

DSL and UML profiles

Sébastien Gérard

Laboratory of model driven engineering
for embedded systems (LISE)

Modeling Wizards

1st International Master Class on MDE
Oslo, Norway, 2010-09-06

Thanks to Bran Selic from Malina Software
and Yann Tanguy from CEA LIST that have
provided materials for this presentation.



Sebastien.Gerard@cea.fr



Standards have traditionally provided major boosts to technological progress !

- **Standards benefits to users**

- Because it enable vendor independence
 - » Users have a choice of different tool vendors (no vendor "tie-in")
- Forces vendors into competing and improving their products

- **The Object Management Group (OMG) has created the Model-Driven Architecture initiative:**

- A comprehensive set of standards in support of MBE including standard modeling languages: **UML2, MARTE, SysML, QVT, SPEM, BPMN, etc.**





• Abstract Syntax

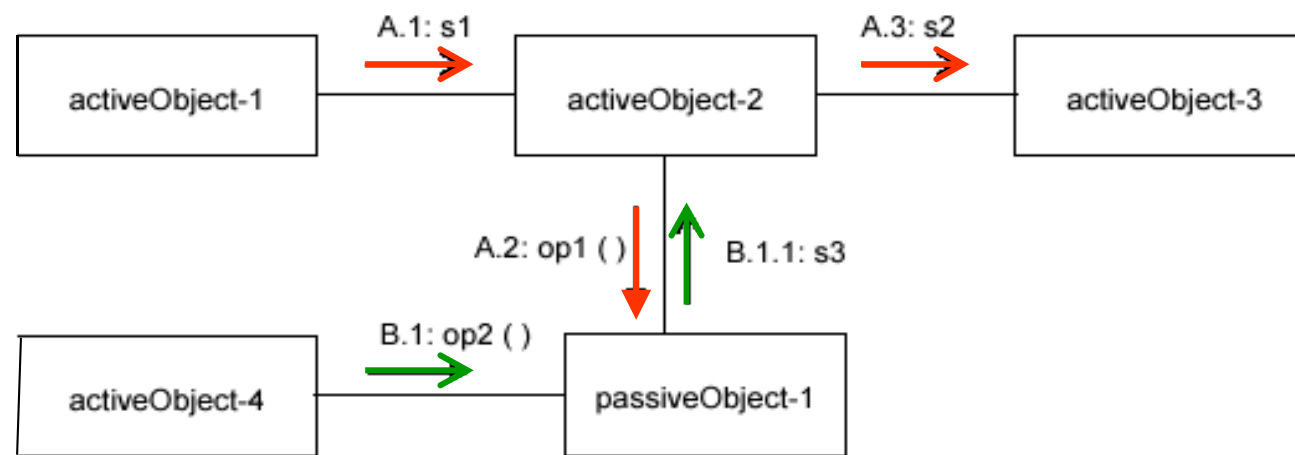
- Described by a metamodel
 - » Concepts, their features, and mutual relationships
- Additional constraints (OCL or English), e.g.:
 - » English: If a Class is concrete, all the Operations of the Class should have a realizing Method in the full descriptor.
 - » OCL: `not self.isAbstract implies self.allOperations->forAll (op | self.allMethods->exists (m | m.specification-> includes(op)))`
- Language Semantics
 - » The meaning of UML models and the concepts used to express them
 - » English (+ fUML model + mathematical model)

• Concrete Syntax

- Notation (diagrams, text, tables, graphical representation)
- UML concrete syntax definition is incomplete and informally defined

