctionnalités » (*feature model view*) pour définir et gérer celles-ci. Des composants sont associés aux fonctionnalités, ils peuvent prendre diverses formes comme : un fichier, une classe, etc. Ces composants sont gérés au travers d'une vue « modèle de famille de composants » (*component family model*). Finalement une vue « modèle de description de variants » (*variant description model*) permet de construire une application en choisissant ses fonctionnalités et composants. Cet outil est distribué sous la forme d'un plugin d'Eclipse. Il offre une perspective d'Eclipse, appelée « perspective de gestion des variants » (*variant management perspective*), avec plusieurs vues et des outils. Il peut exporter les modèles en différents formats : HTML, XML, CSV et Directed Graph (DOT) pour GraphViz. Il peut aussi importer du code source Java ou C++, d'autres formats peuvent être ajoutés par plugin. On peut définir de nouveaux modules pour répondre à des besoins spécifiques comme, par exemple, l'interopérabilité avec d'autres outils. Des modules optionnels existent déjà pour faire l'interface avec des systèmes de gestions de configuration (CVS, IBM ClearCase, Subversion), de gestion des besoins (Doors, Borland CaliberRM), d'interface avec les banques de données.

4.2. *Gears*

Gears est un produit commercial offert par BigLever (`www.biglevel.com`). Il offre un environnement de développement de lignes de produits. Les étapes pour créer une ligne de produits et de nouvelles applications sont les suivantes :

1) créer un modèle de fonctionnalités qui déclare plusieurs fonctionnalités. Ces fonctionnalités représentent les parties optionnelles ou qui varient dans la famille de produits. Ce sont en fait des variables qui représentent des concepts du domaine. Les valeurs que peut prendre une variable peuvent être restreintes à partir d'un langage simple de contraintes. Par exemple, on peut dire que si la valeur d'une fonctionnalité est dans un intervalle donné, alors la valeur d'une autre fonctionnalité devra être xyz. Ces contraintes expriment les relations entre les fonctionnalités ;

2) définir les artefacts de la famille : code source, cas de test, documents, besoins, cas d'utilisation, architecture, description de projet, etc. Ces items sont générés, copiés ou transformés dans l'application finale en accord avec les spécifications de petits programmes (dans un langage spécifique) qui leur sont associés. Les programmes décrivent comment traiter un artefact selon les valeurs des fonctionnalités. Un point de variation est composé d'artefacts et des programmes décrivant comment les instancier dans une application ;

3) créer un profil d'application en donnant une valeur aux différentes fonctionnalités. Les contraintes sur les valeurs des fonctionnalités servent comme aide à la définition de ce profil ;

4) exécuter les activateurs qui vont automatiquement générer l'application à partir de l'application des programmes des artefacts (étape 2) et des valeurs des fonctionnalités (étape 3).

4.3. *OpenArchitectureWare*

OpenArchitectureWare (oAW, `http://www.openarchitectureware.org/`) est une plate-forme de MDD (développement dirigé par les modèles) et de développement par aspects intégrée dans Eclipse et permettant la création de lignes de produits. Le cœur de oAW est un moteur de script de tâches qui permet de définir et transformer des tâches en organisant entre eux des séquences de composants de tâche. Il propose des tâches préconstruites pour faciliter la lecture et l'instanciation de modèles, vérifier qu'ils ne violent pas des contraintes prédéfinies ou les transformer en d'autres modèles ou en code source. La génération et transformation de script des tâches se fait par des fichiers XML qui décrivent les étapes qui doivent être exécutées. Dans oAW les transformations entre modèles sont définies en utilisant le langage `Xtend`. C'est un langage textuel et fonctionnel pour interroger et naviguer dans les modèles et pour construire de nouveaux modèles. oAW offre un support pour la programmation par aspect. Un composant processus (`workflow`) facilite la définition des intercepteurs pour activer les aspects réalisant des modifications des règles de transformations. Un exemple d'application à la ligne de produits « smart home » est décrit dans (Völter *et al.,* 2007). Il dispose également d'une communauté de développeurs très active.

