HAL
open science

Vers des Lignes de Produits Flexibles

Jean-Marc Jézéquel, Gilles Perrouin

To cite this version:

Jean-Marc Jézéquel, Gilles Perrouin. Vers des Lignes de Produits Flexibles. Revue des Sciences et Technologies de l'Information - Série L'Objet : logiciel, bases de données, réseaux, 2008, 14 (3), pp.33–45. inria-00456480

HAL Id: inria-00456480

https://inria.hal.science/inria-00456480v1

Submitted on 15 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Vers des Lignes de Produits Flexibles

Apports de l'Ingénierie Dirigée par les Modèles à la Dérivation de Produits

Jean-Marc Jézéquel — Gilles Perrouin

Équipe Projet Triskell
IRISA/INRIA Rennes
Campus de Beaulieu
35042 Rennes, France
{gilles.perrouin,jezequel}@irisa.fr

RÉSUMÉ. *Afin de faire face à la complexité du logiciel due à la variabilité de ses environnements et de ses utilisations, l'ingénierie des lignes de produits permet d'important gains en termes de coûts et de qualité de développement en systématisant la réutilisation d'éléments communs. Néanmoins, les approches actuelles manquent de flexibilité dans la prise en compte des exigences particulières à un utilisateur. Nous illustrons ici comment, en utilisant des techniques d'ingénierie des modèles telles que la composition et la transformation et en les outillant dans l'environnement de métamodelisation Kermeta, il est possible de concilier flexibilité et efficacité lors de la dérivation de produits.*

ABSTRACT. *To address current software engineering challenges resulting from the diversity of software environnements and usages, Software Product Lines (SPL) promise greater productivity by proposing to develop applications based on a set of common assets thus fostering reuse. However, software product line derivation fails to meet efficiently unforeseen, customer-specific, requirements. In this paper, we show how model driven engineering contributes to flexibility and support during product derivation. Our approach combines model composition and model transformation techniques in order to assist the developer along the whole derivation process. This process is implemented in Kermeta, a meta-modelling environment.*

MOTS-CLÉS : *Ingénierie Dirigée par les Modèles, Dérivation de Lignes de Produits, Kermeta*

KEYWORDS: *Model Driven Engineering, Software Product line Derivation, Kermeta*

1. Introduction

Les applications logicielles sont devenues indispensables dans un grand nombre d'activités humaines et se sont répandues grâce à l'avènement des réseaux (ADSL, WIFI, GSM...) ou de l'informatique nomade (ordinateurs portables, assistants digitaux personnels, téléphones mobiles etc.) qui ont rendu l'interaction avec des systèmes informatiques possible en presque tout lieu. La distribution et la mobilité sont des facteurs de *variabilité* qui sont à prendre en compte tout au long du processus de développement logiciel. Prenons le cas d'une entreprise comme Nokia. Elle doit gérer la variabilité de ses téléphones selon plusieurs dimensions. La dimension matérielle qui inclut le type de processeur utilisé, la quantité de mémoire, la taille et le type d'écran (LCD, OLED etc.), le support d'un certain nombre de protocoles réseaux (GSM, WIFI, Bluetooth etc.) La dimension logicielle qui inclut les fonctionalités offertes à l'utilisateur telles que la présence d'un agenda, d'un navigateur internet, de fonds d'écrans, de sonneries personnalisables, de jeux ainsi que d'un support d'un nombre important de langue afin de toucher un large public. Enfin l'analyse du marché va définir un certain nombre de segments à occuper afin d'aligner au mieux les qualités techniques et le coût d'un téléphone en fonction de l'audience visée. Ces dimensions sont de plus rarement orthogonales. En effet, un complexe réseau de dépendances les relie entre elles ; par exemple, certains jeux nécessitent une grande quantité de mémoire vidéo pour tourner de manière fluide. Dans ce cas, même si cela est possible de les afficher sur un tout petit écran, cela n'a pas de sens du point de vue de l'expérience d'un utilisateur toujours plus exigeant. Il en découle que Nokia doit spécifier, concevoir et maintenir des centaines de composants qu'ils soient logiciels ou matériels ainsi que leurs dépendances pouvant potentiellement générer des milliards de variantes de téléphones. Afin de faire face à une telle complexité et de répondre aux besoins dans des délais et des coûts raisonnables, il est nécessaire d'améliorer nos méthodes de développement.

