# Model-based Engineering and Distributed Realtime Embedded Systems

Ecole Temps Réel: « Conception orientée objet appliquée à l’embarquée et au temps-réel »
Fréjus, 22 novembre 2009

Sébastien Gérard CEA, LIST / LISE
sebastien.gerard@cea.fr

## System development trends

- **Current constraints** → **Design principle candidates**
  - Growing size → Divide-to-conquer
  - Growing complexity → Separation-of-concern
  - Decreasing dev. time → Correct-by-construction
  - ⚠ Decreasing "bugs" → Early validation

**System architecture does enables these principles**

- By **definition**, fosters divide-to-conquer
- By **good practice**, enables separation-of-concern
- By **intent**, facilitates correct-by-construction description
- By **interest,** supports for early validation

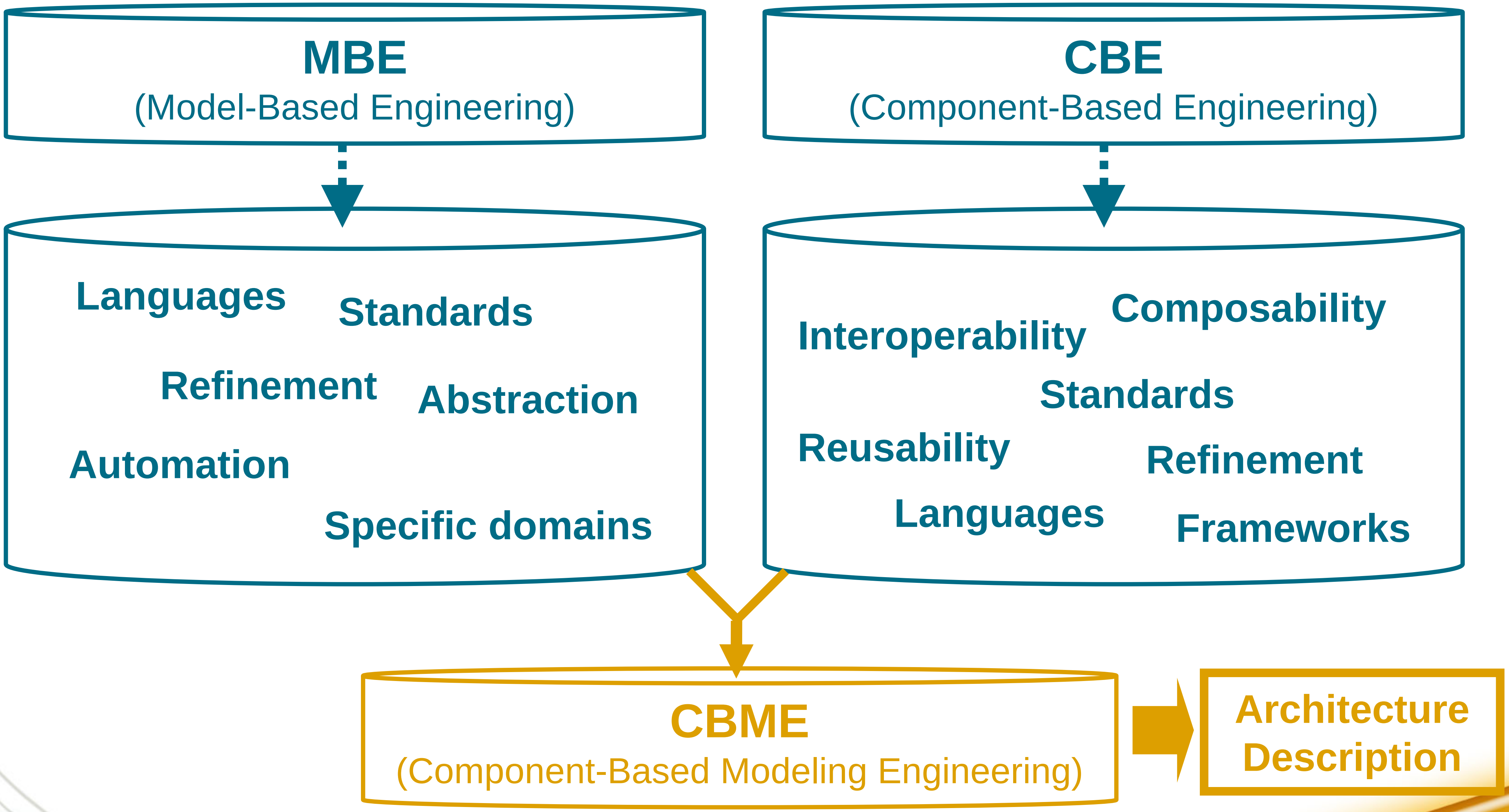
## Other advantages of explicit architecture description

