Master Informatique - Université de Montpellier - Ingénierie logicielle - 2018

Lignes de produits logiciels
Dérivation de produits avec `FeatureIDE`

Objectifs :
- Prise en main du plugin `FeatureIDE` de `Eclipse`
- Modélisation de la variabilité d'une ligne de produits avec les *feature models*
- Génération automatique des variantes logicielles avec une approche basée sur les annotations

1 Contexte

Nous allons appliquer les concepts vus en cours sur la mise en place des lignes de produits logiciels. Pour cela, nous allons créer une ligne de produits logiciels de "générateurs de graphes" qui permettent à un utilisateur de construire des graphes en `Graphviz` [1]. Une variante logicielle de cette ligne de produits prendra la forme d'un projet `Eclipse` et aura pour objectif d'interroger l'utilisateur via la console sur les sommets et les arêtes du graphe qu'il souhaite construire. Une fois le graphe terminé, le programme doit afficher sur la console le code `Graphviz` correspondant ; Le graphe associé à ce code pourra être visualisé en ligne sur le site `Webgraphviz` [2]. Les variantes de "générateurs de graphes" appartenant à cette ligne de produits de logiciels diffèrent de par les types de graphes qu'ils permettent de construire : par exemple, certains générateurs pourront proposer des couleurs sur les sommets et les arêtes, d'autres laisseront le choix de la forme des sommets, etc.

Si un utilisateur (disons, un doctorant la veille d'une *deadline*) souhaite créer rapidement des graphes très simples (sans couleur, non orientés et avec des sommets de forme rectangulaire), un outil lui proposant de rajouter des couleurs ou bien de changer la forme des flèches sur les arêtes/arcs ne lui est d'aucune utilité. Imaginons un second utilisateur souhaitant pouvoir modéliser beaucoup d'informations différentes dans un seul graphe afin d'illustrer la conclusion d'un chapitre de sa thèse. Il souhaite pour cela se reposer sur les couleurs et les formes des éléments : l'outil du premier utilisateur ne lui est alors d'aucune utilité.

À la fin de ce TP, nous devrons pouvoir sélectionner les caractéristiques attendues pour un logiciel de génération de graphes et dériver automatiquement le projet `Eclipse` permettant de créer les types de graphes spécifiés. Pour nous aider dans cet objectif, nous utiliserons `FeatureIDE`.

2 `FeatureIDE` et `Antenna`

2.1 Installation et fonctionnement

FeatureIDE est un plugin `Eclipse` qui a pour but d'assister le développement logiciel **orienté caractéristiques**. Une caractéristique (ou *feature*) est une fonctionnalité ou un comportement visible d'un logiciel. Elles permettent de faire la liaison entre les artefacts logiciels de bas niveau (code, test, documentation) et les exigences de haut niveau formulées par un utilisateur.

`FeatureIDE` peut être installé depuis le *Marketplace* (`FeatureIDE` 3.5.1).

1. http://www.graphviz.org/
2. http://www.webgraphviz.com/

`FeatureIDE` permet de développer du code correspondant à l'architecture commune et aux artefacts réutilisables d'une ligne de produits, ainsi qu'un *feature model* représentant les caractéristiques (*features*) du projet ainsi que sa variabilité. `FeatureIDE` propose ensuite plusieurs approches différentes pour définir des liens entre le code et les *features* du modèle, permettant ainsi de générer automatiquement des produits logiciels autorisés par ce modèle. Les deux approches les plus connues sont l'approche par **composition** et l'approche par **annotation**. Dans l'approche par composition, le code correspondant à chaque *feature* est développé dans un fichier séparé. Pour générer le produit correspondant à une configuration valide du *feature model*, les fichiers correspondant aux *features* sélectionnées sont mis en commun pour former un projet final. Dans l'approche par annotation, un seul projet regroupant le code de toutes les *features* est mis en place, et ses lignes de code sont annotées en fonction des *features* auxquelles elles correspondent. Pour générer un produit correspondant à une configuration valide du *feature model*, le code des *features* non sélectionnées est commenté, ne laissant que les caractéristiques désirées dans le projet final.