Une possibilité est de procéder par analogie avec l'approche mise en place dans des domaines comme l'industrie automobile où la production de véhicules s'organise en gammes partageant des pièces similaires et offrant un certain nombre d'options. Dans le monde du logiciel, nous nommons *ligne de produits* (LdP) (Clements *et al.*, 2001) un ensemble d'applications, partageant des points communs mais exhibant aussi des différences, qui sont développées à partir de composants communs dans un domaine déterminé. Ainsi en maximisant la réutilisation de composants conçus avec soin, il est possible de satisfaire simultanément des critères de qualité et de temps de développement. A partir de ces concepts, une ingénierie des lignes de produits est née et un certain nombre d'approches outillées ont été introduites afin de générer une application en fonction d'un ensemble de choix proposés au développeur. Cette ingénierie a été couronnée de succès dans des domaines divers comme l'automobile, la téléphonie mobile, ou l'imagerie médicale[1].

1. Voir `http://www.sei.cmu.edu/productlines/plp_hof.html` pour un aperçu de ces réussites.

Néanmoins, un logiciel doit très souvent répondre à un besoin qui émane d'un utilisateur particulier. Il est donc nécessaire de prendre en compte des attentes qui sont parfois spécifiques à cet utilisateur et il n'est ni possible ni souhaitable de définir et de supporter celles-ci dans l'ensemble de composants réutilisables à partir desquels les applications sont dérivées.

Certaines méthodes orientées lignes de produits proposent des approches pour la dérivation de produits par trop restrictives qui excluent de manière indue des produits qui, bien que pouvant être développés à partir des composants de la LdP, n'ont pas été envisagés lors de sa définition. Cela nous prive donc de la possibilité d'adresser facilement les exigences spécifiques des utilisateurs. Si quelques méthodes reconnaissent la nécessité de prendre en compte ces exigences particulières, elles ne fournissent aucune solution systématique pour intégrer ces exigences dans le cycle de développement d'un produit.

Dans cet article, nous nous attachons à définir une approche d'ingénierie de ligne de produits permettant la dérivation semi-automatique et flexible de produits. Nous utilisons pour ce faire l'*ingénierie dirigée par les modèles* (IDM), qui permet la description des applications logicielles et de leur variabilité tout au long du cycle de développement ainsi que leur génération par transformations successives. Notre processus est organisé en deux phases ; la première consiste à générer un squelette du produit à développer à partir de choix réalisés par le développeur dans un diagramme de variabilité. La génération du squelette procède par composition de modèles représentant chacun un choix effectué par le développeur. La deuxième phase consiste à compléter ce squelette en utilisant des techniques de transformation de modèles afin de répondre aux exigences spécifiques du produit. Ainsi, ce processus conjugue l'efficacité du développement orienté ligne de produits avec la flexibilité requise pour répondre aux attentes des utilisateurs.

2. Etat des lieux de l'ingénierie des lignes de produits

L'ingénierie des *lignes de produits* de logiciels appelée aussi *famille de produits*[2] est une transposition des chaînes de production au monde du logiciel. Le principe est de minimiser les coûts de construction de logiciels dans un domaine d'application particulier en ne développant plus chaque logiciel séparément, mais plutôt en le concevant à partir d'éléments réutilisables (Clements *et al.*, 2001). L'ingénierie des lignes de produits consiste à mettre en œuvre deux activités interdépendantes. La première, nommée *ingénierie du domaine* consiste à concevoir les éléments réutilisables ainsi qu'une architecture de base pour construire les produits lors de *l'ingénierie d'applications*.

2. van der Linden (2002) justifie l'origine de deux appellations par l'emplacement géographique : l'appellation « *ligne de produits* » est utilisée aux États-Unis d'Amérique et « *famille de produits* » est utilisée en Europe. Depuis 2003, les deux appellations ont convergé sous le terme de « *ligne de produits* » comme en témoigne l'historique des principales conférences sur le sujet : `http://www.splc.net/`

2.1. *Ingénierie du domaine*

Les membres d'une LdP sont caractérisés par leurs points communs, mais aussi par leurs différences (aussi appelées *variabilités* (Coplien *et al.*, 1998; van Gurp *et al.*, 2001)). La gestion de cette variabilité est l'une des activités-clés (Bosch *et al.*, 2001) des lignes de produits. Par exemple, dans une chaîne de production de véhicules, des voitures sont fabriquées à partir d'un ensemble d'éléments communs (roues,volant, vitres, etc.) mais peuvent comporter certaines caractéristiques qui les différencient (nombre de chevaux du moteur, présence ou non de la climatisation, etc.). Dans le monde logiciel, les différences peuvent apparaître de manière analogue, en fonction de choix techniques (utilisation d'un type particulier de matériel), commerciaux (création d'une version limitée), régionaux (produits destinés à plusieurs pays).

