Modélisation de Lignes de Produits en UML*

Tewfik ZIADI[1], Loïc HELOUET[1], Jean-Marc JEZEQUEL[2]

IRISA, Campus de Beaulieu
35042 RennesCedex, France
Tewfik.Ziadi@irisa.fr Loic.Helouet@irisa.fr, Jezequel@irisa.fr

[1] INRIA
[2] Université de Rennes1

RÉSUMÉ. *L'ingénierie des lignes de produits est une approche récente du génie logiciel. Elle regroupe les activités de développement d'un ensemble de logiciels appartenant à un domaine particulier. L'objectif principal de cette approche est d'augmenter la productivité et diminuer le temps de réalisation. L'une des difficultés est de représenter dans un modèle la variabilité, c'est à dire les parties communes ou distinctes dans différents produits. Cet article propose une approche pour modéliser les lignes de produits en UML, en intégrant à la fois les aspects fonctionnels, statiques et dynamiques. Dans un premier temps, nous proposons une extension des cas d'utilisations, des diagrammes de classes, et des diagrammes de séquences pour tenir compte de la notion de variabilité. La cohérence des modèles est ensuite assurée par des contraintes écrites en OCL (Object Constraint Language).*

ABSTRACT. *Software Product Line Engineering is a recent approach of software engineering. It gathers activities of development of a set of software belonging to a particular domain. The main goal of this approach is to increase the productivity and to decrease the time-to-market. One of the main difficulties for representing software product lines is variability management, ie taking into account common and variable parts of products. This paper proposes a new approach for modeling product lines with UML integrating functional, static and dynamic aspects. First, an extension to use cases, class diagrams and sequence diagrams is proposed. The coherence between parts of the product line models is then ensured by OCL (Object Constraint Language) constraints.*

MOTS-CLÉS: *Ligne de Produits, Variabilité, UML, Contrainte, OCL, diagrammes de séquences*

KEYWORDS: *Product Line, Variabillity, UML, Constraint, OCL, sequence diagrams.*

This work has been partially supported by the ITEA project ip00004, CAFE in the Eureka Σ! 2023Programme

1 Introduction

L'approche « Lignes de produits logiciels » 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, appelés « assets ». Un **asset** peut ainsi être une exigence, un modèle, un composant, un plan de test ou tout simplement un document.

La première difficulté liée à cette approche réside dans la conception d'une architecture permettant de définir plusieurs produits. Les membres d'une ligne de produits sont caractérisés par leurs points communs, mais aussi par leurs différences (aussi appelées « points de variation »). La gestion de cette variabilité est un des concepts clé des lignes de produits. 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, …). 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), …

Une autre difficulté réside dans l'utilisation d'une ligne de produits. La construction d'un produit logiciel (on parle aussi de «dérivation de produit ») consiste en partie à figer certains choix (cristallisation) vis-à-vis des points de variation définis dans la ligne de produits. De toute évidence, certains choix sont incompatibles entre eux. Si l'on reprend l'analogie ci-dessus, 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 ligne de produits doit donc aussi intégrer des contraintes de cohérence permettant de faciliter les choix lors de la dérivation.

Cet article propose une technique de modélisation de lignes de produits fondée sur UML, intégrant à la fois des aspects fonctionnels, statiques, dynamiques, et des contraintes de cohérence. La section 2 fait un rapide état de l'art sur l'ingénierie des lignes de produits, la section 3 décrit l'approche proposée pour la modélisation de lignes de produits en UML. La section 4 propose un ensemble de contraintes assurant la cohérence du modèle proposé. La section 5 conclut ce travail et donne quelques perspectives.

2 Etat de l'art

L'ingénierie des lignes de produits logiciels, telle qu'elle est définie dans la littérature (ESAPS 2001-a), (Northrop, 2002), distingue deux niveaux, illustrés par la Figure 1: l'« Ingénierie du domaine » et « Ingénierie de l'application ». L'ingénierie du domaine consiste à développer et construire les assets qui seront réutilisés pour

la construction de produits (développer *pour* la réutilisation). L'ingénierie de l'application consiste à utiliser les assets pour la construction d'un produit ou d'une application particulière après cristallisation des points de variation, il s'agit d'un développement *par* la réutilisation.

