Master Informatique - Université de Montpellier - Ingénierie Dirigée par les Modèles - 2016

# Lignes de produits logiciels
# Transformation de modèles de caractéristiques

Dans ce TP, nous allons d'abord créer notre propre méta-modèle à partir duquel nous pourrons instancier plusieurs modèles de caractéristiques différents. Ensuite, nous implémenterons une transformation vers des modèles de caractéristiques conformes à un autre méta-modèle : `FAMILIAR`. `FAMILIAR` est un *DSL* qui s'accompagne d'un environnement graphique permettant entre autre de visualiser un modèle de caractéristiques. Nous nous en servirons pour visualiser les modèles instanciés depuis notre méta-modèle.

## 1 Rappel : les modèles de caractéristiques

Les modèles de caractéristiques (*feature models*) sont une famille de langages de description visuels qui permettent de décrire un ensemble de caractéristiques ainsi que des dépendances entre ces caractéristiques. Ils sont utilisés pour représenter un ensemble de produits appartenant à une même famille, de manière compacte et compréhensible.

Un modèle de caractéristiques organise de manière hiérarchique un ensemble de caractéristiques dans un arbre : la caractéristique racine est la plus générale (elle représente souvent le nom de la famille modélisée) alors que les caractéristiques les plus basses dans la hiérarchie sont les plus spécialisées. Une arête de l'arbre représente donc une relation de raffinage entre une caractéristique mère et une (plusieurs) caractéristique(s) fille(s). Un produit correspond à un ensemble de caractéristiques sélectionnées dans cet arbre. Ces arêtes peuvent être décorées pour contraindre la sélection. Si on sélectionne une caractéristique qui possède une sous-caractéristique : une arête *optionnelle* définit que la sous-caractéristique n'est pas obligatoirement sélectionnée, et une arête *obligatoire* force la sous-caractéristique à être sélectionnée. On trouve aussi des contraintes sur les groupes de caractéristiques. Si on sélectionne une caractéristique qui possède un groupe de sous-caractéristiques : dans un groupe *or*, au moins une des sous-caractéristiques doit être sélectionnée, alors que dans un groupe *xor*, exactement une doit être sélectionnée. Les arêtes correspondant à ces 4 relations sont exposées dans la Figure 1 (gauche). Des contraintes qui ne peuvent être exprimées dans la hiérarchie (implications *cross-tree*, exclusions) peuvent être rajoutées textuellement.

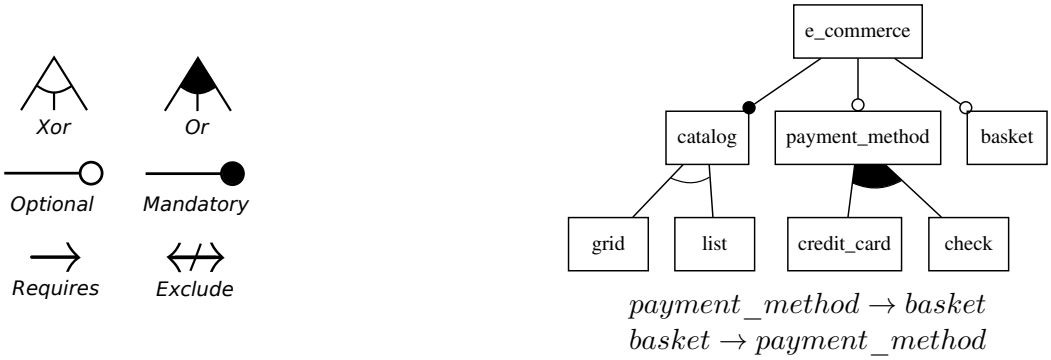


FIGURE 1 – Exemple de modèle de caractéristiques (droite) et ses différentes relations (gauche)

La Figure 1 (droite) représente un exemple de modèle de caractéristiques sur une famille d'applications de e-commerce. On peut lire que : une application de e-commerce possède obligatoirement un catalogue ; ce catalogue peut être affiché sous forme de grille ou de liste, mais pas les deux ; l'application peut optionnellement proposer des méthodes de paiement ; elle peut proposer un paiement par carte, par chèque, ou bien les deux ; une application de e-commerce peut éventuellement proposer une gestion de panier ; si la gestion de panier est sélectionnée, l'application doit posséder des méthodes de paiement, et inversement.

La configuration $\{e\_commerce, catalog, grid, basket, payment\_method, credit\_card\}$ respecte l'ensemble des contraintes du modèle, et représente donc une variante possible de la famille de produits. Cependant, la configuration $\{e\_commerce, catalog, grid, basket\}$ ne représente pas une variante possible, car la contrainte $basket \rightarrow payment\_method$ n'est pas respectée. La liste des 8 configurations valides du modèle de la Figure 1 sont listées dans la Table 1.