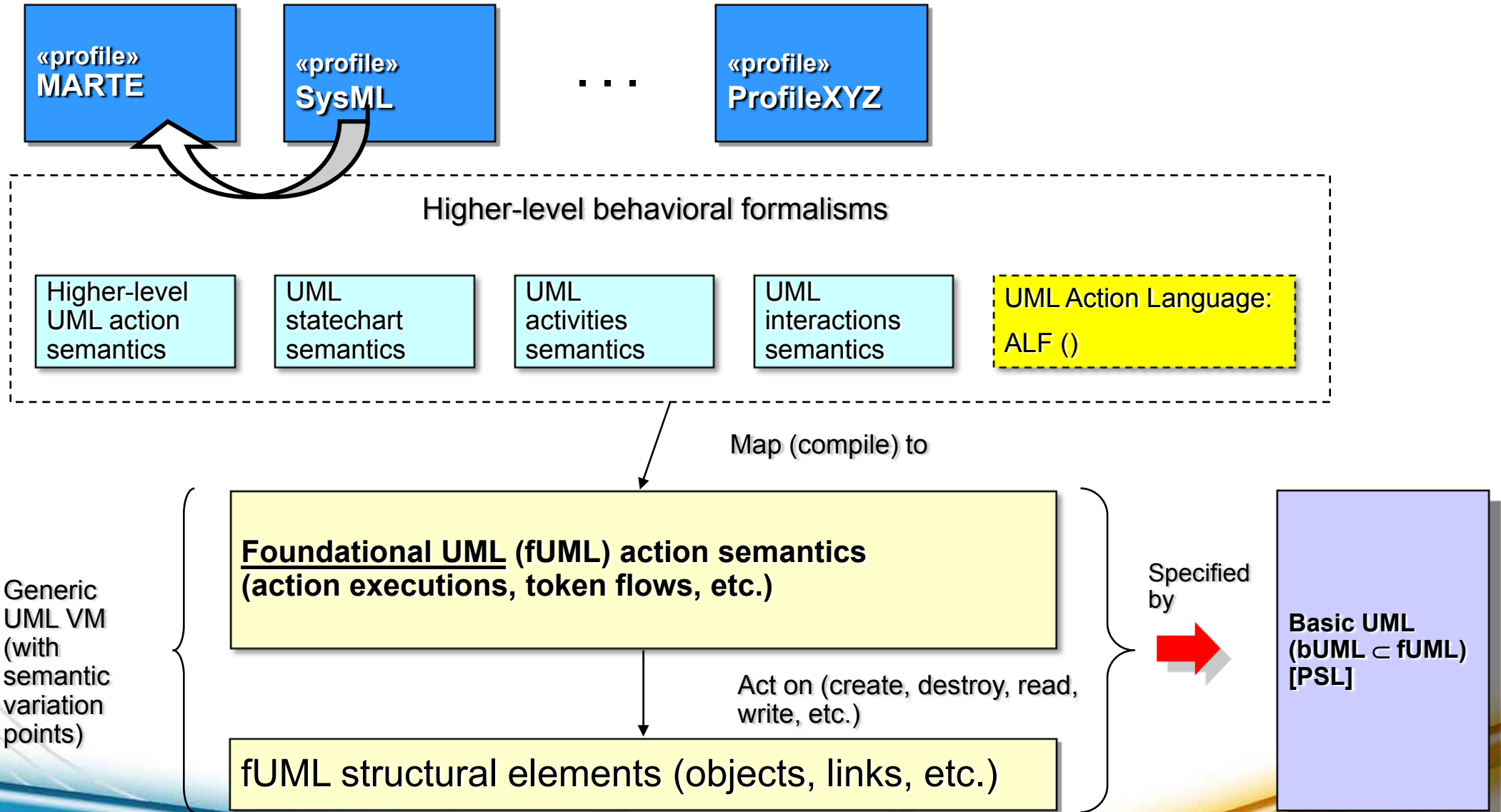


- Causality model = specification of how things happen at run time.
- Principles
 - Objects respond to messages that are generated by objects executing communication actions.
 - When messages arrive, the receiving objects eventually respond by executing the behavior that is matched to that message, and may consequently send other messages, etc...
- Example:





- UML 2 has semantics?





Structure Diagrams

- Class Diagram
- Component Diagram
- Object Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram
- Profile Diagram



Behavior Diagrams

- State Machine Diagram
- Activity Diagram
- UseCase Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram



Interaction Diagrams



- **For modelling fine-grained behavioural phenomena which manipulates and accesses UML entities (objects, links, attributes, operations, etc.)**
 - E.g. create link, write attribute, destroy object
 - A kind of UML “assembler”
- **The UML standard defines:**
 - A set of actions and their semantics
 - » i.e., what happens when the actions are executed).
 - A method for combining actions to construct more complex behaviours
- **The UML standard does not include** a notation for individual kinds of actions
 - But at last OMG meeting (one week ago!), OMG Architecture Board has endorsed a new standard: “**Concrete Syntax for a UML Action Language**”.



- **Capabilities covered:**

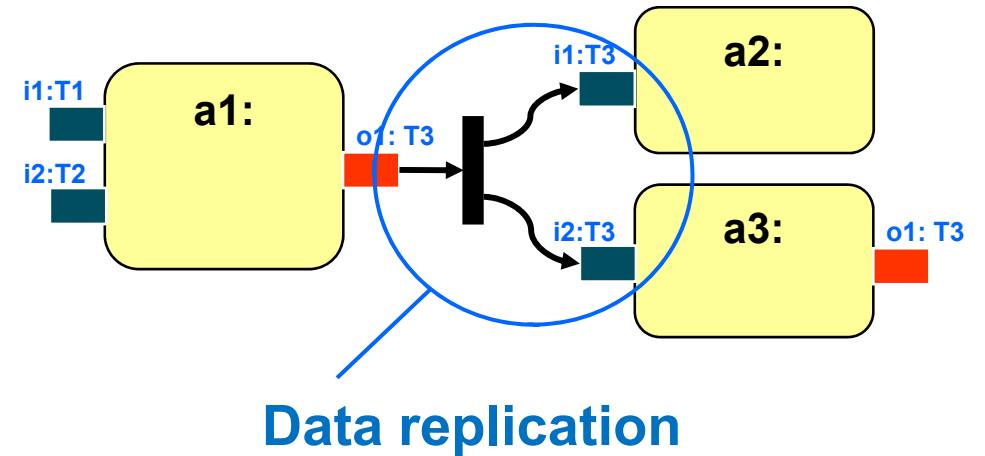
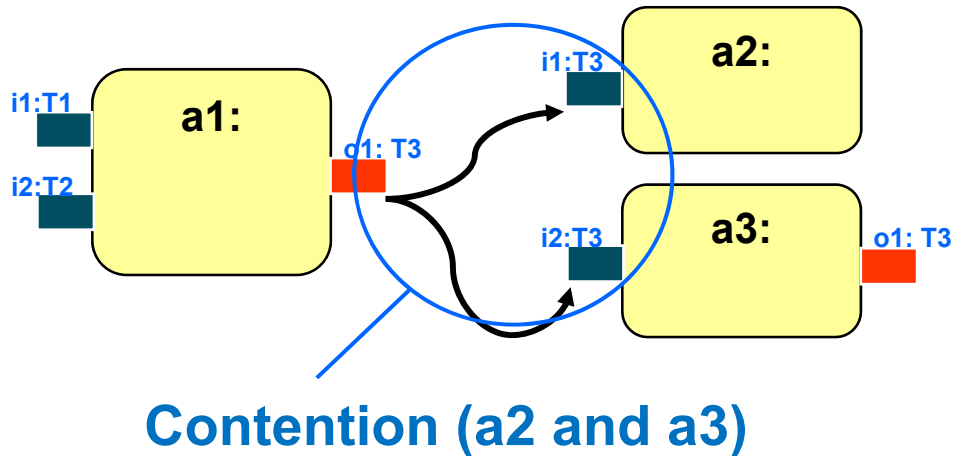
- Communication actions (send, call, receive,...)
- Primitive function action
- Object actions (create, destroy, reclassify,start,...)
- Structural feature actions (read, write, clear,...)
- Link actions (create, destroy, read, write,...)
- Variable actions (read, write, clear,...)
- Exception action (raise)

- **Capabilities not covered:**

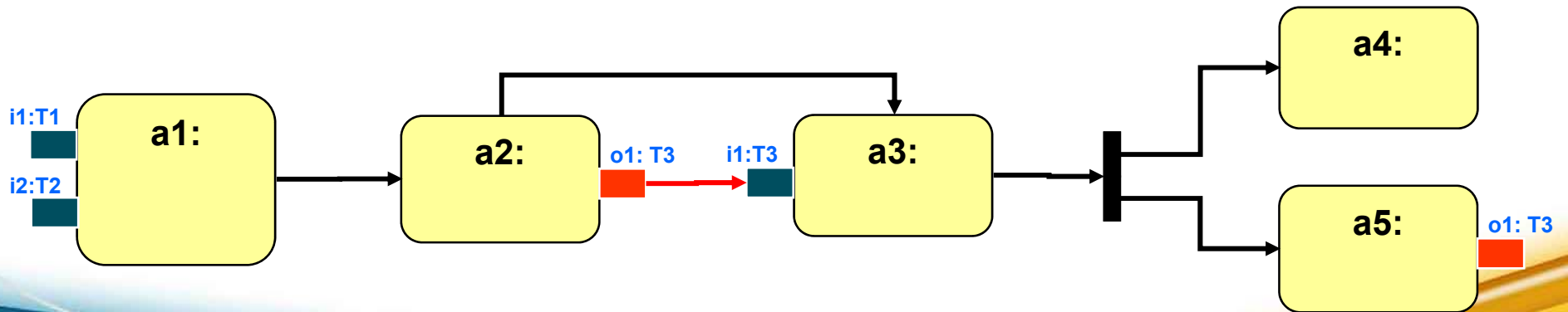
- Standard control constructs (IF, LOOP, etc. – handled through Activities)
- Input-output
- Computations of any kind (arithmetic, Boolean logic, higher-level functions)



