Comprendre la nature exécutable des modèles

Eric Cariou, Olivier le Goaër, Franck Barbier

Université de Pau / LIUPPA

4ème Conférence en IngénieriE du Logiciel (CIEL) 09 Juin 2015

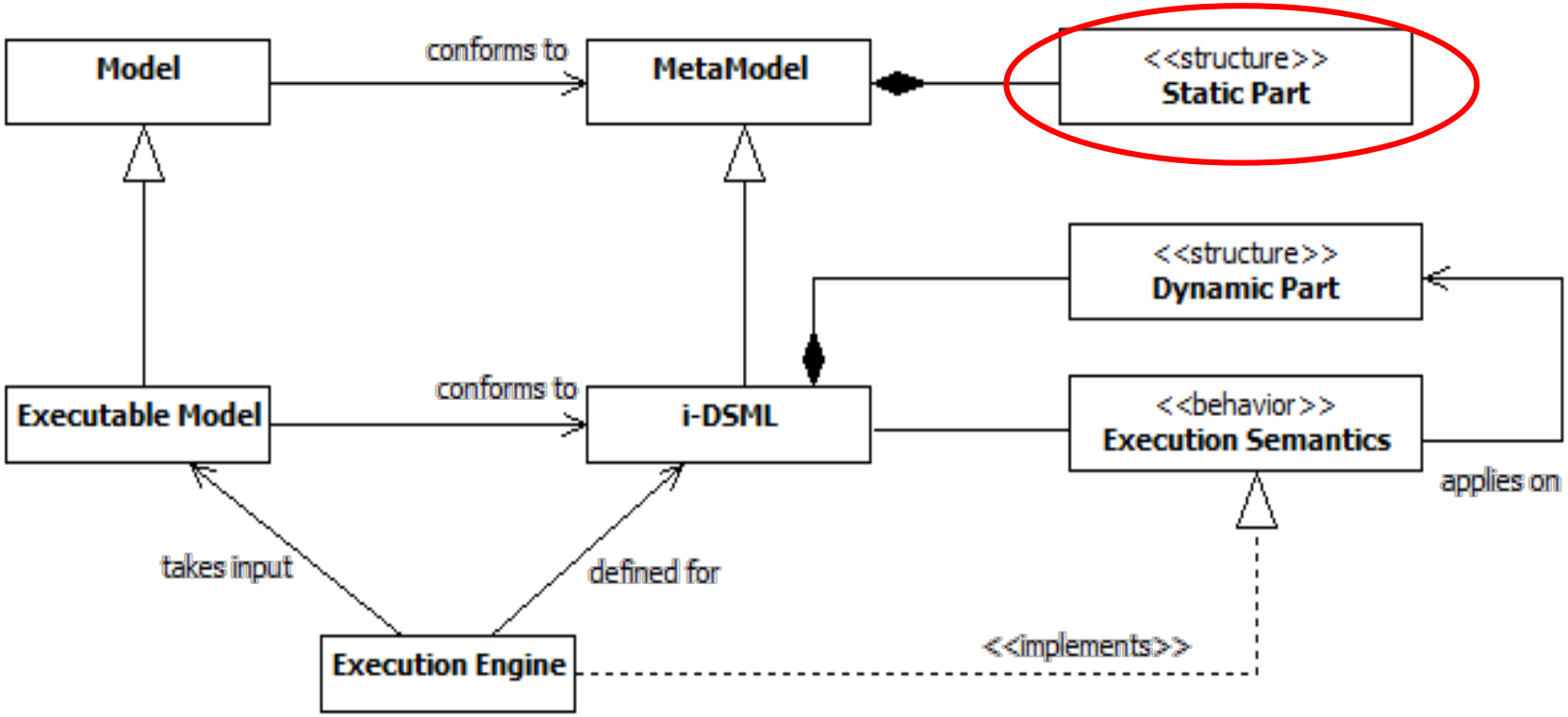
Introduction

- Modèles exécutables
  - Ex : machines à états, workflows, réseaux de Petri...
- Intérêts
  - Simulation pendant la conception
    - Détection anticipée de problèmes
  - L'exécution du modèle est le système
    - Supprime l'étape d'implémentation
- En Ingénierie Dirigée par les Modèles (IDM)
  - Création de ses propres modèles exécutables
  - i-DSML : *interpreted-Domain Specific Modelling Language*

Introduction

- Comment définir un i-DSML ?
  - On sait faire depuis longtemps
    - Méta-modèle + sémantique d'exécution
    - Moteur qui interprète le modèle
      - Ou on génère du code vers une plate-forme dédiée
      - Ex : librairie PauWare en Java implémentant la sémantique des machines à états UML
- Dans cet article, on tente de répondre à une autre question
  - Peut-on savoir si un DSML donné est un i-DSML ?
  - Quelle est la nature exécutable des modèles ?

