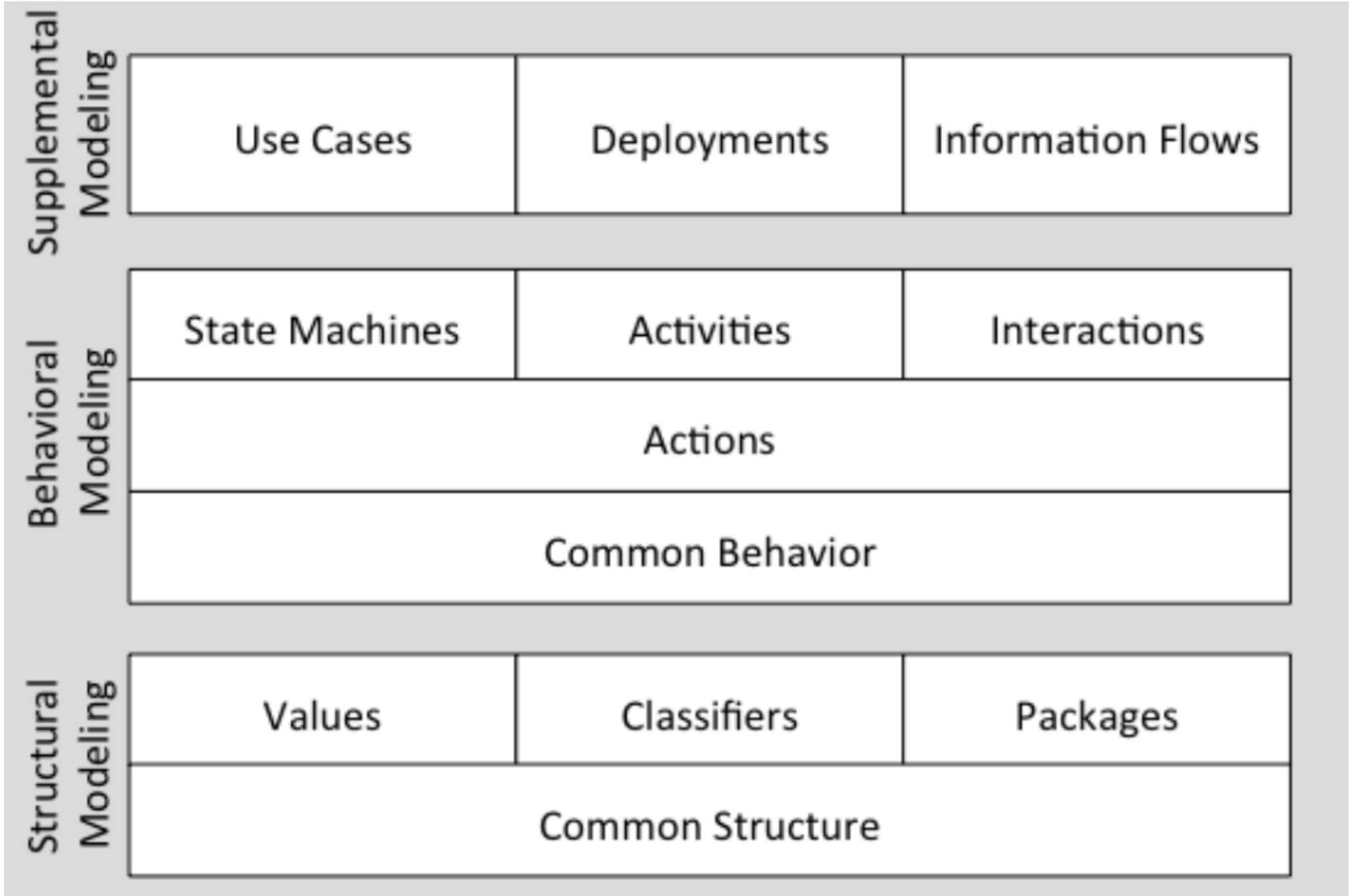
Introduction to modeling

Asma Smaoui, Ansgar Radermacher
asma.smaoui@cea.fr, ansgar.radermacher@cea.fr

Acknowledgments – contains material from my CEA colleagues Ansgar Radermacher, Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

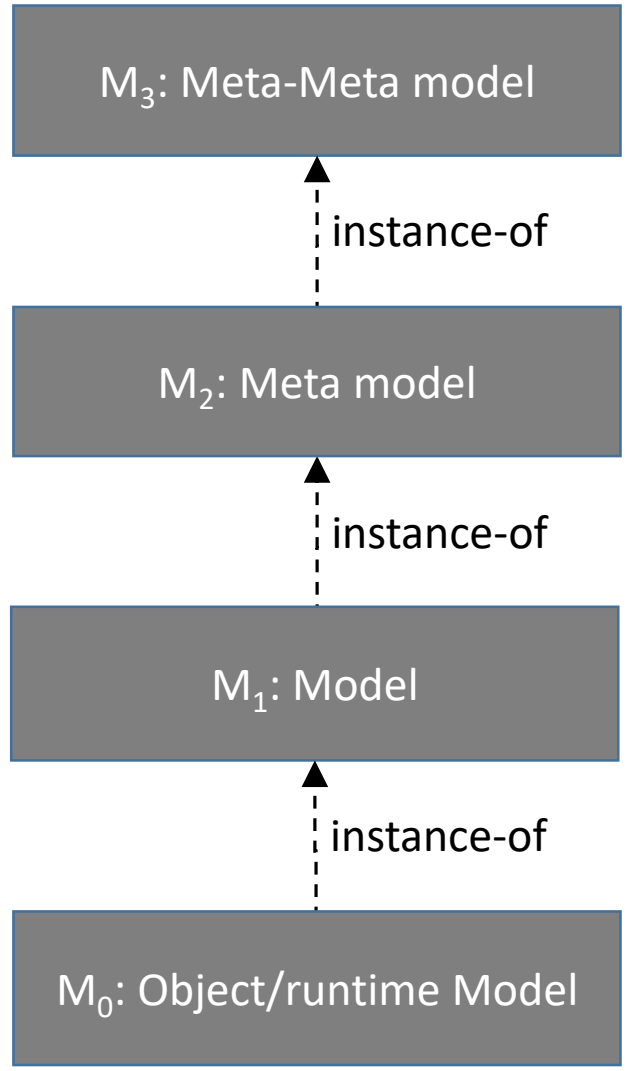

Supplemental Modeling
Use Cases
Deployments
Information Flows
Behavioral Modeling
State Machines
Activities
Interactions
Actions
Common Behavior
Structural Modeling
Values
Classifiers
Packages
Common Structure

UML, now with semantics!

