UML 2.0 State Machines

F. Mallet

Frederic.Mallet@unice.fr

Université Nice Sophia Antipolis

M1 IFI – Embedded & Mobile Computing

UML State Machines

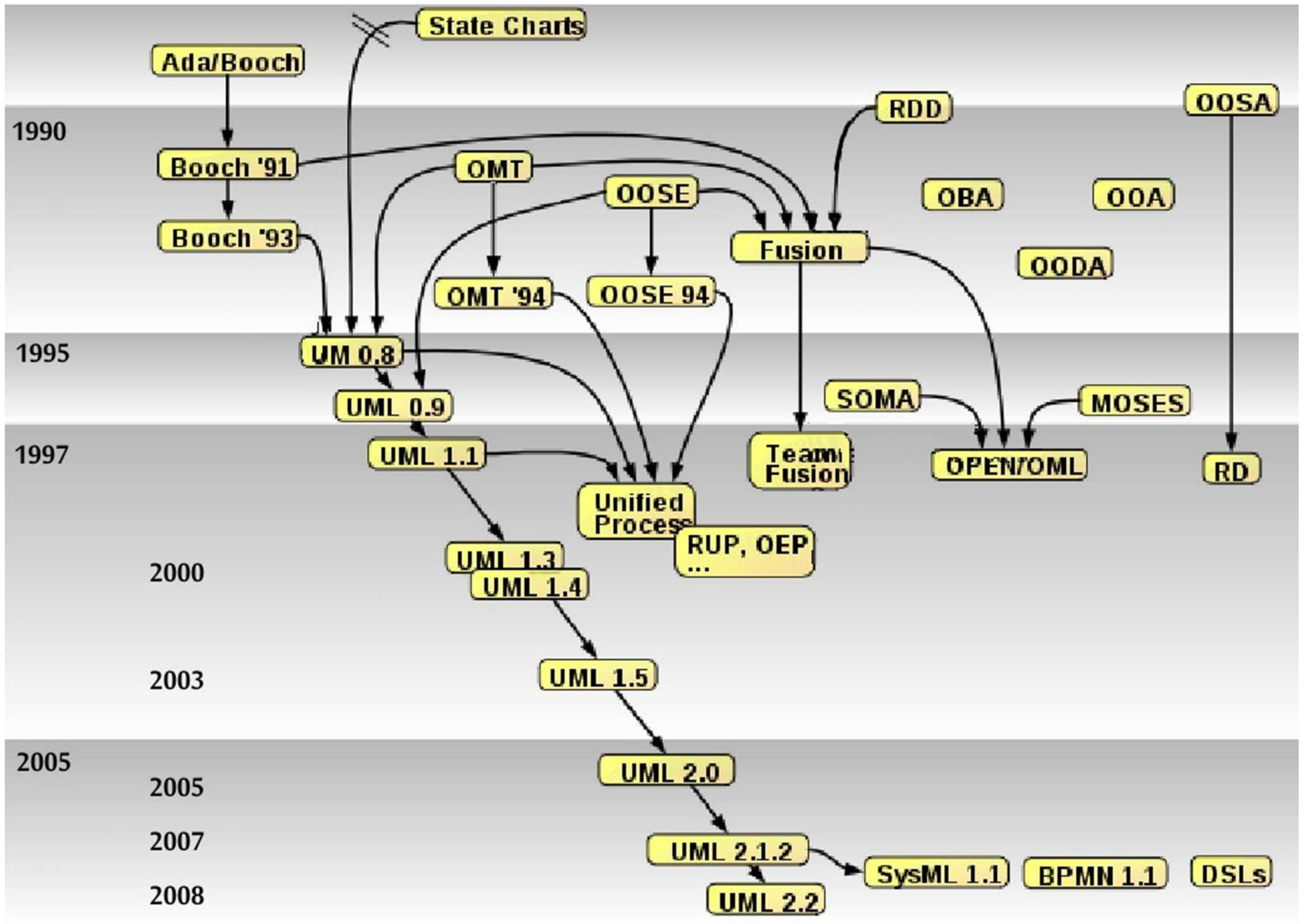
- Objectives
  - UML, OMG and MDA
  - Main diagrams in UML
    - Focus on State Machines here !
  - New Constructs in UML 2.0
  - Profiling mechanism

- Practical: Transforming UML state machines into formal counterparts

Unified Modeling Language Genesis

- In 1994
  - Object-orientation was becoming popular
  - Too many methods/languages to describe similar concepts (>5000)
    - Metamodels were very similar
    - Graphical notations were completely different
  - The Industry was asking for a standard notation
- Rational Software Corporation starts a process
  - Booch method (Grady Booch) and OMT (Jim Rumbaugh)
  - Followed by OOSE (Ivar Jacobson from Objectory)
    - Use cases

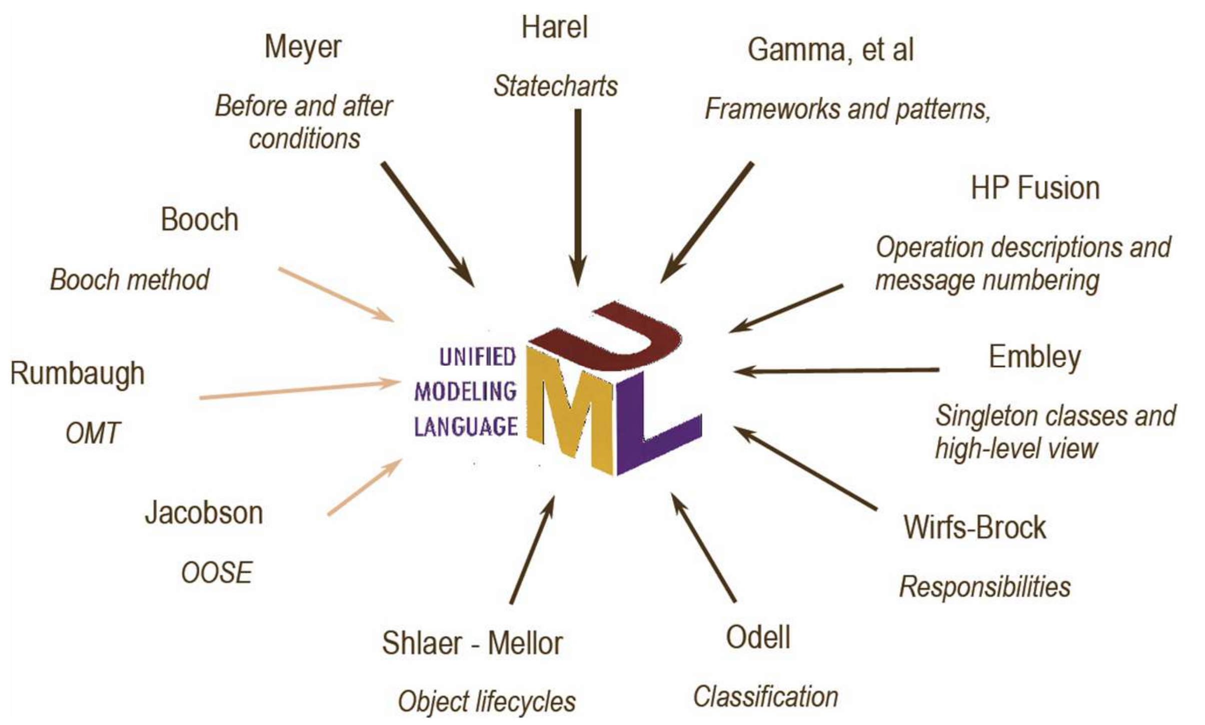
Unified Modeling Language Genesis



http://en.wikipedia.org/wiki/File:OO-historie.jpg

Unified Modeling Language (UML)

- Specified by the OMG
  - Object Management Group
    - OMG™ is an international, open membership, not-for-profit computer industry consortium since 1989
    - Most famous specifications: CORBA, UML, MDA, MOF, IDL
    - http://www.omg.org



Model-Driven Architecture (MDA)

- MDA is an approach to using models in software development
  - Specify the system independently of the platform that supports it (Platform-Independent Model - **PIM**)
  - Specify platforms
  - Choose one particular platform for the system
  - Transform the PIM into a Platform-Specific Model (**PSM**)
- UML is a core element of the MDA
  - Both for PIM and PSM
  - May need some extensions depending on the domain
    - Profiles = annotate UML models with domain-specific information

MDA viewpoints (1/2)

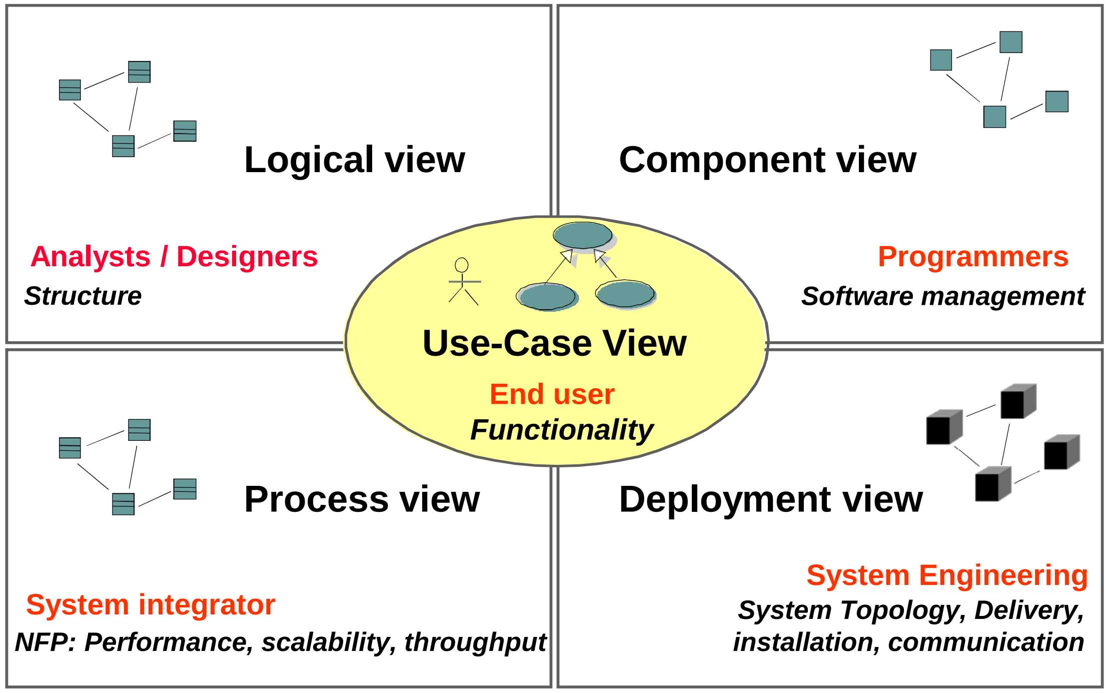
- **Computation Independent Model (CIM)**
  - The requirements for the system: *domain/business model*
  - Shows the system and its environment
    - Understand the problem
    - Source of shared vocabulary
    - Should be traceable to PIM and PSM
  - E.g., may describe the processors and communication media but it does not mean that there will be a class Processor in the final system
- **Platform-Independent Model (PIM)**
- **Platform-Specific Model (PSM)**

MDA viewpoints (2/2)

- **C**omputation **I**ndependent **M**odel (**CIM**)
- **P**latform-**I**ndependent **M**odel (**PIM**)
  - Shows the part of the system that does not change from one platform to another
    - Assumes some abstract features from the platform
    - For RTES, expected QoS (Quality of Services) and NFP (Non Fonctional Property) must be there !
  - E.g., may not need to know if we use Corba, .net or J2EE but needs to know that we use a middleware
- **P**latform-**S**pecific **M**odel (**PSM**)
  - Merge between PIM and Platform
  - Should be generated

