Examen (avec documents)

NOM : **Prénom :** **Signature :**

Barème indicatif :

exercice	1	2	3	4	5
points	2	5	3	5	5

Exercice 1 Soit le programme ci-dessous :

```
1  void foo(boolean a, boolean b) {
2     if (a && b) {
3        out.println("ok␣if");
4     }
5     while (a || b) {
6        out.println("ok␣while");
7        break;
8     }
9     out.println("fin");
10 }
```

1.1 Donner les éléments à couvrir pour chacun des critères suivants : instructions (I), décisions (D) et conditions (C). Vous rappellerez en 2 ou 3 phrases le principe de chaque critère.

1.2 Donner les jeux de tests du programme couvrant les critères et illustrer qu'ils sont différents.

Exercice 2 Dans cet exercice, nous nous intéressons à une partie de la transformation ATL qui a certainement été écrite pendant le BE. Son texte est donné au listing 1.

2.1 Expliquer chaque élément ATL présent dans le code du listing 1 correspondant à cette transformation. Il est conseillé de répondre directement sur le sujet.

2.2 Dessiner la structure des modèles d'entrée et de sortie manipulés par cette transformation. On ne devra faire apparaître que les éléments qui sont explicitement présents dans la transformation. On les représentera en s'inspirant de la syntaxe concrète graphique utilisée par l'OMG pour les modèle MOF ou par l'éditeur Ecore d'Eclipse.

Diagrammes à état d'UML (version simplifiée)

Dans ces exercices, nous nous intéressons à une version simplifiée des diagrammes à état d'UML 2 que nous appellerons DET pour Diagramme États-Transitions.

Exercice 3 : Méta-modèle DET

La figure 1 présente le méta-modèle de DET dans le formalisme graphique Ecore inspiré du diagramme de classe UML.

La figure 2 donne un exemple de diagramme à état pour un feu clignotant qui est initialement dans l'état « allumé » et qui passe alternativement à l'état « éteint » puis « allumé » sur la réception d'un événement « TOP ».

3.1 Les deux références nommées « source » et « sortantes » entre les méta-classes « État » et « Transition » sont dites opposées (*opposite* en Ecore). Il en est de même pour « cible » et « entrantes ». Indiquer ce que signifie cette notion.

Listing 1 – Extrait de la transformation SimplePDL vers PetriNet

```
1  module SimplePDL2PetriNet;
2  create OUT : PetriNet from IN : pdl;
3
4  ...
5
6  rule Parameter2PetriNet {
7     from p : pdl!Parameter
8     to
9        a_r2s : PetriNet!Arc (
10          kind <- #normal
11          , nbJetons <- 1
12          , source <- p.ressource
13          , cible <- thisModule.resolveTemp(p.workDefinition,'t_start')
14       )
15       , a_f2r : PetriNet!Arc (
16          kind <- #normal
17          , nbJetons <- 1
18          , source <- thisModule.resolveTemp(p.workDefinition,'t_finish')
19          , cible <- p.ressource
20       )
21 }
```

3.2 Montrer que le diagramme à état de la figure 2 est bien conforme au méta-modèle de la figure 1. On pourra directement répondre sur le sujet en explicitant les liens entre le modèle et le méta-modèle.

3.3 Le méta-modèle de la figure 1 a trois références de type composition (*containment* en Ecore). Indiquer comment il aurait été possible de définir une seule référence de type composition tout en gardant une notion de composition entre Diagramme d'une part et État, Transition et Événement d'autre part.

Exercice 4 : Contraintes OCL

Nous allons maintenant définir quelques contraintes OCL.

4.1 Écrire une contrainte OCL qui vérifie la propriété suivante : « tout diagramme à état a au moins un état ».

4.2 Écrire une contrainte OCL qui vérifie la propriété suivante : « si un diagramme à état possède plusieurs états alors chacun de ses états doit être relié par une transition à un autre état ».

4.3 Écrire une contrainte OCL qui vérifie la propriété suivante : « tout diagramme à état est déterministe ». Un diagramme est déterministe si pour tout événement et tout état, il y a au plus une transition sortante de cet état déclenchée par cet événement.

4.4 Indiquer si, étant donné le méta-modèle DET donné, il est pertinent de définir les contraintes précédentes sous forme de contraintes OCL.

Indiquer quel est l'intérêt d'OCL.

Exercice 5 : Modèle d'erreur

On peut utiliser ATL pour vérifier des propriétés sur un modèle. Au lieu de se contenter de renvoyer vrai ou faux, on peut construire un modèle d'erreur qui pour chaque élément concerné du modèle va indiquer si la propriété est vérifiée ou non. Dans la négative, une explication pourra être fournie (chaîne de caractères).

Ainsi, un modèle d'erreur contient plusieurs verdicts. Un verdict correspond à une propriété (décrite de façon informelle par une chaîne de caractères, par exemple « déterminisme »), un résultat (vrai ou faux) et, dans le cas où le résultat est faux une explication du problème détecté (chaîne de caractères).

