Examen terminal d'Ingénierie des Modèles

Durée : 2h – Tous documents (non-électroniques) autorisés.
Le barème est donné à titre indicatif. La partie 2 est indépendante de la partie 1, la question 5 est indépendante de la partie 2.

1 OCL (2 points)

Un diagramme de classes modélisant les PIEs et les parcours vous est donné à la figure 1. On supposera ici pour simplifier qu'un PIE correspond à une liste de modules.

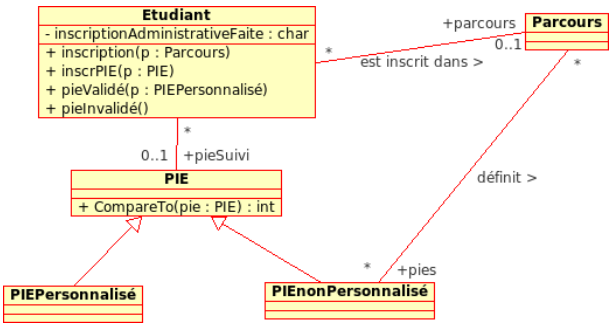

FIG. 1 – Diagramme de classes pour les étudiants, les parcours et les PIEs

Question 1. Exprimer en OCL les contraintes suivantes :

1. Le PIE d'un étudiant est soit personnalisé, soit inclus dans la liste des PIEs du parcours dans lequel est inscrit l'étudiant.
2. Un étudiant ne peut être inscrit à aucun parcours ni suivre aucun PIE s'il n'est pas inscrit administrativement.

2 Machines à états en UML (7 points)

On va maintenant s'intéresser à la machine à états attachée à la classe `Etudiant`. Elle vous est donnée à la figure 2.

2.1 OCL (1 point)

Question 2. Sur le diagramme de la figure 2, il manque deux gardes notées C1 et C2. Exprimer ces deux gardes en OCL. La garde C1 (respectivement C2) doit exprimer que le PIE passé en paramètre de l'inscription ne fait pas partie (respectivement fait partie) de la liste des PIEs du parcours auquel est inscrit l'étudiant.

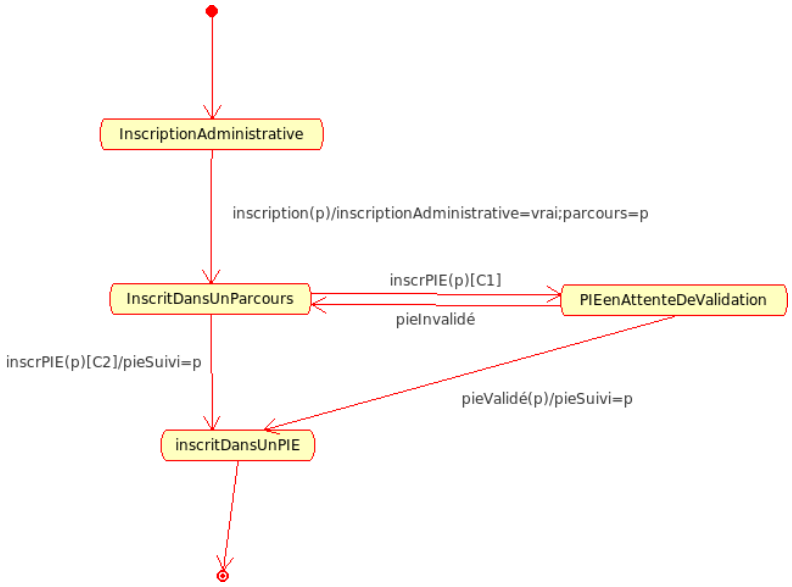

FIG. 2 – Machine à état attachée à la classe Etudiant

2.2 Métamodèle

Un extrait drastiquement simplifié du métamodèle des machines à états UML vous est donné à la figure 3. Vous y reconnaîtrez des éléments du métamodèle qui correspondent à la partie statique d'UML étudiée en cours et en TD, notamment la métaclasse `Class`, `NamedElement`, et `Operation`. Les liens qu'entretiennent ces métaclasses n'ont pas été explicités mais bien sûr ils existent néanmoins.

Une machine à états est un comportement. À ce titre, elle est reliée à un contexte (un classifieur à comportement) qui dans notre cas sera une classe (ce pourrait être aussi un cas d'utilisation par exemple). La machine à états peut accéder à toutes les propriétés et associations de son contexte. Une machine à états est composée de régions. Pour simplifier ici nous ne travaillerons qu'avec des machines à états à une seule région. Une région se compose de sommets (`Vertex`) et de transitions. `Vertex` est une classe abstraite factorisant les états et les pseudo-états. Un vertex est lié à ses transitions sortantes (outgoing) et entrantes (incoming).