L'ingénierie du domaine est supportée par de nombreuses techniques pour la définition de la variabilité, avec l'approche *feature diagram* (Kang *et al.*, 1990; Schobbens *et al.*, 2007; Griss *et al.*, 1998; Czarnecki *et al.*, 2005a) ou par des extensions (Ziadi *et al.*, 2003; Gomaa, 2004) à des langages de description de logiciel existants. Concernant la conception de l'architecture en elle-même, des canevas à composants (Atkinson *et al.*, 2002; Batory *et al.*, 2000; Perrouin, 2007; Gomaa, 2004) sont généralement employés.

2.2. *Ingénierie d'applications*

La deuxième activité concerne la construction du produit logiciel (on parle aussi de *dérivation de produit* (Ziadi, 2004)) qui consiste en partie à figer certains choix vis-à-vis de la variabilité définie dans la LdP. De toute évidence, certains choix sont incompatibles entre eux. Si l'on reprend l'analogie ci-avant, une voiture comporte généralement un seul moteur, et il faut alors choisir entre une motorisation essence ou diesel. De la même manière, un choix particulier lors de la dérivation d'un logiciel peut exclure certaines variantes. Par exemple le choix d'un cabriolet à toit amovible exclura la possibilité de choisir un toit ouvrant. Une LdP doit donc aussi intégrer des contraintes de cohérence permettant de faciliter les choix lors de la dérivation. Bien que cette activité soit fondamentale à la réussite de l'ingénierie des lignes de produits dans le monde du logiciel, elle n'a reçu une attention approfondie que relativement récemment (Ziadi *et al.*, 2006). Depuis plusieurs années, nous nous intéressons à supporter l'ingénierie des applications (Ziadi, 2004; Ziadi *et al.*, 2006; Guelfi *et al.*, 2007; Perrouin, 2007; Perrouin *et al.*, 2008). Dans la suite, nous dressons un état des lieux des techniques de dérivation de produits qui peuvent être classées en deux sous-catégories, *configuration* et *transformation*.

2.2.1. *Dérivation par configuration*

Les approches par configuration ou *personnalisation de masse* (Krueger, 2002; Krueger, 2006) se fondent sur le principe que les produits peuvent être obtenus directement par configuration (*via* un certain nombre de choix prédéterminés) de l'ar-

chitecture de la LdP. La plupart de ces approches se basent sur des « *feature diagrams* » (Kang *et al.*, 1990) pour décrire les choix proposés à l'utilisateur. Sur base des choix réalisés d'une part, et grâce à des liens de traçabilité définis ente les « features » (Kang *et al.*, 1990; Czarnecki *et al.*, 2005a) (représentant des caractéristiques fonctionnelles ou non fonctionnelles permettant de distinguer un produit d'un autre (Bosch, 2000; Czarnecki *et al.*, 2000)) et les composants de la LdP d'autre part, il est possible d'automatiser la dérivation du produit grâce à un *configurateur* (Hollmann *et al.*, 2000). Ce configurateur peut soit générer des scripts de configuration ou de compilation (Hotz *et al.*, 2006), soit assembler les composants individuels pour former le produit (Voelter *et al.*, 2007; Perrouin *et al.*, 2008). Cette approche a donné lieu à des outils commerciaux (PureSystems, 2006; BigLever, 2006) et a été appliquée pour la confection de systèmes électroniques pour l'automobile (Hotz *et al.*, 2006).

2.2.2. *Dérivation par transformation*

Les approches par transformation, plus récentes et dans lesquelles s'inscrivent nos travaux (Guelfi *et al.*, 2007; Perrouin, 2007; Ziadi *et al.*, 2006; Ziadi, 2004) s'appuient tout particulièrement sur l'IDM (*model-driven engineering* (MDE) en anglais (Kent, 2002)) pour dériver des produits à partir d'une modélisation de la LdP.

Nous définissons un *modèle* comme une abstraction soit d'un système à l'étude soit du problème (cahier des charges) auquel ce système doit répondre (Perrouin, 2007). La construction d'un modèle est motivée par un but précis (par exemple l'analyse de son architecture ou la vérification de certaines propriétés de fiabilité) et est encadrée par un *metamodèle*. Un métamodèle définit les éléments utilisables dans un modèle ainsi que des règles regissant l'assemblage de ces éléments. Un métamodèle est à un modèle ce que la liste des mots et un ouvrage de grammaire est à une langue. Mais le principal intérêt d'un modèle par rapport à d'autres formes de documentation est sa capacité à intervenir comme brique logicielle à partir de laquelle on peut obtenir n'importe quel constituant d'un logiciel que ce soit du code source, des fichiers de configuration, ou des schémas de base de données. Cette conversion, au cœur de l'IDM (Sendall *et al.*, 2003), est appelée *transformation de modèles* et suscite un intérêt grandissant, tant dans la recherche que dans l'industrie. Il existe de nombreuses approches d'IDM (Sztipanovits *et al.*, 1997; Greenfield *et al.*, 2004; Soley *et al.*, 2000). Une des plus célèbres est certainement l'initiative « *model-driven architecture* (MDA) » de l'OMG qui a servi de catalyseur à la recherche ainsi qu'aux vendeurs d'outils afin de supporter l'IDM. Elle se base sur des métamodèles standardisés comme UML (OMG, 2007), une hiérarchisation de niveaux de modélisation couvrant le cycle de développement, une infrastructure permettant de de créer des métamodèles spécifiques comme le MOF (OMG, 2006) et un langage de transformation comme QVT (OMG, 2005).