Five complementary views

- Different model elements and diagrams can be in different views
  - With more or less details



14 diagrams

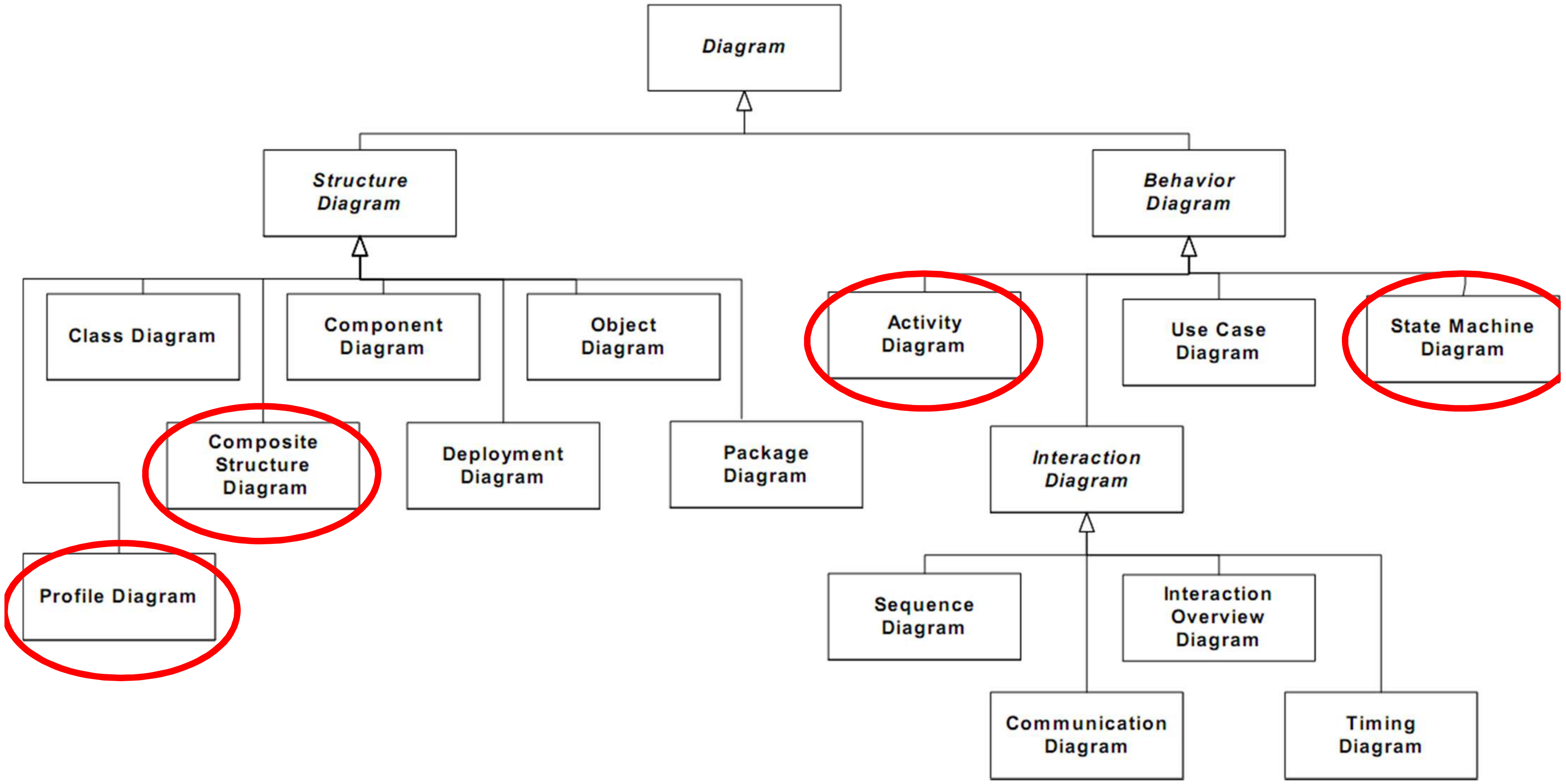
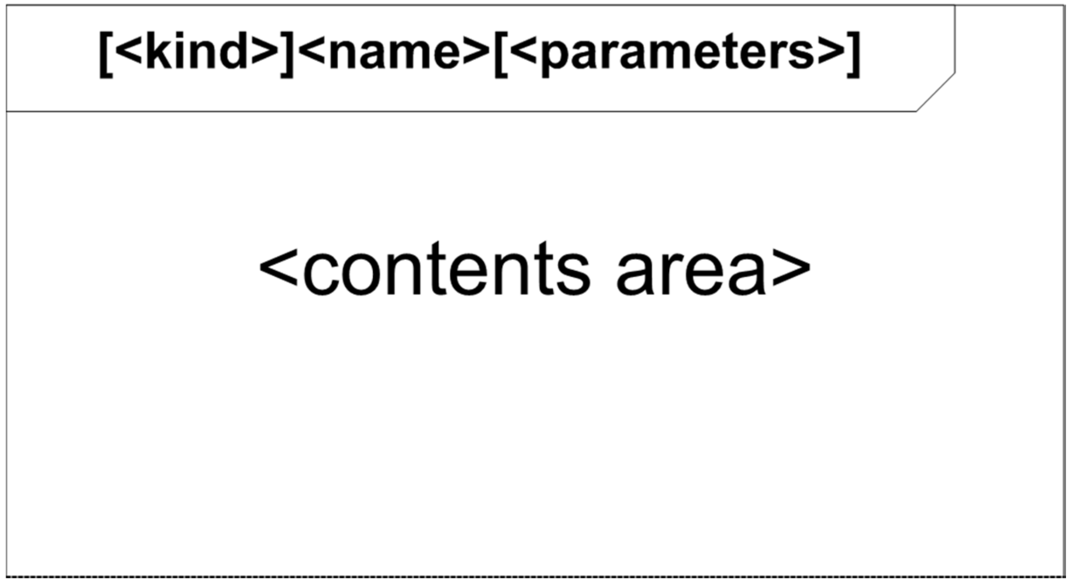


Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Interaction Overview Diagram
Communication Diagram
Timing Diagram

Diagram and frames

- Diagrams should be within frames
  - Heading should give a name, kind and parameters if any
    - kind ∈ { activity, class, component, deployment, interaction, package, state machine, use case }
    - Short form { act, class, cmp, dep, sd, pkg, stm, uc }



What process to use ?

- The UML is a language, not a process
  - The UML is independent of the process
- The process may benefit from using the UML when
  - Use-case driven
    - SysML has improved a lot the way requirements are captured
  - Architecture centric
    - Refine the architecture or execution platform
  - Iterative and incremental
    - Several executables delivered throughout the development
    - Continuous Testing (unit testing, integration, regression tests)
- Most famous process: **R**ational **U**nified **P**rocess

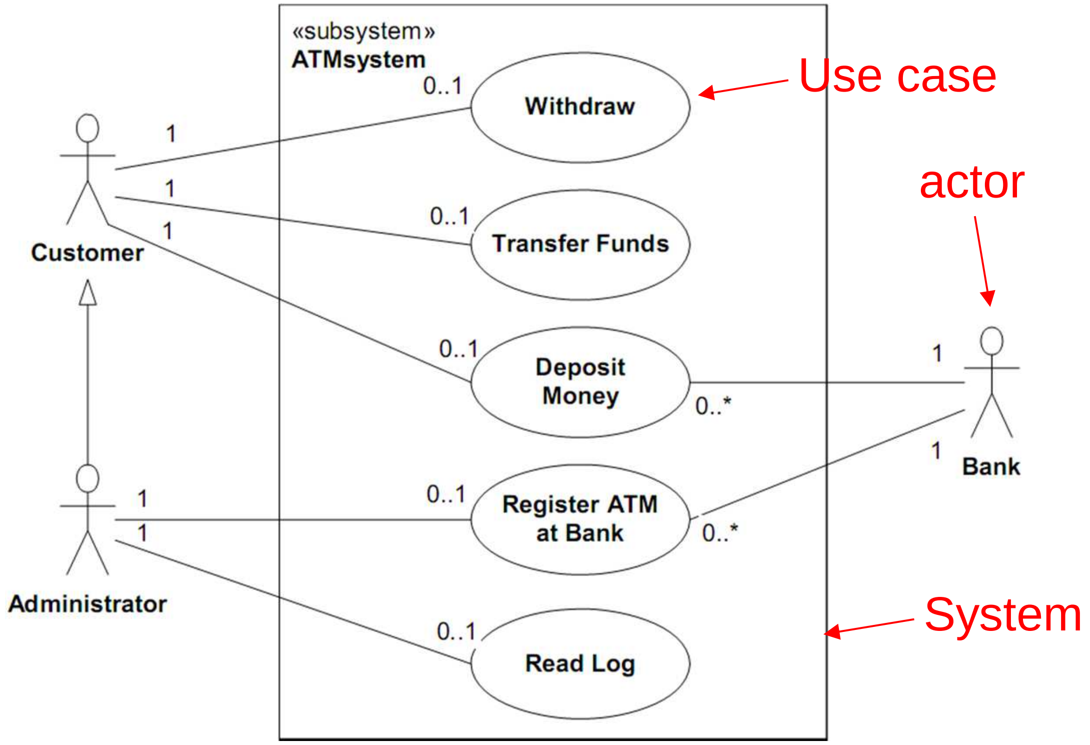
One process

- Describe Requirements with Use cases or SysML
  - Describe the behavior of requirements with state machines and activities
  - Describe the abstract feature of the platform (Non Fonctional Properties)
- Build possible scenarios for each Use cases
  - Using interactions (sequence or collaboration diagrams)
  - This requires to identify
    - the objects and classes
    - the methods and their parameters
  - In parallel, build a class diagram
- Describe the state of classes
  - With state machines or activities
- Make bundles
  - Component and deployment diagrams

USE CASES

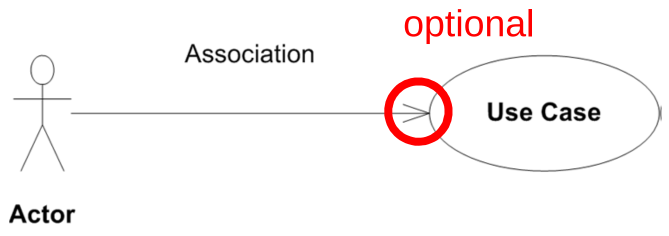
Use Case Modeling (1/4)

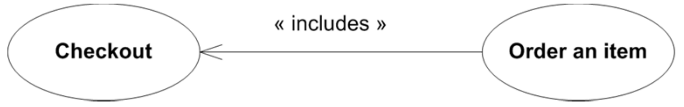
- Use cases define
  - What the system contains and what it does not contain
  - Who is responsible for what
  - What are the boundaries of the system
  - Must be approved by various stakeholders (actors)



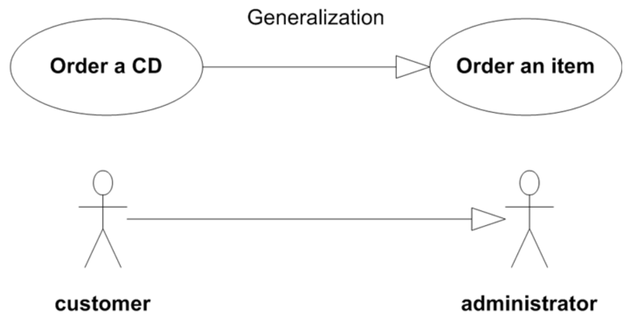
Use Case Modeling (2/4)