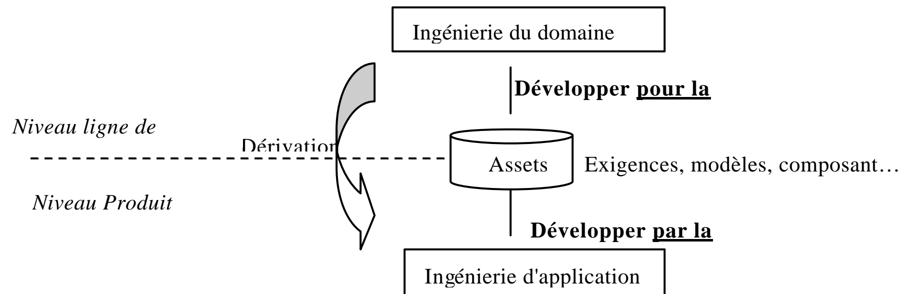

Figure.1. Ingénierie des lignes de produits

Bien que l'approche des lignes de produits soit un nouveau paradigme dans le contexte du génie logiciel, le problème de la gestion de la variabilité dans les logiciels est déjà identifié, surtout dans le domaine de la gestion des configurations. (Anastapoulos 2000), et (Svahnberg, 2000) citent quelques techniques pour l'implémentation de la variabilité. Parmi les techniques et approches utiles pour la gestion de la variabilité, on peut citer :

-Les techniques de compilation : elles permettent la dérivation d'un produit pendant la phase de compilation. La compilation conditionnelle et le chargement de bibliothèques sont des exemples de ces techniques. Elles sont utiles si les produits diffèrent dans les parties de code à inclure ou à exclure et dans les bibliothèques qu'ils utilisent.

-Les techniques liées aux langages de programmation : les langages à objets (LAO) ont apporté quelques techniques utiles pour implémenter la variabilité. Citons l'abstraction à travers la notion de polymorphisme, la surcharge et la liaison dynamique. Les points de variation peuvent être définis comme abstraits dans la ligne de produits et redéfinis par chaque produit d'une manière spécifique. Certains LAO permettent de définir des classes paramétrées, appelées classes templates. La variabilité peut être implémentée en utilisant les classes templates lorsque les variantes ne diffèrent que sur un ensemble de paramètres (taille de mémoire, type d'un paramètre, …). Les diagrammes de classes d'UML permettent de définir des classes templates, et donc de gérer ce type de variabilité.

-Les patrons de conception : les patrons de conception (Gamma, 1995) fournissent des solutions réutilisables pour certains type de problèmes. Dans (Jézéquel, 1998) le patron de conception fabrique abstraite (Abstract Factory) est utilisé pour la réification des variants. La fabrique abstraite permet de définir une

interface pour la création des produits concrets. (Keepence et al , 1999) proposent un ensemble de patrons pour modéliser la variabilité dans les lignes de produits. Ces patrons sont basés sur des types de dépendances entre un ensemble de propriétés.

-Des approches de programmation : des approches récentes de génie logiciel peuvent être utilisées pour implémenter et gérer la variabilité dans les systèmes. La séparation des aspects (Kiczales, 1997) est une approche permettant de réduire la complexité des systèmes. Le principe est de décomposer le problème en un ensemble de composants fonctionnels et d'aspects transversaux. Quelques travaux (Anastapoulos, 2000) (Bayer, 2000) (Griss, 2000) proposent d'utiliser cette approche pour la gestion de la variabilité dans les LdPs. L'idée est que pour implémenter la variabilité les aspects peuvent être vus comme des points de variation et chaque produit, membre de la LdP, est différencié par l'ensemble des aspects qu'il utilise. La programmation générative (Czarnecki, 2000) est une approche qui s'intéresse au développement des familles de systèmes, elle est basée sur la notion de « générateur ». La variabilité dans les LdPs peut être implémentée en développant des générateurs comme des artefacts génériques, leur instanciation permet de générer l'implémentation d'un produit.

Les techniques présentées ci-dessus sont généralement liées aux langages de programmation, et relèguent l'activité de dérivation d'un produit à une activité de programmation. Mais il est aussi important de montrer et de spécifier la variabilité à un niveau plus abstrait, en particulier au niveau des modèles, par exemple en s'appuyant sur UML (Unified Modeling Language) (OMG, 2001). UML propose en effet un ensemble de notations, sous forme de diagrammes, pour l'analyse et la conception orientée objet. Mais UML est dédié à la modélisation d'un seul système à la fois, et ne supporte donc pas explicitement la modélisation de la variabilité essentielle aux lignes de produits. Cependant, UML est un langage extensible, grâce certains mécanismes tels que les *stéréotypes*, *les tagged values* et les *contraintes* (OMG, 2001 pp 274). Kobra (Atkinson,2000) propose une approche pour la création des lignes de produits basée sur la définition de composants par des vues UML comportant des points de variation.