- Data flow MoC: output to input connections**

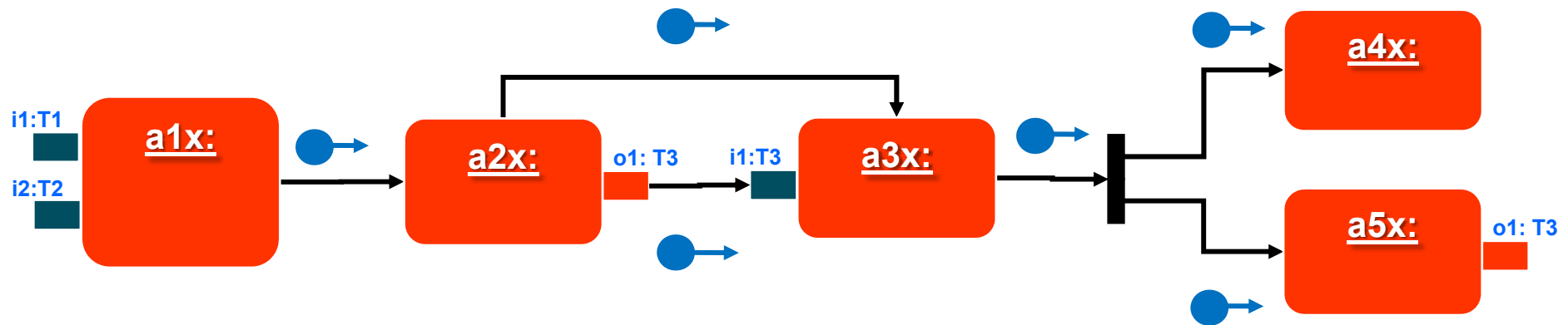


- Control flow MoC: identifying successor actions**





- Execution order can be modeled as an exchange of data/control “tokens” between nodes

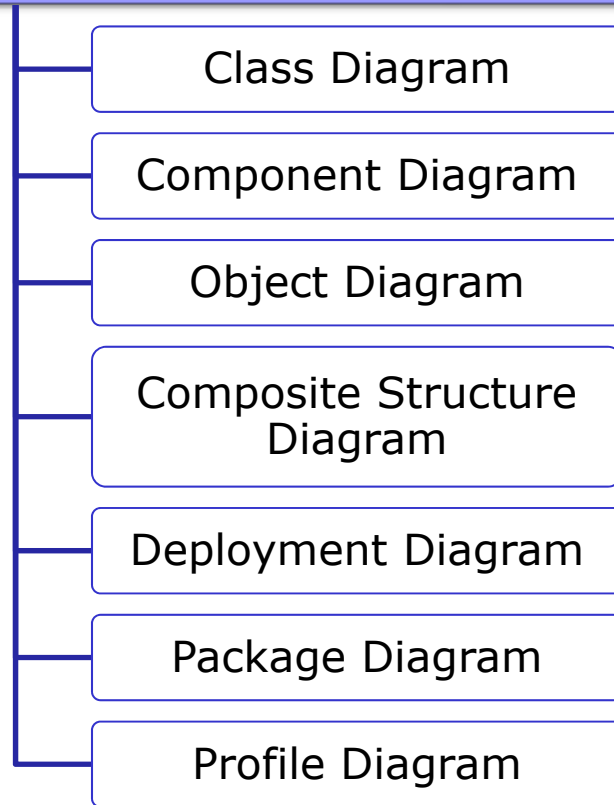


- **General execution rules:**

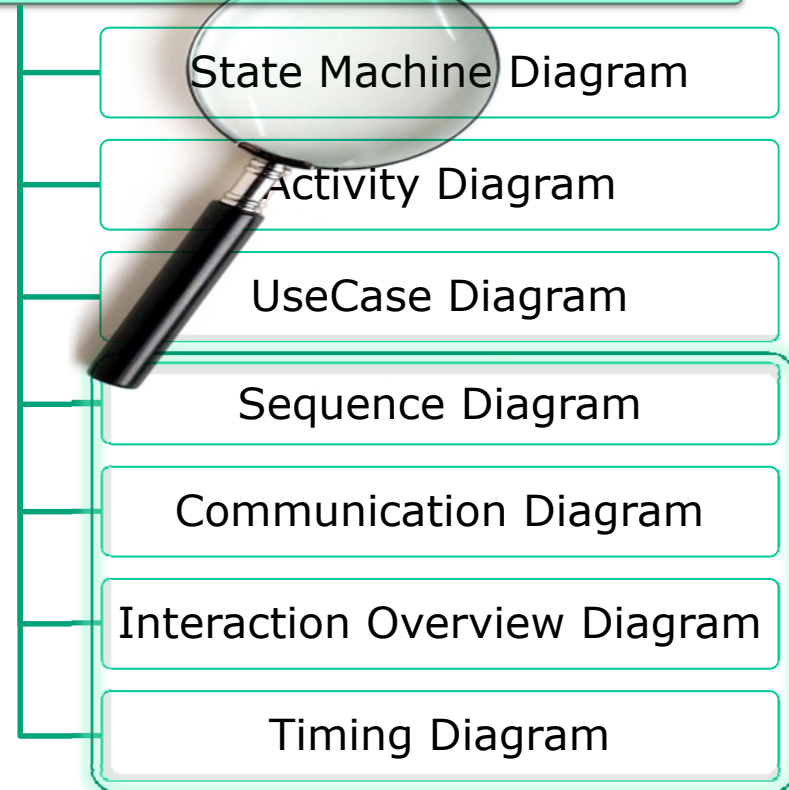
- **All** tokens have to be available before actions execute
- Tokens are offered only after action execution completes



