A software development process based on UML state machines

Eric Cariou[1], Léa Brunschwig[2], Olivier Le Goaer[1], Franck Barbier[1]

[1] Université de Pau / LIUPPA, France
[2] Universidad Autónoma de Madrid, Spain

MegaM@Rt2

ECSEL Joint Undertaking
European Commission
Horizon 2020
European Union funding
for Research & Innovation

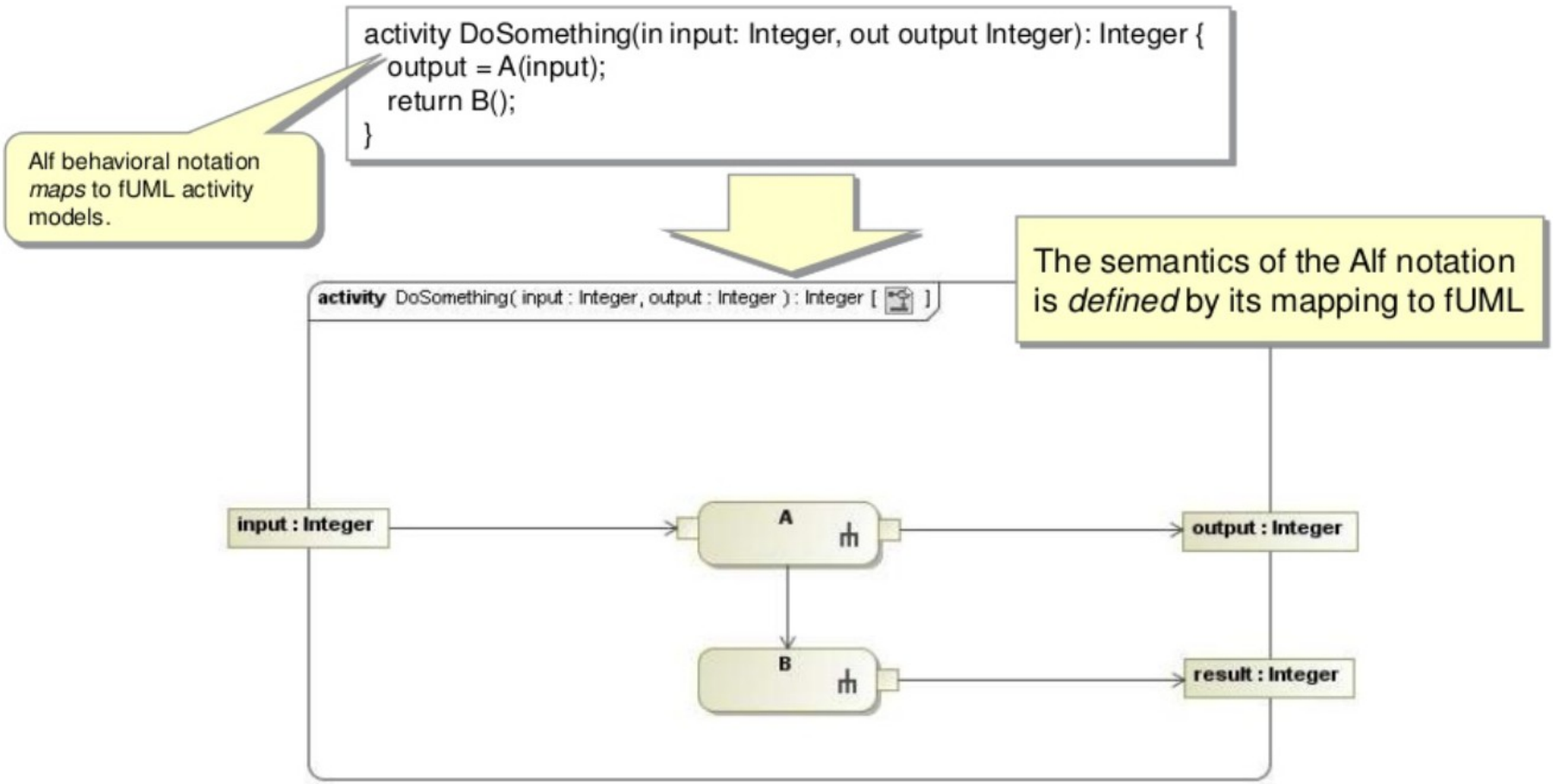
The 4th International Conference on Advanced Aspects of Software Engineering (ICAASE'20)