Plusieurs travaux (ESAPS, 2001-b) (Clauß, 2001) s'appuyant sur les mécanismes d'extension d'UML, se sont intéressés à la spécification des points de variation à l'aide de stéréotypes spécifiques : <<*Optional*>> pour désigner les éléments optionnels, <<*variationPoint*>> pour les points de variation avec plusieurs alternatives. Notons que la majorité de ces propositions s'est intéressée aux diagrammes statiques d'UML. La contribution de cet article est de montrer comment ce type d'extensions visant à modéliser des lignes de produits peut aussi être utilisé sur les diagrammes de cas d'utilisation et les diagrammes dynamiques, et comment leur sémantique peut être en partie capturée par des contraintes OCL de méta-niveau ouvrant ainsi des perspectives pour la vérification automatique des modèles dérivés.

3 Modélisation de lignes de produits avec UML

Pour illustrer notre approche, nous avons choisi l'exemple d'une Ligne de Produits pour la Vente aux Enchères (LdPVE) sur Internet. Il s'agit d'une famille de serveurs de ventes aux enchères permettant à des utilisateurs de négocier l'achat et la vente de biens par le web (inspirée de systèmes tels que E-Bay) dont l'objectif est d'imiter le plus possible le déroulement de séances d'enchères dans des salles de ventes. Une approche ligne de produit est justifiée par la quantité de variations réglementaires selon les pays de déploiement, et selon des profils des administrateurs de tel serveurs. On distingue différents points de variations:

- La gestion des comptes des utilisateurs dans le système est effectuée en faisant directement des débits et des crédits sur les cartes bancaires des utilisateurs, ou bien en utilisant une monnaie virtuelle.
- Certaines variantes du logiciel permettent d'établir des prix de réserve pour les ventes définies par les vendeurs et / ou des prix d'augmentation minimum pour accepter les enchères des visiteurs.
- Des versions plus sophistiquées permettent à certains utilisateurs privilégiés de participer virtuellement aux ventes en programmant un "robot" avec une stratégie de participation paramétrée.
- En fonction des variantes, la commission retenue sur le prix de vente peut être paramétrée. Dans certains produits, elle peut même être nulle.

Pour modéliser cette ligne de produits, nous sommes intéressés aux diagrammes de cas d'utilisation, aux diagrammes de classes et aux scénarios décrits par des MSC (Message Sequence Chart). Nous présentons aux paragraphes suivants cette modélisation ainsi que les éléments d'extension que nous utilisons.

3.1 Variabilité dans les diagrammes de cas d'utilisation

La première étape dans une analyse et conception à base d'UML est l'identification des exigences et des besoins des utilisateurs. Les cas d'utilisation d'UML représentent un moyen adéquat pour l'expression des besoins. Une ligne de produits doit inclure les cas d'utilisation de chaque produit, même s'ils ne sont pas pertinents pour le reste de la famille. Pour cette raison, il est nécessaire de pouvoir considérer certains cas comme optionnels. L'optionalité dans les cas d'utilisation est spécifiée par le stéréotype <<Optional>> qui peut être associé aux acteurs et/ou aux cas d'utilisation. Dans notre exemple, le cas d'utilisation "programmer robot" ne concerne que certains produits, il est défini comme optionnel (Figure.2).

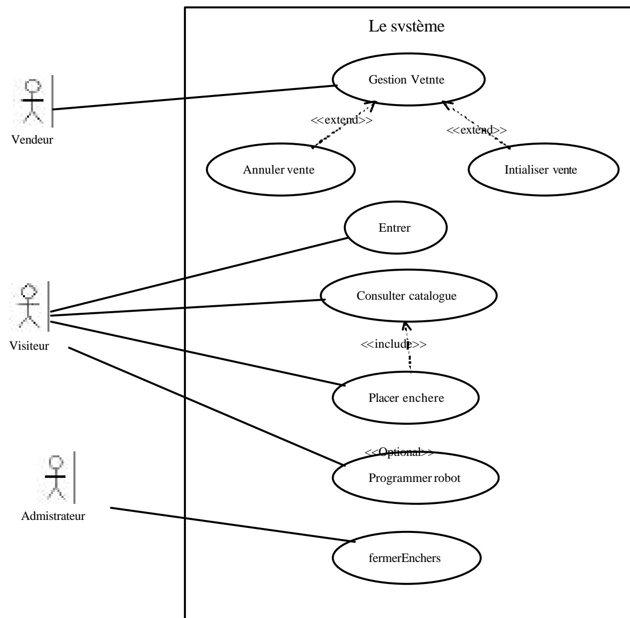

Figure.3. Cas d'utilisation de la LdPVE

3.2 Variabilité dans les diagrammes de classes d'UML

La majorité des travaux sur la modélisation des lignes de produits s'est intéressée aux modèles statiques d'UML, et tout particulièrement aux diagrammes de classes. Trois mécanismes sont présentés dans cette section pour spécifier la variabilité dans les modèles de classes d'UML :

-L'abstraction*.* La façon naturelle de modéliser les points communs et les points de variation dans les diagrammes de classes est d'utiliser le polymorphisme. En effet, le polymorphisme permet de regrouper les propriétés communes dans une classe abstraite (ou une interface) et de spécifier la variabilité dans des sous classes.