- **Stakeholder communication**
  - Used as a focus of discussion by stakeholders of the system development
- **Integration medium**
  - Architectural description used to drive integration of its implemented subsystems
- **Architecture analysis / validation**
  - Architecture description may serve to analysis for validation of whether the system can meet its non-functional requirements is possible => very important result for RT/E!
- **Large-scale reuse**
  - The architecture description is one key stone of product lines
    - Has to be reusable across a range of systems (gathered in one family system)

## A standard for system architecture: IEEE 1471

- **IEEE 1471**
  - Recommended Practice for Architectural Description of Software-intensive Systems
    - http://www.iso-architecture.org/ieee-1471/docs/
- **Scope**
  - Architecture descriptions of systems, including software systems.
- **Two definitions**
  - “Architectural description (AD): A collection of products to document an architecture.”
  - “Architecture: The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.”

## Two main current paradigms: Model & Compoment



## Agenda

- **Part I**
  - Introduction to CBME for real-time and embedded
- **Part II**
  - ~~MARTE in details~~ ➔ Some details on MARTE
- **Part III**
  - MBE in action, examples

## Agenda

- **Part I: Introduction to CBME for real-time and embedded**
  - Component-based Engineering,
  - Model-based Engineering,
  - Standards for Component-based Modeling Engineering
- **Part II: Some details on MARTE**
- **Part III: MBE in action, example.**

## Component paradigm for architecture design

- **Architecture Description relies on two key features**
  - Interoperability
    - Needs to deal with heterogeneity of systems
  - Reusability
    - Grail quest of software engineering (one starting point of OO introduction)
- **Both aforementioned concepts lead to adopt a view where a system consists of a set of interacting components**

  ➔ CBSE: Component-Based System/Software Engineering
- **Adoption of CBE for architecture leads to following evolutions**
  - For implementation:
    - Component-based execution framework: ex. CCM, EJB, Think…
  - For Modeling:
    - Model-based approach relies on Component: ex. Component evolution from UML1.x to UML2.x, AADL, EAST-ADL, AUTOSAR…

## Basic concepts behind components

- **Component**
  - Unité de composition et de déploiement, qui remplit une fonction spécifique et peut être assemblé avec d'autres composants. À cet effet, il porte une description de ses points d'interaction (port/interface).
- **Connector**
  - Élément permettant d'assembler des composants en utilisant leurs interfaces fournies et requises.
  - Remplit 2 fonctions ➔ liaison et communication
- **Configuration**
  - Un assemblage de composants
  - Selon le modèle de composant adopté, une configuration peut être vue comme un composant.

- ***N.B.:***
  - *La notion de composant est préservée à l'exécution pour faciliter l'adaptation dynamique et la répartition.*
  - *La différence entre composant et connecteur est une différence de fonction, non de nature: Un connecteur peut lui-même être un composant!*

## Two views for component

- **An external view (or "black-box" view)**
  - Publicly visible features (operations & properties).
  - Behavior may be attached to interface, port or/and to the component itself.
  - Component wiring via assembly-connectors between ports.
    - Possible detailed wiring specification via composite structure to specify the role or instance level collaboration.