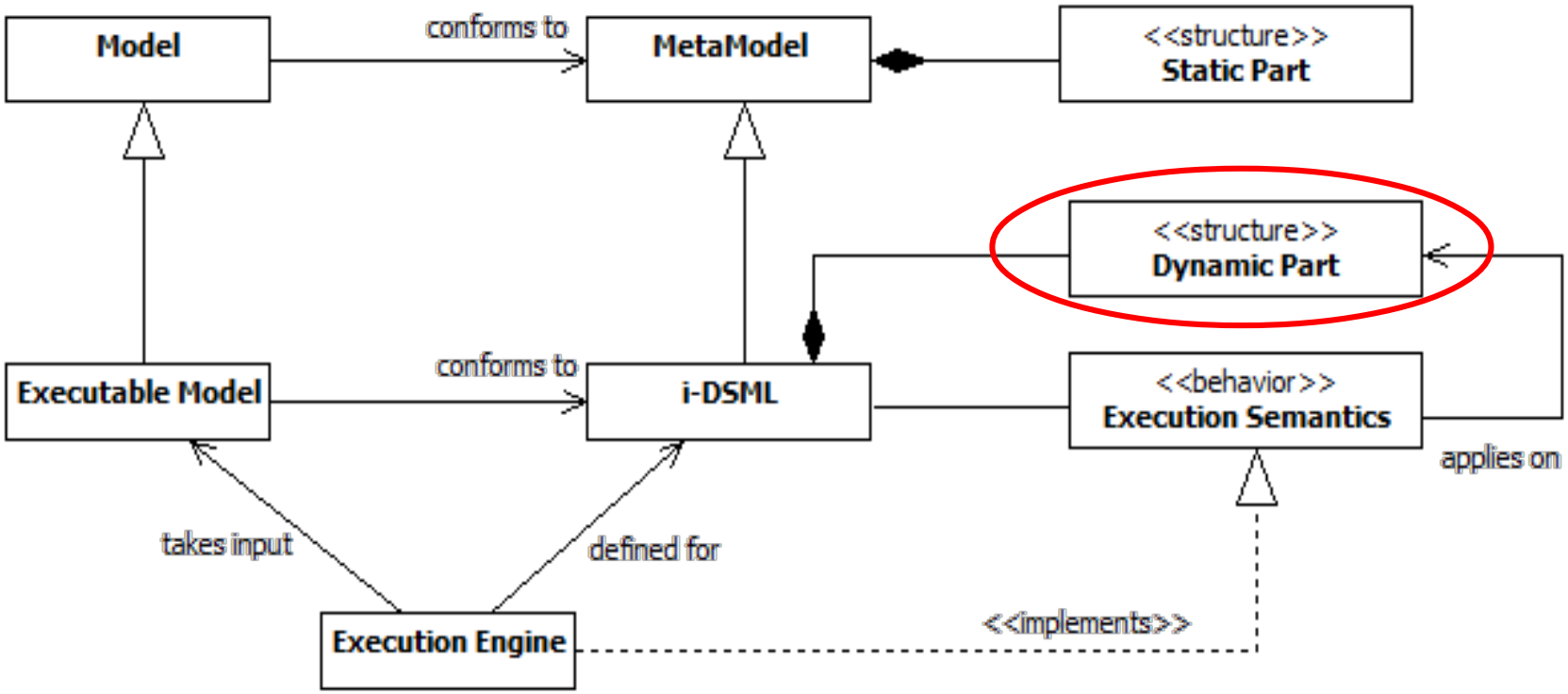
Caractérisation des i-DSML*



- Partie statique du méta-modèle
  - Ex : états et transitions d'une machine à états