-La Paramétrisation. une classe peut être générique, et comporter un ensemble de paramètres, instantiés d'une manière spécifique par chaque produit. Nous utilisons les classes template d'UML pour spécifier les classes génériques.

-L'optionalité. Un modèle de ligne de produit doit contenir l'ensemble des éléments nécessaires pour la construction de chaque produit de la ligne. Lorsqu'un modèle d'un produit est dérivé, certains de ces éléments, dits optionnels seront supprimés. Pour montrer l'optionalité dans les diagrammes de

classe d’UML, nous utilisons un stéréotype spécifique : << Optional>>. Ce stéréotype peut être associé à une classe, un package ou une interface.

La Figure.4. montre le diagramme de classes associé à la LdPVE. Une enchère est associée à une vente particulière et concerne un bien. Un membre peut être un vendeur d’un bien ou un participant aux enchères. Il a un compte dans le système définissant des opérations en unités de monnaie virtuelle ou des opérations sur sa carte bancaire. La participation automatique dans une vente est gérée par la classe *Robot*. Les mécanismes présentés ci-dessus sont utilisés pour spécifier la variabilité dans le modèle de classe de la LdPVE. Polymorphisme et abstraction sont utilisés pour définir la variabilité du prix de réserve (les classes : *EnchereAvecPrixReserve* et *EnchereSansPrixReserve*). Comme mentionné ci-dessus, certains produits gèrent les compte des utilisateurs en utilisant une monnaie virtuelle d’autre les gèrent directement par des débits et crédits sur les cartes bancaires. Les classes : *MonnaieVirtuelle* et *CarteBancaire* sont définies comme optionnelles (le stéréotype << Optional>>). La classe *Robot* est elle aussi définie comme optionnelle, car elle n’apparaît pas dans tous les produits. La classe *SystemVenteEncheres* est définie comme une classe template avec un paramètre « Commission » pour désigner la commission retenue pour chaque vente.

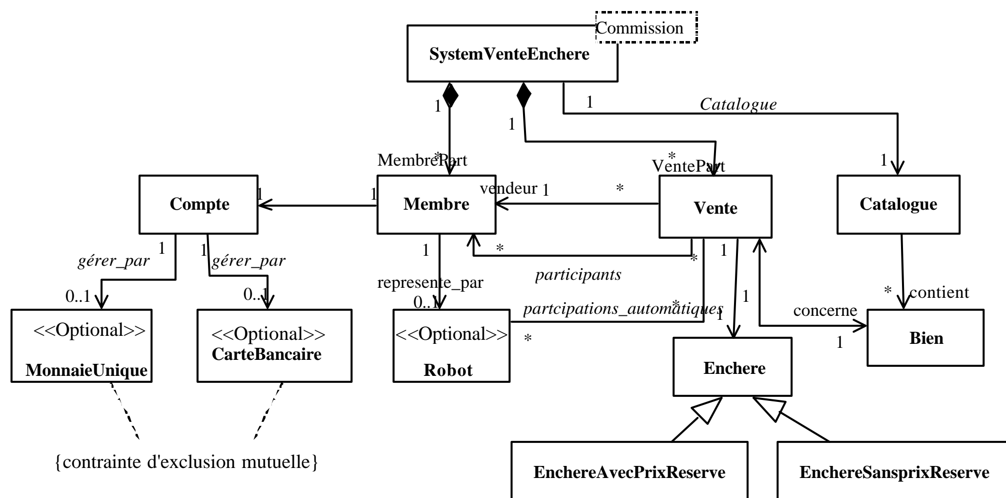

Figure.4. Diagramme de classe de la LdPVE

3.3 Variabilité dans les diagrammes de séquences d’UML

Les diagrammes de séquences d’UML permettent de modéliser et de spécifier le comportement d’un système avec un haut niveau d’abstraction. Ils sont généralement utilisés pour capturer les exigences, mais peuvent ensuite être utilisés pour documenter un système, produire des tests,… Dans leur forme actuelle, les

diagrammes de séquence ne permettent pas de modéliser la variabilité. Il semble cependant intéressant d'y introduire de façon similaire des points de variation dans, ce qui permettrait de générer en même temps qu'un produit particulier la liste des exigences de ce produit, un ensemble de jeux de tests, etc.. Les diagrammes de séquence d'UML 1.4 étant assez pauvres, nous faisons une première proposition pour étendre les Message Sequence Charts (relativement proches graphiquement, mais plus expressifs) afin qu'ils intègrent la notion de variabilité. Le lecteur intéressé pourra trouver cette proposition dans (Ziadi, 2002). Le prochain standard UML 2.0 (U2 Partners, 2002 pp 34-306) devrait faire des diagrammes de séquence un langage très similaire aux actuels MSC. Les extension proposées ici pour les MSC pourront alors être définies comme un stéréotypage particulier d'opérateurs de composition de UML 2.0. Les principaux moyens que nous proposons pour modéliser la variabilité sur le plan des comportements dynamiques sont :