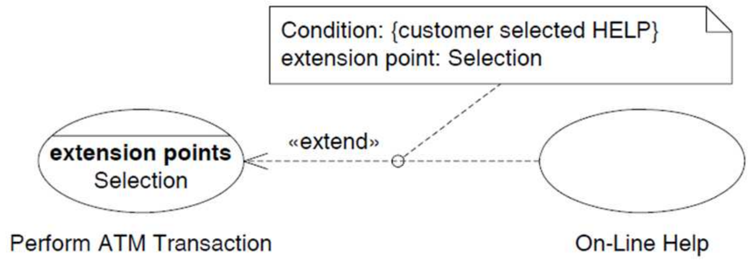
- Several relations between actors and use cases
  - Association, specialization, extension, inclusion





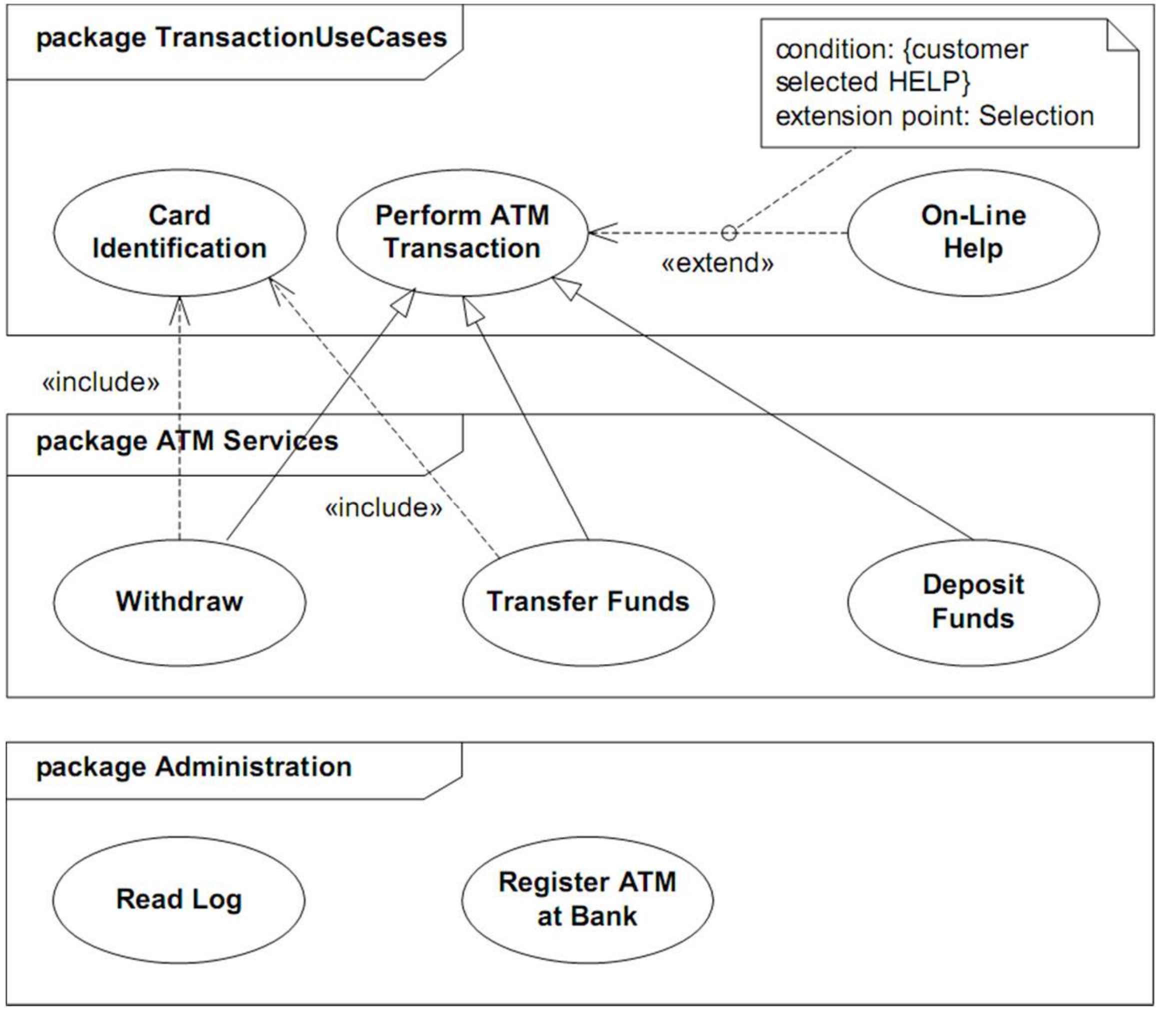
Cannot order without going through check-out





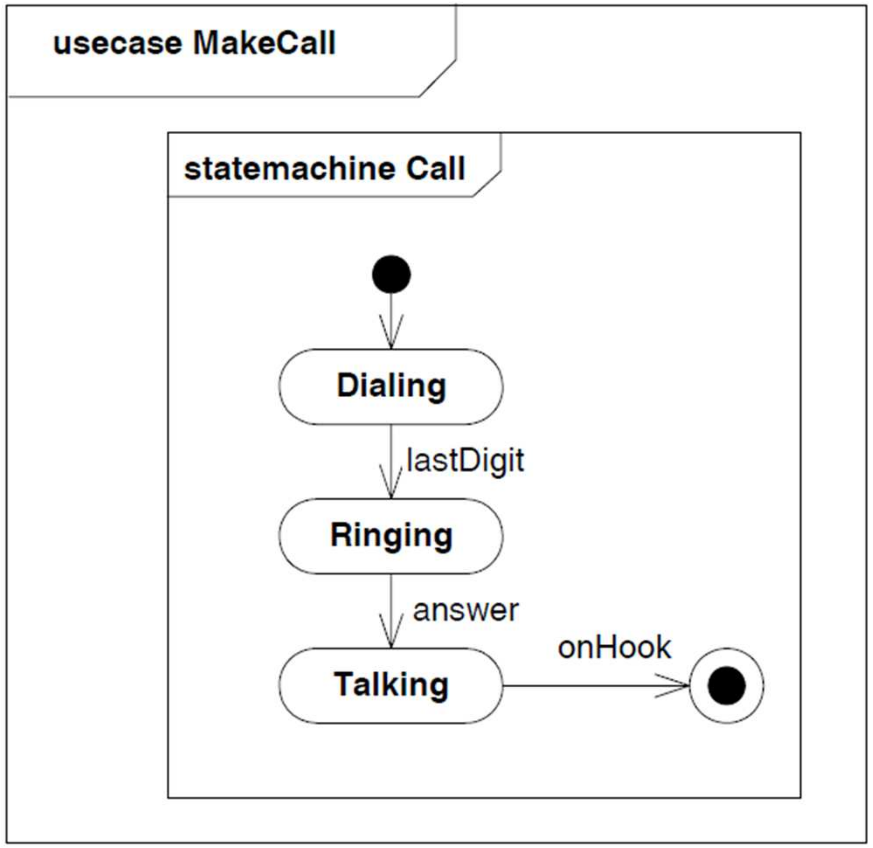
Use Case Modeling (3/4)

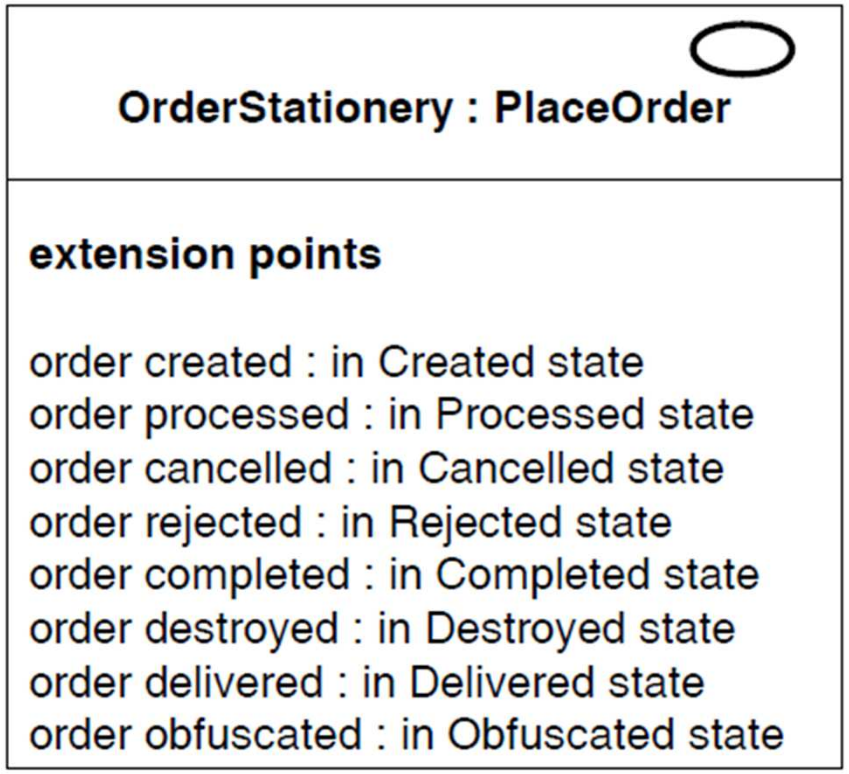
- Several relationships among actors and use cases
  - Association, specialization, extends, includes



Use Case Modeling (4/4)

- Several relationships among actors and use cases
  - Association, specialization, extends, includes





extension points

order created : in Created state
order processed : in Processed state
order cancelled : in Cancelled state
order rejected : in Rejected state
order completed : in Completed state
order destroyed : in Destroyed state
order delivered : in Delivered state
order obfuscated : in Obfuscated state

Semantics of Use Cases

- An execution of a use case is an occurrence of emergent behavior.
- Every instance of a classifier realizing a use case must behave in the manner described by the use case.
- Use cases may have associated actors, which describes how an instance of the classifier realizing the use case and a user playing one of the roles of the actor interact. Two use cases specifying the same subject cannot be associated since each of them individually describes a complete usage of the subject. It is not possible to state anything about the internal behavior of the actor apart from its communications with the subject.
- When a use case has an association to an actor with a multiplicity that is greater than one at the actor end, it means that more than one actor instance is involved in initiating the use case. **The manner in which multiple actors participate in the use case depends on the specific situation on hand and is not defined in this specification.** For instance, a particular use case might require simultaneous (concurrent) actions by two separate actors (e.g., in launching a nuclear missile) or it might require complementary and successive actions by the actors (e.g., one actor starting something and the other one stopping it).