Lorsque l'on crée un projet avec `FeatureIDE`, celui-ci nous propose de choisir un *composer*, qui agit comme une extension permettant d'appliquer une des approches citées ci-dessus. Nous allons utiliser le *composer* `Antenna`, un préprocesseur pour les fichiers Java, qui va nous permettre d'appliquer l'approche par annotation.

Antenna permet d'annoter des morceaux de code avec les *features* auxquelles ils correspondent. Ces annotations permettent d'indiquer que le code encadré ne sera pris en compte que lorsque la *feature* correspondante sera sélectionnée. Par exemple, si l'on a une *feature* `color` représentant la gestion de la couleur des graphes, on pourra trouver dans le code :

```
//#if color
public Color getColor() {
return color;
}
public void setColor(Color color) {
this.color = color;
}
//#endif
```

2.2 Anatomie d'un projet

Question 1 : Créez un projet `FeatureIDE` : New > Project > FeatureIDE project. Dans la fenêtre qui s'affiche, sélectionnez le *composer* `Antenna`, et validez.

Un projet `FeatureIDE + Antenna` est composé des éléments suivants :

- Un dossier `src` dans lequel on va mettre les packages et les classes structurant le code de la ligne de produits.
- Un fichier `model.xml` qui représente le *feature model* associé à la ligne de produits. Par défaut, il ne présente que deux *features*.
- Un dossier `configs` contenant des fichiers .xml, chacun représentant une configuration du *feature model*. Ces fichiers prennent la forme de "configurateurs" assistant l'utilisateur dans la sélection des *features* en fonction des contraintes définies par le *feature model*. Par défaut, ce dossier ne contient qu'un seul fichier appelé `default.xml`.
- Un dossier `products` qui n'est pas présent à la création du projet et qui sera créé automatiquement lors de la phase de génération des variantes logicielles. Dans ce dossier seront créés différents dossiers contenant le code représentant les variantes logicielles générées. Chaque dossier correspond à une configuration définie dans les fichiers .xml précédents.

3 Ingénierie du domaine : analyse et représentation du domaine

3.1 Spécifier un *feature model*

Avant de commencer à développer, nous allons définir les *features* de notre projet, ainsi que le *feature model* représentant sa variabilité.

Question 2 : À partir du cahier des charges suivant, identifiez les *features* et leur contraintes. Ensuite, définissez un *feature model* représentant ces informations. Vous utiliserez l'éditeur graphique de `FeatureIDE`.

> Cahier des charges : Tous les générateurs de graphes permettent de créer des sommets. Les sommets d'un graphe peuvent être de forme rectangulaire, ou bien de forme elliptique. Les deux formes peuvent se retrouver dans un même graphe, comme l'exemple présenté précédemment. Un graphe possède des arêtes dont la représentation est uniforme dans tout le graphe : un générateur de graphes ne peut donc générer des graphes n'ayant qu'un seul et même type d'arêtes. Notons que, par abus de langage, nous utilisons ici le mot "arête" pour définir à la fois des arêtes (cas non orienté) et des arcs (cas orienté). Ces arêtes peuvent avoir des flèches standards (pointe fermée), des flèches en forme de "v", ou pas de flèche (i.e., cas non orienté).
>
> Enfin, un générateur de graphes peut éventuellement gérer des couleurs, mais tous les générateurs ne gèrent pas nécessairement les mêmes couleurs. Pour simplifier notre modèle, nous ne chercherons à gérer que deux couleurs différentes (rouge et vert par exemple).

Question 3 : On veut maintenant rajouter une caractéristique indiquant si les générateurs de graphes réalisent des graphes orientés ou non. Quelles contraintes vous faut-il rajouter pour que le modèle soit cohérent ? Pour ajouter une contrainte, faire un clic droit sur la *feature* puis "Create Constraint starting with".

