Université de Lille

Software Product Lines

2017–2018
Master 2 IAGL

Séance 2 - Gestion de la variabilité

Cette séance a pour objectif de vous faire découvrir de manière concrète le processus d'ingénierie des Lignes de Produits Logiciels (LPL) en vous faisant manipuler la variabilité, tant au niveau du modèle que du code associé. L'ensemble du processus sera réalisé via FeatureIDE, et sera illustré à travers deux exemples distincts. L'objectif n'étant pas d'implémenter la variabilité à proprement parler mais plutôt d'en comprendre les enjeux et mécanismes, le travail demandé est plutôt faible, la séance s'orientant plutôt vers un tutoriel qu'un TP "classique". Ne vous pressez pas, prenez le temps d'analyser et de comprendre ce que vous faites.

Exercice 1 : Prenons l'ascenseur

1. Récupérer le projet existant en cliquant ici. Importez-le dans FeatureIDE. Assurez-vous que le projet s'exécute correctement: *Run As* → *Java Application*. L'interface s'affiche, vous êtes prêts pour la suite.

Sabbath

2. L'ascenseur étant inactif, on va donc lui donner un peu de mouvement en modifiant son comportement. Ouvrez le feature model de votre projet. Comme vous pouvez le constater, il n'y a aucune feature qui implémente l'ascenseur. Ajoutez une feature optionnelle nommée Sabbath. Sabbath est un mode automatisé qui permet de parcourir tous les niveaux de l'ascenseur de bas en haut et de haut en bas sans tenir compte d'aucun élément extérieur. Remarquez l'ajout automatique par FeatureIDE d'un dossier Sabbath dans le répertoire features. C'est dans ce répertoire que sera située l'implémentation de la feature.

3. Récupérez le code implémentant la feature Sabbath en cliquant ici. Créez une classe du même nom dans le répertoire associé à la feature et copiez/collez le code fourni. **Attention**: respectez la hiérarchie indiquée par l'import du package. Votre classe doit donc se situer sous *Sabbath/elevator/core/controller*. C'est une convention adoptée par FeatureIDE, il en sera donc de même pour **TOUS** les fichiers qui seront modifiés par la suite.

4. Complétez l'implémentation de la méthode *calculateNextState*(). L'objectif de la manipulation est de raffiner le code de cette méthode (présente au même emplacement dans le répertoire *src/*) afin d'activer un comportement différent du comportement par défaut lorsque le mode Sabbath sera activé. Sélectionnez la feature Sabbath (*configs/default.xml*) et lancez l'application pour tester votre code. Selon la version de FeatureIDE utilisée, il vous sera peut être nécessaire de créer un nouveau fichier de configuration, *default.xml* pouvant alors être supprimé.

Service

5. On souhaite désormais ajouter une fonctionnalité, optionnelle, appelée Service. Cette fonctionnalité permet aux personnes autorisées d'effectuer de la maintenance sur l'ascenseur, en envoyant ce dernier au rez-de-chaussée si un appel sur le bouton correspondant est effectué. Faites la modification nécessaire au niveau du feature model pour intégrer cette nouvelle fonctionnalité.

6. L'interface graphique changeant légèrement (ajout du bouton et du listener associé), le code de l'UI et les changements qui en découlent vous sont fournis ici. Remplacez votre code par le code des 3 features que vous avez récupéré et qui prend en compte la nouvelle interface graphique. Il vous reste alors à compléter un peu de code pour faire fonctionner le tout. L'unité de contrôle devant savoir si l'ascenseur est en mode service ou non, l'information sur son état doit donc être accessible. Modifiez donc le modèle de façon adéquate, et utilisez cette information dans le contrôleur pour permettre l'activation/désactivation du mode Service.

FIFO

7. On continue dans la configuration de l'ascenseur. Sabbath n'étant pas très optimisée, nous proposons un mode alternatif nommé FIFO. Un ascenseur est donc forcément caractérisé par un mode (au choix) et peut toujours être éventuellement mis en mode service. Faites les changements nécessaires au niveau du feature model (HINT: *Create Feature Above* vous évitera de casser votre modèle).

8. L'ascenseur peut être dirigé vers un étage soit depuis l'intérieur de la cabine, soit en étant appelé de l'extérieur. Lorsque plusieurs appels sont émis simultanément, il doit donc les prioriser: c'est le mode FIFO. Pour visualiser ce mode, il faut donc ajouter les éléments graphiques nécessaires. Ils vous sont fournis ici. Répétez le même procédé qu'à l'étape 6.

9. Configurez votre ascenseur de sorte qu'il utilise FIFO et lancez l'application. Complétez le code de la classe Request pour résoudre les problèmes. Testez votre ascenseur: FIFO ou Sabbath, avec ou sans Service, appels internes et externes "simultanés", etc.

10. Si vous souhaitez vous amuser un peu, vous pouvez essayer de rendre le nombre d'étage paramétrable...

Exercice 2 : Retrouvons leur trace

Comme vous avez du le constater, il peut devenir complexe de gérer la variabilité au niveau du code et de comprendre l'impact de la sélection d'une feature sur celui-ci. FeatureIDE permet de "tracer" visuellement une feature.

1. Faites clic droit sur votre projet, *FeatureIDE* → *Color Scheme Menu* → *Add Color Scheme*. Donnez-lui un nom et validez. Dans votre feature model, faites clic droit sur vos features et attribuez-leur une couleur.

2. Ouvrez le Project Explorer et observez les couleurs. Ouvrez une classe et remarquez l'*Outline* FeatureIDE (à droite en général). Déroulez les différentes méthodes pour voir comment elles sont raffinées, le cas échéant. Ouvrez aussi l'onglet *Collaboration Diagram* (en bas en général) et observez l'impact des features sur les différentes classes: clic droit sur le bouton default, *Show unselected features*. Dans *Show Fields and Methods*, décochez *Fields without Refinements* et *Methods without Refinements* pour une meilleur lisibilité.

Exercice 3 : Mettons un peu d'ordre

Considérez *Hello*, *Hello world*, *Hello beautiful world* et *Hello wonderful world* 4 produits d'une LPL.

1. Récupérez le projet FeatureIDE fourni ici. Complétez le feature model afin de pouvoir construire les 4 produits ci-dessus. Implémentez les features que vous venez d'ajouter en vous inspirant de celles existantes. Testez vos différentes configurations. Que constatez-vous?

2. Vous venez de tomber sur un des problèmes majeurs des LPL: différencier les produits "légaux" (vis-à-vis du feature model, qui correspondent à des configurations valides) des produits "composables" (qui n'ont de sens que lorsque la composition suit un ordre logique). Ouvrez le feature model. Dans l'onglet *Feature Order*, cochez la case vous permettant de définir un ordre de composition personnalisé et modifiez-le. Testez à nouveau votre LPL pour vous assurer qu'elle offre bien les 4 produits décrits précédemment.