On the executable nature of models

Eric Cariou, Olivier le Goaër, Franck Barbier

University of Pau / LIUPPA, France

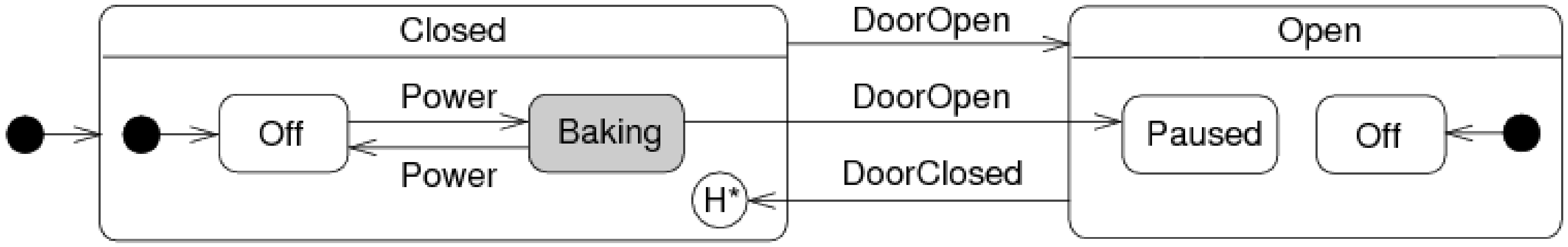
2nd International Workshop on Executable Modeling at MoDELS 2016

October 3, 2016

Introduction

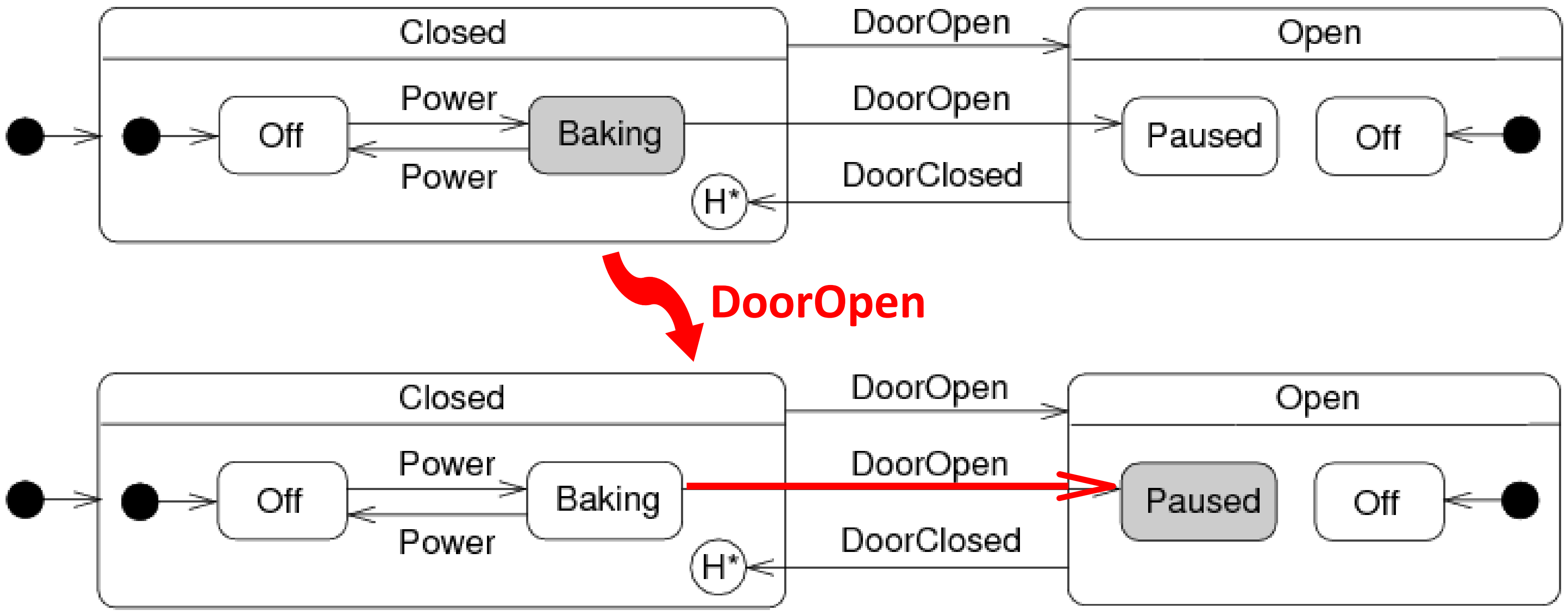
- Executable models
  - Ex: state machines, activity diagrams, Petri nets, ...
- In a MDE context
  - Definition of dedicated languages of executable models
    - i/x-DSML : interpreted/executable-Domain Specific Modelling Language
  - How to build an i-DSML?
    - It is well-known
- In this paper, we try to answer symmetrical questions
  - If facing a model, can we know if it could be executable?
    - How knowing that its DSML can actually be an i-DSML?
  - What is the executable nature of models?
    - Two main criteria found