Dans la suite de cet exercice, on ne s'intéresse qu'à la propriété « déterminisme » (voir question 4.3).

Le modèle DET de la figure 2 est un diagramme déterministe car ses deux états sont déterministes. En revanche, le modèle DET de la figure 3 est non déterministe à cause de l'état E1 sur l'événement b et l'état E3 sur d.

5.1 Proposer un méta-modèle d'erreur en Ecore.

5.2 Indiquer en français, sans les formaliser en OCL, les contraintes qu'il faudrait ajouter pour s'assurer de la validité du modèle d'erreur.

5.3 Donner le modèle d'erreur correspondant au modèle DET de la figure 3. On pourra utiliser un formalisme proche du diagramme d'objet d'UML pour le représenter ou une description arborescente comme le fait l'éditeur arborescent engendré avec EMF.

5.4 Écrire une transformation ATL qui vérifie qu'un modèle DET est bien déterministe. Pour chaque état d'un modèle DET, on devra indiquer s'il provoque ou non un indéterminisme.

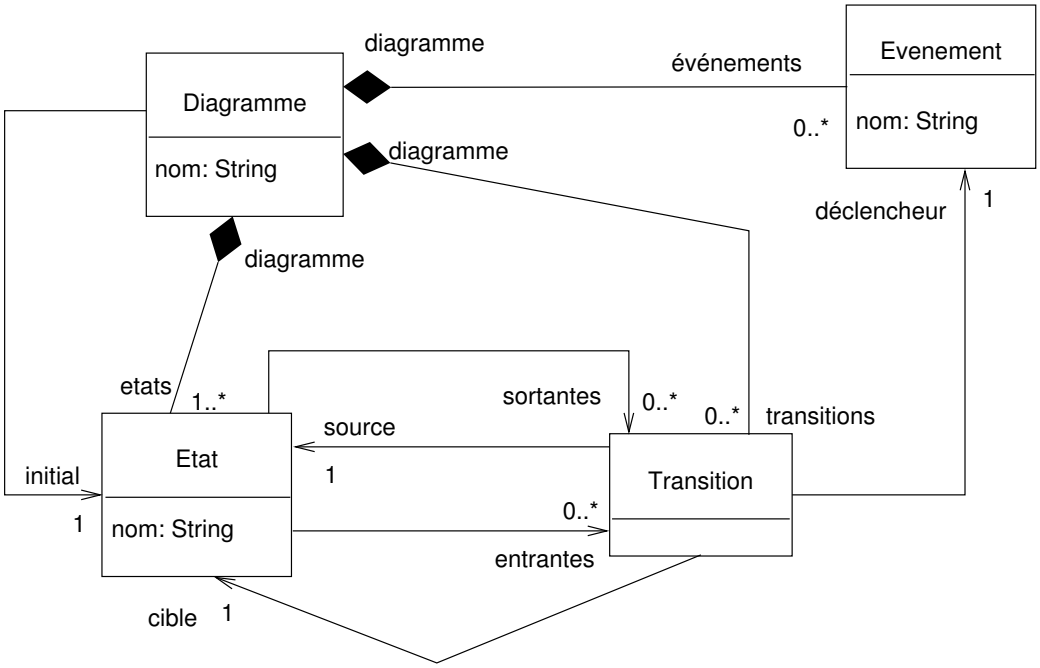

FIGURE 1 – Méta-modèle de DET

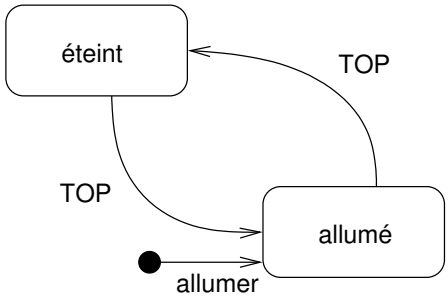

FIGURE 2 – Diagramme à état d'un feu clignotant

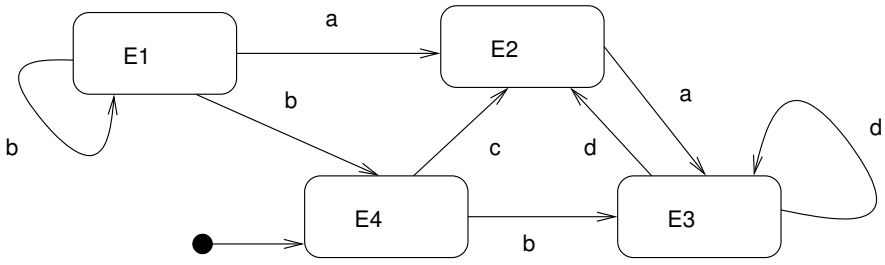

FIGURE 3 – Exemple de modèle DET non déterministe