Les pseudo-états peuvent être de plusieurs sortes (jonction, branchement, etc) mais la seule sorte qui nous intéresse ici est `initial` qui permet de représenter l'état initial de la machine à états (représenté graphiquement par un disque noir). La métaclasse `State` permet de représenter les états de la machine à états (graphiquement représentés par des rectangles à coins arrondis). `FinalState` est la métaclasse qui permet de représenter les états finaux d'une machine à états (graphiquement représentés par un disque noir entouré d'un cercle)

Chaque transition relie 2 Vertex (`Source` et `Target`). Une transition se compose d'une éventuelle garde (sous forme d'une contrainte) et d'un ensemble de déclencheurs ou gachettes (`trigger`). Chaque déclencheur est lié à l'événement permettant de le déclencher. On ne s'intéresse ici qu'aux événements `CallEvent`, qui représentent les événements liés aux appels de méthode. On transite d'un vertex V1 à un vertex V2 liés par une transition T quand l'un des événements des déclencheurs liés à la transition T est levé et quand la garde est vérifiée. S'il n'y a pas de déclencheurs, on dit que la transition est spontanée, et on transite dès que la garde est vraie. S'il n'y a pas de garde, cela est équivalent à une garde valant vrai. Une transition se compose également d'un éventuel effet, sous forme

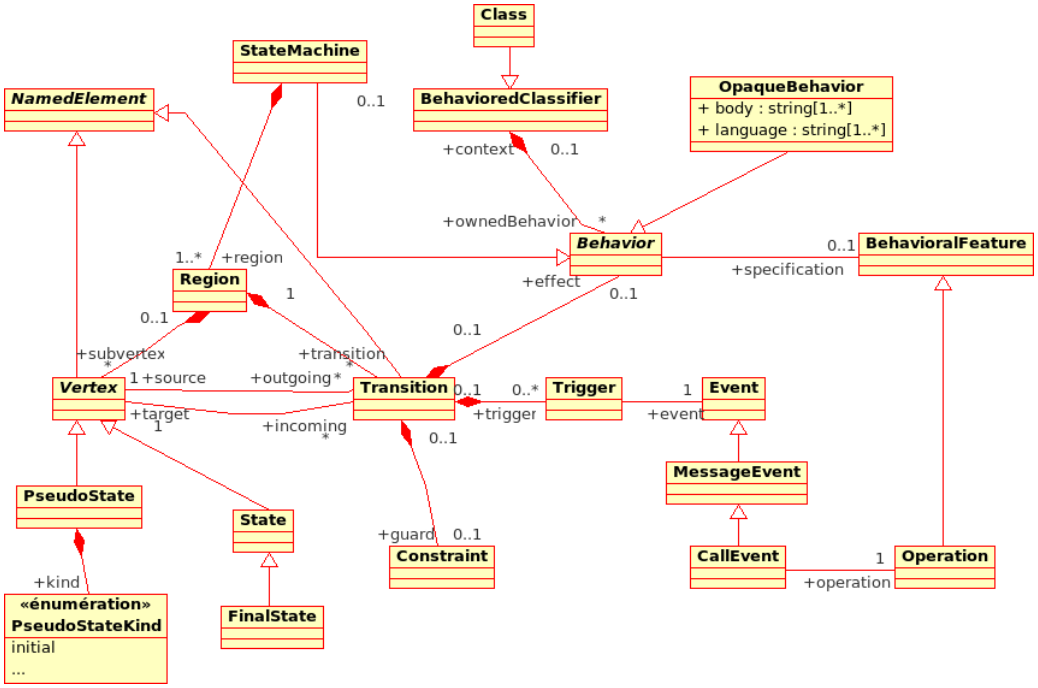

FIG. 3 – Extrait du métamodèle UML pour les machines à états

d'un comportement. Cela correspond au comportement à déclencher lorsque la transition est tirée (i.e. lorsque l'on transite effectivement par la transition). Ce comportement peut être spécifié par une `BehavioralFeature` comme par exemple une opération. Un comportement peut être un comportement opaque, qui est spécifié par 2 chaînes : `body` (la description du comportement) et `language` (le langage utilisé pour la description). On peut donner plusieurs descriptions dans plusieurs langages, d'où les tableaux de chaînes pour `body` et `language`.