C'est sur cette dernière technologie que sont basés certains travaux de dérivation (Haugen *et al.*, 2004). A partir de la modélisation d'une architecture de LdP hiérarchisée suivant les différents niveaux MDA (CIM,PIM,PSM) et dont les éléments variables sont décrits à l'aide de stéréotypes UML, une transformation QVT assure la dérivation. Une approche similaire est proposée par Kim *et al.* (2005).

Nous utilisons aussi UML pour la modélisation de l'architecture de la LdP (Perrouin, 2007; Ziadi, 2004). Ziadi *et al.* (2006) définissent la dérivation comme l'application d'un *patron de conception* (Gamma *et al.*, 1995) sur l'architecture de la LdP qui entérine les choix par rapport à un produit et génère les éléments correspondants et supprime les éléments inutiles pour le produit désiré. De plus, l'aspect comportemental, très peu abordé par ailleurs est pris en compte. Ce comportement est modélisé sous forme de scénarios UML 2.0 (permettant la définition de séquences communes et de séquences variables) qui sont ensuite synthétisés en machines à état qui peuvent ainsi être associées aux composants du produit.

3. Un processus de dérivation flexible

3.1. *Allier automatisation et flexibilité*

Les approches que nous venons de présenter sont en général outillées et permettent donc d'obtenir un bon niveau d'efficacité lors de la dérivation. Néanmoins nous estimons qu'elles souffrent d'un certain manque de flexibilité. Cela se traduit à la fois pendant l'ingénierie du domaine et celle d'application. Au niveau de l'ingénierie du domaine nous avons relevé les difficultés suivantes :

– *identification de la variabilité* : la plupart des approches existantes suggèrent que nous devons identifier de manière exhaustive tous les points de variations (*loci* de variabilité où les différents choix sont offerts) pour la LdP. Comme il n'est pas possible d'anticiper totalement les attentes spécifiques des utilisateurs, une pratique courante consiste à offrir un nombre élevé de ces points de variation en espérant couvrir les besoins spécifiques. Cependant, il est impossible de fournir des garanties sur ce point. De plus, un grand nombre de variants implique aussi une complexité accrue au niveau de leur conception au sein de la LdP ainsi que pour leur maintenance,

– *modélisation de la variabilité* : la modélisation de la variabilité se fait le plus souvent dans les modèles des éléments réutilisables de la LdP. Cette pratique est à la source de deux problèmes. Premièrement, ces modèles deviennent vite complexes du fait de la documentation exhaustive de ces points de variation. Ensuite, cela « fige » un certain nombre de choix spécifiques à une LdP pour un élément. De fait, la réutilisation des modèles entre différentes LdPs s'en trouve limitée.

Du fait des rapports étroits existant entre l'ingénierie du domaine et l'ingénierie d'application, les problèmes mentionnés ci-avant impactent directement l'ingénierie d'application. Dans les approches mentionnées, La dérivation semi-automatique se base exclusivement sur les points de variation définis, il n'est donc pas possible de prendre en compte des exigences spécifiques pour un produit en utilisant le processus proposé. Deux cas se présentent. On peut intégrer ces exigences spécifiques dans la LdP afin de pouvoir les prendre en compte (Hotz *et al.*, 2006). Cependant il n'est pas optimal de faire évoluer ainsi la LdP si les exigences ainsi prises en compte ne sont dérivées que pour un seul développement, ce qui peut arriver dans le cas d'exigences métiers particulièrement spécifiques. L'autre possibilité est de considérer un

développement supplémentaire pour prendre en compte la partie spécifique des exigences après avoir utilisé un processus de dérivation pour les exigences supportées par la LdP. Certaines approches, comme KoBra (Atkinson *et al.*, 2002), ont explicitement inclus dans leur processus de développement ce type d'activité. Néanmoins, elles ne proposent aucune forme de support, que ce soit pour assister le développement ou pour vérifier la cohérence de ce développement par rapport à la LdP.