- For a long period of time
  - UML was considered as “semantic free” (considered as a major flaw)
    - Until UML 2.4 its semantics was only described in prose
    - This assertion does not hold at all for UML 2.5
- Semantics of UML
  - Defined using an operational approach
  - Two normative documents
    - fUML defining the semantics of classes and activities
    - PSCS defining the semantics of composite structures
    - PSSM defining the semantics of behavior state-machines
  - Models build using classes, activities, composite structures and state-machines are by construction executables

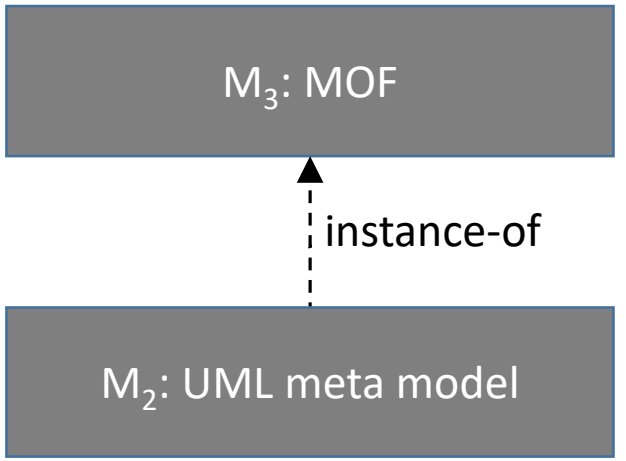
Meta-models and languages

- The elements of a model are defined in a meta-model, e.g. a Class in case of UML
- A model at M_n is instance of a M_{n+1} model
- Typical (design) models = M_1 models



UML Meta-models and languages

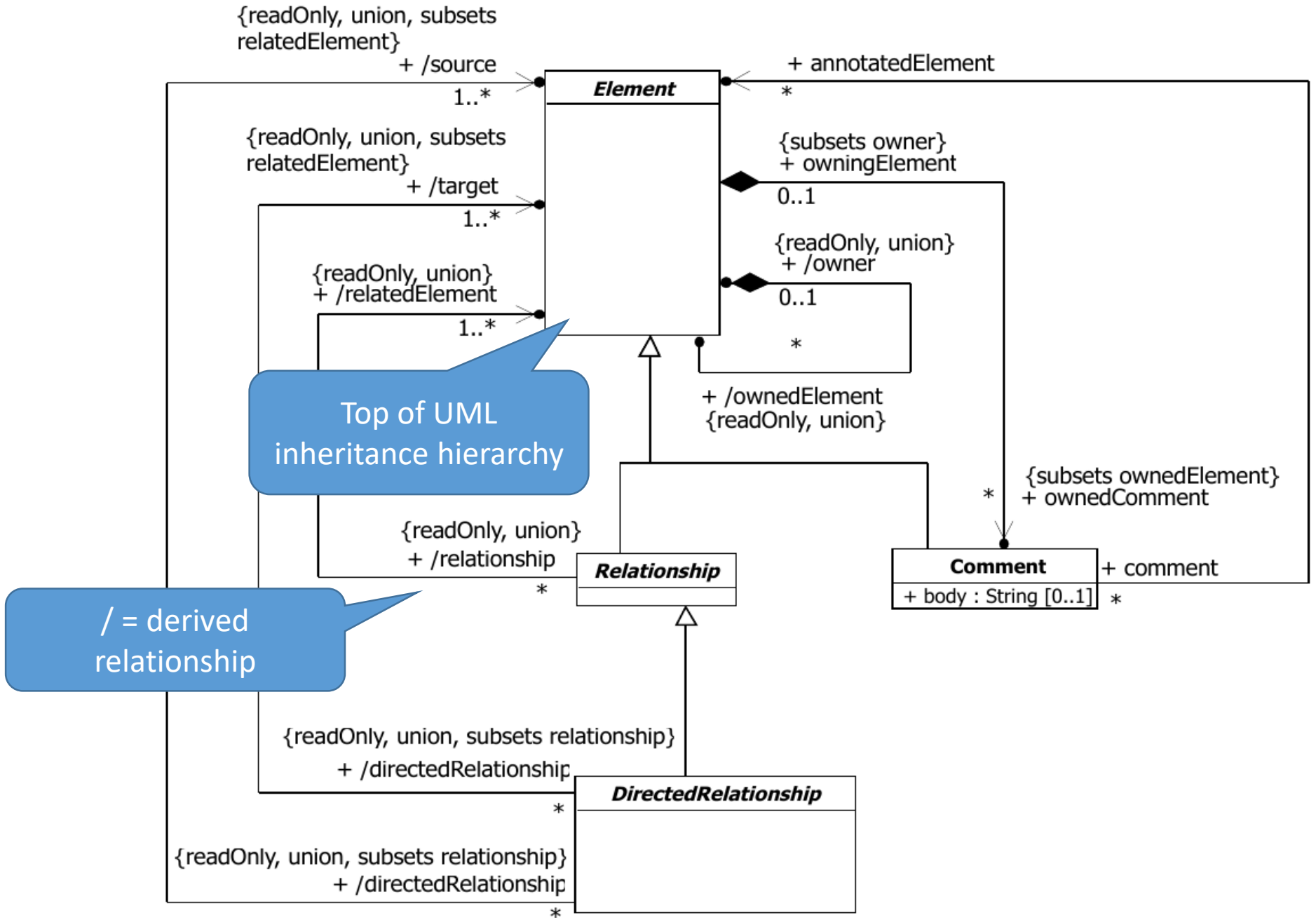
- Meta-Object facility (MOF, OMG standard)
  - Imports basic MM concepts from UML (Class, Property, Association, Generalization)
  - Specification becomes kind of recursive