2.2.1 OCL (3 points)

Question 3. Écrivez en OCL les contraintes sur le métamodèle des machines à états :

1. Le pseudo-état initial ne peut avoir au plus qu'une transition sortante.
2. Dans un `OpaqueBehavior`, il y a autant d'éléments dans `body` que dans `language`.
3. Une classe ne peut être le contexte que d'une seule machine à états.

2.2.2 Compréhension du métamodèle (3 points)

Question 4. Dessinez sous forme d'instances du métamodèle de la figure 3 (diagramme d'objets) une partie de la machine à états de la figure 2 composée du pseudo-état initial, de l'état `InscriptionAdministrative`, et de l'état `InscritDansUnParcours` ainsi que des deux transitions reliant ces 3 états. L'effet de la transition d'inscription sera représenté par un `OpaqueBehavior` en langage naturel, non spécifié par une `BehavioralFeature`.

3 Transformation de modèles (11 points)

On souhaite écrire une transformation de modèle qui, à partir d'une classe disposant d'une machine à états, génère la structure statique engendrée par l'application du patron

de conception State à la classe. Vous n'avez pas besoin de connaître le patron State pour réussir cet exercice. Lors de l'application du patron de conception à une classe A :

- Une classe Etat est créée, reliée par une association à la classe A : la classe A connaît son état. La classe Etat possède toutes les méthodes dont le comportement dépend de l'état dans lequel se trouve une instance de A. Il s'agit ici de toutes les méthodes présentes comme déclencheurs dans les transitions de la machine à états. Souvent, la classe Etat est laissée abstraite, et la plupart de ses méthodes également. On ne détaillera pas ici cet aspect : tout sera ici laissé concret.
- Pour chaque état de la machine à états, une sous-classe de la classe Etat est créée, qui possède les mêmes méthodes que la classe mère. On ne s'intéressera pas ici au corps de ces méthodes (dans lequel le comportement adéquat doit être implémenté en fonction de l'effet de la transition, et qui doit permettre le changement d'état).

Ainsi, pour l'exemple de la classe Etudiant et de sa machine à états, on obtient le diagramme de classes de la figure 4.

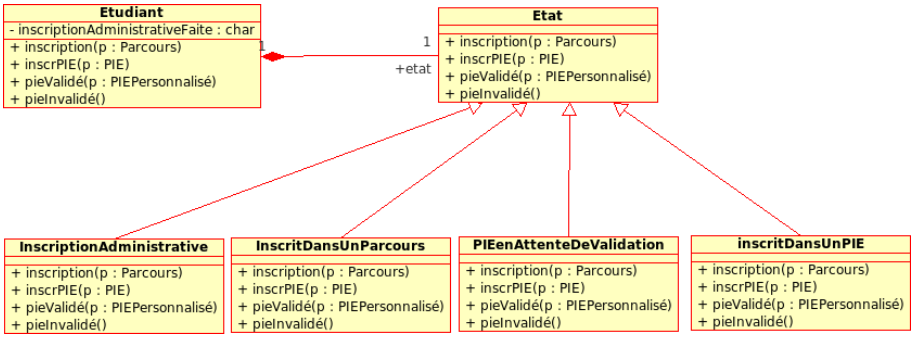

FIG. 4 – Résultat de l'application du patron de conception State à l'exemple des étudiants

Question 5. (3 points) Représentez sous forme d'instance du métamodèle UML (la version vue en cours et en TD) un extrait de la figure 4 composé :

- des classes Etudiant, Etat, et InscriptionAdministrative,
- de la méthode inscription de la classe Etudiant (et seulement cette méthode),
- de la généralisation entre InscriptionAdministrative et Etat,
- et de l'association entre Etudiant et Etat.

On se place dans la suite sur la métaclasse `Class`.

Question 6. (2 points) Soit la méthode `getAllStates` qui retourne une collection d'états attachés à la classe si la classe est reliée à une (et une seule) machine à états et qui échoue sinon. Spécifiez cette méthode en OCL. On suppose par la suite disposer d'une telle méthode.

Question 7. (3 points) Soit la méthode `createStateHierarchy:Class` qui crée la hiérarchie des états issue de l'application du patron de conception *State* et qui retourne la superclasse `Etat`. Implémentez cette méthode dans le langage de votre choix : J [1], Java/EMF, ou Kermeta.

Question 8. (3 points) Soit la méthode `linkWithState(etat : Class)` qui crée l'association entre la classe courante et l'état passé en paramètre. Implémentez cette méthode dans le langage de votre choix : J, Java/EMF, ou Kermeta.

[1] On se basera sur le métamodèle donné à la figure 3 et pas celui d'Objecteering. On appliquera toutefois les règles de navigation spécifiques à J.