Formal semantics of Use Cases

- Many attempts to define in a formal language one possible partial interpretation of the UML specification, in relation with other diagrams
  - F. Mokhati, M. Badri: *"Generating Maude Specifications From UML Use Case Diagrams",* in Journal of Object Technology, vol. 8, no. 2, March-April 2009, pp. 119-136
    - Analyze Use Cases in respect of their relationships to classes and collaboration diagrams
  - D. Sinnig, P. Chalin, F. Khendek: *"LTS semantics for use case models"*, SAC '09, pp. 365-370
    - Describe Use cases in LTS,
  - W. Grieskamp, M. Lepper, W. Schulte, N. Tillmann: *"Testable Use Cases in the Abstract State Machine Language"*. APAQS 2001: 167-172
  - W. Grieskamp, M. Lepper: "Using Use Cases in Executable Z". ICFEM 2000: 111-120
    - Describe Use cases in Z,
  - Peter Fröhlich, Johannes Link: "Automated Test Case Generation from Dynamic Models". ECOOP 2000: 472-492
    - Derive UML State Charts from a set of Textual Use Cases
  - S. Meng and B.K. Aichernig, "Towards a Coalgebraic Semantics of UML:Class Diagrams and Use Cases", UNU/IIST Report No. 272, January 2003
    - Analyze Use Cases

Behavioral or protocol

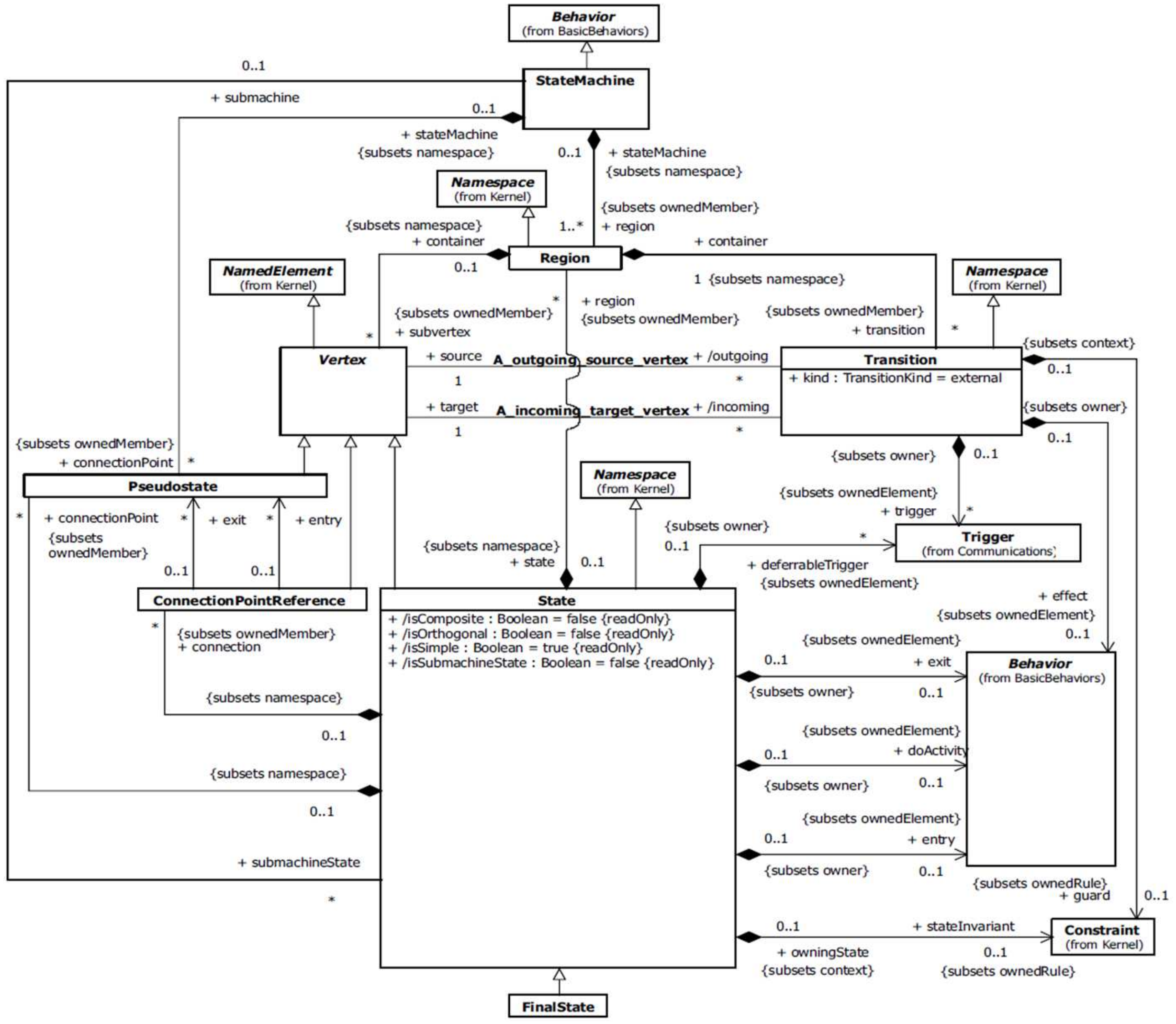
STATE MACHINES

UML State Machines

- **Behavioral** State Machines
  - Behavior of individual entities (e.g., class instances, operations, actions, use cases)
    - Associated with a classifier or a behavioral feature
  - Object-based variant of Harel statecharts
  - Behavioral states and behavioral transitions
- **Protocol** State Machines
  - Specialize Behavioral State Machines
  - Associated with a classifier (class, interface, port)
  - Usage Protocols
    - Legal transitions than a classifier can trigger, life cycle
    - Order of invocation of methods
  - Protocol states and protocol transitions

UML Behavioral State Machines

- Specialization of Behavior
  - Discrete behavior through finite state transitions


Behavior (from BasicBehaviors)
StateMachine
0..1
+ submachine
0..1
+ stateMachine {subsets namespace}
0..1
+ stateMachine {subsets namespace}
Namespace (from Kernel)
{subsets ownedMember} + region
{subsets namespace} + container
1..*
+ container
Region
0..1
1 {subsets namespace}
NamedElement (from Kernel)
Namespace (from Kernel)
*
{subsets ownedMember} + subvertex
*
+ region {subsets ownedMember}
{subsets ownedMember} + transition
*
{subsets context}
Vertex
+ source
A_outgoing_source_vertex
+ /outgoing
Transition
+ kind : TransitionKind = external
0..1
1
*
+ target
A_incoming_target_vertex
+ /incoming
{subsets owner}
1
*
0..1
{subsets ownedMember} + connectionPoint
*
Pseudostate
{subsets owner}
0..1
Namespace (from Kernel)
{subsets ownedElement} + trigger
*
*
+ connectionPoint {subsets ownedMember}
*
+ exit
*
+ entry
{subsets owner}
0..1
*
Trigger (from Communications)
0..1
0..1
{subsets namespace} + state
0..1
+ deferrableTrigger {subsets ownedElement}
ConnectionPointReference
State
+ /isComposite : Boolean = false {readOnly}
+ /isOrthogonal : Boolean = false {readOnly}
+ /isSimple : Boolean = true {readOnly}
+ /isSubmachineState : Boolean = false {readOnly}
+ effect {subsets ownedElement}
0..1
*
{subsets ownedMember} + connection
{subsets ownedElement}
0..1
+ exit
Behavior (from BasicBehaviors)
{subsets namespace}
{subsets owner}
0..1
0..1
{subsets ownedElement}
+ doActivity
{subsets namespace}
0..1
{subsets owner}
0..1
0..1
{subsets ownedElement}
0..1
+ entry
+ submachineState
{subsets owner}
0..1
*
{subsets ownedRule} + guard
0..1
0..1
+ stateInvariant
Constraint (from Kernel)
+ owningState {subsets context}
0..1
{subsets ownedRule}
FinalState

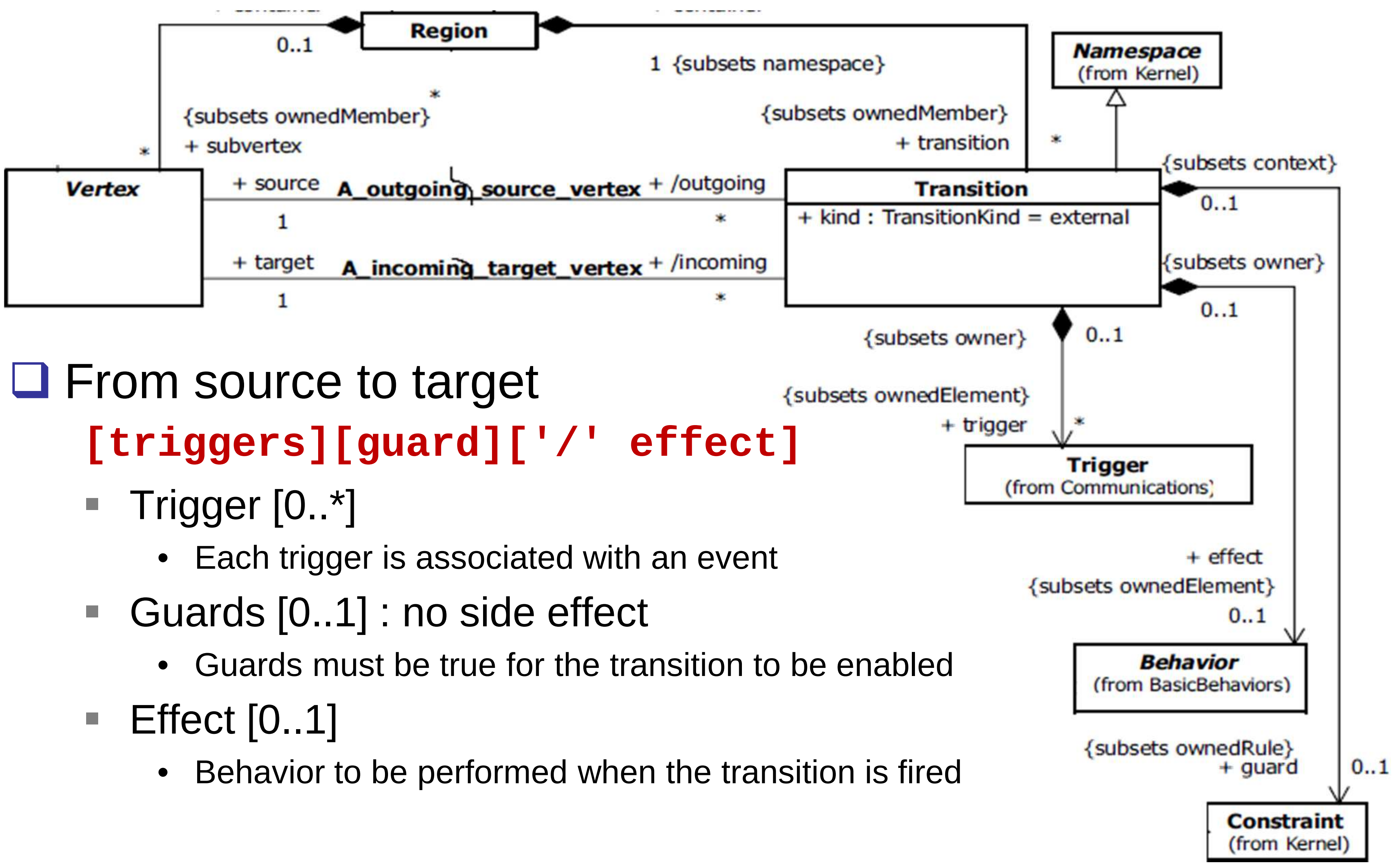
UML Behavioral State Machines

- Specialization of Behavior
  - Discrete behavior through finite state transitions
  - Traversal of graph of states connected with transitions
  - Transitions are triggered by event (occurrence)s
  - During the traversal, the SM executes a series of activities
- Context
  - Usually a *behaviored* classifier
  - Defines the signal and call **triggers** available [trigger]
  - Defines the attribute and operations are available in activities [effect]
- Behavioral features and methods
  - A SM can be the method associated with a behavioral feature (operation, reception)
    - Parameters of behavioral feature => parameter of the state machine

Regions and transitions

- Regions
  - A SM contains one or more regions
  - Regions contain vertices and transitions
- Transition
  - At each step, at most one transition is selected and fire
  - Conflicting transitions
    - A implicit priority is given depending on the state hierarchy
      - The lower in the hierarchy, the higher the priority
  - Transition selection : maximal set of transitions such that
    - All transitions are enabled
    - There is no conflicting transition within the set
    - There is no transition outside the set that has higher priority