Structure Diagrams



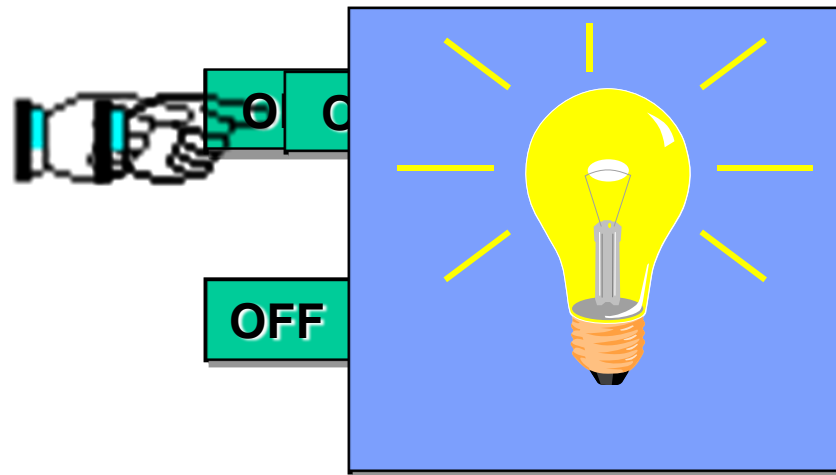
Behavior Diagrams



Interaction Diagrams



- A machine whose output behavior is not only a direct consequence of the current input, but of some past history of its inputs
- Characterized by an internal state which represents this past experience

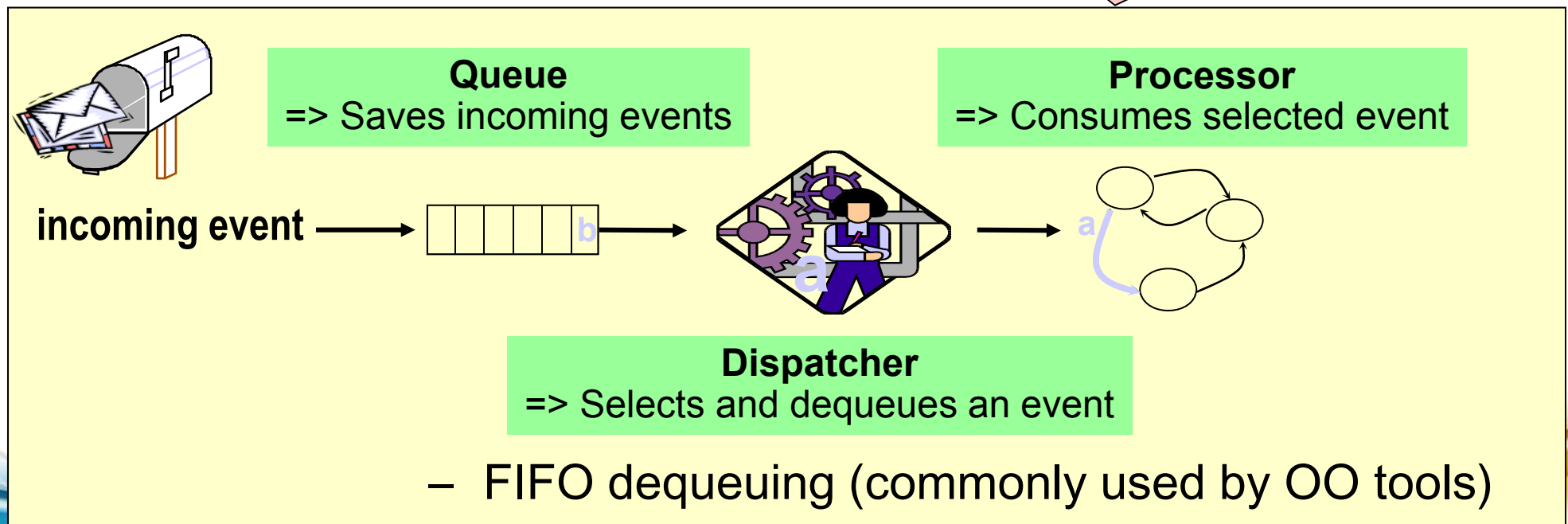




• UML state machines = hypothetical machine that:

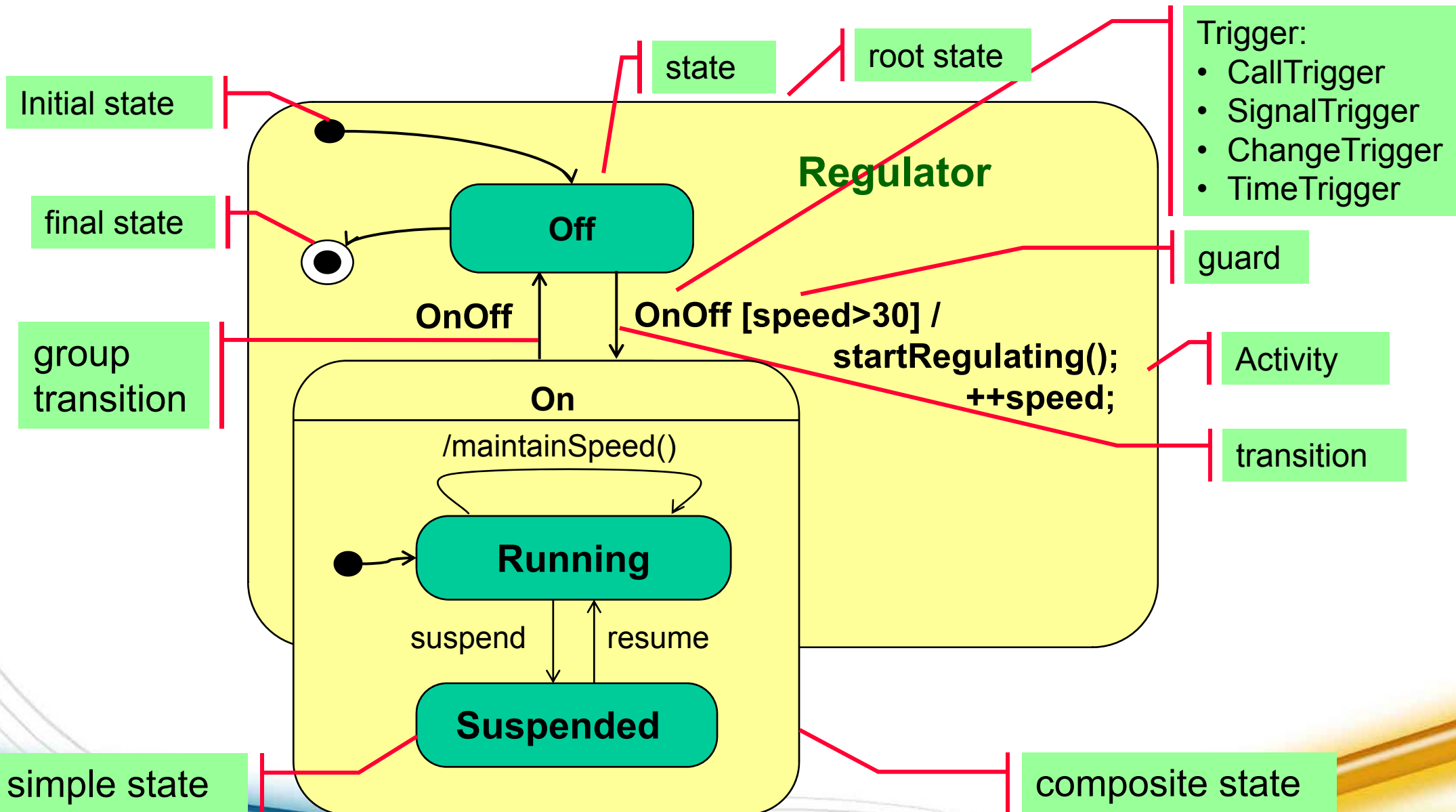
- ⇒ queues event
- ⇒ dispatches event
- ⇒ processes event