3.2 Quelques réflexions sur la cohérence du *feature model*

Question 4 : Quelles sont les *features* de votre modèle représentant le cœur de la ligne de produits ?

Question 5 : Ajoutez une contrainte d'implication depuis une *feature* obligatoire vers une *feature* optionnelle. Quel est le problème de modélisation détecté ?

Question 6 : Ajoutez une contrainte d'implication depuis une *feature* obligatoire vers une *feature* appartenant à un groupe XOR. Quel est le problème de modélisation détecté ?

3.3 Spécifier une configuration du *feature model*

Ouvrez le fichier `default.xml` présent dans le dossier `configs`. Les différentes *features* de votre modèle apparaissent à côté de cases à cocher. Elles sont organisées sous la forme d'une hiérarchie équivalente à celle du modèle.

Question 7 : Assurez-vous de la cohérence des combinaisons valides en fonction du cahier des charges précédent. Cochez uniquement la case représentant la *feature* "couleur" : la configuration est elle valide ? Pouvez vous sélectionner n'importe quelle combinaison de couleurs ? Pouvez-vous sélectionner deux types de flèches ? Pouvez-vous sélectionner la *feature* "graphe orienté" et des arêtes sans flèches ?

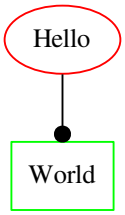
Question 8 : Combien de variantes de "générateurs de graphes" différentes pouvez vous dériver avec ce modèle ?

4 Ingénierie du domaine : implémentation de l'architecture commune et des artefacts réutilisables

4.1 Introduction à Graphviz

`Graphviz` est un langage permettant de définir des graphes. Il propose, entre autres, de définir les couleurs et les formes des sommets et des arêtes. L'exemple ci-dessous définit un graphe avec deux sommets : vous pouvez copier le code et le coller dans la zone de texte de webgraphviz.com pour visualiser le graphe correspondant.

```
digraph G {
Hello [color=red]
World [color=green, shape=box]
Hello -> World [arrowhead=dot]
}
```



Une partie de la documentation se trouve aux pages suivantes :
— gérer la forme des arêtes : https://www.graphviz.org/doc/info/arrows.html
— gérer la forme des sommets : https://www.graphviz.org/doc/info/shapes.html
— gérer les couleurs : http://www.graphviz.org/doc/info/colors.html
La page https://renenyffenegger.ch/notes/tools/Graphviz/examples/index propose des exemples d'utilisation du langage.

4.2 Diagramme de classes UML

La Figure 1 représente le diagramme de classe UML d'un générateur de graphes.
La classe `Graph` compte deux méthodes :
— `createNewGraph()` qui demande à l'utilisateur des informations sur le graphe qu'il veut construire (nombre de sommets, label, forme, etc.) et crée les instances de `Node` et `Edge` correspondantes.
— `printInGraphViz()` qui affiche les informations précédentes au format Graphviz dans la console.

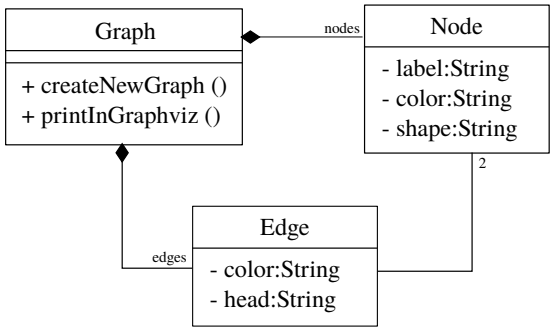


FIGURE 1 – Diagramme de classes UML du générateur de graphes

Ce diagramme prend en compte toutes les *features* que l'on peut trouver dans un générateur de graphes. Grâce au *feature model* défini précédemment, nous allons pouvoir créer des variantes n'ayant qu'un sous ensemble de ces *features*.

Question 9 : Implémentez les trois classes de la Figure 1 dans le dossier `src` du projet `FeatureIDE`.
Gardez en tête les différentes *features* et leur variabilité, et écrivez votre code de sorte qu'il soit facile de distinguer les parties correspondantes à ces *features* : cela devrait faciliter la phase d'annotation.