Behavioral transitions



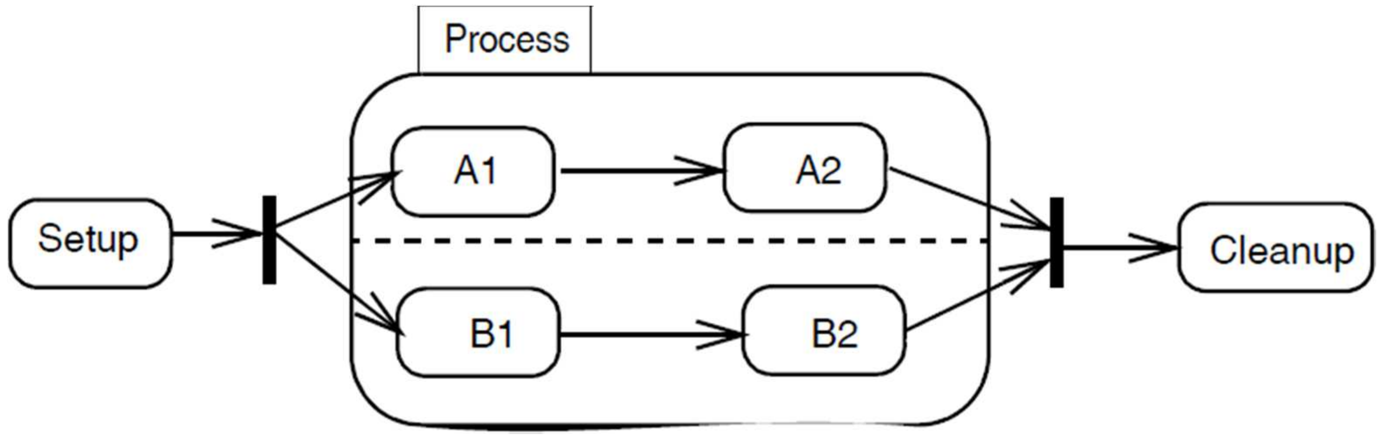
- From source to target

  `[triggers][guard]['/' effect]`

  - Trigger [0..*]
    - Each trigger is associated with an event
  - Guards [0..1] : no side effect
    - Guards must be true for the transition to be enabled
  - Effect [0..1]
    - Behavior to be performed when the transition is fired

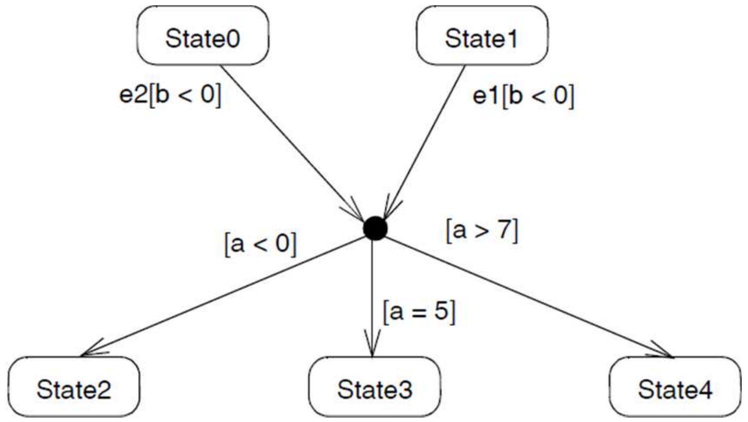
Pseudo-states (1/3)

- Pseudo states are transient vertices
  - Connect multiple transitions into more complex state transition paths
    - ● Initial [0..1]: default state of a composite, no trigger, no guard
    - H* deepHistory [0..1]
    - H shallowHistory [0..1]
    - join: join transitions from orthogonal regions, no trigger, no guard on the entering transitions
    - fork: split one transition into several ones, no trigger and no guard on the outgoing transitions

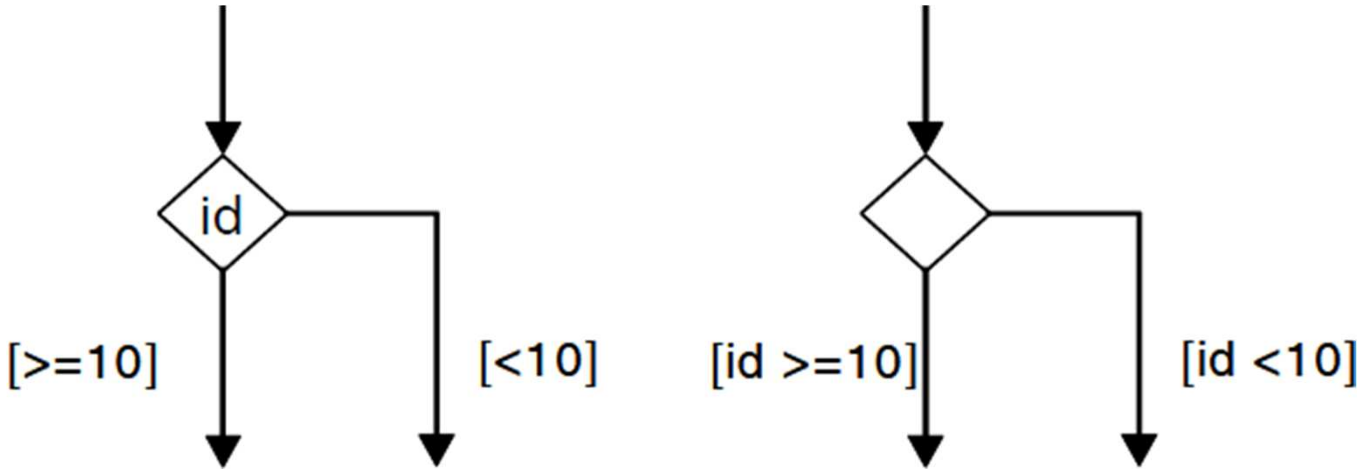


Pseudo-states (2/3)

- Pseudo states are transient vertices
  - Connect multiple transitions into more complex state transition paths
    - junction: semantic-free vertices

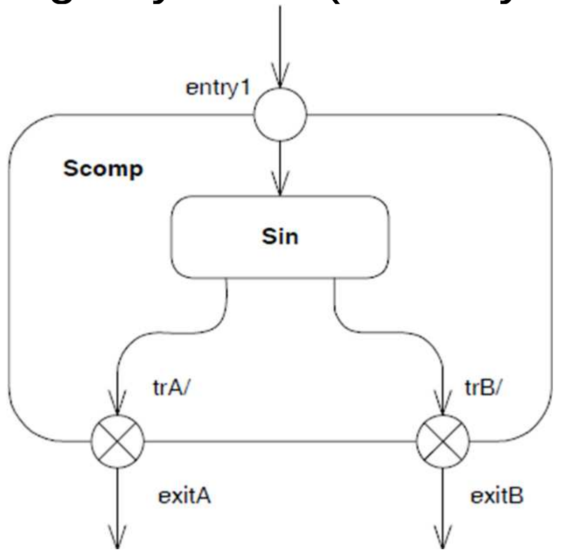


    - choice : dynamic conditional branch, select one amongst the outgoing guards evaluated to true, should have [else]



Pseudo-states (3/3)

- Pseudo states are transient vertices
  - Connect multiple transitions into more complex state transition paths
    - ○ Entry point: at most one a single transition to a vertex within the same region (allows for submachines)
    - ⊗ Exit point: entering an exit point within any region implies the exit of the composite state or submachine state
    - ✕ Terminate: the execution is terminated without performing exit actions or exiting any state (DestroyObjectAction)



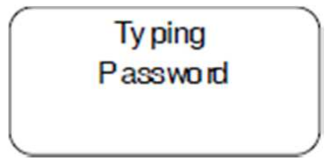
Final state

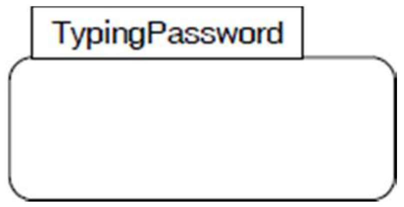
- Not a pseudo-state !
  - At most one per region
  - When a final state is reached, the enclosing region is completed
  - A state machine is completed when all its regions are completed
- Completion transitions (whose target is a final state)
  - Unlabeled transitions
- Notation:

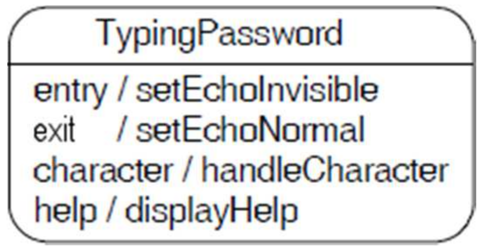
◉

Behavioral **Simple** States

- **Simple** states
  - Name (String)
  - Entry/do/Exit actions
- Composite states
- Submachine states