5. Informations additionnelles

5.1. *Quelques projets*

5.1.1. *ITEA CAFÉ (2001-2003)*

L'objet du projet ITEA CAFÉ (« *from Concept to Application in system-Family Engineering* », `http://www.esi.es/Cafe/`) est de développer une infrastructure permettant de définir et de gérer des familles d'architectures de logiciels correspondant à des lignes de produits. Ces familles d'architectures doivent exprimer les carac-

téristiques communes à un ensemble de logiciels et les moyens de les spécialiser pour des plates-formes ou des applications particulières. Le projet repose sur l'idée que les quatre pôles : activités commerciales, processus de développement logiciel, technologies cibles et organisation de déploiement sont intimement liés. Le projet CAFÉ a regroupé un nombre important d'industriels européens (notamment Philips, Siemens, Thales, Alcatel, esi, Bosch Gmbh, etc.). Les domaines d'applications privilégiées par le projet sont : les systèmes communicants, les systèmes d'assistance médicale, le contrôle du trafic aérien, les systèmes de contrôle et de commande.

5.1.2. *Feasiple (2006-2008)*

Feasible (`http://www.feasiple.de/`) est un projet allemand regroupant des universités et des éditeurs logiciels comme pure::variants ou SAP. Le consortium part du constat que les lignes de produits semblent une approche permettant une meilleure gestion de l'économie du projet : réduction des coûts et des temps de développement. Toutefois il y a actuellement peu de vrai projet industriel qui se font avec ce genre d'approche. Les raisons essentielles semblent le manque d'outils supports notamment pour la maîtrise de la chaîne complète de développement, la gestion de la variabilité et la dispersion des fonctionnalités dans le code ou les spécifications. La méthodologie mais également la qualité sont des besoins très forts dans l'industrie. Le projet a pour but d'évaluer les approches existantes dans le domaine des lignes de produits et d'explorer la combinaison de nouvelles techniques comme les aspects et l'ingénierie des modèles. Le projet propose une extension AO-MDSD permettant la modélisation des variantes et un processus orienté par le modèle des fonctionnalités et reposant sur l'utilisation d'outils comme openArchitectureWare.

5.1.3. *Le projet AMPLE (2006-2009)*

Le projet européen AMPLE (*Aspect-oriented Model-driven Product Line Engineering*, `http://www.ample-project.net/`) se propose de combiner les techniques de pointes en développement par aspects (AOSD) et en développement (de logiciel) dirigé par les modèles (MDD) pour avancer l'état de l'art en lignes de produits logiciels (SPL). Le thème général de ce projet est l'étude des lignes de produits logiciels, c'est-à-dire une approche du développement où l'accent est mis sur le développement d'une famille de systèmes logiciels plutôt que d'un seul logiciel comme dans les approches traditionnelles. Le fait de travailler sur une famille permet de mieux capitaliser et de réutiliser les éléments qui sont produits pendant le développement. Cela nécessite tout de même une approche plus complexe des projets qui sont découpés en une partie concernant l'ingénierie du domaine (ou l'étude de la famille en question) et un autre niveau qui est celui du développement individuel d'un produit. La production d'un produit logiciel final se fait par dérivation à partir de la famille et assemblage ou configuration de composants déjà existants. Dans ce contexte le projet cherche à améliorer la méthodologie en introduisant des techniques récentes comme la programmation par aspects ou l'ingénierie des modèles. Ces techniques doivent permettre une meilleure modularisation des logiciels mais également améliorer l'automatisation des tâches. Un autre enjeu du projet est de fournir un support au développement des lignes de pro-

duits en intégrant la traçabilité depuis l'ingénierie des besoins jusqu'à la conception des produits. Parmi les résultats notables obtenus jusqu'à présent :

– un atelier d'aide à l'identification de la variabilité : dans une ligne de produits les artefacts sont nombreux et variés et la documentation du domaine est généralement importante en taille. L'atelier permet d'identifier les variations à partir d'une analyse des besoins fonctionnels écrits en langage naturel. L'approche utilise les techniques d'analyse du langage naturel et d'extraction d'information ainsi que de leur organisation en hiérarchies ;

– un langage d'expression de la variabilité dans les architectures : les langages usuels (UML ou autres ADL) ne prennent pas en compte les aspects de variabilité. Le langage VML propose de décrire la variabilité et des moyens de composition à travers un petit nombre de primitives pour connecter, déployer, ajouter ou enlever des éléments pour aider à la construction des architectures ;

– l'atelier de traçabilité : cet atelier offre une architecture ouverte pour intégrer des fonctions relatives à la traçabilité en générale. Il repose sur une base de données avec des interfaces permettant de gérer les artefacts et les liens de traces. Le système est couplé avec Subversion et permet de gérer les informations de traces et les configurations des artefacts d'une façon cohérente. Une interface utilisateur générale a été définie ainsi que des outils pour la visualisation graphique des liens, des mesures du graphe des traces, des conversions vers des formats comme DOT, XML ou Excel. D'autres extensions proposent des extracteurs pour analyser différents dépôts d'informations et nourrir la base de données avec les artefacts et traces extraits ;

– une extension de Java permettant divers modèles de programmation (classes, mixins, classes virtuelles, aspects, types dépendants) dans un même contexte. L'objectif est de fournir un langage de haut niveau intégrant la plupart des techniques d'implémentation de la variabilité et pouvant servir de langage cible dans un développement de ligne de produits.