-la virtualité. Un MSC décrivant le comportement d'une LdP peut définir des parties définissables dites parties virtuelles. Une partie virtuelle est spécifiée par une zone du MSC étiquetée par le mot clé « Virtual ». Elle peut être redéfinie par un MSC de raffinement associé à chaque produit. Notons que cette construction existe déjà dans le standard Z.120 (ITU, 1999).

-la variation. Il s'agit de définir plusieurs alternatives comportementales dans un MSC de la LdP, chaque produit ayant un seul comportement à la fois. Le mot clé « Variation » est utilisé pour spécifier ce type de variabilité. Notons que la variantion est différente de l'alternative (définie par le mot clé alt): une alternative définit un branchement possible qui peut se retrouver dans tout produit, tandis qu'une variation définit plusieurs produits.

-l'optionalité. L'optionalité proposée dans les MSCs comporte deux aspects : l'optionalité des instance et l'optionalité des comportements. Les instances optionnelles sont annotées par le stéréotype « Optional ». Un MSC d'une LdP peut aussi inclure des comportements optionnels, définis par une zone étiquetée par le mot clés « Optional ».

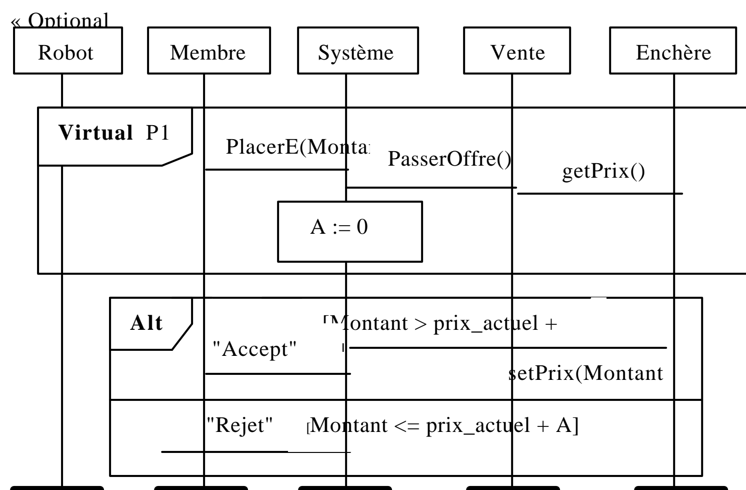

Figure. 5: Scénario générique

La Figure 5 illustre un cas d'utilisation de la virtualité : La partie P1 du MSC est virtuelle, et représente le placement d'une enchère. Cette partie peut être redéfinie par un autre comportement, par exemple pour remplacer un membre par un robot programmé pour suivre l'enchère. Le stéréotype <<Optional>> est utilisé pour montrer l'optionalité de l'instance *Robot* dans l'interaction. Un scénario associé à un produit particulier peut être dérivé en instanciant ces deux points de variation (absence ou présence de l'instance *Robot*, définition d'un comportement raffinant la partie virtuelle P1). On peut ainsi obtenir le MSC de la Figure 6 décrivant une enchère pour un produit sans robot et qui fixe un seuil d'augmentation des enchères à partir du MSC générique de la Figure 5.

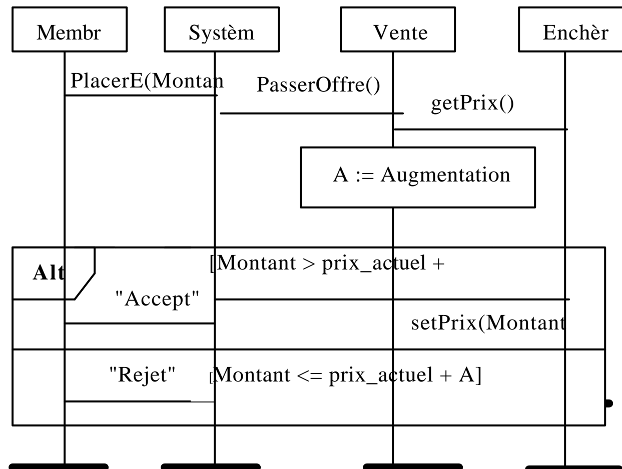

Figure. 6 : Scénario dérivé pour un produit particulier

4 Contraintes dans les architectures de lignes de produits

Revenant aux chaînes de production de voiture, on peut identifier quelques règles de fabrication établissant des liens de dépendances entre les différents points de variation. Par exemple, l'option de l'air conditionné exige une puissance de moteur supérieure à un certain seuil. De telles contraintes peuvent également apparaître dans le contexte des lignes de produits. (Bass, 1998), considère d'ailleurs que les contraintes font partie de l'architecture des LdP. Une approche définissant des relations de dépendance (présence et exclusion mutuelle) à l'aide d'extensions d'UML a également été proposée par (Clauß, 2001).