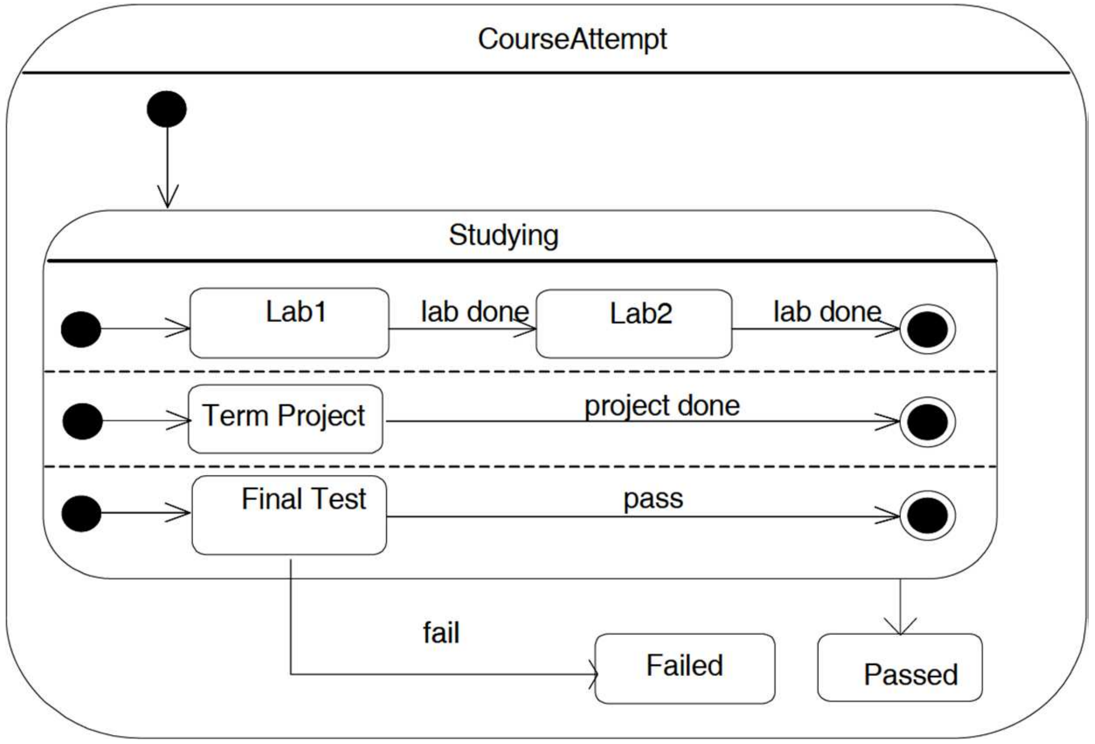






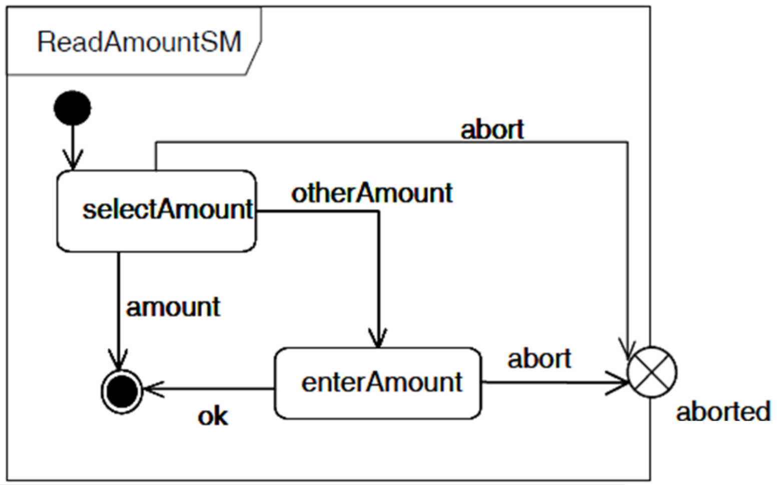
Behavioral **composite** states

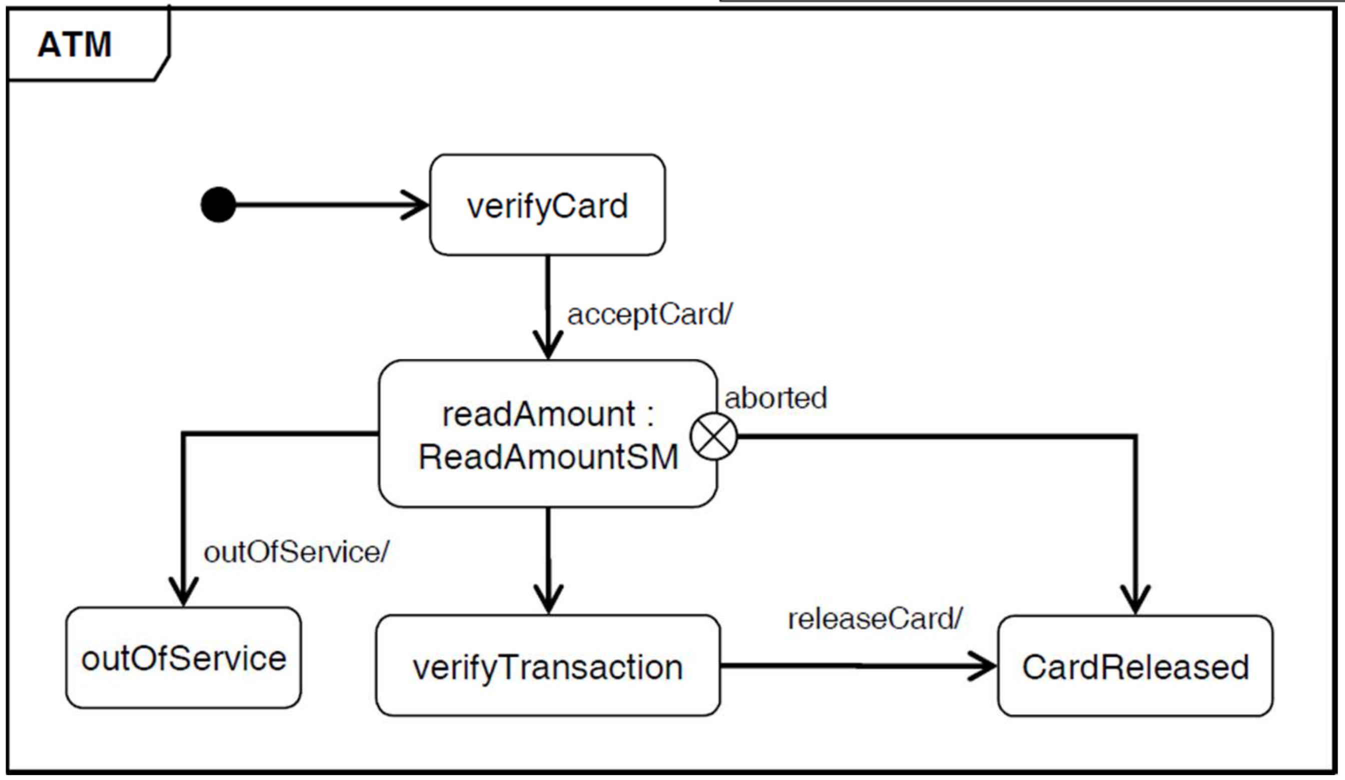
- Simple states
- Composite states
  - Either contains one region
  - Or decomposed into two or more orthogonal regions
- Submachine states



Behavioral **Submachine** States

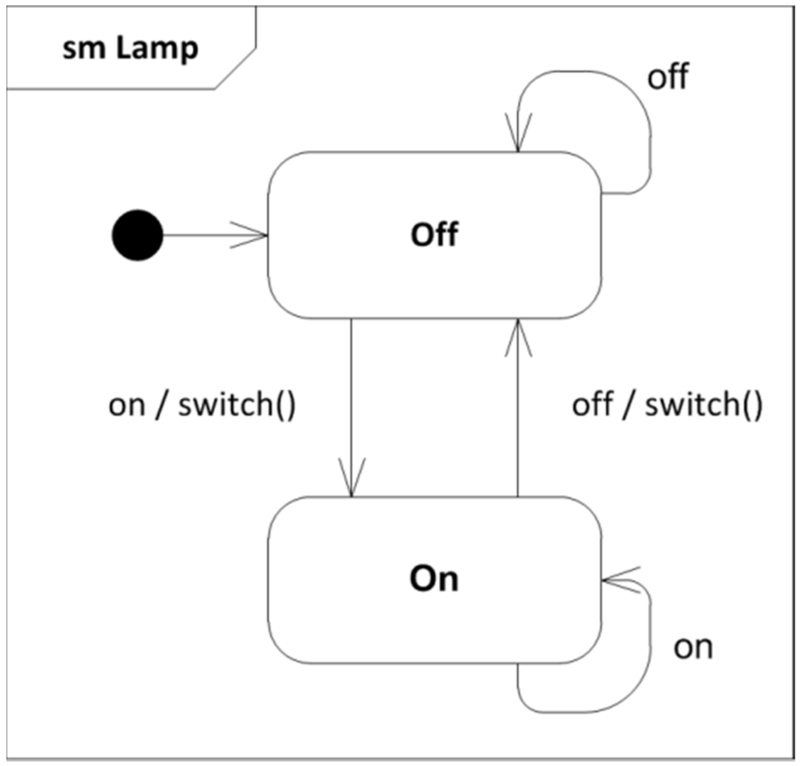
- Simple states
- Composite states
- **Submachine** states