5.1.4. *ITEA MoSiS (2007-2010)*

Le projet MoSiS (`http://itea-mosis.org`) s'intéresse à la production du logiciel embarqué en adoptant une approche ligne de produit. Le projet vient de démarrer il y a un an et les informations disponibles sont actuellement assez sommaires. Le projet devrait aboutir à : i) la standardisation d'un langage MoSis pour la modélisation et la gestion de la variabilité, ii) la définition de bonnes pratiques d'ingénierie des modèles pour les systèmes MoSiS, et iii) un prototype ouvert et libre supportant le langage et le processus MoSiS. Le projet espère des avancées dans le domaine de la variabilité, de l'expression des propriétés non fonctionnelles et de l'adaptation dynamique des systèmes.

5.2. *Lectures Complémentaires*

– (Czarnecki *et al.,* 2000) est une référence très connue sur le sujet du développement génératif de logiciels ;

– (Pohl *et al.,* 2005) ont un bon livre d'introduction sur les lignes de produits logiciels. Il décrit les idées de base et toutes les étapes du processus de développement d'une ligne de produits ;

– (Stahl *et al.,* 2006) ont un livre qui intègre les lignes de produits et le développement dirigé par les modèles d'un point de vue pratique ;

– (Greenfield *et al.,* 2004) intègrent dans leur livre différentes approches d'ingénierie des lignes de produits telles que usines logicielles ou développement par patrons (*patterns*). Le livre donne une bonne introduction aux lignes de produits ;

– la revue « Communication of the ACM » a publié un numéro spécial (Volume 49, numéro 12) sur les lignes de produits en décembre 2006 ;

– l'institut de génie logiciel (SEI, *Software Engineering Institute*) de l'université Carnegie Melon a un site internet (`http://www.sei.cmu.edu/productlines`) sur les lignes de produits, avec des liens sur des conférences, des définitions, et d'autres ressources ;

– Wikipedia `http://fr.wikipedia.org/` a plusieurs articles (principalement en anglais) sur :

- la programmation générative : `http://en.wikipedia.org/wiki/Generative_programming` ;

- le développement dirigé par les modèles : `http://fr.wikipedia.org/wiki/Ingenierie_dirigee_par_les_modeles` (en français) ; `http://en.wikipedia.org/wiki/Model-driven_development` (en anglais) ;

- les lignes de produits logiciels : `http://en.wikipedia.org/wiki/Software_product_line`.

5.3. *Lexique*

Un petit lexique de traduction des expressions anglaises est proposé ici. Il ne s'agit en aucun cas d'une traduction officielle, mais seulement de nos meilleurs efforts pour recenser les expressions qui paraissent les plus courantes.

– Asset, Artefact : désigne les éléments de base du catalogue d'une ligne de produit. Cela peut être tout élément intervenant à un niveau quelconque de la construction d'une application : un modèle UML, une spécification formelle, une architecture, un besoin en langage naturelle, une classe Java, un composant binaire exécutable, etc.

– DSL, Domain Specific Language : langage dédié à un domaine d'application précis.

– Feature Model : modèle de fonctionnalités.

– Generative software development : développement génératif de logiciel.

– MDA, Model Driven Architecture : architecture dirigée par les modèles.

– MDD, Model Driven Developement : développement (de logiciel) dirigé par les modèles.

– MDE, Model Driven Engineering : ingénierie dirigée par les modèles (aussi : ingénierie des modèles)

– PIM, Platform Independent Model : modèle indépendant de la plate-forme.

– PSM, Platform Specific Model : modèle spécifique à une plate-forme.

– Software Factory : usine logicielle.

– SPL, Software Product Line : ligne de produits logiciels (aussi : ligne de produits logiciels).