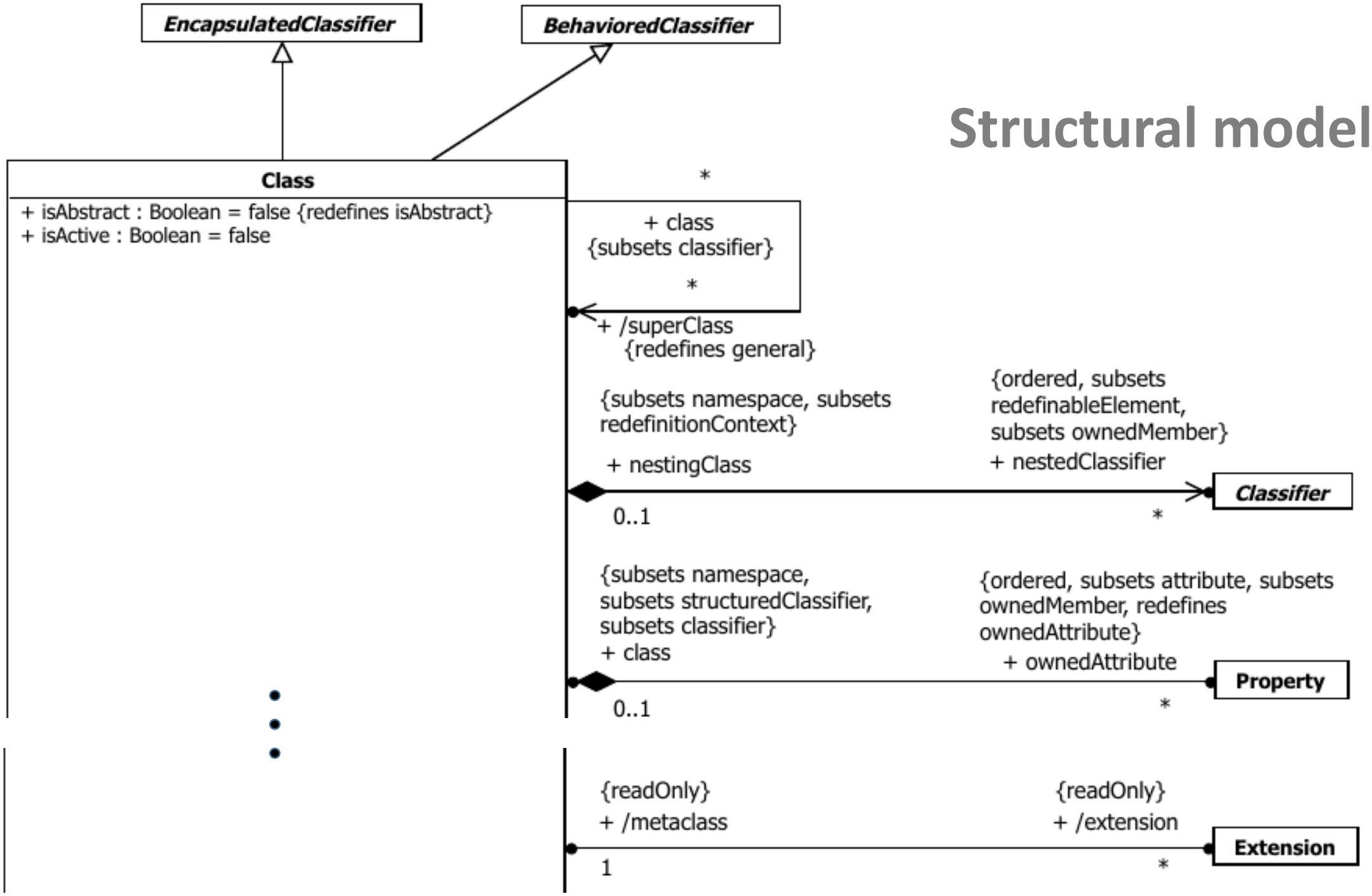
- UML-Meta-Model, 3 main parts
  - Common infrastructure
  - Structural modeling
  - Behavioral modeling

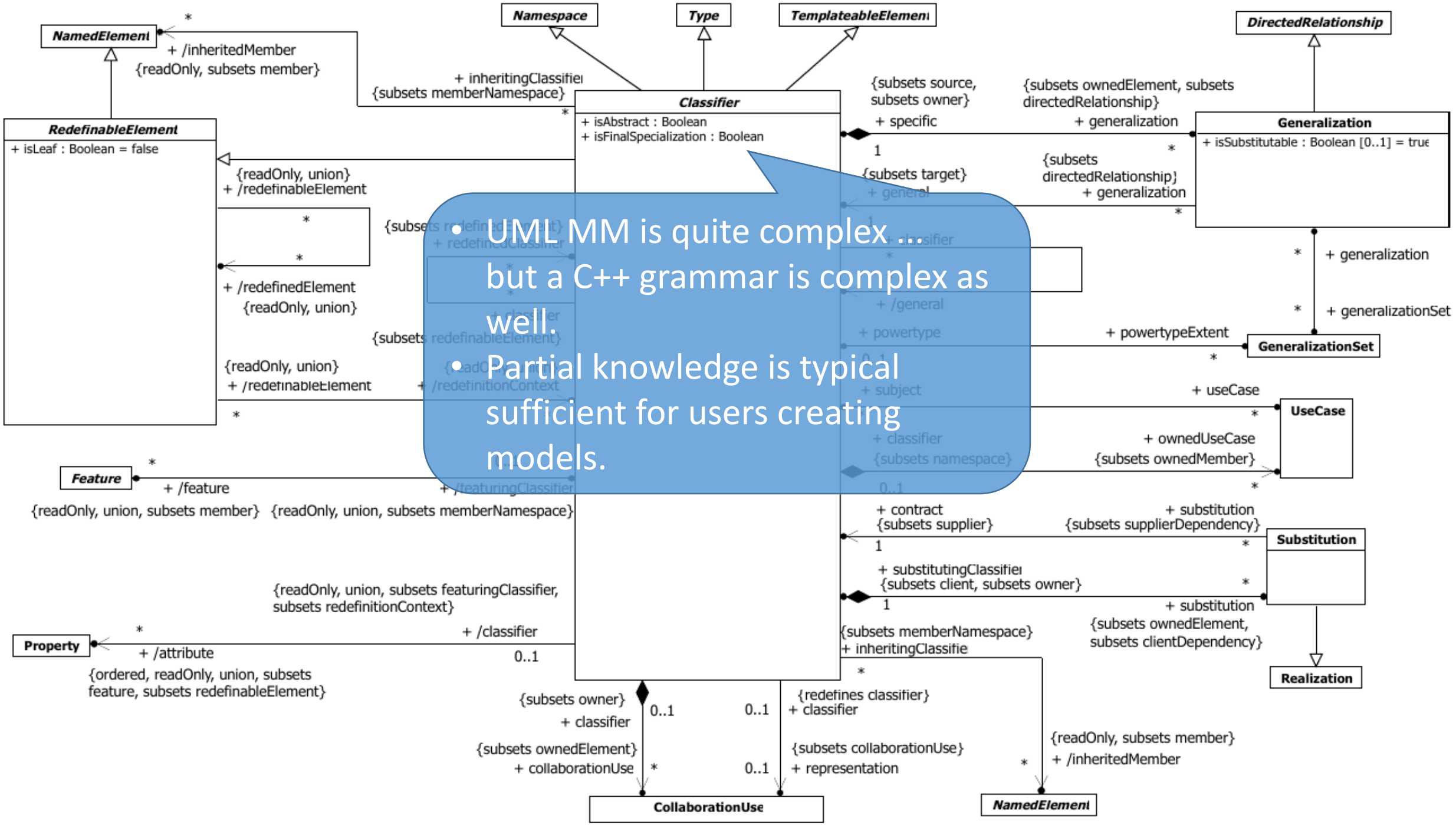
- Additional (OCL) constraints limit what can be “legally” modeled

Common structure



Structural modeling





- UML MM is quite complex ...
  but a C++ grammar is complex as well.
- Partial knowledge is typical sufficient for users creating models.