**Run-To-Completion:
only one event at a time**



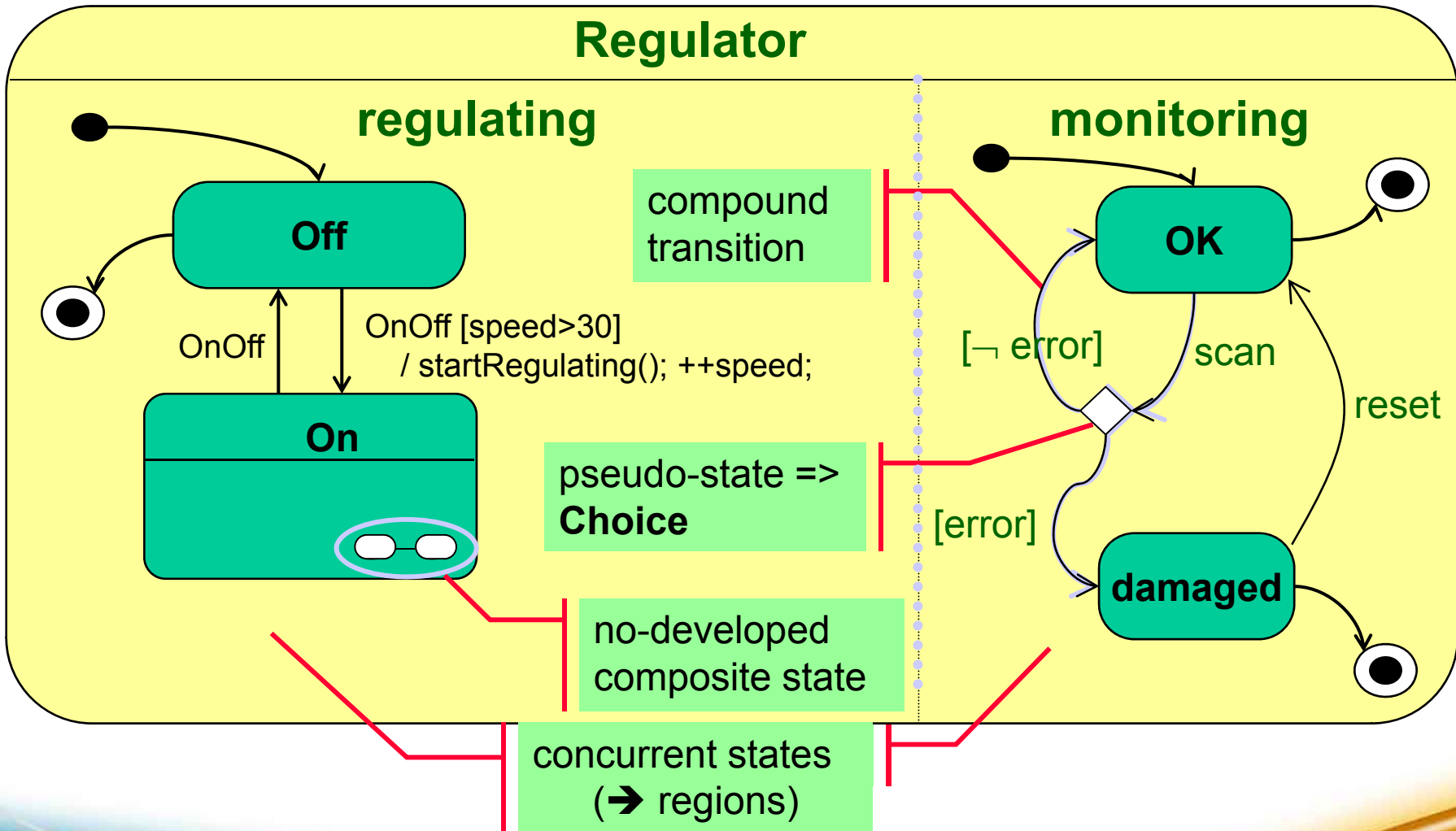


State diagram



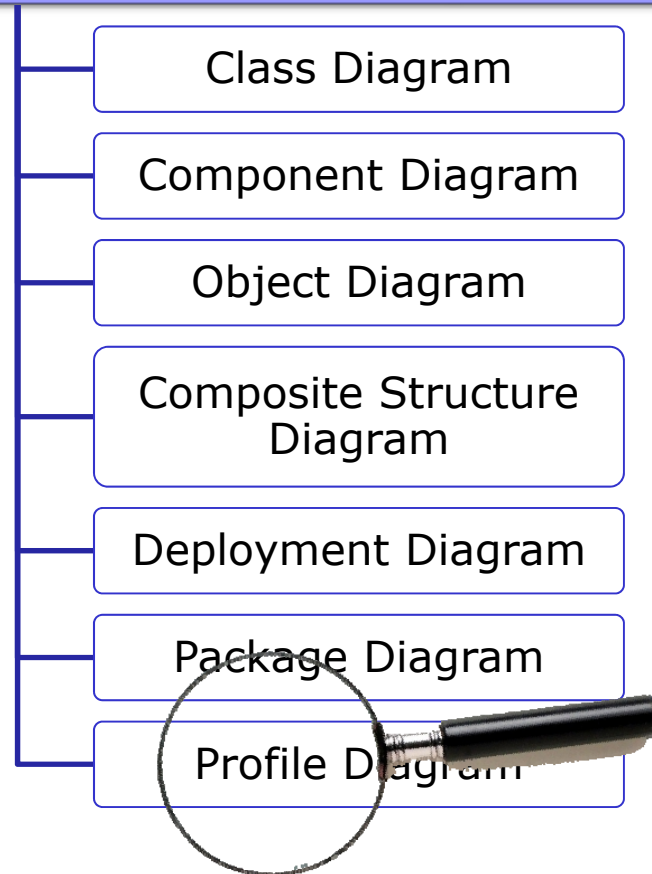


State diagram (seq.)

