# Transformation de modèles

## 1 Métamodèle pour les machines à états UML

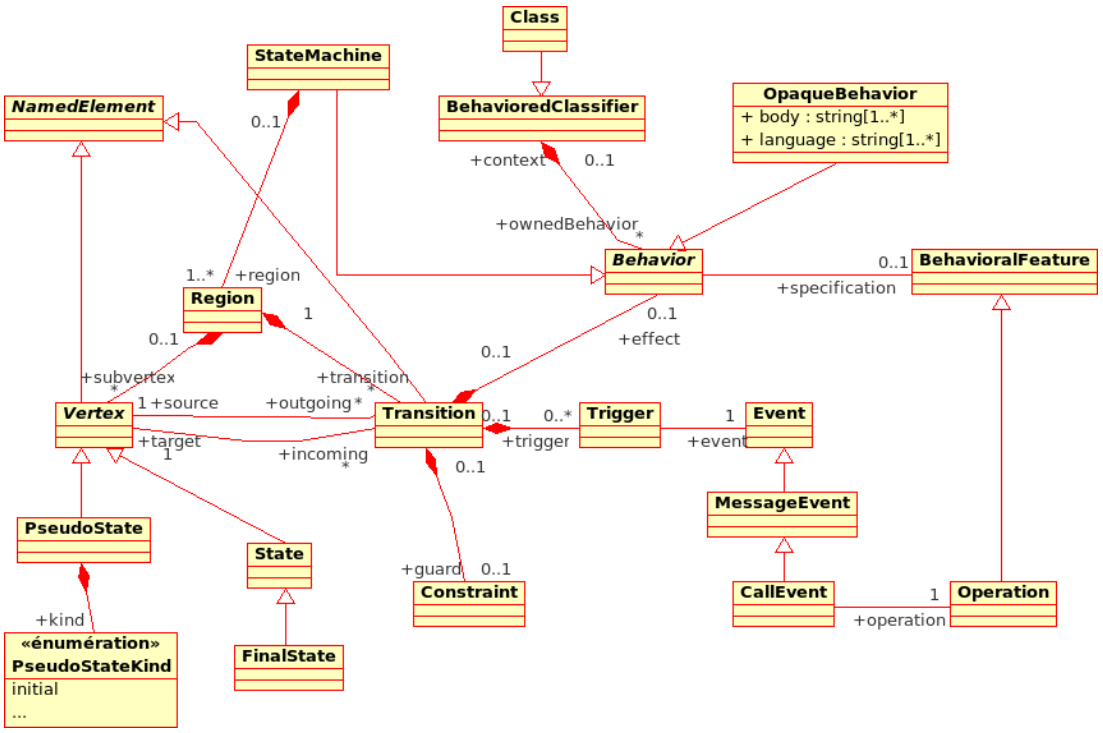


FIGURE 1 – Extrait du métamodèle UML pour les machines à états

Un extrait drastiquement simplifié du métamodèle des machines à états UML vous est donné à la figure 1. Vous y reconnaîtrez des éléments du métamodèle qui correspondent à la partie statique d'UML déjà rencontrés lors du précédent TP, notamment les métaclasses `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. 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 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`.

**Question 1.** Dans ce métamodèle, certaines associations sont des compositions, d'autres pas. Expliquez comment les liens de composition doivent être répartis sur le métamodèle de manière à ce que celui-ci soit pertinent.

**Question 2.** Montrez sous forme d'instance du méta-métamodèle Ecore la métaclasse Region, la métaclasse Transition, et la composition entre ces deux métaclasses.

**Question 3.** Soit la machine à états de la figure 2, qui est un comportement pour une classe de nom Cl, et possédant une méthode m1 et une méthode m2, toutes 2 sans paramètres ni type de retour. Donnez la représentation de cette machine à états en syntaxe abstraite sous forme de diagramme d'instances (en représentant les éléments du modèle sous forme d'instances d'éléments du métamodèle), ainsi que la représentation de la classe Cl et des méthodes m1 et m2. On représentera la transition sous forme d'OpaqueBehavior, et la transition sera spécifiée par la méthode m2. On supposera disposer dans la méta-classe Constraint d'un attribut `text:String` pour reporter le texte de la contrainte.

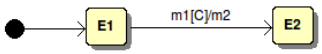


FIGURE 2 – Une petite machine à états

## 2 Transformations de modèles

Pour simplifier, dans cette partie, nous ne travaillerons qu'avec des machines à états à une seule région.

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 abstraite nommée EtatA est créée, reliée par une association à la classe A : la classe A connaît son état. La classe EtatA possède toutes les signatures de 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. Ces méthodes sont abstraites dans EtatA.
- Pour chaque état de la machine à états, une sous-classe de la classe EtatA 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).

**Question 4.** Ecrivez une méthode prenant en paramètre une classe, et qui retourne les machines à état la décrivant.

**Question 5.** Ecrivez une méthode prenant en paramètre une machine à états et qui vérifie qu'elle est correctement formée pour notre exercice, c'est-à-dire qu'elle ne contient qu'une seule région.

**Question 6.** Ecrivez une méthode prenant en paramètre une machine à état bien formée pour notre exercice, et qui retourne la liste des états la composant.

**Question 7.** Ecrivez une méthode prenant en paramètre une machine à état bien formée pour notre exercice, et qui retourne la liste des opérations se trouvant comme trigger dans la machine à état.

**Question 8.** Ecrivez une méthode qui appliquerait le patron State à une classe donnée.