Nov 2020

Introduction

- 20 years ago, the OMG launched the MDA initiative
  - Full-model approach for software development
  - UML-based models embed the complete application definition
    - Code generation for specific implementation technologies
  - Limits
    - UML, OCL and MOF models are not sufficiently precise to define detailed or algorithmic parts of an application
  - In 2011: fUML specification v1.0
    - « *executable subset of standard UML [...] to define, in an operational style, the structural and behavioral semantics of systems* »
    - Specific activity diagrams with equivalent textual syntax ALF

Introduction



Source: Programming in UML: An Introduction to fUML and Alf, Ed Seidewitz, 2011

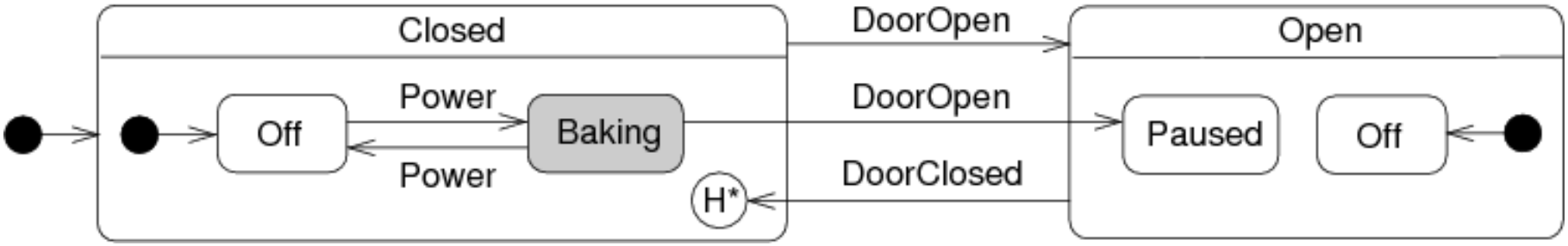
Introduction

- Thanks to fUML (and other specifications of the OMG)
  - Abstract code is added in the UML models
    - Interpretable or compilable towards implementation platforms
  - Implemented in tools such as Papyrus (EMF/Eclipse-based)
- Open questions
  - Is it efficient and even feasible to define a complete application at the specification/model level?
    - Does the approach scale for equivalent of millions LoC?
  - How to manage existing libraries, legacy code or specific IDE?
    - How to embed fUML spec. within an Android Studio project?