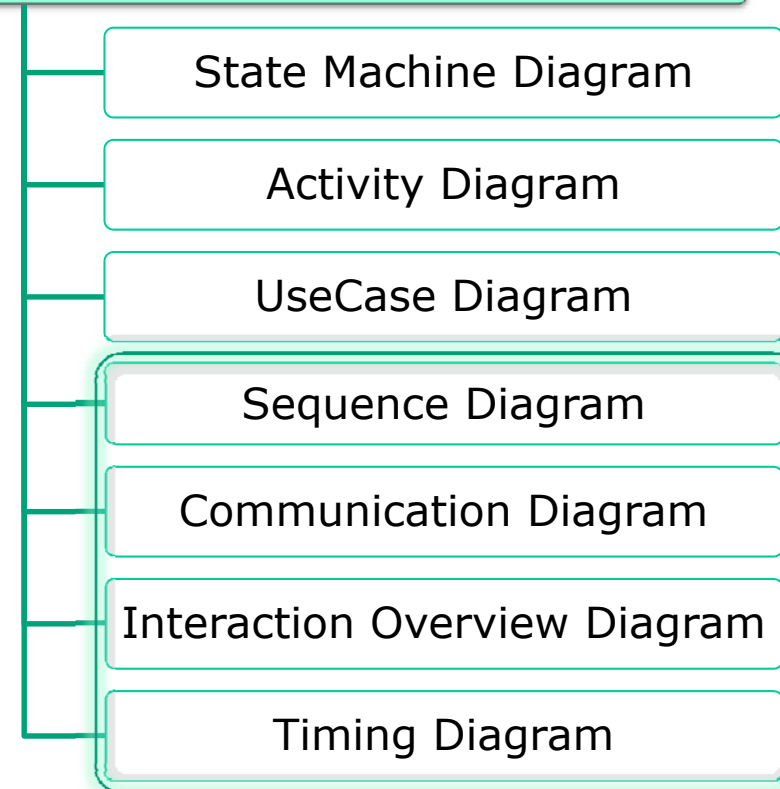




Structure Diagrams



Behavior Diagrams



Interaction Diagrams



- **Originally intended for modeling software-intensive systems:**
 - UML models capture different views of a software system (information model, run-time structure/behavior, packaging, deployment, etc.)
 - Inspired primarily by the concepts from object-oriented languages (class, operation, object, etc.)
- **However, the general nature of its concepts made UML2 suitable for extensions to any specific domains.**

Domain Specific Modeling by profiling the UML2!





- **Meta-modelling via MOF**

- For heavyweight extension mechanisms
- Ensures full manipulation of MMs
 - » Add, remove meta-classes and relationships between

- **Meta-models profiling**

- For lightweight extension mechanisms
- Adaptation of existing meta-models...
 - ...no modifications of existing concepts!
 - » e.g. UML MM constraints may not be suppressed, but additional one (compatible with existing one) may be added.
- May extend any standard MM of the OMG
 - » e.g. UML profiles, BPMN profiles...

- **Profile vs. MOF → ?**

- Depend on your project context, e.g.:
 - » Scope of the extensions
 - » Tooling constraints
 - » Engineer level of experiment/education
 - » Etc.

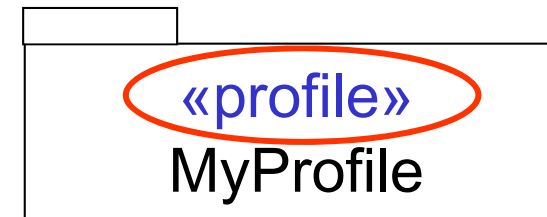
**"There is no simple answer for when you should create a new metamodel and when you instead should create a new profile."
(extracted from UML super-structure doc.)**



- **Define a domain specific terminology, i.e. a domain specific notation instead of the plain UML2 notation.**
 - Possibly including a domain specific concrete syntax.
- **Complete/specialize the UML2 semantics for dealing with:**
 - UML Semantics Variation Points,
 - » e.g. On BroadcastSignalAction, scope of target objects in Signal.
 - For clarifying ambiguous definition,
 - For specializing an existing semantics aspect of UML2.
- **Define usage constraints of the UML2 in order to drive its usage**
 - e.g., for defining a domain specific methodology
 - » e.g. to force behavior specification via protocol state-machine
- **Define new meta-information for annotating a model for a given purpose**
 - e.g. for code generation purpose, for enabling model-based analysis such as quality performance analysis, etc.
 - » e.g. for C++, «virtual», «inline»...



- Profile is a stereotyped package



- Applying a profile

- All extensions are then available for modelling



- If multiple profiles are applied:

- Referenced MMs have to be identical...
... and the model has also to refer the same MM.
- Their constraint sets do not have to conflict
- In case of naming conflict, use namespace notation:
 - » <ProfileName>::<StereotypeName>
 - e.g. «MyProfile1::name» & «MyProfile2::name»

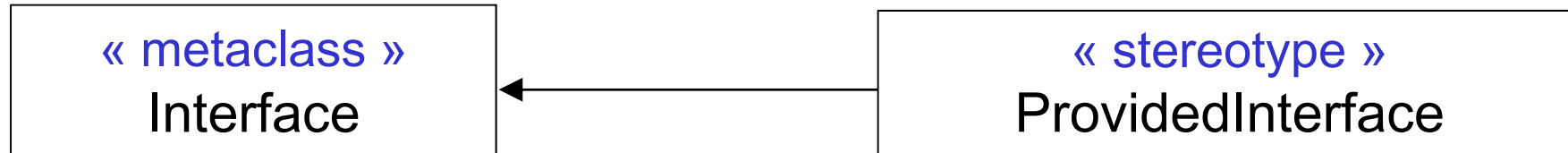


- **Defines how a specific metaclass may be "extended" for:**
 - **Adding semantics information for:**
 - » Extension-purpose (but with a limited scope!),
 - » Restriction-purpose,
 - » Clarification-purpose.
 - **Supporting domain specific terminology including specific notations**
 - » E.g. EAST-ADL, a UML profile for automotive ECUs (<http://www.east-eea.net/>)
- **Several stereotypes may be applied to one model element,**
- **Stereotypes may have properties that are often referred as "tagged values"**