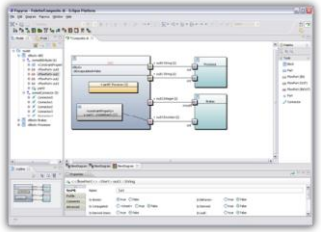
Meta-model implementations - Ecore

- Part of Eclipse Modeling Framework (EMF)
- Meta model for describing models and runtime support
  - Change notification,
  - Persistence support with default XMI serialization
  - A reflective API for manipulating generic EMF objects
- Ecore tools: Sirius (graphical tool)

Agenda

1. Modeling languages, motivation
2. UML, the software modeling language
3. Formalize models, meta-modeling
4. **Papyrus UML modeler (+ class and state-machine diagram)**
5. Model transformation, principles
6. Model transformation, languages

Papyrus UML modeler



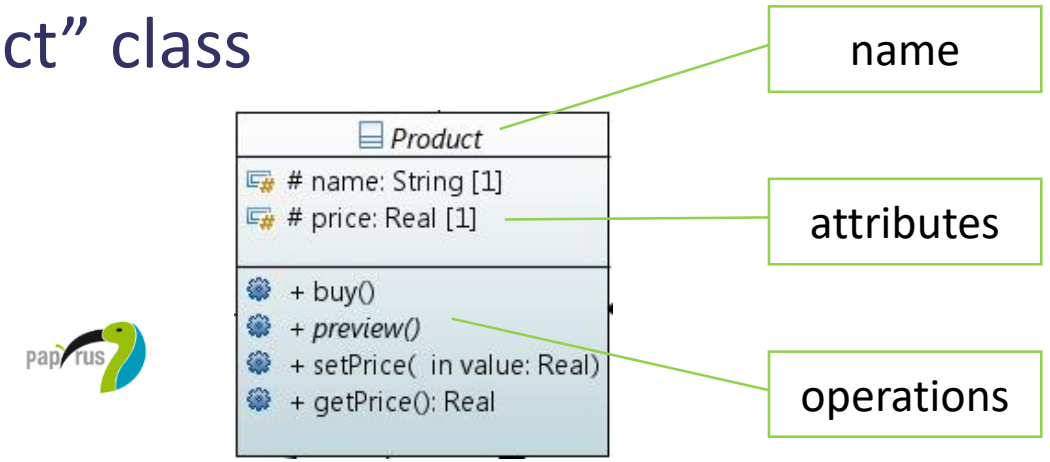
- Papyrus in a nutshell
  - Industrial-grade **open source** Model-Based Engineering tool
  - Part of the Eclipse release train
  - **Standard based** (OMG UML, MARTE, …)
  - Customizable to address domain-specific concerns
  - Multiple extensions exist (some shown later)

- Get started: https://www.eclipse.org/papyrus/

Classes and class diagrams

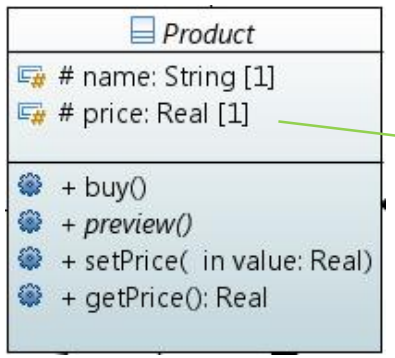
Overview

- Description of a set of objects with common semantics, features, and constraints
- Among features, a class has attributes and methods
- In UML, features are gathered within compartments: a class has a name, a compartment of attributes, and a compartment of operations
- Example: a “Product” class



Classes and class diagrams – attributes

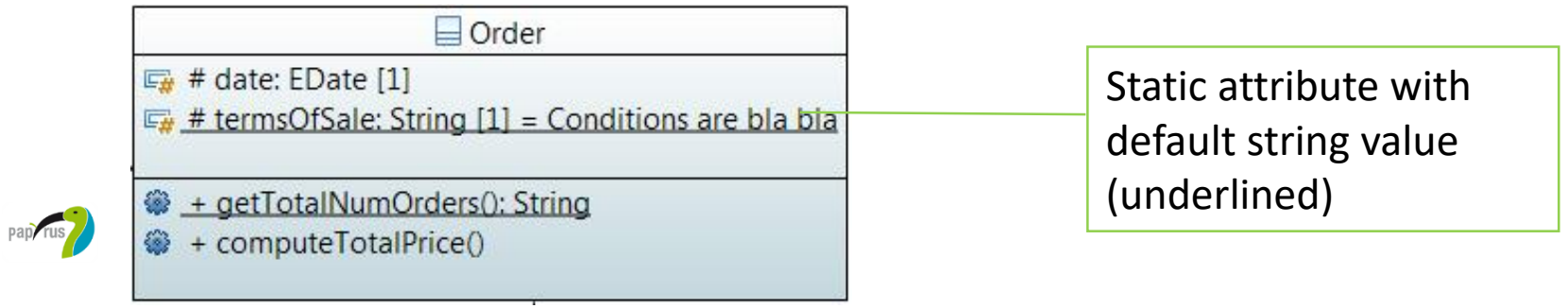
- An attribute describes data of the class when instantiated as an object
- Specification of an attribute
  - Visibility: constraint on the access of the attribute (more on this later)
  - Name
  - Type: class that the attribute instantiates at runtime
  - Multiplicity: cardinality, i.e. number of elements
    - Syntax: [lower..upper] (* means several, n..n is also noted n, 0..* is also noted *)
  - Default value
- In UML, an attribute is modeled with a Property element
- Example:



Visibility = protected
Name = price
Type = Real
Multiplicity = [1..1] ➔ [1]