Deux types de contraintes apparaissent immédiatement dans le contexte des LdP: des contraintes génériques appliquées à toutes les modélisations de ligne de produits à l'aide des stéréotypes présentés précédement, et des contraintes spécifiques associées à une ligne de produits particulière. Ces contraintes peuvent être exprimées en OCL (Object Constraint Language) (Warmer, 1998). Les avantages sont multiples : OCL a une puissance d'expression raisonnable, et permet de spécifier de contraintes de niveau méta-modèle, essentielles pour exprimer des propriétés d'un modèle de ligne de produits.

4.1 Contraintes génériques

La possibilité de définir des éléments optionnels et des points de variation n'assure pas automatiquement la cohérence d'un modèle de ligne de produits. Il existe quelques règles structurelles que toute ligne de produits doit respecter. Par exemple, un élément non optionnel ne doit pas pouvoir utiliser d'élément optionnel (le risque étant de dériver un produit incomplet sans s'en rendre compte). Les contraintes génériques sont des contraintes de correction «syntaxiques » d'un modèle de ligne de produits, définies au niveau méta-modèle, et évaluées pour toute ligne de produit (cf Figure 7).

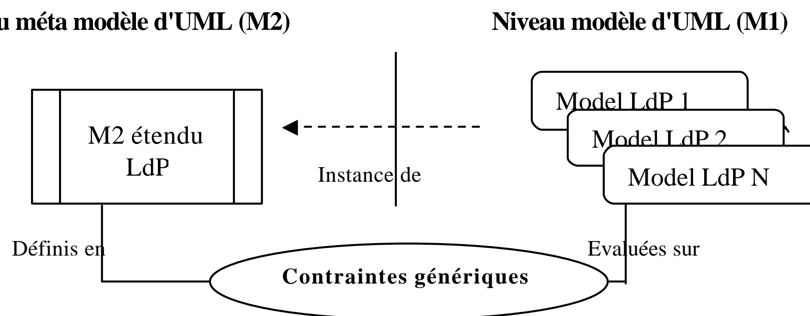

Figure.7. Contraintes OCL génériques exprimées au niveau méta-modèle

Exemples de contraintes génériques associées aux cas d'utilisation

Le méta-modèle d'UML (OMG, 2001 pp 2.135) définit deux types de relation entre les cas d'utilisation : La première est une relation d'extension, spécifiée par une relation stéréotypée <<extend>>, qui indique qu'un cas d'utilisation en étend un autre. La deuxième relation est celle d'inclusion, spécifiée par une relation stéréotypée <<include>>, et dans laquelle un cas d'utilisation en contient un autre. Ces deux relations de dépendance entre les cas d'utilisation engendrent deux contraintes génériques :

- *une contrainte d'extension*. Pour assurer la cohérence des cas d'utilisation de produits dérivés, un cas d'utilisation non optionnel ne doit pas étendre un cas

d'utilisation optionnel. On ajoute cette contrainte comme un invariant de la méta-classe *Extend* qui spécifie cette relation d'extension dans le méta-modèle d'UML :

```
context Extend
--L'extension d'un cas d'utilisation optionnel, doit aussi être optionnel
inv     self.extension.isStereotyped ("Optional")
   implies  self.base.isStereotyped("Optional")
```

- *une contrainte d'inclusion*. Un cas d'utilisation non-optionnel ne doit pas être contenu dans un cas d'utilisation optionnel. Cette contrainte est identique à la contrainte d'extension, à ceci près qu'elle est ajoutée comme un invariant de la méta-classe *Include,* qui représente la relation d'inclusion dans le méta-modèle d'UML.