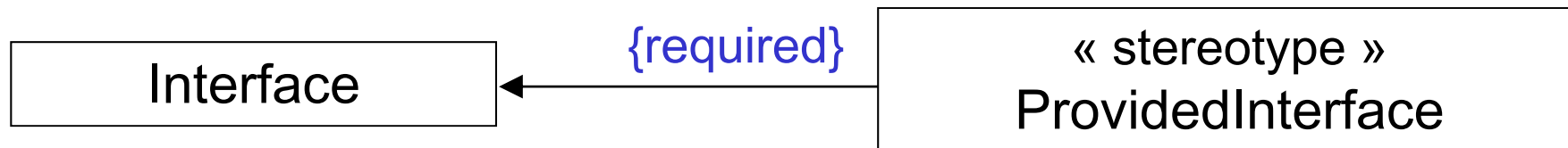
Notation for stereotype definition (Metamodel-level)

- **Stereotype definition**



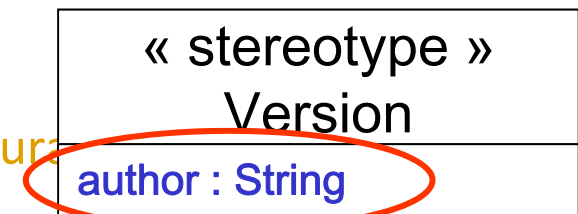
- **Required stereotype**

- Extended class may only be instantiated under its stereotyped form



- **Definition of stereotype properties**

- UML primitive types (Integer, String, Boolean, UnlimitedNatural)
- UML Meta classes
- Possible to define or import additional specific types (Enumerations, Datatype or PrimitiveTypes)





Notation for stereotype usage (User model-level)

- Applying a stereotype

« providedInterface »
MyInterface

- Applying several stereotypes

« providedInterface, version »
MyInterface

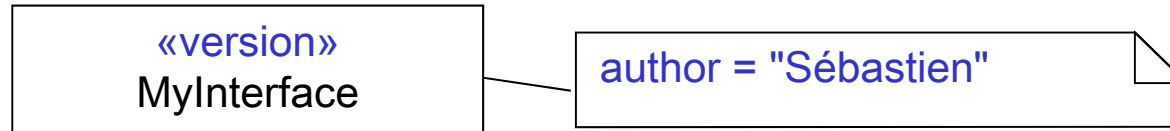
or

« providedInterface », «version »
MyInterface

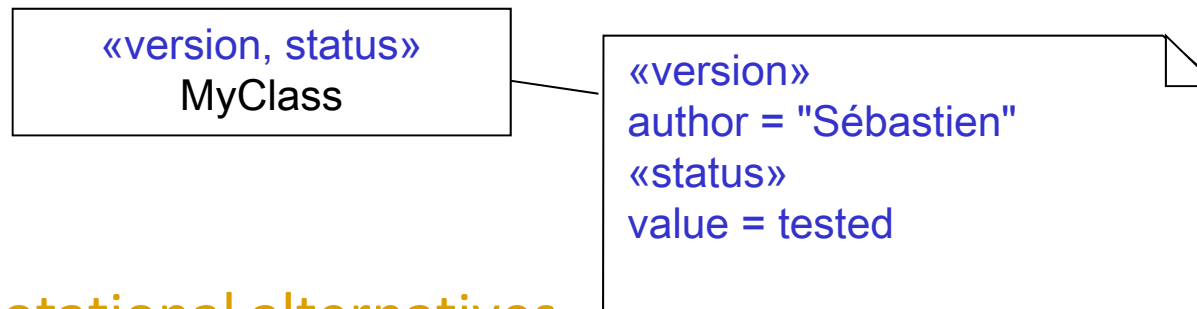


Notation for specifying values of stereotype properties

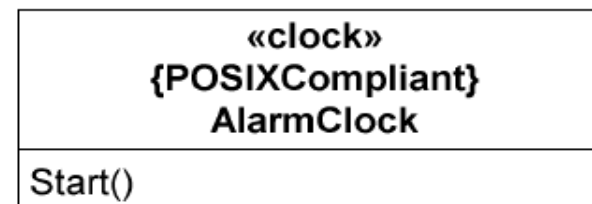
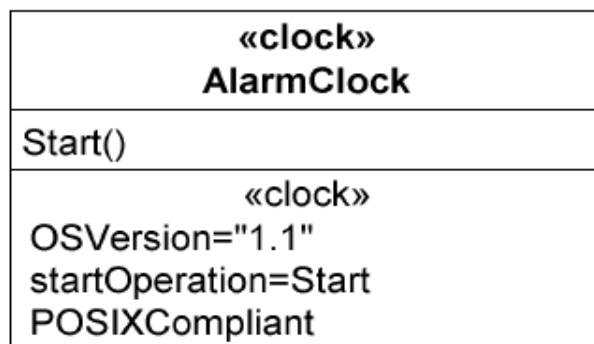
- Specifying values of a stereotype properties (also called "tagged-values")



- Add name of stereotypes when possible confusion



- Two other notational alternatives





- A profile package may import external resources

- Import of packages (possibly stereotyped as « modelLibrary »)

» e.g., external pkgs defining specific types for a profile



- Import of profiles



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



One simple profile example

Profile definition (Language definition level)

Specific notation



« metaclass »
UML::Class

Extension

« stereotype »
Semaphore

Specific properties

limit: Integer
getSema: Operation
relSema: Operation

Usage constraint

« Constraint »
limit < UpperLimit

Profile application (User model level)

« semaphore »
DijkstraSem

p ()
v ()