Classes and class diagrams – static attributes

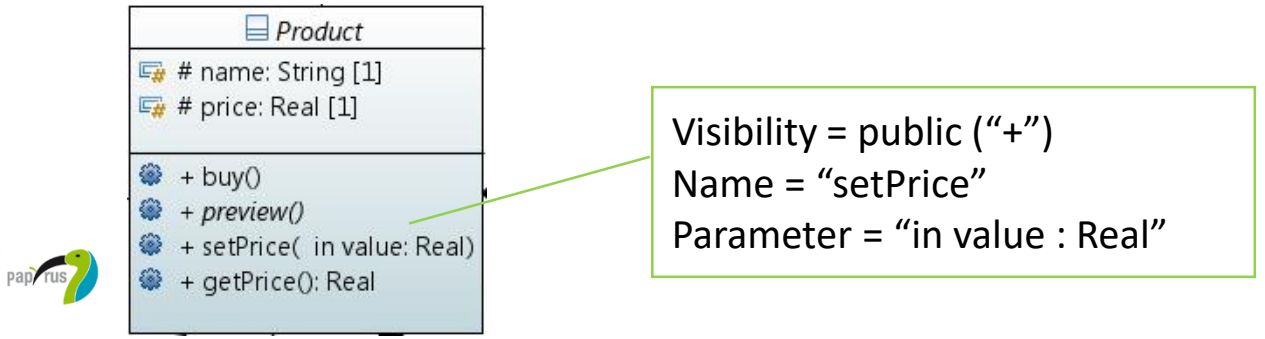
- An attribute takes value when the class is instantiated as an object…
- …unless the attribute is static
- A static attribute can be accessed without instantiating the class
- A static attribute has a visibility, multiplicity and type as well
- Example: “termsOfSale” is a static attribute since we want to access it without having to instantiate any “Order”



Classes and class diagrams – operations

- A class has methods with signature and body
- In UML, an operation is the specification of a method, i.e. the method signature, independently of its implementation
- In UML, the word “method” is used to designate an implementation of an operation, i.e. the body of a method in a class
- Specification of an operation
  - Visibility
  - Name
  - Parameters: name, direction (in, out, inout, return), type, multiplicity, default value
- Example

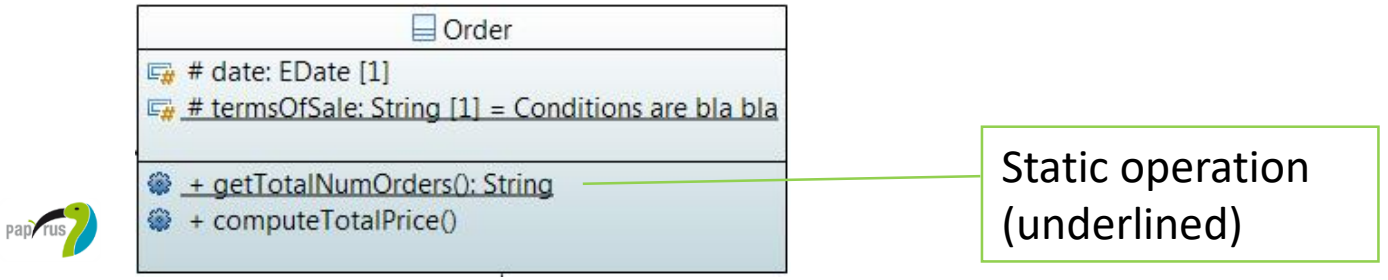
Careful, the same “method” word in UML and OOP do not designate the same things. Yes this can be confusing…



Visibility = public (“+”)
Name = “setPrice”
Parameter = “in value : Real”

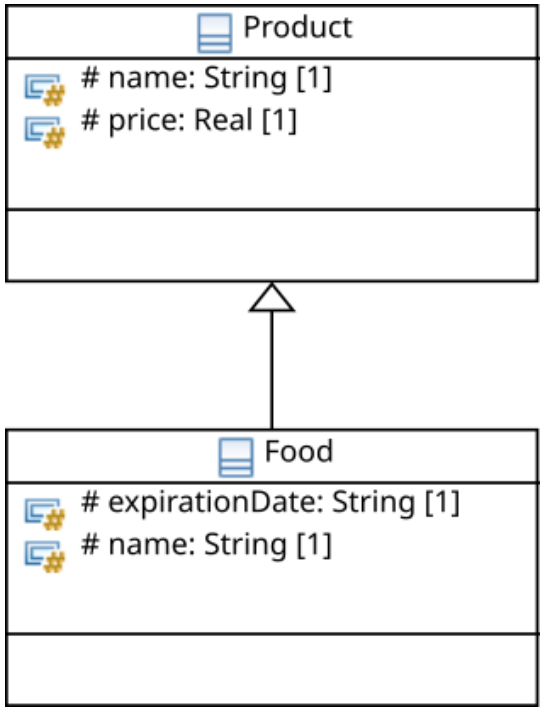
Classes and class diagrams – static operations

- Usually an operation is accessed from the object instantiating the class…
- … unless the operation is static
- A static operation can be accessed without instantiating the class
- Example: “getTotalNumOrders” is a static operation since we want to access the total number of orders without instantiating an “Order”



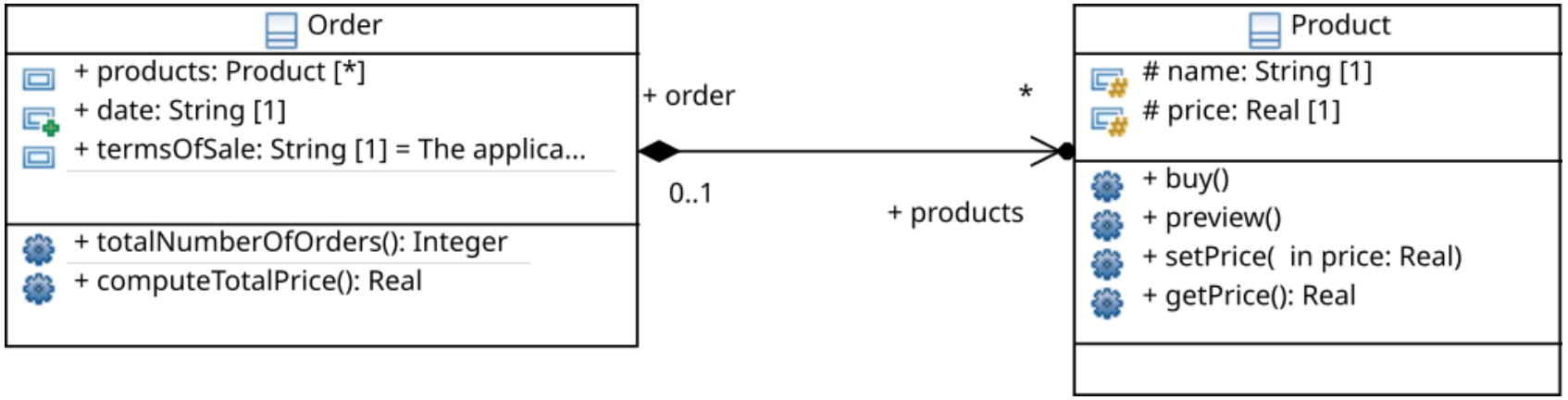
Inheritance / Generalization

- A class might inherit from another
- Notation uses hollow arrow
- **Optionally** show inherited features (Feature is a superclass of Operation and Property in the UML MM)