Software development process

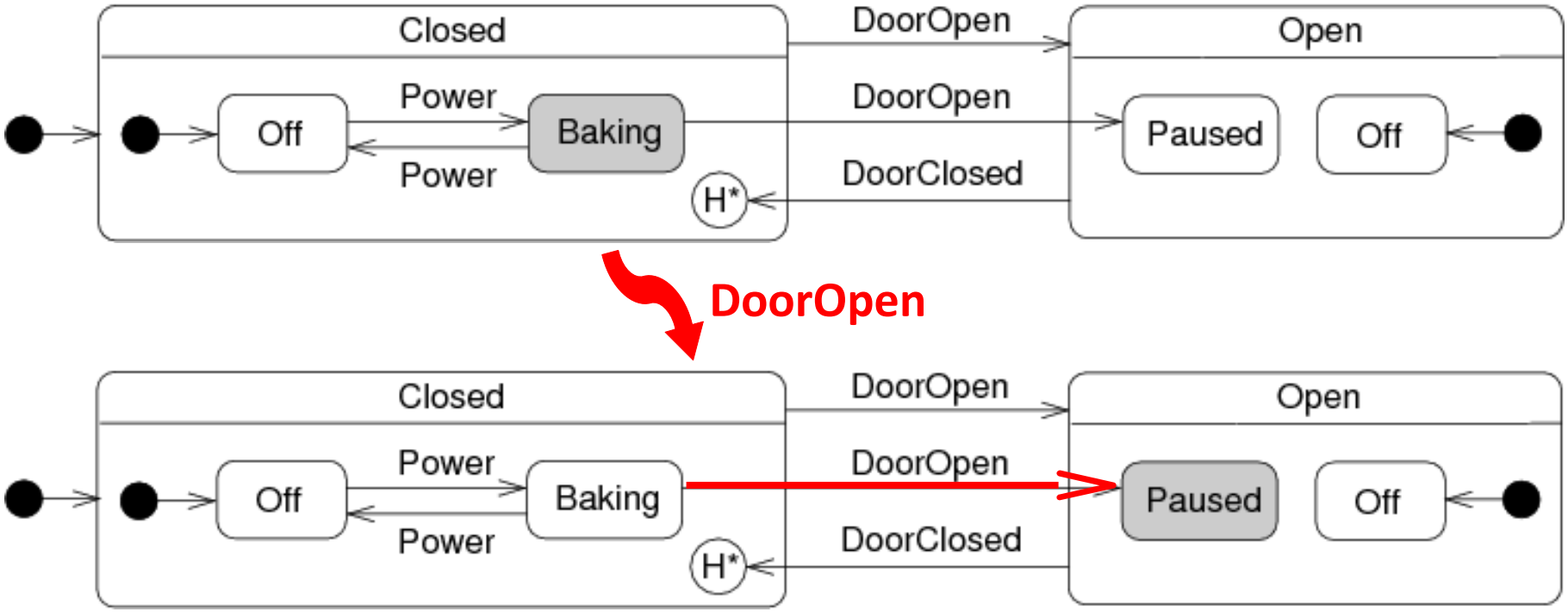
- A model-based software development process
- Pragmatic and intermediate approach between modeling and programming
  - The behavior of the system is a model
    - An executable model: UML state machines here
    - High-level of abstraction for defining the behavior
    - Can also be simulated at design for early detection of problems
  - The rest of the system is developped with standard programming languages
    - Most suitable way to implement technical parts (data persistence, distribution...) or business operations

UML state machines



- Example: state machine of a microwave oven
  - Composite states based on the door: closed or open
  - Transitions between states associated with events

UML state machines



- Can be executed
  - Process of events (''DoorOpen'' here) and trigger transitions
  - Change of the current active state: from''Baking'' to ''Paused''

Executable models

- But what will concretely execute my model at runtime?
  - Currently nothing without the weaving of business operations!
  - Ex: a Java business method associated with the "baking" state

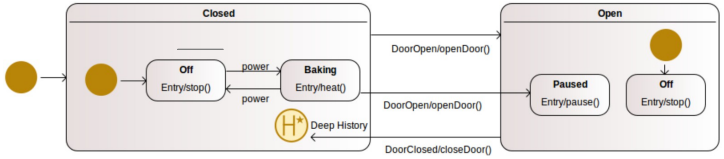
```
public void heat() {
   setLightOn();
   setMagnetronOn();
}
```

- PauWare: for "programming" UML state machines in plain Java
  - API to define states, transitions associated with business operations

```
baking = new Statechart("Baking");
baking.set_entryAction(businessObject, "heat");
```