Exemples de contraintes génériques associées aux modèles statiques

De la même manière que pour les cas d'utilisation, les diagrammes de classes peuvent permettre la génération de produits incohérents si les dépendances entre éléments optionnels et non optionnels ne sont pas accompagnées de contraintes. On peut ainsi distinguer :

- *une contrainte de dépendance*. Un élément optionnel ne doit pas être dépendant d'un élément optionnel (un exemple de dépendance dans UML est l'utilisation, qui apparaît lorsqu'un package utilise un autre package). La dépendance dans UML est spécifiée par la méta-classe *Dependency*, (OMG, 2001 pp 2.15). Nous ajoutons cette contrainte comme un invariant de la méta-classe *Dependency* :

```
context Dependency
inv self.supplier→ exists(S:ModelElement|S.isStereotyped  ("Optional"))
implies self.client → forAll( C:ModelElement | C.isStereotyped("Optional") )
```

- *une contrainte d'héritage*. Lorsqu'une classe non optionnelle hérite d'une classe optionnelle, il est probable que le modèle contienne une incohérence. Cependant, dans certains cas particuliers, notamment lorsque le modèle de la ligne de produit contient de l'héritage multiple, ce type de construction peut être cohérent. Il est cependant plus prudent de souligner le parties du modèle qui contiennent ce type d'héritage, en associant la contrainte suivante à la méta-classe *Generalization* (voir (OMG, 2001 pp 2.14) qui représente l'héritage dans le méta-modèle d'UML :

```
context Generalization
-- Si la classe parente dans un héritage est optionnelle, la sous-classe est
également optionnelle
```

Inv: self.parent.isStereotyped ("Optional") **implies** self.child.isStereotyped("Optional")

4.2 Contraintes spécifiques

Un concepteur doit pouvoir exprimer des relations entre les éléments du modèle, spécifiques à une ligne de produit particulière. Ainsi, dans l'exemple de la ligne de produits vente aux enchères, nous exigeons qu'un compte soit exclusivement géré en monnaie virtuelle, ou à l'aide de transactions sur une carte bancaire. Tout produit dérivé doit satisfaire l'ensemble des contraintes spécifiques définies par le concepteur. Ces contraintes sont également spécifiées au niveau méta-modèle. (Figure.8).

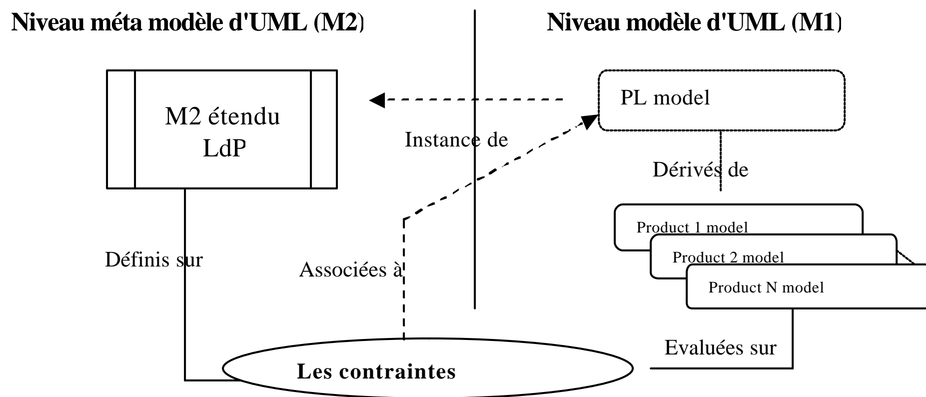

Figure.8. Contraintes spécifiques

Exemples de contraintes spécifiques

Les éléments de la ligne de produits peuvent être fortement dépendants, ou au contraire incompatibles. Pour une ligne de produit donnée, il faut donc savoir exprimer des relations de dépendance ou au contraire d'exclusion mutuelle.

- *La contrainte de présence*. permet d'exiger la présence d'une partie du modèle de la ligne de produit dans un produit dérivé lorsqu'une autre partie s'y trouve. Nous ajoutons cette contraintes comme invariant de la classe *Namespace* qui représente l'espace de nommage des éléments d'un modèle :

context Namespace
-- *la présence de la classe C2 requiert la présence de la classe C1*
inv self.presenceClass(C1) **implies** self.presenceClass(C2)

- *La contrainte d'exclusion mutuelle*. spécifie un lien d'exclusion mutuelle entre deux parties du modèle de la ligne de produit. Dans l'exemple de la LdPVE une contrainte d'exclusion mutuelle peut être ajoutée entre les deux classes optionnelles (*MonnaieVirtuelle* et *CarteBancaire*) :

context Namespace
inv (self.presenceClass(MonnaieVirtuelle) **implies not** self.presenceClass(CarteBancaire)) **and** (self.presenceClass(CarteBancaire) **implies** **not** self.presenceClass(MonnaieVirtuelle))

Les contraintes définies précédemment utilisent les primitives OCL suivantes :

- *isStereotyped(S)*. Vérifie si un élément donné est stéréotypé par la chaîne S.
context ModelElement :: isStereotyped(S : String)
post :**result** = self.stereotype → exists(s | s.name = S)