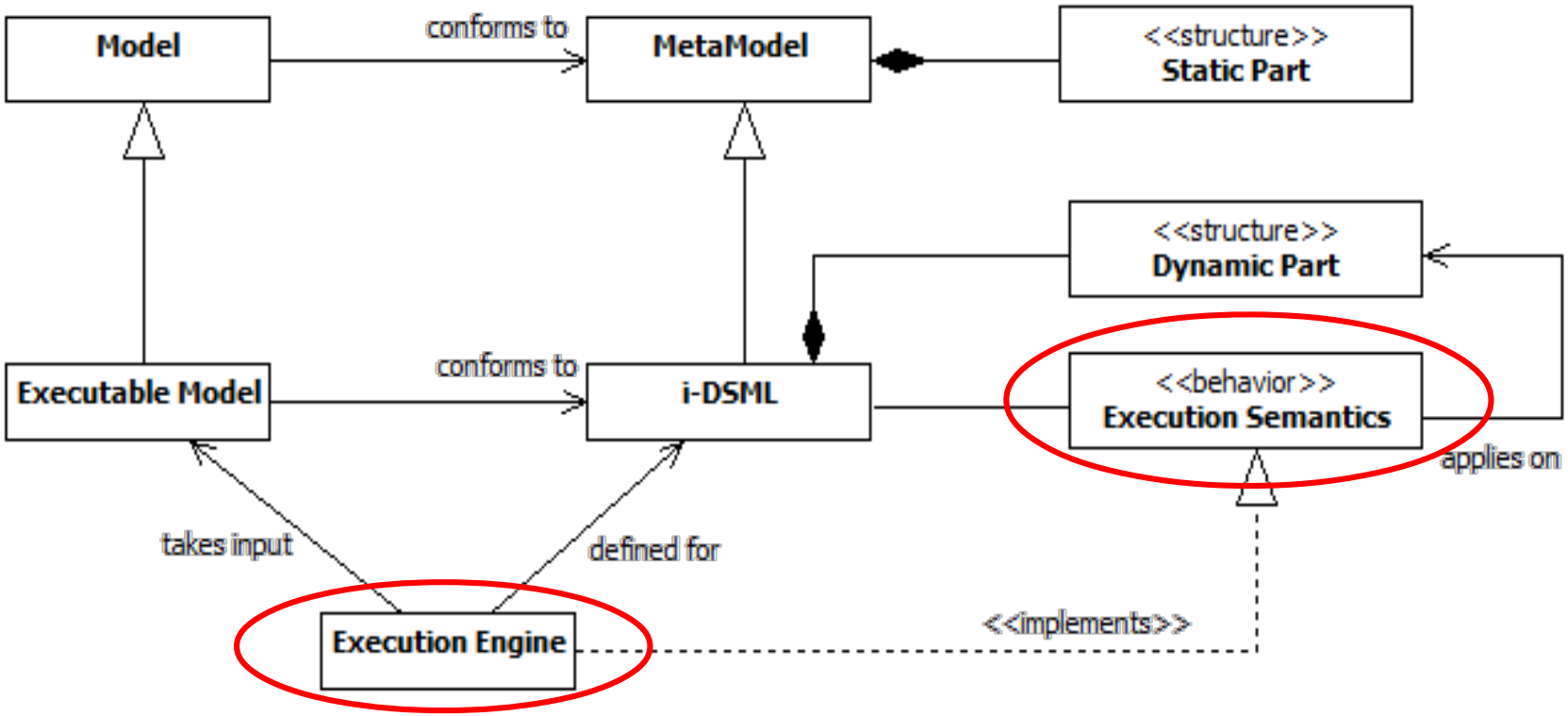
  (*Caractérisation reprise de : Cariou *et al.*, characterization of adaptable interpreted-DSML, ECMFA 2013)

Caractérisation des i-DSML



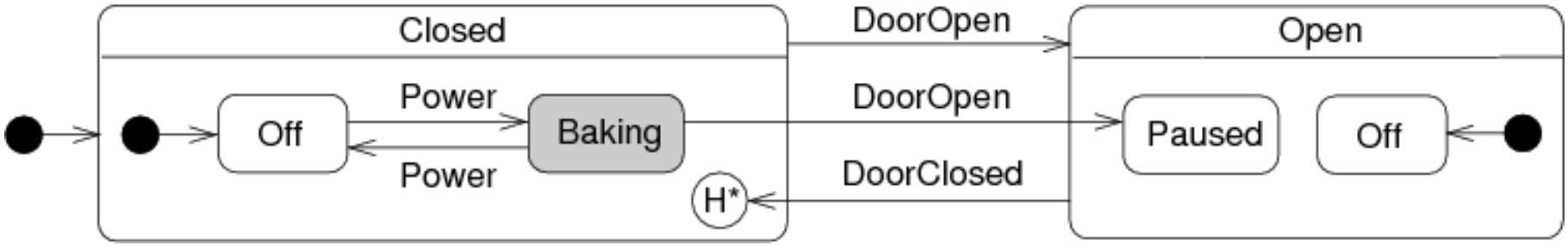
- Partie dynamique du méta-modèle
  - Définit l'état courant du modèle pendant son exécution
    - Ex : les états actifs d'une machine à états
  - Peut être éventuellement gérée en dehors du modèle