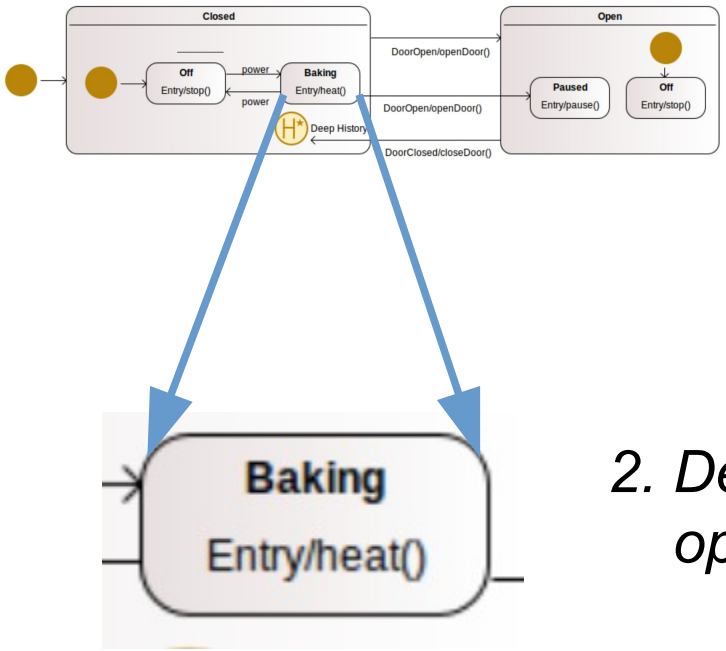
  - Engine to execute UML state machines and business operations by processing events
    - Simple to use and powerful but it is code not really a model ...

Concrete development process



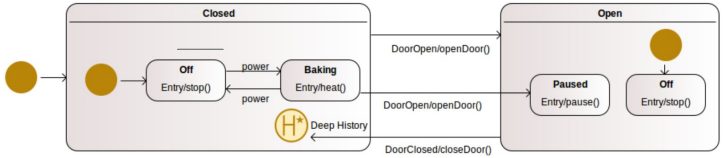
1. *Define with your favorite UML modeler the UML state machine specifying the behavior of the application*