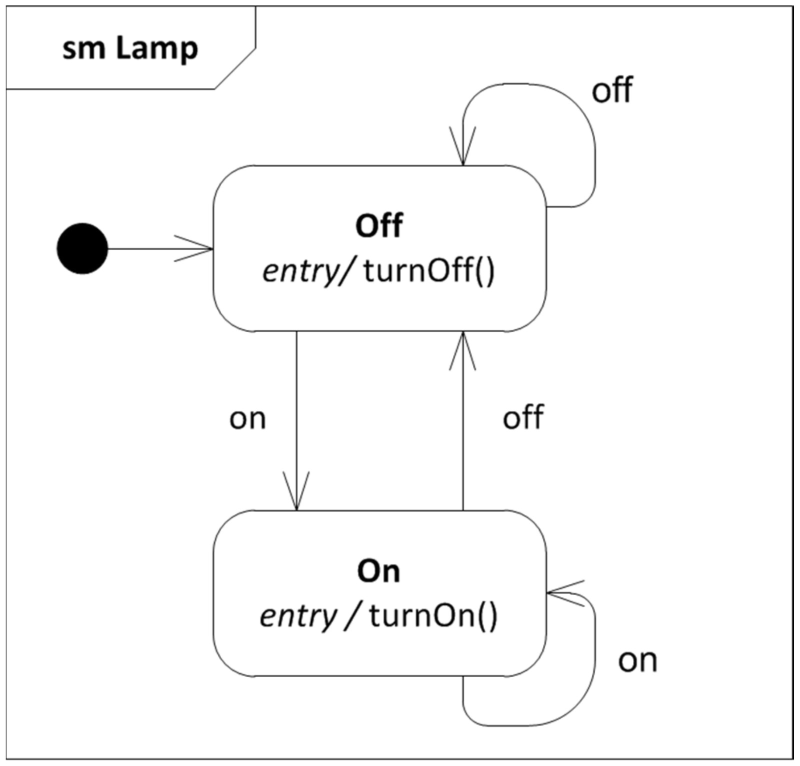


Mealy vs. Moore Machines

Mealy machines

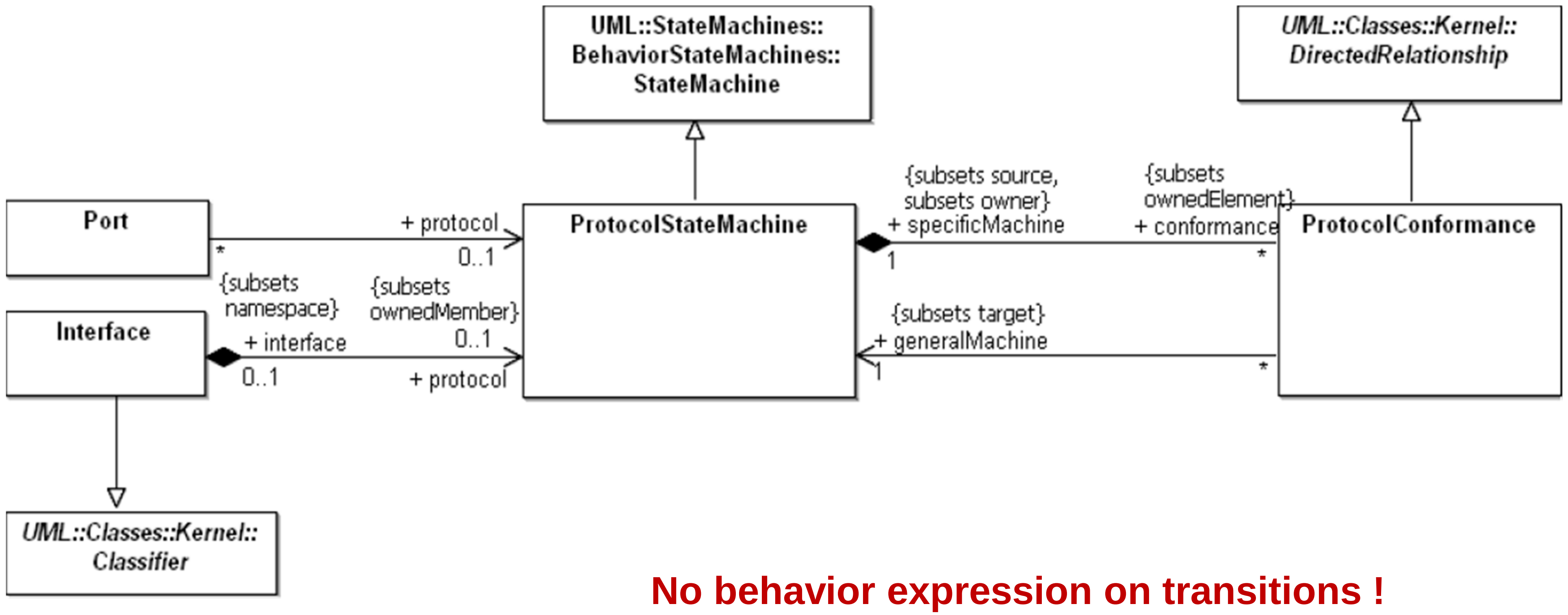


Moore Machines



UML Protocol State Machines

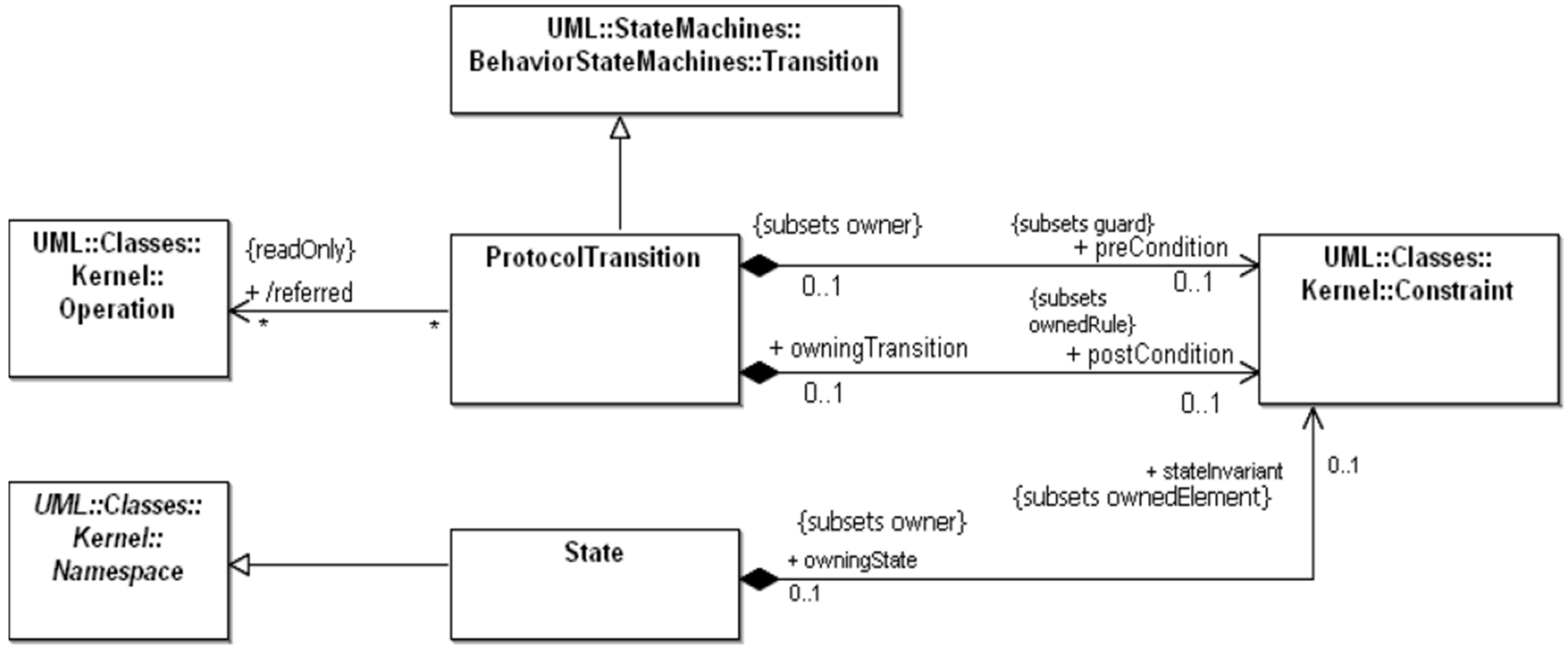
- Specialization of StateMachine
  - Legal transitions than a classifier can trigger, life cycle



No behavior expression on transitions !
No entry/do/exit within states !
No deep/shallow history !

Protocol States & Protocol transitions

- A protocol states only contains protocol states and protocol transitions



Protocol Transition

- Transitions of protocol state machines

  `[pre-condition] trigger / [post-condition]`

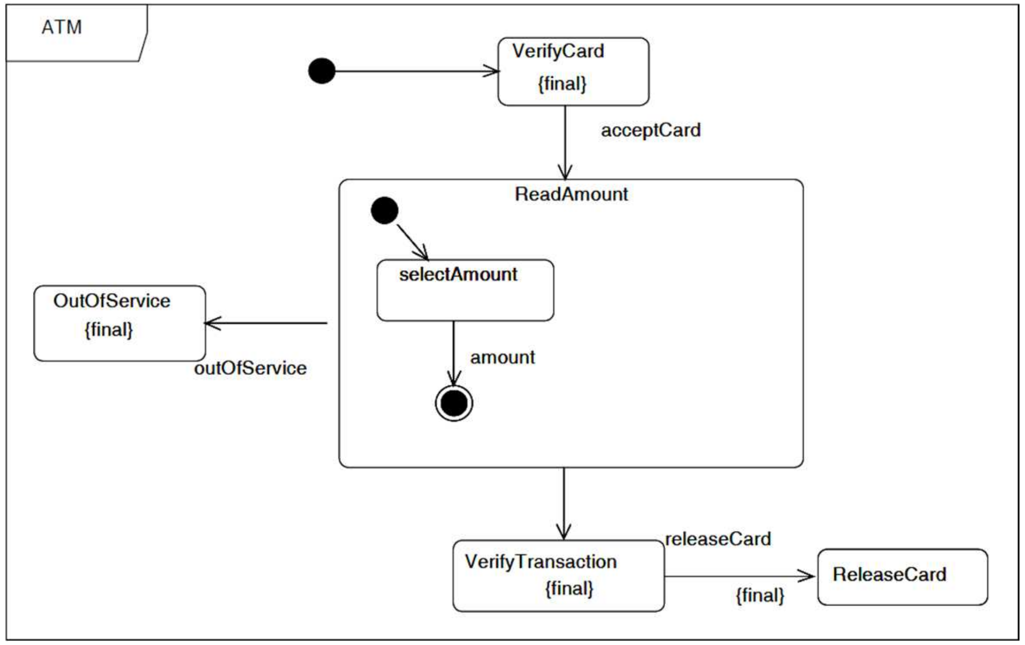
  - No effect action
    - When the trigger is a call action, the effect is the operation called
    - Otherwise, no effect
      - only specifies that a given event can be received under a specific state and pre-condition, and that a transition will lead to another state under a specific post-condition, whatever action is made
  - Unexpected event reception
    - Current state, state invariant, and pre-condition
      - **Pre-condition violation**: can be ignored, rejected or deferred
  - Unexpected behavior
    - Wrong final state, final state invariant or post-condition
      - **Error of the implementation**

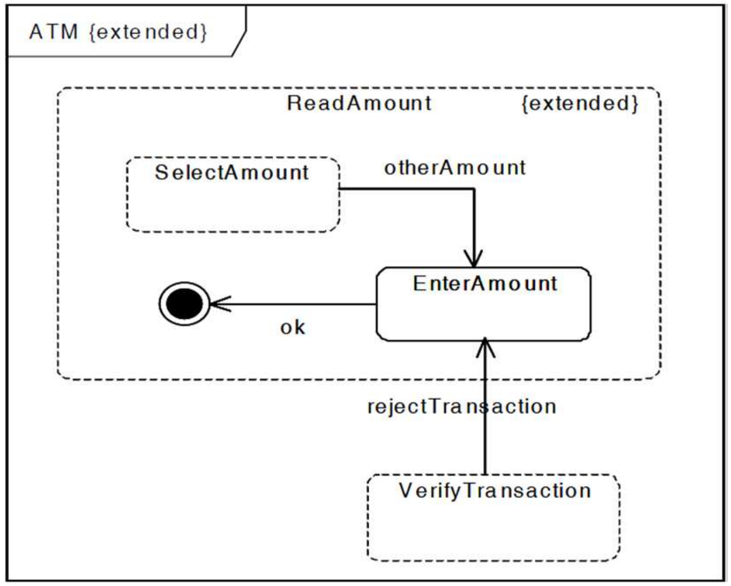
Protocol State

- Expose a stable condition of its context classifier
  - stateInvariant [0..1]
    - Specifies conditions that are always true when this state is the current state