TABLE 1 – Liste des 8 configurations décrites par la modèle de caractéristiques de la Fig. 1

| | e_commerce | catalog | grid | list | paym_method | credit_card | check | basket |
|---|---|---|---|---|---|---|---|---|
| c1 | x | x | x | | | | | |
| c2 | x | x | | x | | | | |
| c3 | x | x | x | | x | x | | x |
| c4 | x | x | x | | x | | x | x |
| c5 | x | x | x | | x | x | x | x |
| c6 | x | x | | x | x | x | | x |
| c7 | x | x | | x | x | | x | x |
| c8 | x | x | | x | x | x | x | x |

Les modèles de caractéristiques sont le standard *de facto* pour modéliser et représenter les connaissances liées à la variabilité d'une famille de produits. Ils permettent une meilleure gestion des éléments communs d'un ensemble de variantes, et des éléments spécifiques de certaines variantes, favorisant ainsi la réutilisation : les coûts et temps de développement sont alors réduits, et la qualité des produits ainsi obtenus est meilleure.

## 2 Un méta-modèle pour les modèles de caractéristiques

La Figure 2 représente un méta-modèle de modèles de caractéristiques. Il se situe au niveau M2 de l'architecture d'IDM. Le niveau M0 (monde réel), représente l'ensemble des systèmes logiciels similaires appartenant à une même ligne de produits logiciels. Au niveau M1, on retrouve les modèles capables de décrire l'ensemble des ces systèmes logiciels, comme par exemple les modèles de caractéristiques : tout ensemble de systèmes logiciels correspondant à une LDPL peut donc être représenté par un modèle de caractéristiques. Au niveau supérieur (M2) se situe alors les méta-modèles de ces modèles de caractéristiques, permettant de représenter tout modèle de caractéristiques.

**Question 1 :** Ouvrez dans Eclipse un projet Java classique. Dans ce TP, nous n'utiliserons pas EMF. Implémentez le méta-modèle de la Figure 2 : créez les classes, attributs et méthodes nécessaires à son utilisation.

**Question 2 :** Créez une nouvelle classe `TestFM` possédant une méthode `main`. Elle sera le point de départ de notre programme.

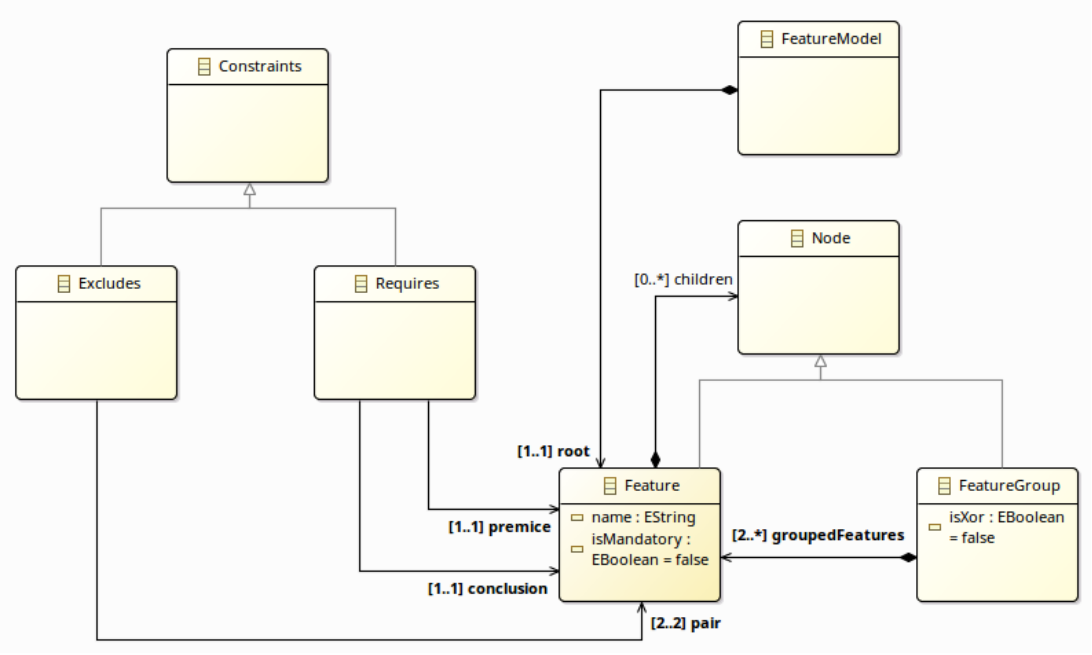


FIGURE 2 – Méta-modèle de modèles de caractéristiques

**Question 3 :** Utilisez les classes créées précédemment pour instancier le modèle de caractéristiques représenté dans la Figure 1.

# 3 FAMILIAR

`FAMILIAR` (pour *FeAture Model scrIpt Language for manIpulation and Automatic Reasoning*) est un *domain-specific language* pour la manipulation de modèles de caractéristiques. Il permet entre autre de créer des modèles de caractéristiques conformes à ceux décrits en Section 1, de réaliser des opérations de modification (édition, fusion de modèles) ainsi que des opérations d'analyse telles que la vérification de la validité du modèle, la détection des caractéristiques inutilisées, la génération des configurations possibles, la comparaison de modèles, etc.