Nous pensons qu'il est possible de trouver un compromis entre le degré d'automatisation que l'on souhaite fournir et la flexibilité requise pour intégrer des exigences utilisateurs spécifiques lors de la dérivation. Considérons la figure 1. Les produits marqués « V » sont directement atteignables à partir des modèles de la LdP. Les produits marqués « O » comportent certaines exigences qui n'ont pas été prises en compte dans la LdP initiale. Néanmoins, la majorité des fonctionalités de ces produits sont supportées par la LdP, de sorte qu'utiliser les éléments existants procure un gain en temps et en coûts significatif. Enfin, les produits marqués « X » sont considérés comme « en dehors » de la LdP ; c'est-à-dire que pour des raisons techniques ou liées au positionnement de la LdP par rapport à un marché, on ne souhaite pas réaliser de tels produits.

Notre objectif est de fournir un processus de dérivation de produits qui permet le développement des produits notés « O » (avec une efficacité proche des approches existantes pour les produits notés « V ») tout en excluant les produits notés « X ». Nous allons voir dans les sections suivantes comment une telle approche peut être formulée en combinant plusieurs techniques de dérivation dans un processus à base d'IDM.

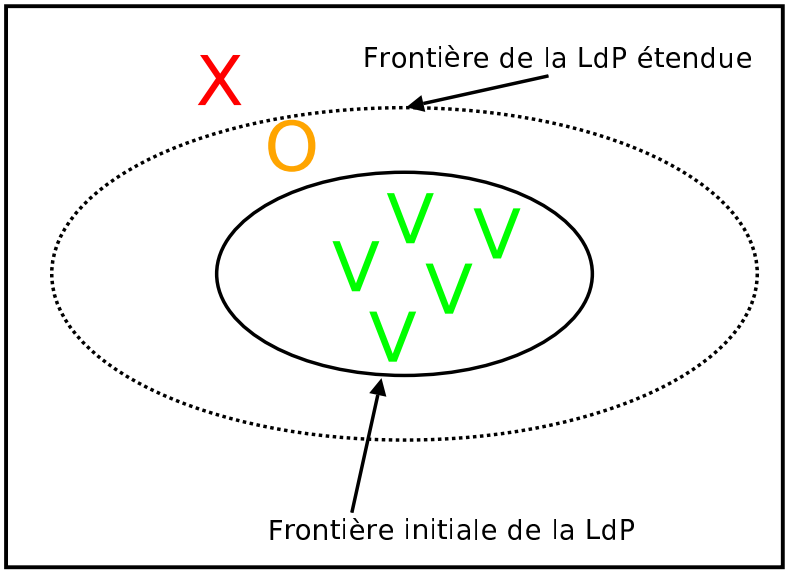

Figure 1. *Frontières d'une LdP*

3.2. *Préconfiguration*

Notre processus de dérivation débute par une phase dite de *préconfiguration*. L'objectif de cette phase est d'identifier les fonctionalités du produit qui sont directement supportées par la LdP et d'en dériver un squelette. En d'autres termes, cela consiste à obtenir la partie « V » d'un produit. Le choix se fait à l'aide d'un « feature diagram » qui définit les points de variations supportés par la LdP. Chaque point de variation offre un certain nombre de variants qui sont eux-mêmes reliés à des modèles constituant la LdP et produits lors de l'ingénierie du domaine. Une fois les choix effectués, nous appliquons une technique de configuration à base de composition de modèles (Fleurey *et al.*, 2007; France *et al.*, 2007) afin d'obtenir un modèle unique du produit intégrant les divers aspects sélectionnés.

Cette dérivation est encadrée par un certain nombre de contraintes, qui permettent à la fois de garantir une sélection appropriée des variants dans le « feature diagram » et une composition consistante des modèles y afférents.

3.3. *Personnalisation*

La phase dite de *personnalisation* consiste à adapter le squelette ainsi généré aux exigences spécifiques du produit qui ne pouvaient pas être prises en compte directement dans l'infrastructure de la LdP. Cela correspond donc à effectuer la transition d'un produit noté « V » à un produit noté « O ». Par définition, cette phase ne peut être automatisée et incombe aux ingénieurs responsables de l'implantation du produit. La personnalisation s'effectue par l'intermédiaire de l'écriture d'une transformation de modèles opérant sur le squelette de l'application. Cette transformation peut soit ajouter des éléments nouveaux dans le squelette existant soit adapter des éléments de la LdP afin qu'ils répondent aux mieux aux exigences du produit. Bien entendu, il s'agit de ne pas laisser l'ingénieur « se débrouiller » et ainsi courir le risque d'obtenir un produit non conforme à la LdP (marqué « X ») à l'issue d'une phase peu efficace, comme cela est le cas dans les approches flexibles existantes (Atkinson *et al.*, 2002). Pour ce faire, nous avons mis en place les mécanismes suivants :

– *librairie de transformation de modèles* : une librairie de transformations de haut niveau et adaptée aux types de modèles employés permet à l'ingénieur de s'abstraire de la gestion de base des éléments de modèle et simplifie l'écriture de la transformation. Nous avons proposé une telle librairie dans le cadre de diagramme de classes UML (Perrouin, 2007),

– *contraintes* : les contraintes définies sur les éléments de la LdP permettent d'exclure toute adaptation incorrecte de ces éléments lors de la transformation et peuvent servir de guides à l'ingénieur lors de la conception de la transformation.