« semaphore »

limit = MAXlimit
getSema = p
relSema = v

DijkstraSem

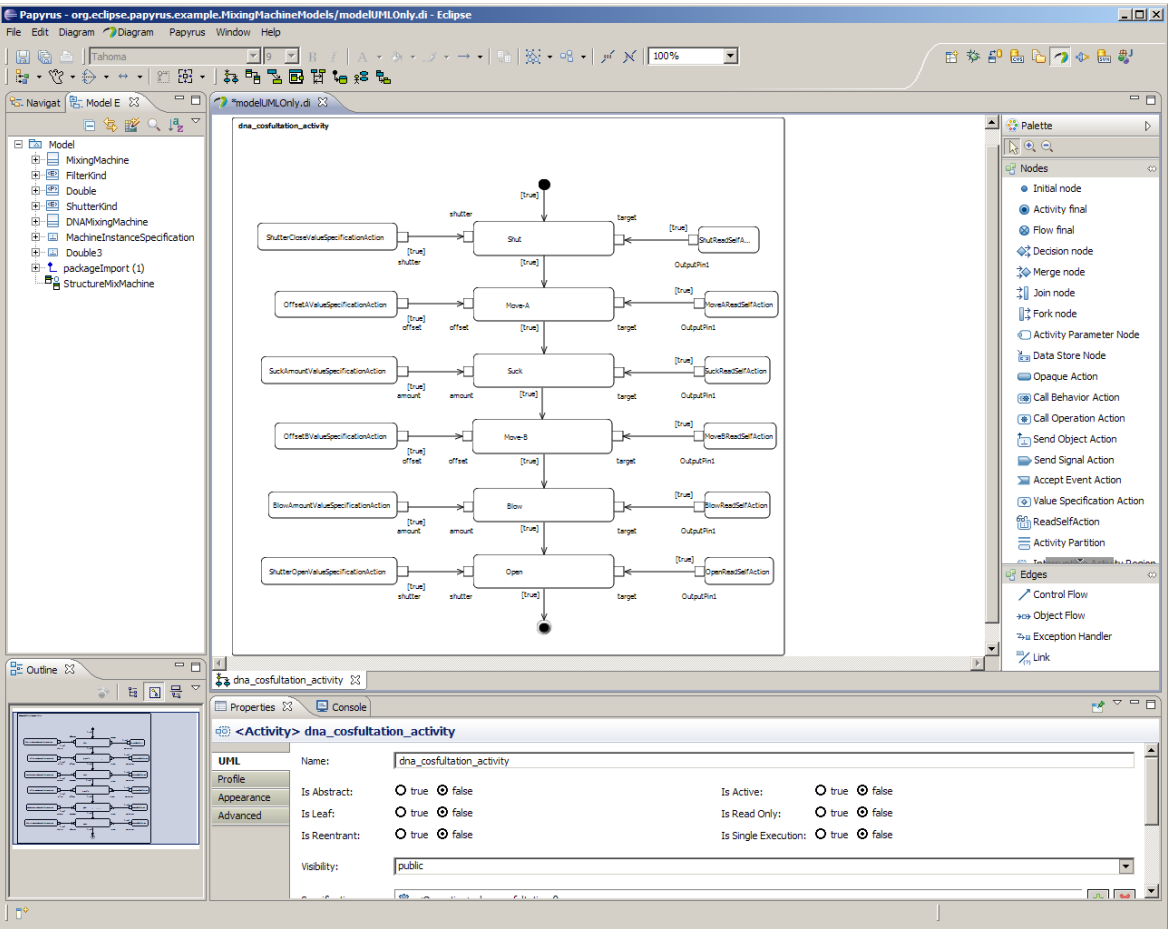


DijkstraSem

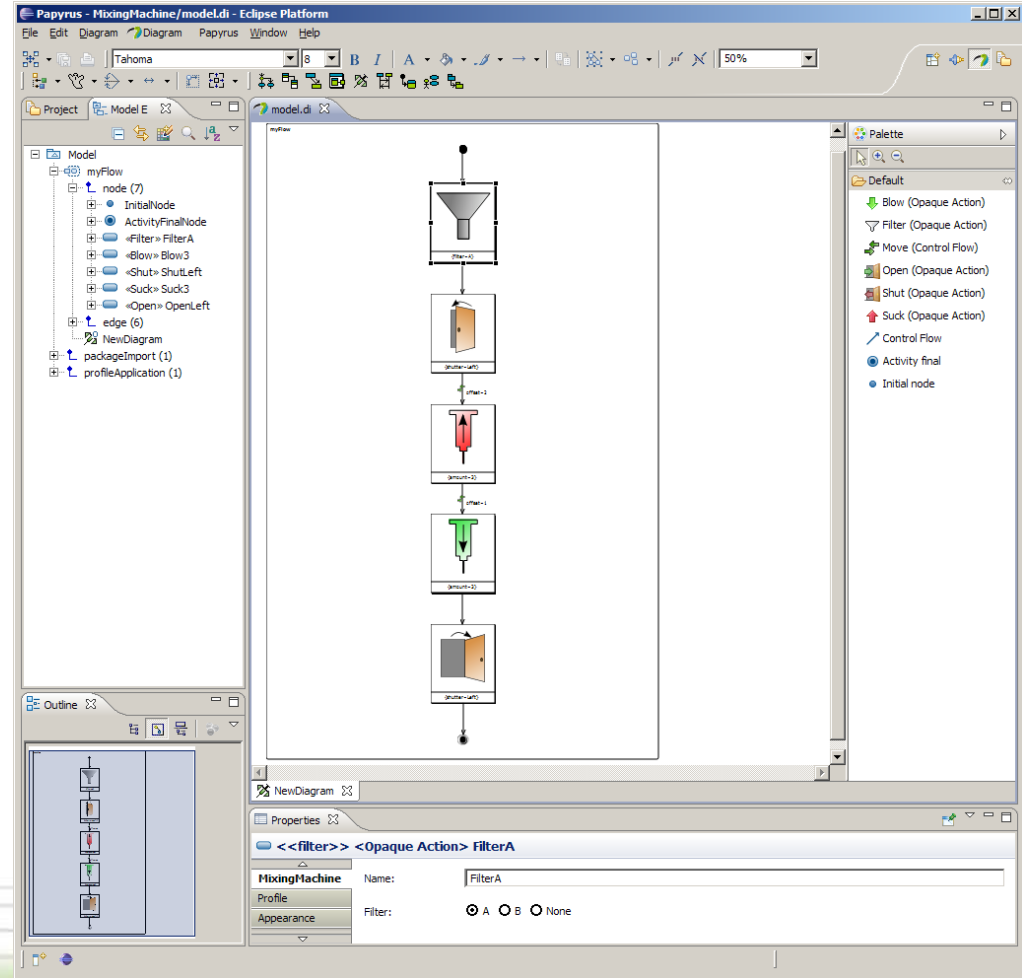


Mini project with UML-Profile within Papyrus

2nd version in specialized UML2



1st version in plain UML2





For using Papyrus (it is free ;-), let's go to: www.eclipse.org/papyrus

• Standard Papyrus installation via the Eclipse Modeling Platform

- Download the Eclipse Modeling Platform (www.eclipse.org/downloads),
- Unzip the downloaded file and start Eclipse.exe,
- Launch the Modeling discovery site update,
- Check Papyrus and start installation.