6. Conclusion

Dans cet article nous avons introduit les concepts de base ainsi que les principales différences entre la notion d'usine logicielle et celle de ligne de produits logiciels. Si toutes les deux utilisent des composants et des principes de la programmation générative, les lignes de produits promettent plus de bénéfices, mais ont aussi une gestion plus complexe que celle des usines logicielles. En effet une ligne de produit suppose une analyse précise du domaine d'application et l'expression de la variabilité des produits. Elle a pour but de gérer de façon rationnelle différents produits partageant un ensemble de fonctionnalités. Elle utilise des notations adaptées comme les modèles de fonctionnalités. L'ensemble de la chaîne est loin d'être complètement automatique et automatisable. Toutefois les approches de types dirigées par les modèles y trouvent une place importante par l'aide à la construction des artefacts ou à la configuration des produits qu'elles proposent. Plusieurs projets passés ou récents s'intéressent à ce type de développement mais il n'a pas encore trouvé beaucoup d'écho dans la communauté de la recherche et de l'industrie en France tout au moins. Notre conviction est que les lignes de produits vont apparaître dans bon nombre de secteurs applicatifs et prendre le pas sur les approches plus traditionnelles de développement.

7. Bibliographie

Bosch J., Florijn G., Greefhorst D., Kuusela J., Obbink J. H., Pohl K., « Variability Issues in Software Product Lines », *PFE '01 : Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, vol. 2290 of *LNCS*, p. 13–21, 2002.

Czarnecki K., Bednasch T., Unger P., Eisenecker U. W., « Generative Programming for Embedded Software : An Industrial Experience Report », *Proceedings of the 1st Conference on Generative Programming and Component Engineering*, Springer-Verlag, p. 156–172, 2002.

Czarnecki K., Eisenecker U., *Generative Programming : Methods, Tools and Applications*, Addison-Wesley, 2000.

Czarnecki K., Helsen S., Eisenecker U., « Staged Configuration Using Feature Models », *Proceedings of the Third Software Product Line Conference 2004*, Springer, LNCS 3154, p. 266–282, 2004.

Greenfield J., Short K., CookS., Kent S., *Software Factories : Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley, 2004.

Ilroy M. M., « Mass Produced Software Components : Software Engineering Concepts and Techniques », *Proceedings of NATO Conference on Software Engineering*, p. 88–98, 1989.

Kang K. C., Kim S., Lee J., Kim K., Shin E., Huh M., « FORM : A feature-oriented reuse method with domain-specific reference architectures », *Annals of Software Engineering*, vol. 5, p. 143–168, 1998.

Kang K., Cohen S., Hess J., Nowak W., Peterson S., Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21, 1990.

Loughran N., Snchez P., Garcia A., Fuentes L., « Language Support for Managing Variability in Architectural Models », *7th Intl. Symposium on Software Composition*, number 4954 in *LNCS*, p. 36–51, 2008.

Neighbors J. M., *Software Reusability, Concepts and Models*, vol. I of *ACM Frontier*, Addison-Wesley, chapter Draco : A method for reingineering reusable software systems, p. 295–319, 1989.

Pohl K., Bckle G., van der Linden F., *Software Product Line Engineering - Foundations, Principles, and Techniques*, Springer, Heidelberg, 2005.

Riebisch M. K., Böllert D. S., Philippow. I., « Extending Feature Diagrams with UML Multiplicities », *Proc. of the 6th Conf. on Integrated Design and Process Technology (IDPT)*, June, 2002.

Stahl T., Czarnecki M. V. K., *Model-Driven Software Development : Technology, Engineering, Management*, John Wiley & Sons, 2006.

Sugumaran V., Park S., Kang K., « Introduction », *Communication of the ACM*, vol. 49, n^o 12, p. 28–32, 2006.

Trigaux J.-C., Heymans P., « Software Product Lines : State of the art », 2003, 1990.

Völter M., Groher I., « Product Line Implementation using Aspect-Oriented and Model-Driven Software Development », *Software Product Lines Conference*, IEEE Computer Society, p. 233–242, 2007.

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   RSTI - L'objet – 14/2008. Lignes de produits logiciels

2. AUTEURS :
   Nicolas Anquetil, Hugo Arboleda, Fabricio de Alexandria Fernandes, Angel Nuñez, Jean-Claude Royer

3. TITRE DE L'ARTICLE :
   Lignes de produits logiciels et usines logicielles

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Lignes de produits logiciels

5. DATE DE CETTE VERSION :
   20 octobre 2008

6. COORDONNÉES DES AUTEURS :

   – adresse postale :

   ASCOLA, EMN-INRIA
   École des Mines de Nantes,
   4 Rue A Kaslter, BP 20722
   F-44307 Nantes Cedex 3

   (Nicolas.Anquetil,Hugo.Arboleda,Fabricio.Fernandes,Angel.Nunez,Jean-Claude.Royer)@emn.fr

   – téléphone : 00 00 00 00 00

   – télécopie : 00 00 00 00 00

   – e-mail : Roger.Rousseau@unice.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.2 du 03/03/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com