Caractérisation des i-DSML



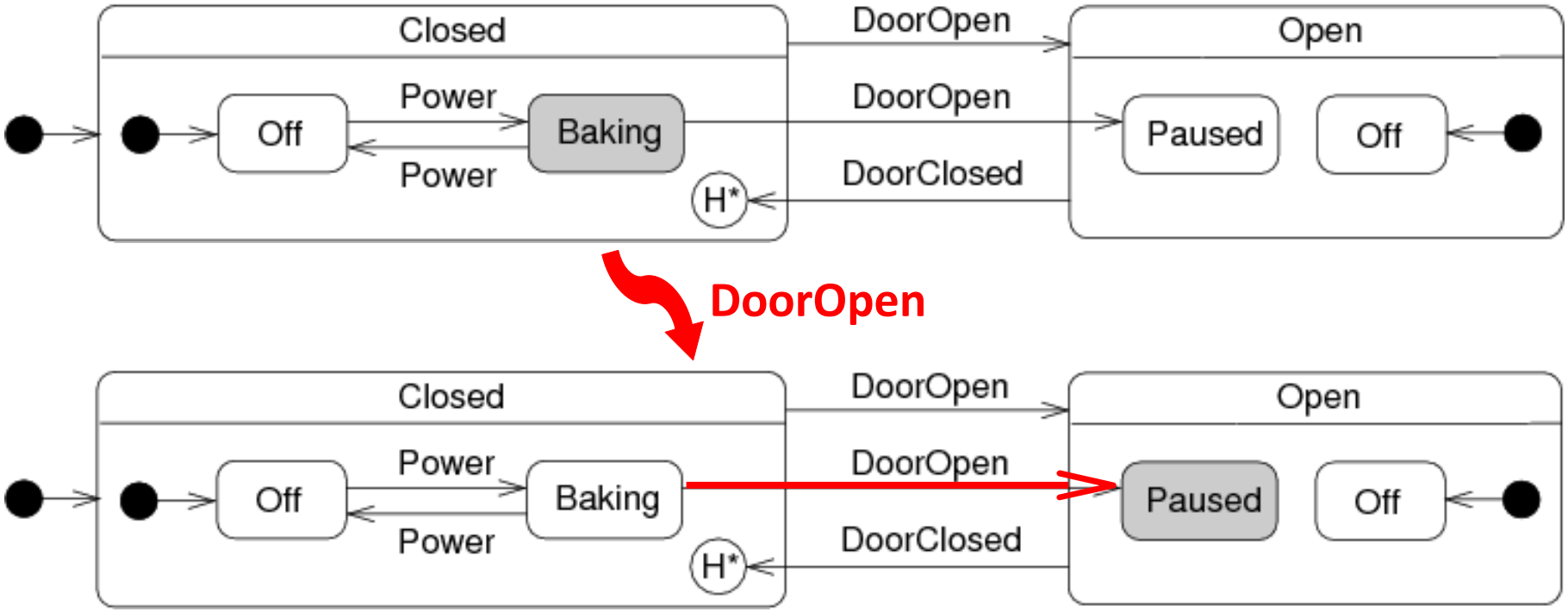
- Sémantique d'exécution
  - Définit comment l'état du modèle évolue pendant son exécution
    - Ex : comment suivre les transitions en fonction des événements
  - Implémentée par le moteur d'exécution qui interprète le modèle

Caractérisation des i-DSML



- Ex : machine à états d'un micro-onde
  - État actif courant : « Baking » de « Closed » (porte fermée)