Elements of an i-DSML



- Metamodel: two kinds of model elements
  - Static: the structural contents of the model
    - State, transition ... (allowing to define the microwave oven state machine)
  - Dynamic: to store the current state of the model under execution
    - Active state of the state machine (here the ''baking'' state)
    - This part is not always embedded in the model, not defined in the MM
      - Has to be managed internally by the execution engine

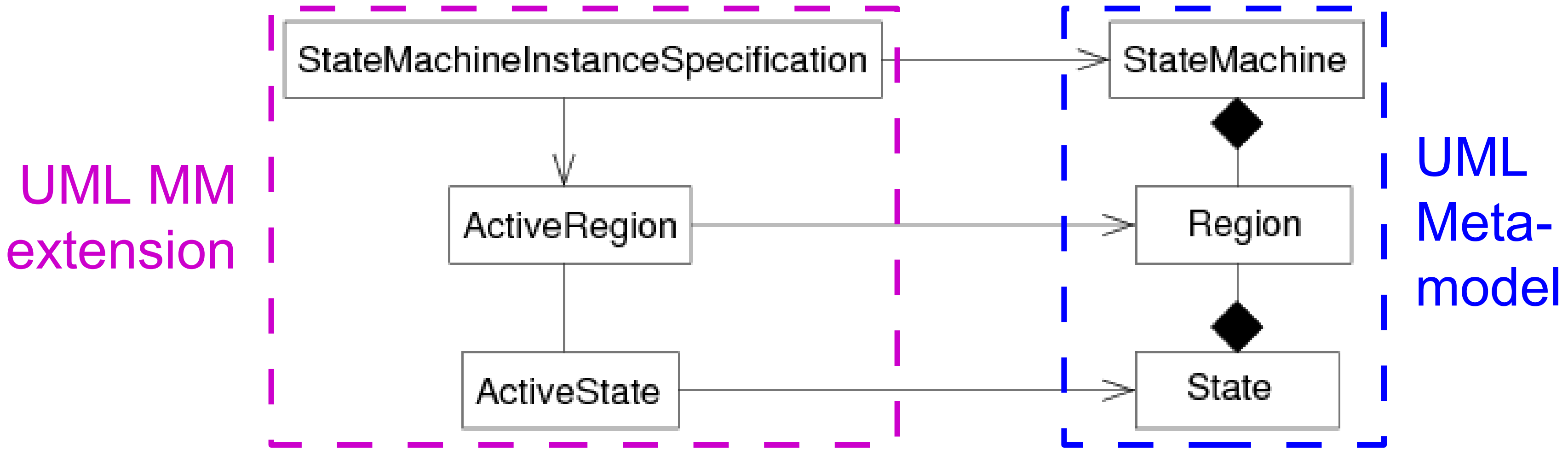
Elements of an i-DSML



- <u>Execution semantics</u>: defines how the dynamic part is evolving in time
  - For state machine: if there is a transition to follow when an event occurs
  - A model evolution = carrying out an <u>execution step</u>
  - Execution semantics implemented by an execution engine