Association

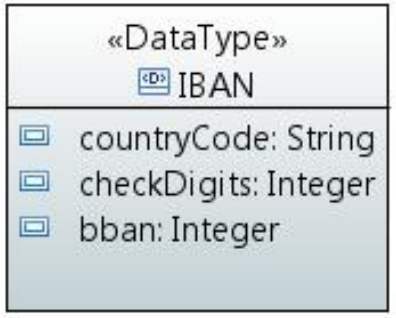
- Link between two classes
- Two ends, can be directed, multiplicity as for properties
- Aggregation kind: none, shared and composition



Data Types

Overview

- A class can type attributes, i.e. the attribute is an instance of the class at runtime
- DataType is similar to a class; it is typically used to represent value types of a certain domain, or primitives, or structured types
- Instances of a data type are identified by the values of the attributes
- Example: “IBAN” is a structured type, defined by a country code, check digits, and a BBAN



Data Types

PrimitiveType

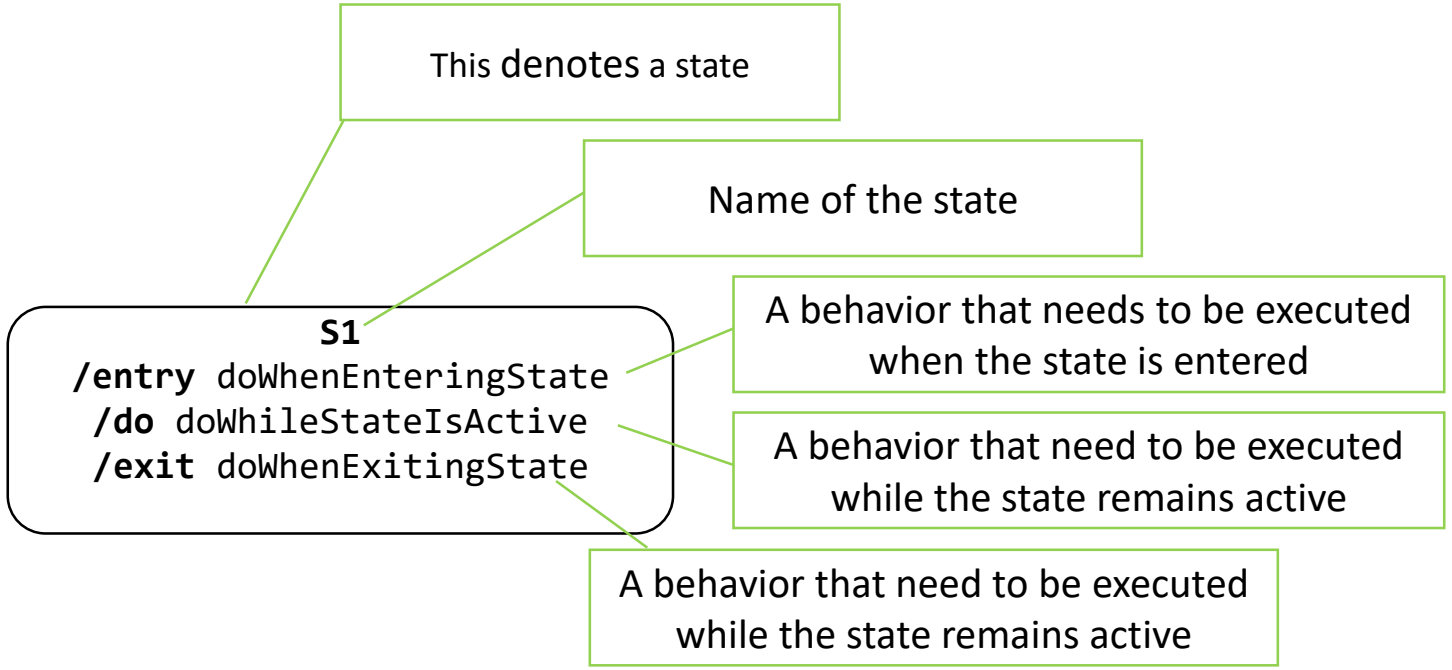
- An atomic data type, i.e. without structure
- UML primitive types: Boolean, Integer, UnlimitedNatural, String, Real
- Example:

Enumeration

- Data type whose values are enumerated as user-defined enumeration literals
- Example:

State – Simple state

- "A state models a situation during which some invariant condition holds"