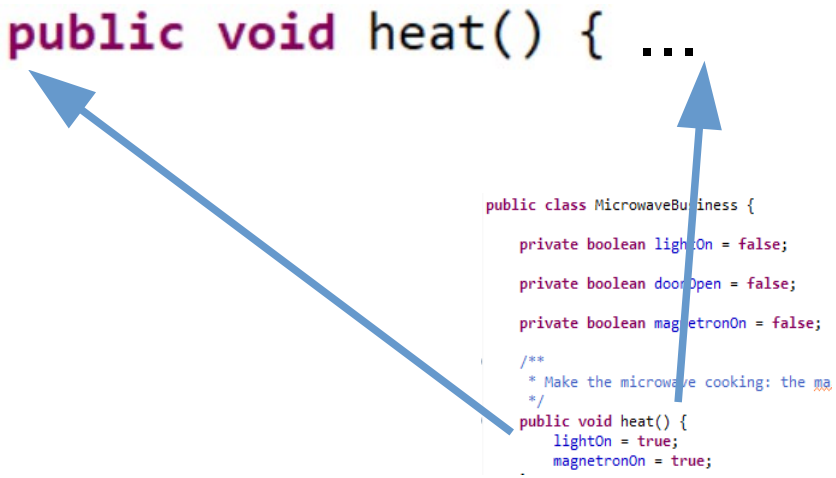
Concrete development process



2. *Define the signatures of business operations on states and transitions*

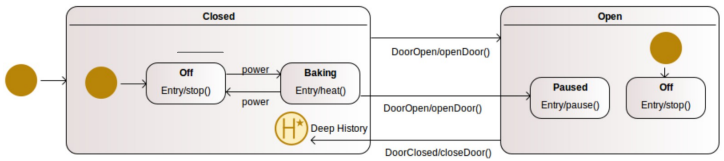
Concrete development process

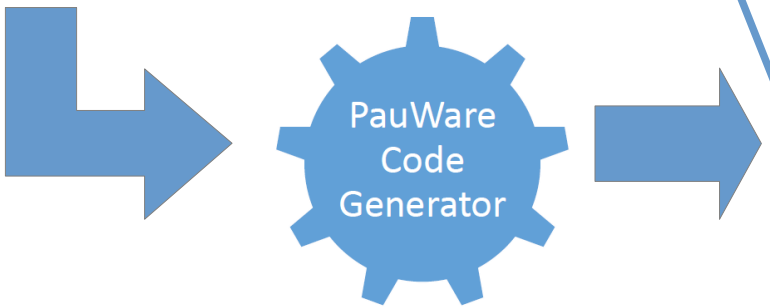




3. *Implement in plain Java the business part with the concrete code of the operations*

Concrete development process





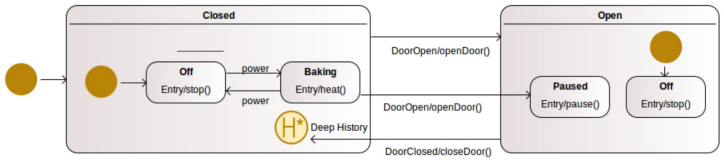

```
baking = new Statechart("Baking");
baking.set_entryAction(businessObject, "heat");
```

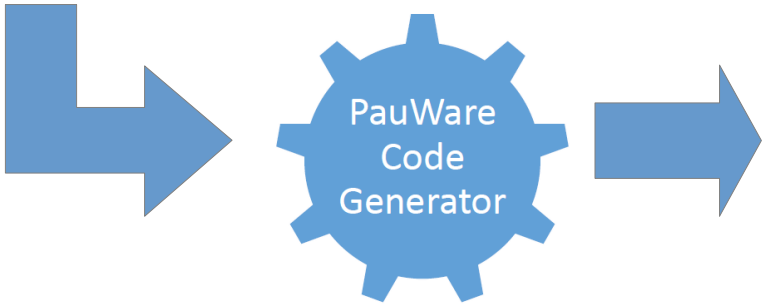
```
offClosed = new Statechart("Off.");
offClosed.set_entryAction(mwb, "stop");
offClosed.inputState();

// The baking state executes the "heat" method
baking = new Statechart("Baking");
baking.set_entryAction(mwb, "heat");

// The paused state executes the "pause" method
paused = new Statechart("Paused");
paused.set_entryAction(mwb, "pause");
```

```
public class MicrowaveBusiness {

    private boolean lightOn = false;

    private boolean doorOpen = false;

    private boolean magnetronOn = false;

    /**
     * Make the microwave cooking: the ma
     */
    public void heat() {
        lightOn = true;
        magnetronOn = true;
```

4. Automatically generate the PauWare API Java code for the UML state machine through the online code generator

Concrete development process






```
offClosed = new Statechart("Off.");
offClosed.set_entryAction(mwb, "stop");
offClosed.inputState();

// The baking state executes the "heat" method
baking = new Statechart("Baking");
baking.set_entryAction(mwb, "heat");

// The paused state executes the "pause" method
paused = new Statechart("Paused");
paused.set_entryAction(mwb, "pause");
```

```
public class MicrowaveBusiness {

    private boolean lightOn = false;

    private boolean doorOpen = false;

    private boolean magnetronOn = false;

    /**
     * Make the microwave cooking: the ma
     */
    public void heat() {
        lightOn = true;
        magnetronOn = true;
```

PauWare

5. *Execute with the PauWare engine the application resulting from the weaving of the UML code with the business part*

Conclusion

- Pragmatic model-based sotfware development process
  - Behavior of the system: executable UML state machine
    - Seamlessness development: the design model is executed at runtime
    - Code compilation : from UML model to PauWare code
  - Rest of the system (business, technical): in regular Java
  - Direct association between the business operations and the state machine
- Independance
  - Of the modeling UML tool: online model compilation
  - Of the IDE, frameworks or libraries for the Java code development
    - PauWare JAR file: ~100 kB