Question 10 : Ajoutez les annotations `Antenna` dans le code.

4.3 Visualiser la répartition des *features* dans le code

Afin de faciliter la visualisation des *features* dans le code et leur répartition, `FeatureIDE` propose une coloration (*color scheme*) des lignes de code en fonction des annotations. Un exemple est présenté en Figure 2.

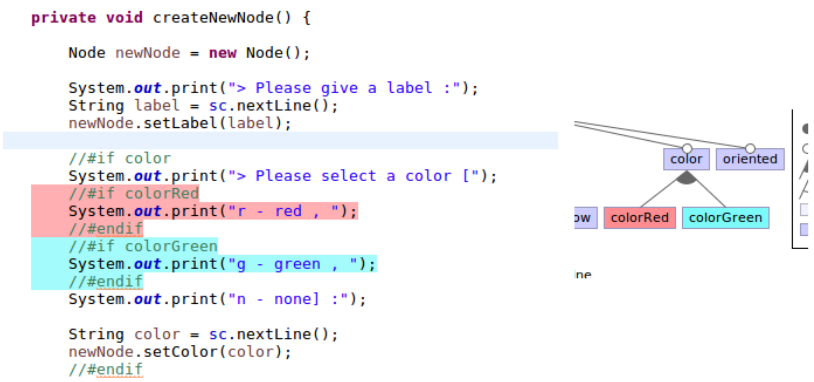


FIGURE 2 – Exemple de *color scheme* sur deux caractérsitiques

Question 11 : Réalisez un *color scheme* pour les *features* représentant la gestion de la couleur dans les graphes. Pour cela, faire un clic droit sur le projet > FeatureIDE > Color scheme menu > add color scheme > new color Scheme. Nommez-le "color1" par exemple. Puis, dans le *feature model*, faire un clic droit sur les *features* auxquelles on veut associer des couleurs puis sélectionner l'option "Feature Color". Choisissez une couleur à associer, sauvegardez puis retournez dans le code Java.

5 Ingénierie de l'application : génération automatique de code

Nous allons à présent définir 3 configurations valides du *feature model*. Elles seront explicitées dans trois fichiers .xml du dossier `configs` :

- `genGraphesSobres.xml` : un générateur de graphes sobres, non orientés, dont les sommets sont en forme d'ellipse. Public cible : les doctorants qui n'ont pas le temps.
- `genFancyGraphes.xml` : un générateur de graphes très détaillés, pouvant avoir des sommets de toutes les formes, des arcs en forme de "v" et proposant toutes les couleurs possibles. Public cible : les doctorants qui veulent faire tenir 3 ans de contributions scientifiques sur une seule diapo.
- `genWeirdGraphes.xml` : un générateur de graphes pouvant avoir des sommets rectangulaires, des flèches standards, mais ne proposant que la couleur verte. Public cible : un doctorant dont il faut respecter les choix.

Pour créer une nouvelle configuration, faire un clic droit sur le dossier `configs` > New > FeatureIDE > Configuration file.

Question 12 : Créez les fichiers pour les trois configurations précédentes. Vérifiez bien que les configurations soient valides.

Afin de générer automatiquement les variantes logicielles correspondant à ces trois configurations, réalisez un clic droit sur le projet > Feature IDE > Product generator. Sélectionnez dans le menu déroulant "all current configurations" puis validez. Le code correspondant aux trois configurations se retrouve alors dans le dossier `products`.

Question 13 : Testez les variantes logicielles ainsi produites. Pour cela, créez un nouveau projet Java : New > Java project. Dans la fenêtre, donnez un nom a votre projet. Puis appuyez sur "next" : dans l'onglet "Sources", cliquez ensuite sur "Link additional source" et sélectionnez un des dossiers du dossier `products` qui

se trouve dans votre *workspace* (cheminToWorkspace/TpLpl2018/products/genWeirdGraphes par exemple). Terminez la création du projet et lancez-le.