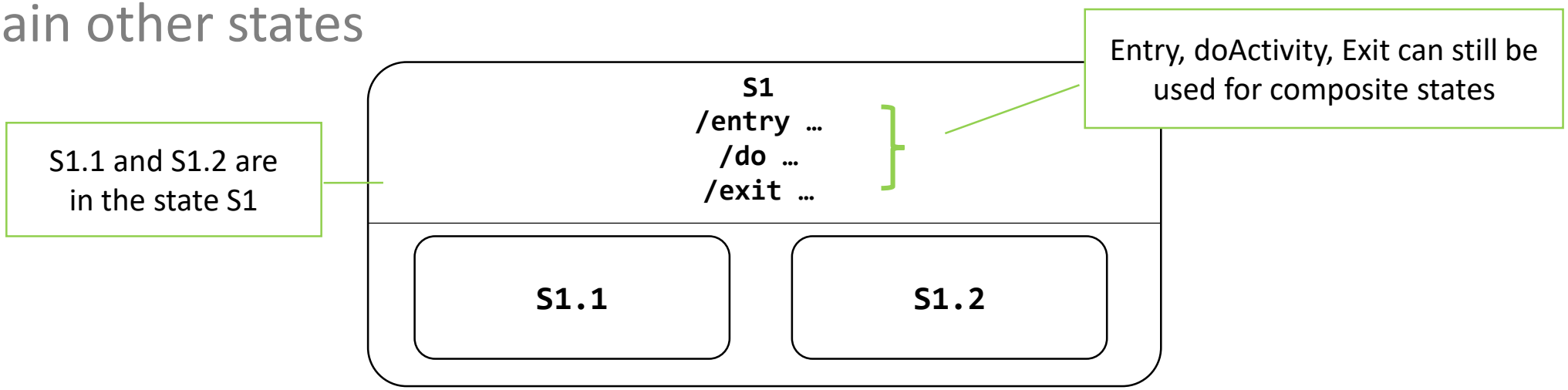


- Example: an order can be Payed, Confirmed, Packed, Cancelled,…

Payment was done and confirmation of the order was sent to the client

State – Composite state

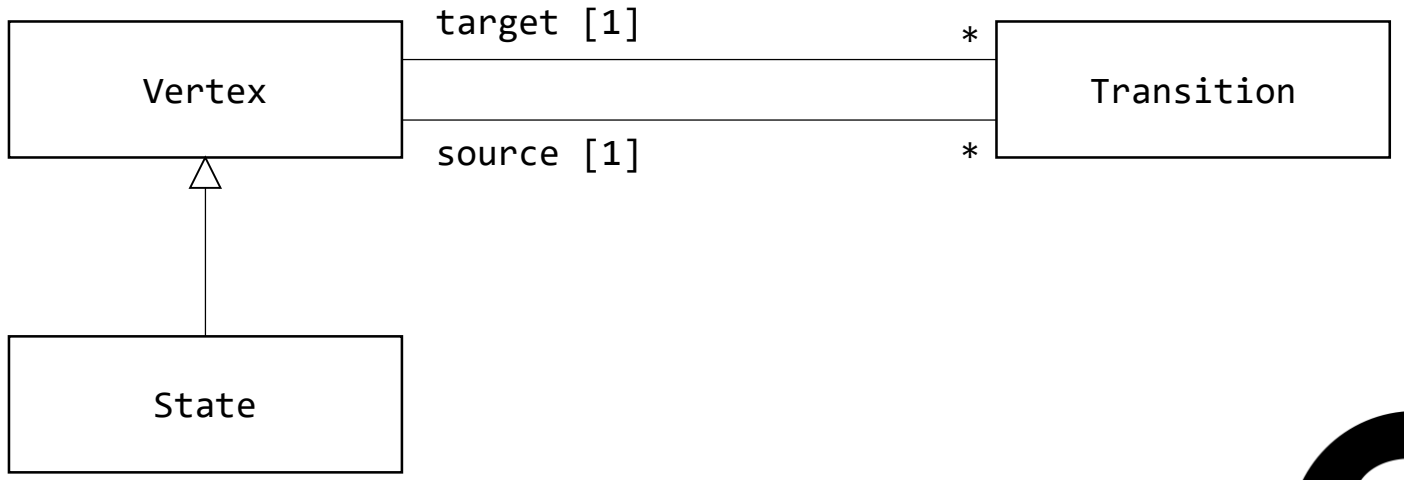
- In addition to what is possible with a simple state, a composite state can contain other states



- If S1 is active that means one of the states included in S1 is also active (e.g., situation where S1 is active as well as S1.1).
- Example: An order can be in the process of being Finalized. This process usually includes the payment and the sending of a confirmation of the order. Therefore Finalized can contain both Payed and Confirmed.

Until now

- We talked about states (simple and composite)
- What is missing here to complete simple state-machines?
  - How do we move from a source state to a target state ?



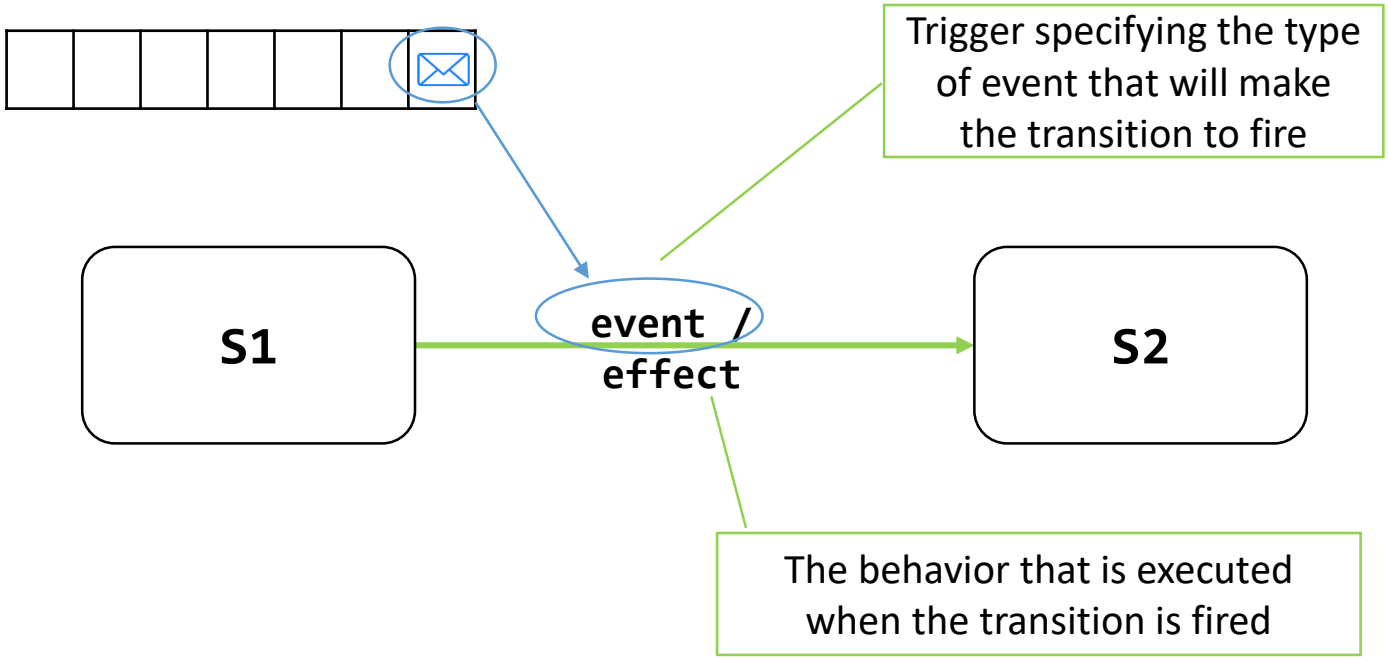
Transition – Completion transition

- A transition is an edge between one source vertex and one target vertex
  - Note: both source and target can be the same
- It denotes part of a path that can be followed during the execution of step of a state-machine.

- Completion transition: There is no constraint(s) to fire this transition. It can be fired immediately after S1 was exited.