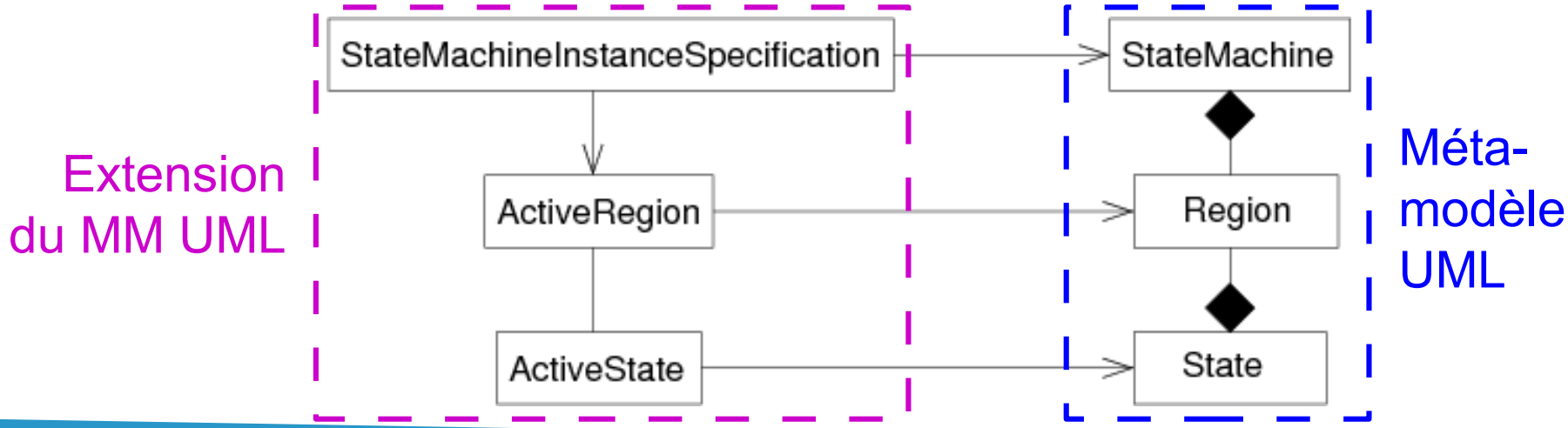
Caractérisation des i-DSML



- Déclenchement d'une transition à l'occurrence d'un événement
  - Nouvel état courant du modèle défini selon la sémantique d'exécution
  - Nouveau pas d'exécution
  - Exécution = série de pas d'exécutions

Paradoxe d'UML

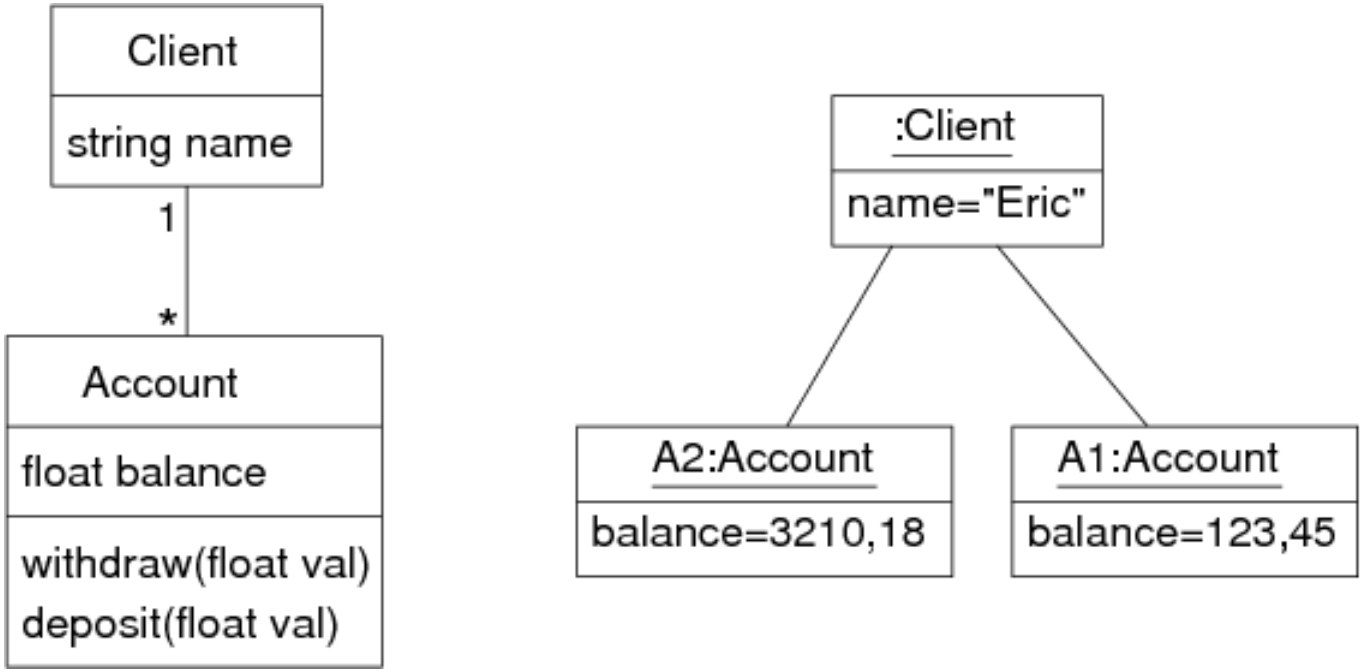
- UML définit de nombreux diagrammes a priori exécutables
  - Diagrammes comportementaux : séquence, activités, machines à états, …
  - Mais le méta-modèle ne définit de partie dynamique pour aucun de ces diagrammes
  - Proposition d'une partie dynamique pour les machines à états (Cariou *et al.*, contracts for model execution verification, ECMFA 2011)