Ces mécanismes permettent donc d'assister l'ingénieur dans la phase de personnalisation. Dans la prochaine section nous allons illustrer les modèles utilisés lors de ces deux phases et comment nous avons implanté ces phases en Kermeta, un environnement de metamodélisation.

4. Modèles et support

4.1. *Modèles*

Dans nos travaux, nous avons principalement considéré deux types de modèles. Le premier permet de modéliser les choix offerts par LdP qui seront effectués lors de la phase de personnalisation. Nous avons choisi de nous baser sur des « feature diagrams » pour des raisons de concision et de simplicité. En effet, malgré les nombreuses notations qui ont émergé ces dernières années (Czarnecki *et al.*, 2005a; Griss *et al.*, 1998), il a été possible de formaliser ce type de modèles (Czarnecki *et al.*, 2005b; Schobbens *et al.*, 2007). Cette formalisation nous a permis d'en dériver un métamodèle générique supportant diverses notations et de concevoir des règles de bonne formation pour ces modèles (Perrouin *et al.*, 2008). Un exemple d'un tel modèle est illustré par la figure 2.

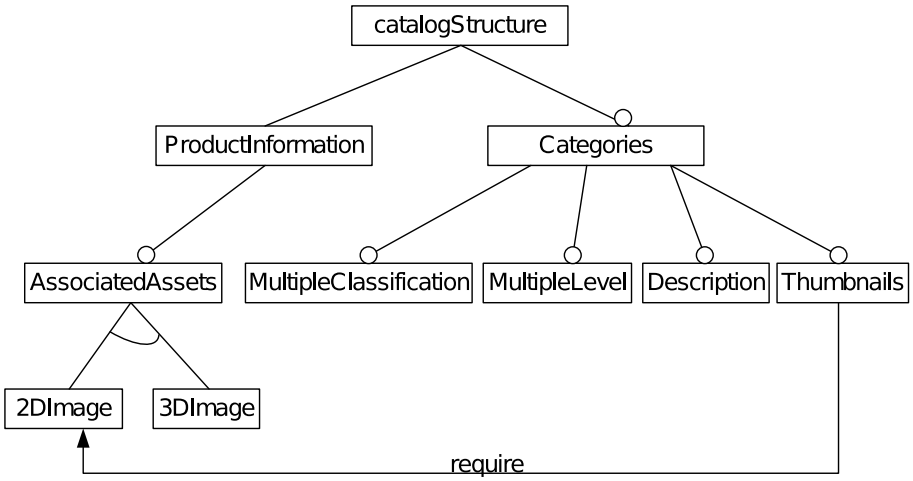

Figure 2. *Exemple de « feature diagram »*

Ce « feature diagram » présente une partie des constituants d'un catalogue de vente de produits en ligne. Pour chaque produit du catalogue il est nécessaire d'en fournir une description (`ProductInformation`). Cette description peut optionellement être complétée par des images en deux ou trois dimensions. Les produits du catalogue peuvent faire l'objet d'une catégorisation (`Categories`) : `MultipleClassification` représente la possibilité pour un produit du catalogue à appartenir à plusieurs catégories, `MultipleLevel` supporte les catégories imbriquées, et `Description` permet d'ajouter une description textuelle à une catégorie. Enfin `Thumbnails` indique que le produit peut être représenté sous forme de vignettes et implique dans ce cas que le support pour les images en deux dimensions ait été choisi.

Le second type de modèles concerne la description des éléments réutilisables de la LdP. Ces modèles peuvent concerner les aspects statiques ou dynamiques (Ziadi *et al.*, 2006), nous nous sommes intéressés au premier dans notre phase de préconfiguration

(Perrouin *et al.*, 2008). Chaque « feature » du « feature diagram » est associé à un ou plusieurs modèles décrivant l'élément associé dans la LdP.

4.2. *Support en Kermeta*

Nous avons utilisé l'environnement Kermeta (Muller *et al.*, 2005) afin de fournir un prototype de support outil à notre processus de dérivation (Perrouin *et al.*, 2008). Kermeta est un langage de metamodélisation open-source qui permet de définir une sémantique opérationelle à un métamodèle. Ce langage possède les fonctionalités d'un langage de programmation impératif ainsi que des fonctions spécifiques dédiées à la manipulation de modèles.