Transition – Triggers and Effects

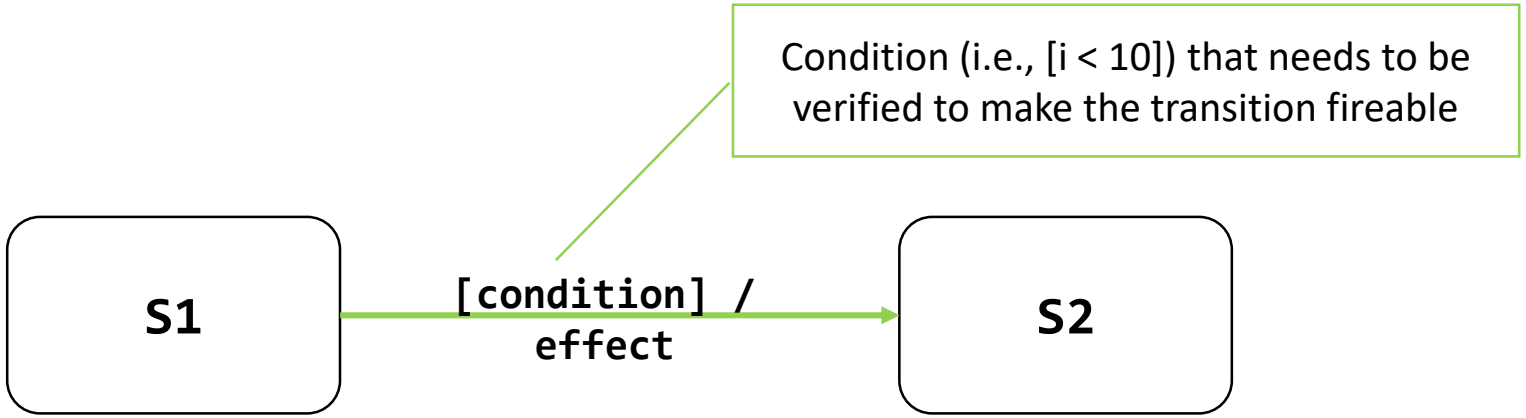
- Can be triggered
  - Fires when a specific event is available (Time, Signal, Call)



- Note: A single transition can specify multiple triggers (i.e. it can fire upon the arrival of many different events)

Transition – Guarded Transition

- Can be guarded
  - Fires when a the condition placed on the guard is verified

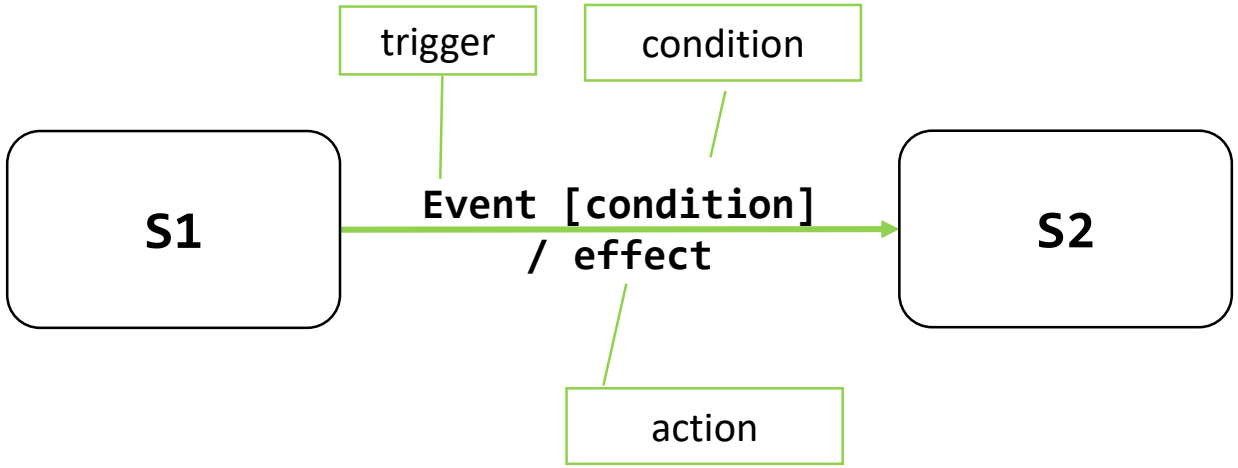


- Note: guards are specified using UML constraints which in turn contain an expression
  - Makes specification unnecessary complex, could directly use expression

Transition – Guarded and Triggered transition

Transition

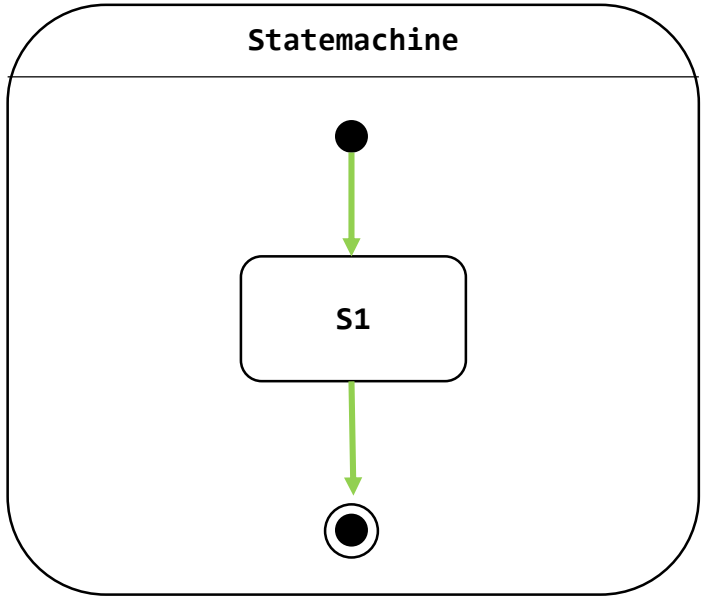
- Can be guarded and triggered
  - Fires when a specific event is available and the condition specified holds

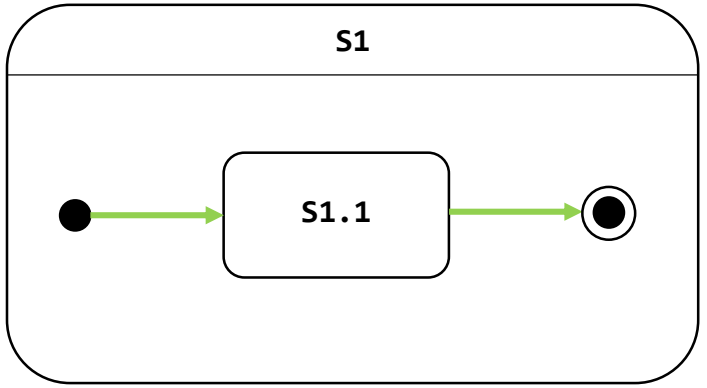


Initial and final pseudo state

- Initial state ●
  - Define the starting point of a state-machine
  - Define the starting point of a behavior owned by a composite state

- Final state ◉
  - Specialization of a state
  - Define the end point of a state-machine
  - Define the end point of a set of states owned by a composite state





Example State-machine diagram