UML paradox

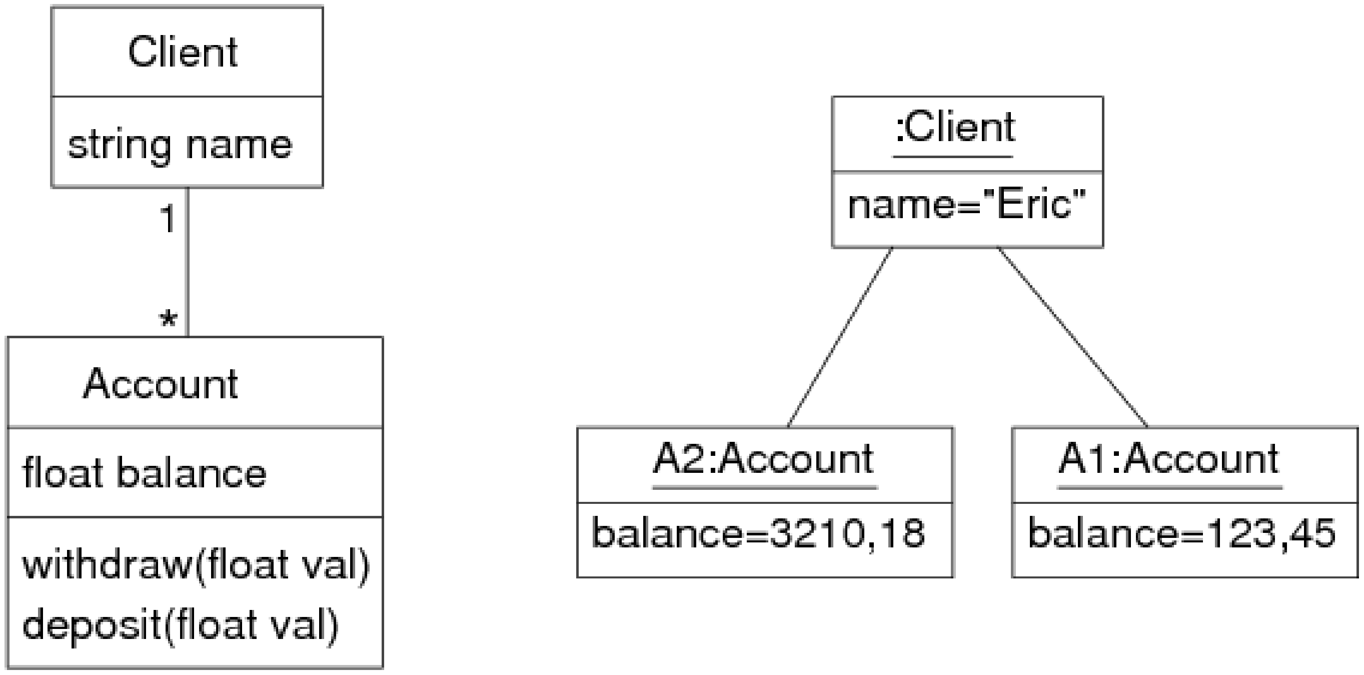
- The UML specification defines several executable models
  - Behavorial diagrams: sequence, activity, state machine ...
  - None of them has a dynamic part defined in the UML metamodel
- Proposition of a dynamic part for OMG's UML state machines (Cariou *et al.*, contracts for model execution verification, ECMFA 2011)



UML paradox

- A class diagram on its own is not executable
  (without associated behavioral diagrams or fUML specification)
- But has a kind of … dynamic part
  - The object diagram
    - Instances of classes with values for attributes and relations between instances
  - Excerpt of the OMG's UML specification
    - *"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****"*
    - It is almost the exact definition of the purpose of a dynamic part!

UML paradox



- How making evolving the current state (the object diagram)?
  - Ex: why and when modifying the balance value of the A2 account?
  - Non determinable, we do not know how to execute the operations

First criterion: execution step

- As just seen, having a current state is not sufficient
- Must be able to compute execution steps
  - Including a potential initial state
  - Enables to define an execution semantics
- (Help to) the definition of execution steps
  - "evolution", "following", "moving forward", "carrying out" or related concepts make sense for the model
  - Explicit: dedicated elements
    - Ex: transitions for state machines or Petri nets (graphically $\longrightarrow$)
  - Implicit
    - Ex: model of business rules in SVBR
    - Engine is responsible for finding and executing the required rules

Second criterion: behavior

- A system implements business actions
  - An elevator opens/closes its door, winds/unwinds cables for reaching a given floor
  - A travel booking system inserts customers data into database or call Web services provided by air transport companies
- Questions
  - Who/what decides when or why calling a given business action?
  - Who is reifying the behavior of the running system?

Second criterion: behavior

- Let suppose that the system is using a model at runtime
  - If this model defines the behavior of the system, it is an executable model
- Examples
  - A state machine controling the elevator ✓
  - A BPEL orchestration calling Web services ✓
  - A model which stores information on the elevator state (daily uses, state of wear parts, …) in the spirit of *models@run.time* ✗
    - Will be used/modified by business actions but does not control them
- A system taking as entry a model refying the system behavior is an execution engine

Some DSML

- Based on the OMG specifications, classification of some DSMLs/diagrams

DSML	Behavior of the system	Current state	Execution step	**Executable?**
BPEL/BPMN	Yes	External	Explicit	**Yes**
Use cases	Yes	Internal	None*	**No**
Class diagram	No	Internal	None	**No**
State machine	Yes	External	Explicit	**Yes**
SBVR	Yes	External	Implicit	**Yes**
Component diag.	No	Internal	None	**No**

* *With the common use of use cases with informal textual description*

Conclusion

- Proposition of two criteria defining the executable nature of models
  - The capability of carrying out execution steps
    - Possible definition of an execution semantics
  - The behavior of the system is reified whithin the model
    - The system *is* the executed model
- These two criretia are required but not necesseraly sufficient
  - Study to extend...