En particulier notre prototype se base sur Kompose (Fleurey *et al.*, 2007; France *et al.*, 2007) pour effectuer la préconfiguration et la personnalisation. Kompose fusionne des modèles en se basant sur leur *signature*. Une signature comprend exclusivement des propriétés syntaxiques comme le nom des classes et des operations et leurs attributs. Par exemple, pour obtenir des catégories ayant à la fois la possibilité d'avoir des sous-catégories et de posséder une description, notre outil va fusionner un modèle comprenant « Category » ayant un lien de composition vers elle-même (correspondant au premier variant) avec un modèle comprenant une classe « Category » ayant l'attribut `description`.

La phase de personnalisation s'appuie sur le mécanisme de *post-directives* de Kompose. Il s'agit d'un langage de transformation qui fournit une syntaxe intuitive et un certain nombre de primitives simplifiant la manipulation de modèles. Bien entendu, il est aussi possible d'utiliser Kermeta directement pour implanter cette transformation. Nous utilisons Kermeta aussi afin de nous assurer que les contraintes définies sur les éléments de la LdP sont respectées à l'issue de la personnalisation.

5. Conclusion

Dans cet article, nous avons fait un panorama des différentes approches concernant la dérivation dans le cadre de l'ingénierie des lignes de produits. Nous avons aussi montré que les propriétés de flexibilité et d'automatisation, bien que disjointes aujourd'hui, n'étaient pas incompatibles. En particulier en se servant de l'ingénierie des modèles comme base unificatrice de ces techniques et en les combinant ensemble, il est possible de concevoir un processus de dérivation de produits offrant à la fois la flexibilité nécessaire pour prendre en compte les exigences particulières des clients de ces produits et le support semi-automatique nécessaire à une dérivation efficace.

Nous pensons raffiner et étendre cette approche flexible de dérivation de lignes de produits à d'autres types de modèles (nous comptons notamment intégrer les aspects dynamiques à notre approche) ainsi qu'à d'autres domaines d'application. Par exemple, nous nous intéressons en ce moment à appliquer ces idées dans le cadre de systèmes reconfigurables tels que des services web ou des applications mobiles temps-réel.

Remerciements

Nous tenons à remercier nos collègues qui ont contribué aux travaux relatés ici, notamment Nicolas Guelfi et Jacques Klein de l'université du Luxembourg.

6. Bibliographie

Atkinson C., Bayer J., Bunse C., Kamsties E., Laitenberger O., Laqua R., Muthig D., Paech B., Wüst J., Zettel J., *Component-based Product Line Engineering with UML*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Batory D., Cardone R., Smaragdakis Y., « Object-Oriented Frameworks and Product Lines », *in* P. Donohoe (ed.), *Proceedings of the First Software Product Line Conference*, p. 227-247, 2000.

BigLever, GEARS Website `http://www.biglever.com/index.html`, 2006.

Bosch J., *Design and use of software architectures : adopting and evolving a product-line approach*, ACM Press/Addison-Wesley Publishing Co., 2000.

Bosch J., Florijn G., Greefhorst D., Kuusela J., Obbink H., Pohl K., « Variability Issues in Software Product Lines », *PFE4*, p. 11-19, 2001.

Clements P., Northrop L., *Software Product Lines : Practices and Patterns*, Addison Wesley, Reading, MA, USA,, 2001.

Coplien J., Hoffman D., Weiss D., « Commonality and Variability in Software Engineering », *IEEE Software*, vol. 15, n° 6, p. 37-45, 1998.

Czarnecki K., Antkiewicz M., « Mapping Features to Models : A Template Approach based on Superimposed Variants », *4th international conference Generative programming and component engineering*, vol. 3676 of *LNCS*, Springer-Verlag, p. 422-437, 2005a.

Czarnecki K., Eisenecker U., *Generative programming : Methods, Tools, and Applications*, ACM Press/Addison-Wesley Publishing, 2000.

Czarnecki K., Helsen S., Eisenecker U., « Staged Configuration through Specialization and Multilevel Configuration of Feature Models », *Software Process : Improvement and Practice*, vol. 10, n° 2, p. 143-169, 2005b.

Fleurey F., Baudry B., France R., Ghosh S., « A Generic Approach For Automatic Model Composition », *Workshop on Aspect-Oriented Modeling, AOM at Models'07*, 2007.

France R., Fleurey F., Reddy R., Baudry B., Ghosh S., « Providing Support for Model Composition in Metamodels », *EDOC*, Annapolis, MD, USA, 2007.