## 3.1 Téléchargement et installation

Le langage `FAMILIAR` s'accompagne d'un environnement disponible en version graphique, téléchargeable sous la forme d'un fichier JAR :

*Version graphique* :
`http://mathieuacher.com/pub/FAMILIAR/releases/FML-environment-1.1.jar`

Dans ce TP, nous utiliserons cet outils pour visualiser nos modèles de caractéristiques instanciés.

Pour utiliser l'environnement graphique, lancez simplement la commande suivante dans un terminal :
`java -jar FML-environment-1.1.jar`

La documentation de `FAMILIAR` peut être trouvée ici :

https://github.com/FAMILIAR-project/familiar-documentation/tree/master/manual

### 3.2 Syntaxe

Pour instancier un nouveau modèle de caractéristiques avec FAMILIAR, il faut utiliser le constructeur "FM" :

```
fm1 = FM (A : B C [D]; B: (E|F) ; D : (J|K)+ ; (!C | D) ; )
```

— A est la caractéristique **racine**
— B, C et D sont les caractéristiques filles de A : B et C sont **obligatoires**, D est **optionnelle**
— E et F forment un **groupe XOR**, et sont les caractéristiques filles de B
— J et K forment un **groupe OR**, et sont les caractéristiques filles de D
— ( !C | D) est l'équivalent de la CTC **requires** : (C → D)
— Une contrainte d'exclusion (**excludes**) entre C et D s'écrirait donc ( !C | !D)

Les détails de la notation textuelle interne sont disponibles au lien suivant :
https://github.com/FAMILIAR-project/familiar-documentation/blob/master/manual/featuremodel.md

### 3.3 Environnement graphique

Lancez l'environnement graphique sur un terminal. Vous devez voir apparaître une fenêtre dans laquelle un modèle de caractéristiques *wiki* est déjà défini. Notez aussi la présence d'une console en bas de la fenêtre.

**Question 4 :** (Prise en main) Visualisez le modèle de caractéristiques `fm1` représenté en langage FAMILIAR dans la sous-section précédente. Pour cela, copiez entièrement la ligne débutant par "`fm1 = FM(A ...`" et collez-la dans la console de l'environnement. Validez l'instanciation en appuyant sur la touche `entrée`. Ensuite, dans le menu en haut de la fenêtre, cliquez sur `Display » Display all ...` : vous devriez voir apparaître un nouvel onglet contenant une version graphique du modèle de caractéristiques `fm1`.

## 4 Transformation

Nous voulons pouvoir transformer n'importe quel modèle de caractéristiques instancié depuis notre méta-modèle en une instance conforme au méta-modèle de FAMILIAR. Pour cela, nous allons définir une fonction dans la classe `TestFM` qui va s'occuper de la transformation. Cette fonction prendra en paramètre une instance de la classe `FeatureModel` de votre méta-modèle, et renverra une chaine de caractères décrivant l'instance au format FAMILIAR.

**Question 5 :** Ajoutez des méthodes dans les classes de votre méta-modèle pour extraire automatiquement une chaine de caractères permettant de décrire ses instanciations au format FAMILIAR.

**Question 6 :** Testez votre transformation sur votre instanciation précédente (Question 3). Affichez la chaine de caractères dans Eclipse afin de la copier, puis coller-la dans la console de l'environnement graphique de FAMILIAR. Visualisez le modèle de caractéristiques. Correspond-il bien à la Figure 1 ?

**Questions 7 :** À présent, instanciez le modèle de caractéristiques de la Figure 3. Tranformez-le au format FAMILIAR et visualisez-le avec l'environnement graphique.

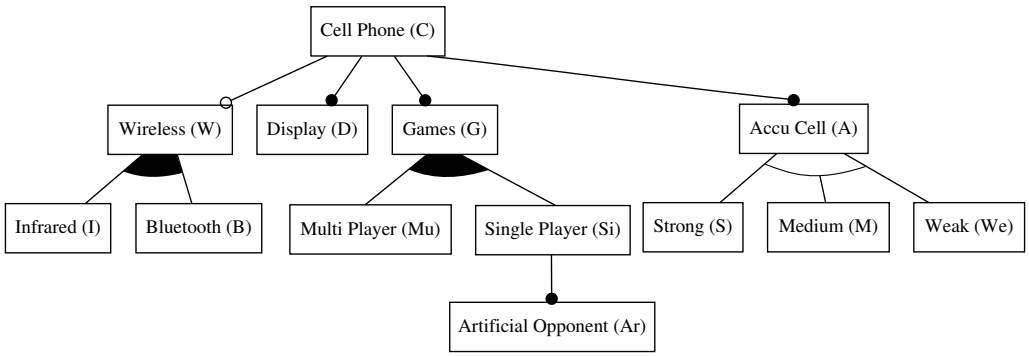


FIGURE 3 – Modèle de caractéristiques sur des téléphones portables