Paradoxe d'UML

- Diagramme de classes : non exécutable
- Mais quelque chose peut jouer le rôle de sa partie dynamique
  - Le diagramme d'objets
    - Instances de classes avec valeurs des attributs et liens entre instances
  - Extrait de la spécification UML de l'OMG
    - « *A static object diagram is an instance of a class diagram ; it shows a* ***snapshot*** *of the detailed* ***state*** *of a system* ***at a point in time*** »
    - C'est presque la définition exacte du but d'une partie dynamique

Paradoxe d'UML



- Comment faire évoluer l'état courant ?
  - Ex : comment, quand ou pourquoi modifier le solde du compte « A2 » ?
  - Non déterminable, ne sait pas comment exécuter les opérations
  - Ne sait pas non plus comment initialiser les objets

Premier critère : pas d'exécution

- État courant seul ne suffit pas
- Doit être capable de calculer des pas d'exécution
  - Incluant un (éventuel) pas initial
  - Permet de définir une sémantique d'exécution
- (Aide à la) définition de pas d'exécution
  - Idées de « évoluer », « suivre », « avancer », « faire » ...
  - Explicite : éléments dédiés à cela
    - Ex : transitions des machines à états ou des réseaux de Petri
  - Implicite
    - Ex : modèle de règles SVBR
    - Moteur cherche parmi toutes les règles lesquelles exécuter

Second critère : comportement

- Système réalise des actions métier
  - Ascenseur ouvre/ferme sa porte, enroule/déroule câble pour monter ou descendre
  - Système de réservation en ligne appelle des web services de compagnies aériennes, fait des requêtes sur des SGBD
- Questions
  - Qui décide de quand ou pourquoi appeler ces actions métier ?
  - Qui réifie le comportement du système qui tourne ?

Second critère : comportement

- Supposons que le système utilise un modèle au runtime
  - Si ce modèle définit le comportement du système, c'est un modèle exécutable (et exécuté)
- Exemples
  - Machine à états qui commande l'ascenseur ✓
  - Orchestration BPEL qui appelle les Web services ✓
  - Modèle qui stocke l'état de l'ascenseur (heures d'utilisation, usure des pièces, ...) dans l'esprit des *models@run.time* ✘
    - Sera utilisé/modifié par les actions métiers mais ne les commande pas
- Le système qui prend un modèle exécutable réifiant un comportement est un moteur d'exécution

Quelques DSML

- En se basant sur les spécifications de l'OMG, étude de quelques DSML/diagrammes

DSML	Comportement du système	État Courant	Pas d'exécution	**Exécutable ?**
BPEL/BPMN	Oui	Externe	Explicite	**Oui**
Use cases	Oui	Interne	Aucun	**Non**
Diag. classes	Non	Interne	Aucun	**Non**
Machines à états	Oui	Externe	Explicite	**Oui**
SBVR	Oui	Externe	Implicite	**Oui**
Diag. composants	Non	Aucun	Aucun	**Non**

Conclusion

- Proposition de deux critères définissant la nature exécutable des modèles
  - La possibilité de réaliser des pas d'exécution
    - Définition possible d'une sémantique d'exécution
  - Le comportement du système est réifié dans le modèle
    - Le système *est* le modèle exécuté
- Ces deux critères sont nécessaires mais peut-être pas suffisants
  - Étude à prolonger ...