Gamma E., Helm R., Johnson R., Vlissides J., *Design patterns : elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

Gomaa H., *Designing Software Product Lines with UML : From Use Cases to Pattern-Based Software Architectures*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

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

Griss M. L., Favaro J., d' Alessandro M., « Integrating Feature Modeling with the RSEB », *ICSR*, Washington, DC, USA, 1998.

Guelfi N., Perrouin G., « A flexible Requirements Analysis Approach for Software Product Lines », *REFSQ*, LNCS-4542, Springer-Verlag, Norway, p. 78-92, 2007.

Haugen Ø., Møller-Pedersen B., Oldevik J., Solberg A., « An MDA-based Framework for Model-Driven Product Derivation », *SEA*, ACTA Press, p. 709-714, 2004.

Hollmann O., Wagner T., Günter A., « EngCon : A Flexible Domain-Independent Configuration Engine », *Proc. ECAI-Workshop Configuration*, p. 94, 2000.

Hotz L., Wolter K., Krebs T., Deelstra S., Sinnema M., Nijhuis J., MacGregor J., *Configuration in Industrial Product Families, The ConIPF Methodology*, IOS Press, 2006.

Kang K., Cohen S., Hess J., Novak W., Peterson S., Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report n° CMU/SEI-90-TR-21, Software Engineering Institute, November, 1990.

Kent S., « Model Driven Engineering », *IFM '02 : Proceedings of the Third International Conference on Integrated Formal Methods*, Springer-Verlag, London, UK, p. 286-298, 2002.

Kim S. D., Min H. G., Her J. S., Chang S. H., « DREAM : A Practical Product Line Engineering Using Model Driven Architecture », *Information Technology and Applications (ICITA)*, Washington, DC, USA, p. 70-75, 2005.

Krueger C. W., « Easing the Transition to Software Mass Customization », *Workshop on Software Product-Family Engineering (PFE)*, Springer-Verlag, London, UK, p. 282-293, 2002.

Krueger C. W., « New Methods in Software Product Line Development », *SPLC*, IEEE, p. 95-102, 2006.

Muller P.-A., Fleurey F., Jézéquel J.-M., « Weaving Executability into Object-Oriented Meta-Languages », *Proc. of MODELS/UML'2005*, LNCS, Springer, Jamaica, 2005.

OMG, MOF QVT Final Adopted Specification, Technical Report n° ptc/05-11-01, OMG, 2005.

OMG, Meta Object Facility (MOF) Core Specification, Technical Report n° 06-01-01, OMG, January, 2006.

OMG, Unified Modeling Language Superstructure (version 2.1.1), Technical Report n° formal/2007-02-03, Object Management Group, February, 2007.

Perrouin G., Architecting Software Systems using Model Transformation and Architectural Frameworks, PhD thesis, FSTC, Université du Luxembourg, and Institut d'Informatique, Université de Namur, Sept, 2007.

Perrouin G., Klein J., Guelfi N., Jézéquel J.-M., « Reconciling Automation and Flexibility in Product Derivation », *12th Software Product Line Conference*, Limerick, Ireland, 2008.

PureSystems, Pure : :Variants Website `http://www.pure-systems.com/`, 2006.

Schobbens P.-Y., Heymans P., Trigaux J.-C., Bontemps Y., « Generic semantics of feature diagrams », *Computer Networks*, vol. 51, n° 2, p. 456-479, 2007.

Sendall S., Kozaczynski W., « Model Transformation : The Heart and Soul of Model-Driven Software Development », *IEEE Softw.*, vol. 20, n° 5, p. 42-45, 2003.

Soley R., OMG, Model Driven Architecture, Technical Report n° omg/00-11-05, OMG, November, 2000.

Sztipanovits J., Karsai G., « Model-Integrated Computing », *Computer*, vol. 30, n° 4, p. 110-111, 1997.

van der Linden F., « Software Product Families in Europe : The Esaps & Café Projects », *IEEE Software*, vol. 19, n° 4, p. 41-49, 2002.

van Gurp J., Bosch J., Svahnberg M., « On the Notion of Variability in Software Product Lines », *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, 2001.

Voelter M., Groher I., « Product Line Implementation using Aspect-Oriented and Model-Driven Software Development », *SPLC*, Washington, DC, USA, p. 233-242, 2007.

Ziadi T., Manipulation de Lignes de Produits en UML, PhD thesis, IFSIC, Université de Rennes 1/IRISA, 2004.

Ziadi T., Hélouët L., Jézéquel J.-M., « Towards a UML Profile for Software Product Lines », *PFE2003 : 5th International Workshop on Software Product-Family Engineering*, vol. 3014 of *Lecture Notes in Computer Science*, Springer, Siena, Italy, p. 129-139, November, 2003.

Ziadi T., Jézéquel J.-M., « Product Line Engineering with the UML : Deriving Products », *Families Research Book*, Springer, 2006.