State Machine redefinition

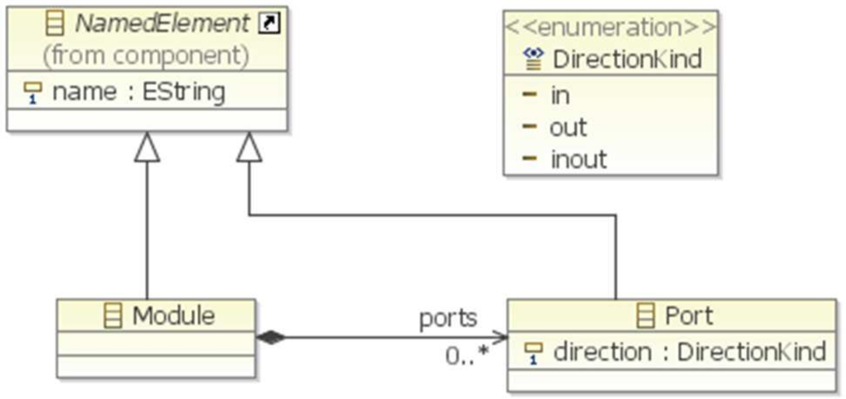
- State Machines can be extended





UML 2.x

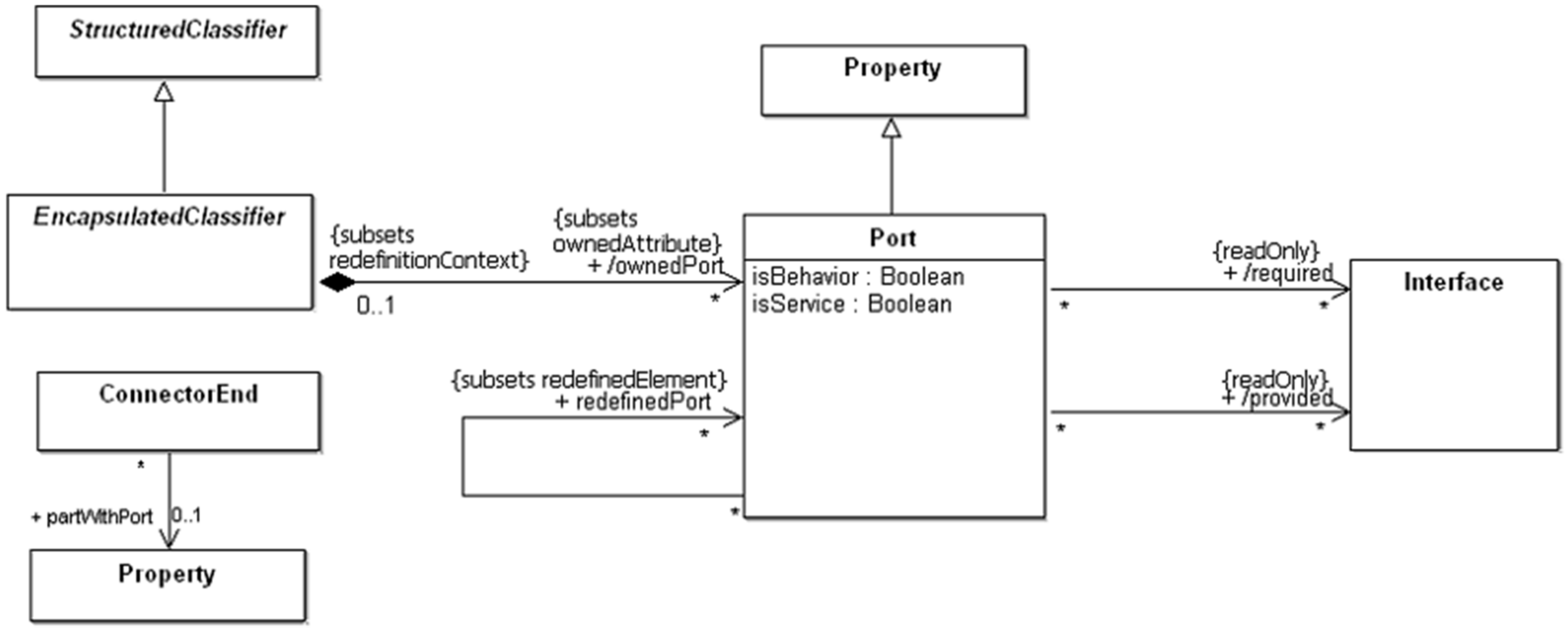
Introduction to the profiling mechanism

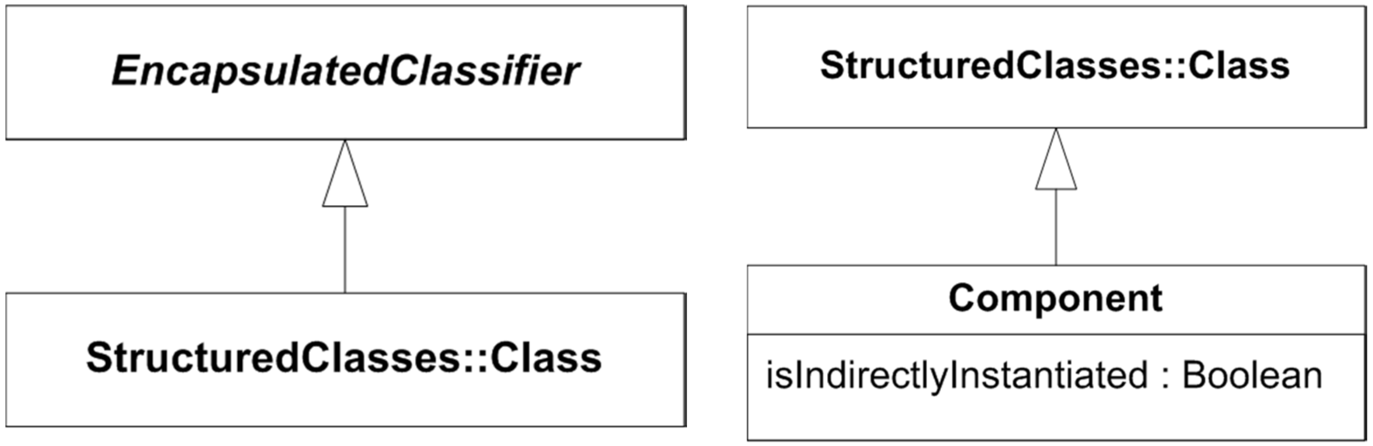


Our Component Model

- Can we model that simple DSL with UML ?
  - No !
- UML has
  - The notion of component
  - The notion of port
  - But not the notion of direction on ports
  - Let us have a look at UML meta model

UML 2.2 – Components & Port





Light- and heavy-weight extensions

- Standard constraints are possible using
  - Stereotype
  - Stereotype properties (formerly Tagged value)
- Profiles: specific 'light-weight' extensions of the UML
- 'heavy-weight' extensions require to alter the UML meta-model
  - Generally use a full meta-modelling approach, DSL
  - No specific tool support for the UML

Profiling the UML for a Domain

- Advantages of UML Profiles
  - Reuse of language infrastructure (tools, specifications)
  - Require less language design skills
  - Allow for new (graphical) notation of extended stereotypes
  - A profile can define model viewpoints
    - E.g., UML activity diagram extended to specify multitask behavior

- Disadvantage
  - Constrained by the UML metamodel

Profiles and stereotypes

- Profiles
  - Define limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain.
  - Consist of stereotypes that extend the metamodel classes (metaclasses).

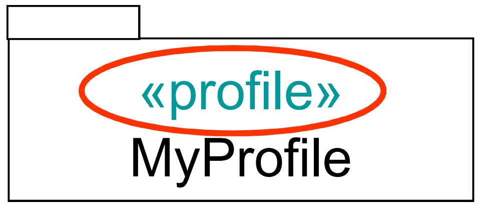
- Stereotypes
  - Define how a specific metaclass may be extended
  - Provide additional semantic information, but only for:
    - Semantic restriction or clarification of existing concept
    - New features (but compatible with exiting one!)
  - Ensure introduction of domain specific terminology
    - E.g., EAST-ADL2, a UML profile for automotive ECUs (http://www.atesst.org)
  - May have values that are usually referred to as tagged values

What a profile can do ?

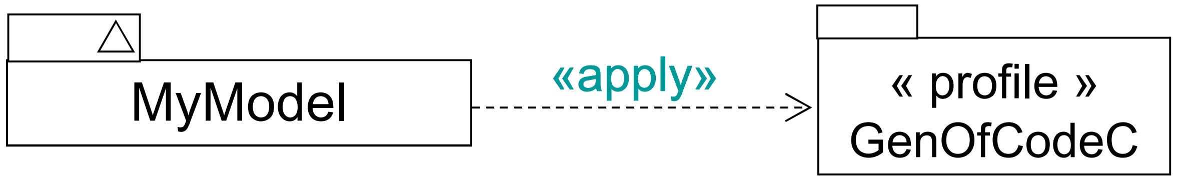
- Give a terminology adapted to a particular platform
  - e.g. IP or VC instead of Class/Component
- Give a syntax for constructs
- Give a different notation for existing symbols
  - e.g. Use an icon for a processor instead of a generic node
- Give a semantics unspecified in the metamodel
  - e.g. What happens when two signals are received simultaneously (priority, aggregation, ...)
- Add semantics (e.g. Timer, Clock, Continuous time)
- Define mapping rules (e.g. Between two platforms)

Profile Notation

- Profile is a stereotyped package



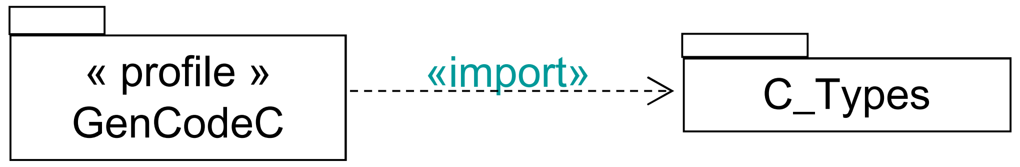
- Applying a profile
  - All extensions are then available for modeling



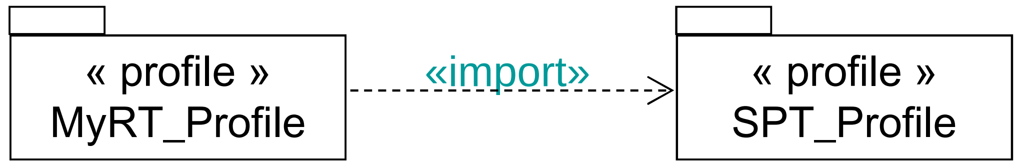
  - If multiple profiles are applied:
    - All profiles and the model should conform to the same MM
    - They must not have conflicting constraints
    - In case of naming conflict, use namespace notation
      - <ProfileName>::<StereotypeName>
      - e.g. «MyProfile1::name» & «MyProfile2::name»

Importing external packages

- A profile package may import external packages
  - "Normal" packages (including Model libraries)
    - e.g. external pkgs defining specific types for a profile

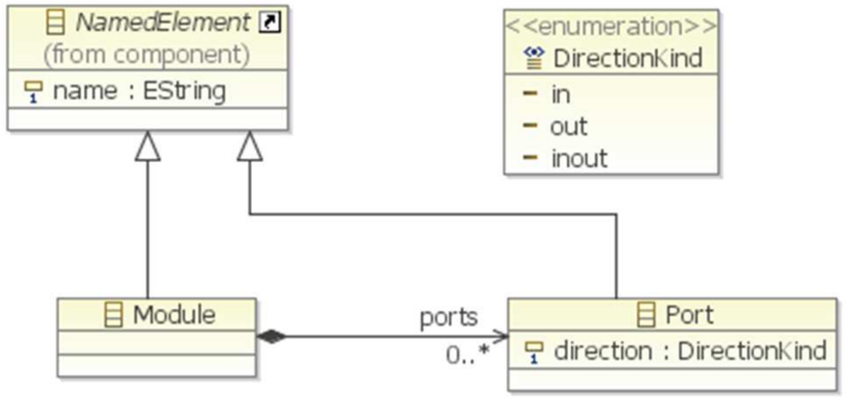


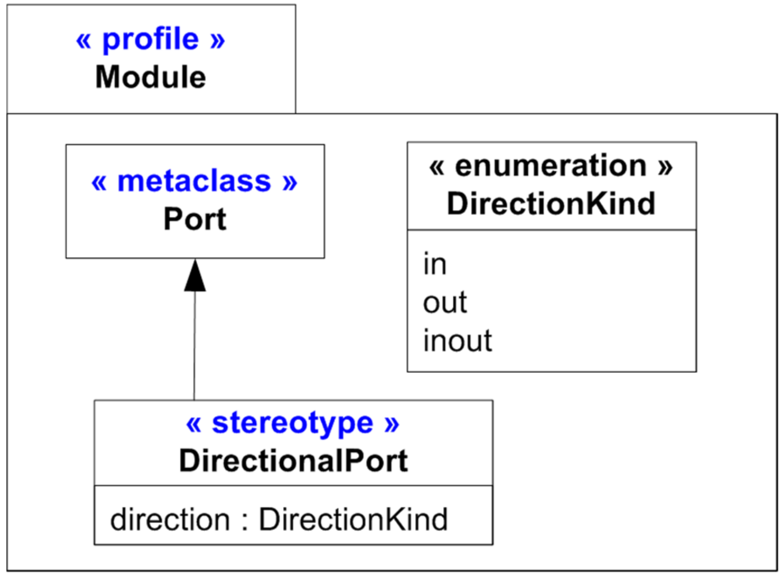
  - "Profile" packages



  - All imported elements may be used in pkgs applying the profile

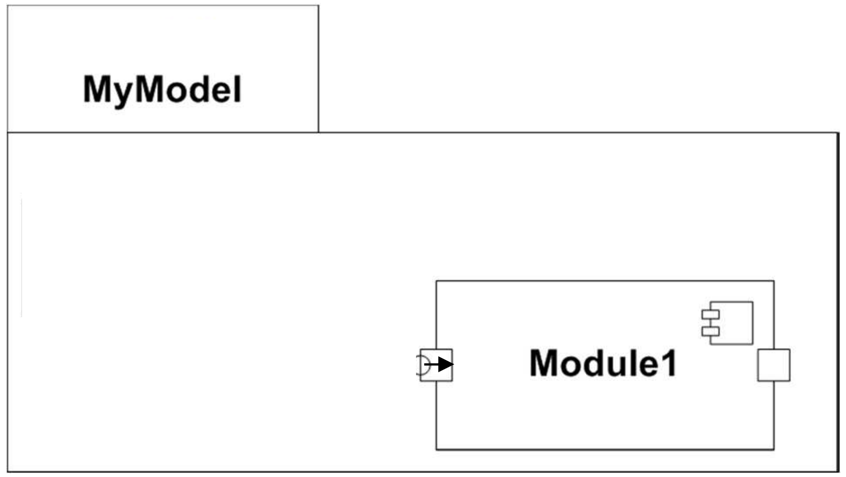
Defining a stereotype: DirectionalPort





- The stereotype DirectionalPort refers to a Port
- It has only one property
  - The direction, whose type is specified by an Enumeration
- An Extension point from the stereotype to the extended metaclass

Example: applying a profile



- The Profile must be applied to the model
  - All stereotypes defined in the profile can then be applied to model elements according to the type of the extended metaclass
- Properties of the stereotype can be shown together with the stereotype
  - Within brackets
  - In a UML note
  - In a specific compartment