- *presenceClass(C)*. Vérifie la présence d'une classe C dans le Namespace courant
context Namespace :: presenceClass(C : Class)
post :**result** = self.owenedElement → **exists**(c : ModelElement | c.oclIsType(Class) **and** c.name = C.name)

5 Conclusions et Perspectives

Nous avons proposé une technique de modélisation de lignes de produits fondée sur une extension des cas d'utilisation, des diagrammes de classes et des futurs diagrammes de séquence d'UML. Ces extensions permettent de prendre en compte la variabilité dans un modèle UML. En plus de ces extensions, il est apparu nécessaire de définir un ensemble de *contraintes architecturales,* exprimées en OCL, décrivant des règles de cohérence et de dépendance entre les éléments d'une ligne de produits. L'avantage de cette approche est qu'elle peut s'appliquer à tout élément de UML décrit dans le méta-modèle. Ainsi, les diagrammes de séquence pourront bénéficier aisément des mêmes possibilités lorsque leur définition dans UML 2.0 sera un peu plus stable.

A partir de ce modèle, de nombreux problèmes intéressants connectés au monde des lignes de produits apparaissent. En amont, on peut se poser la question : comment construire une ligne de produits ? Peut-on envisager une automatisation de la recherche des éléments communs et variables dans des vues UML ?

En aval, ce sont des problème liés à la dérivation de produits qui se posent. La dérivation de produits est en effet bien plus qu'une simple choix d'options dans un ordre indéterminé.

Enfin, on peut envisager d'utiliser les lignes de produit pour vérifier certaines propriétés globales à tous les produits. De telles propriétés peuvent être par exemple la préservation d'un comportement dans les vues dynamiques dérivées de chaque produit.

Bibliographie

Anastapoulos M., Gacek.C., *Implementing Product Line Variability*, Technical report IESE report N°: 089.00/E, Franhofer IESE publication, 2000.
C. Atkinson, J. Bayer, and D. Muthig. *Component-based product line development. the KobrA approach*, In Proc. of the 1st Software Product Lines Conference (SPLC1), pages 289–309,2000.

Bass.L., Clements.P., Kazman.R., *Software Architecture in Practices*, Addison–Wesley, 1998.

Bayer.J., *Toward engineering product line using concerns*, GCSE 2000, Young Workshop, 2000.

Clauß.M., «Modeling variabilities with UML», GCSE 2001, Young Workshop, 2001.

Czarnecki.K., Eisenecker.U.W., *Generative Programming*, Addison-wesley, 2000.

ESAPS (Engineering Software Architectures, Processes and Platforms for System-Families), *Introduction to domain analysis*., Délivrables du projet ESAPS. http://www.esi.es/esaps, 2001

ESAPS *Styles, structures and views for handling commonalities and variabilities* , ESAPS deliverables, 2001.

Gamma.E., Helm.R., Johnson R, and Vlissides.J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

Griss.M L., *Implementing Product-line Features by Composing Component Aspects*, in Proceedings of the First Software Product Line Conference, P. Donohoe, pp. 271-288, 2000.

ITU-T, *Z.120 : Message Sequence Chart (MSC)*, 1999.

Jézéquel.J-M., *Reifying Variants in Configuration Management*, ACM Transaction on Software Engineering and Methodology, pages 526-538, 1998.

Keepence.B, Mannion.M., *Using Patterns to Model Variability in Product Families*, IEEE Software, 16(4): pages 102-108, 1999.

Kiczales.G., et al, *Aspect-Oriented Programming*, In ECOOP'97 –Object Oriented Programming 11th European Conference, 1997.

Northrop.L., *A Framework for Software Product Line Practice –Version 3.0.*, http://www.sei.cmu.edu/pLdP/framework.html#framework_toc, Software Engineering Institute (SEI), 2002

OMG, *Unified Modeling Language specification*, version 1.4, 2001.

Svahnberg.M, Bosch.J, *Issues Concerning Variability in Software Product lines*, in F. van der Linden, editor, Software Architecture for Product Families International Workshop IW-SAPF-3, LNCS 1951, pp. 146-157, Springer 2000.

U2 PARTNERS., *Proposal for UML 2.0 Infrastructure and Superstructure*, ad/00-09-01 and ad/00-09-02. http://www.u2-partners.org/artifacts.htm

Warmer.J., Klepe.A., *The Object Constraint Language- Precise Modeling with UML*, Object Technology Series, Addison-Wesley, 1998.

Ziadi.T., Hélouët.L., Jézéquel.J-M., *Modeling Behaviors in Product Lines*, International Workshop in Engineering Requirement for Product Line (REPL'02), Essen, 2002.