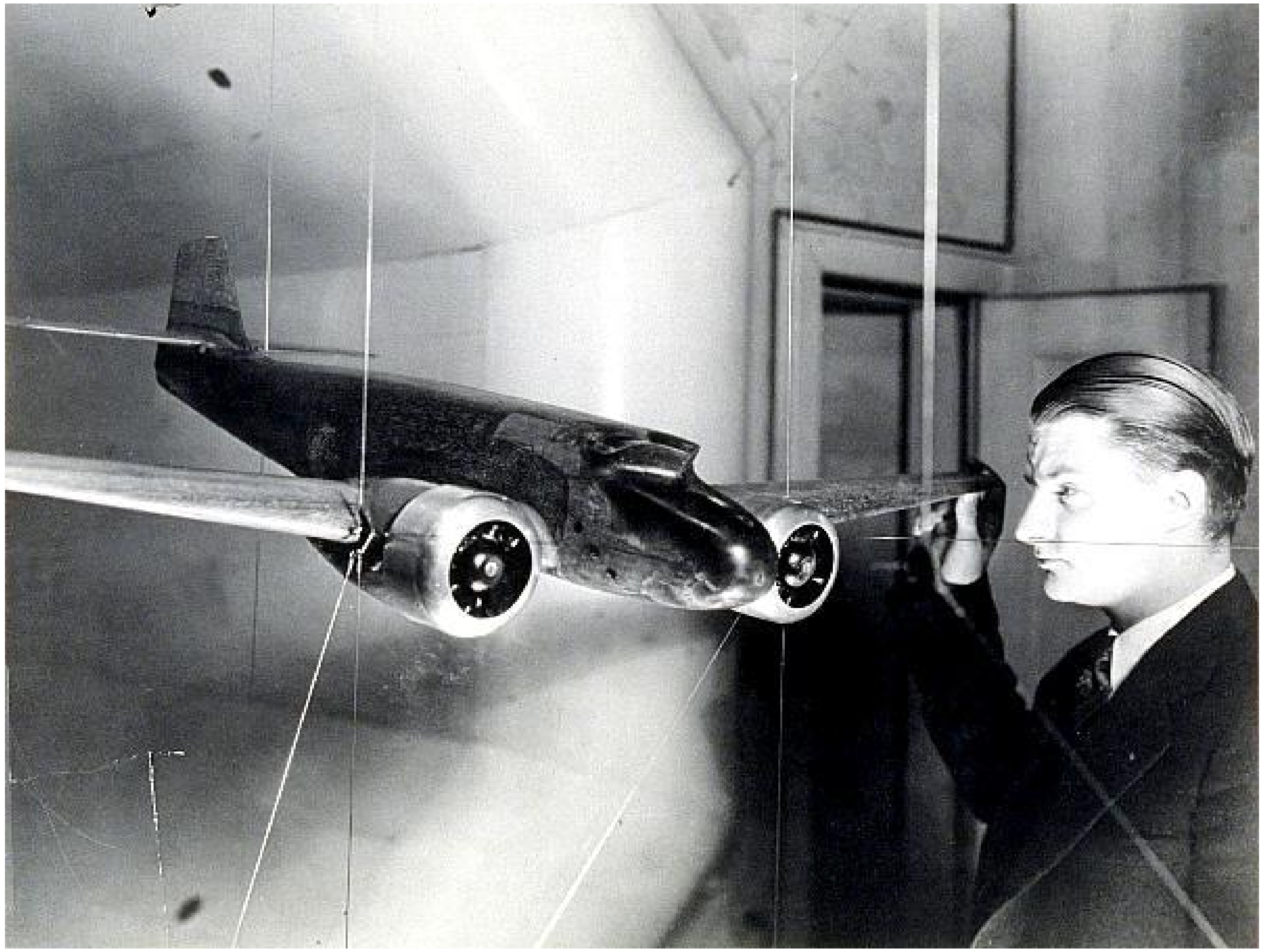
- **An internal view (or "white-box" view)**
  - Private properties and realizing classifiers.
  - External and internal views mapping:
    - Dependencies (on structure diagrams),
    - Or delegation connectors to internal parts (on composite structure diagrams).
    - More detailed behavior specifications (e.g. interactions and activities) may be used to detail the mapping.

## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
  - Component-based Engineering,
  - **Model-based Engineering,**
  - Standards for Component-based Modeling Engineering
- **Part II: MARTE in more details**
- **Part III: One typical usage example of MARTE**

## Models in Traditional Engineering

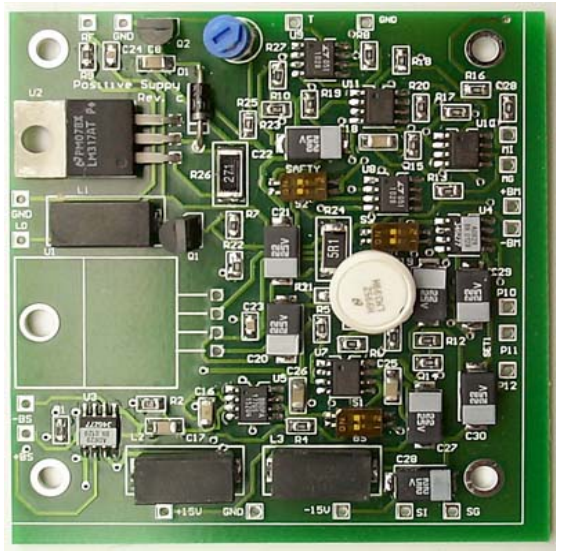
- **Probably as old as engineering**



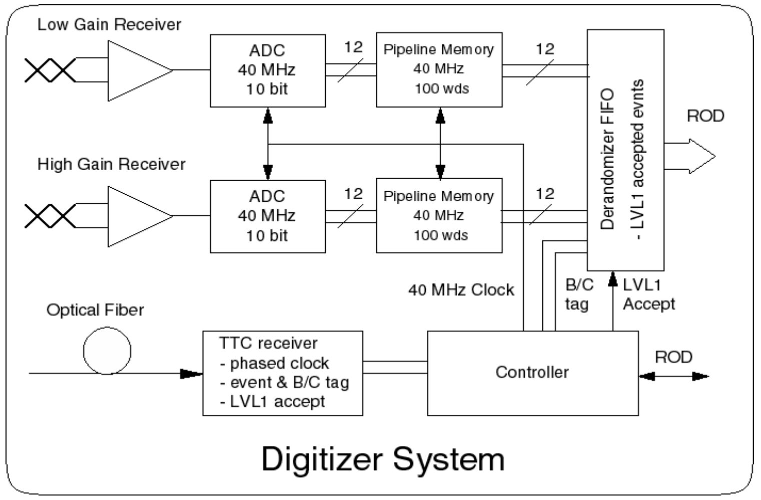
Extracted from B. Selic presentation during Summer School MDD For DRES 2004 (Brest, September 2004)

## Engineering Models

- **What is a model**
  - A reduced representation of some system that highlights the properties of interest from a given viewpoint



Modeled system


Low Gain Receiver
ADC 40 MHz 10 bit
12
Pipeline Memory 40 MHz 100 wds
12
Derandomizer FIFO - LVL1 accepted evnts
ROD
High Gain Receiver
ADC 40 MHz 10 bit
12
Pipeline Memory 40 MHz 100 wds
12
40 MHz Clock
B/C tag
LVL1 Accept
Optical Fiber
TTC receiver - phased clock - event & B/C tag - LVL1 accept
Controller
ROD
Digitizer System

Functional Model

- **Main features**
  - We don’t see everything at once
  - We use a representation (notation) that is easily understood for the purpose on hand

## Why modeling?

- **To deal with complexity**
  - Concrete representation of knowledge and ideas about a system being developed ➔ improve communication around a problem
  - Abstract a problem (omits some aspects) to focus on some particular points of interest ➔ improve understability of a problem

- **To improve communication …**

  **… to foster information sharing and reuse!**

- **To increase control of complexity**
  - Via a set of nearly independent views of a model

    ➔ Separation of concerns
  - A model may be expressed at different level of fidelity

    ➔ Refinement (also called model evolution)

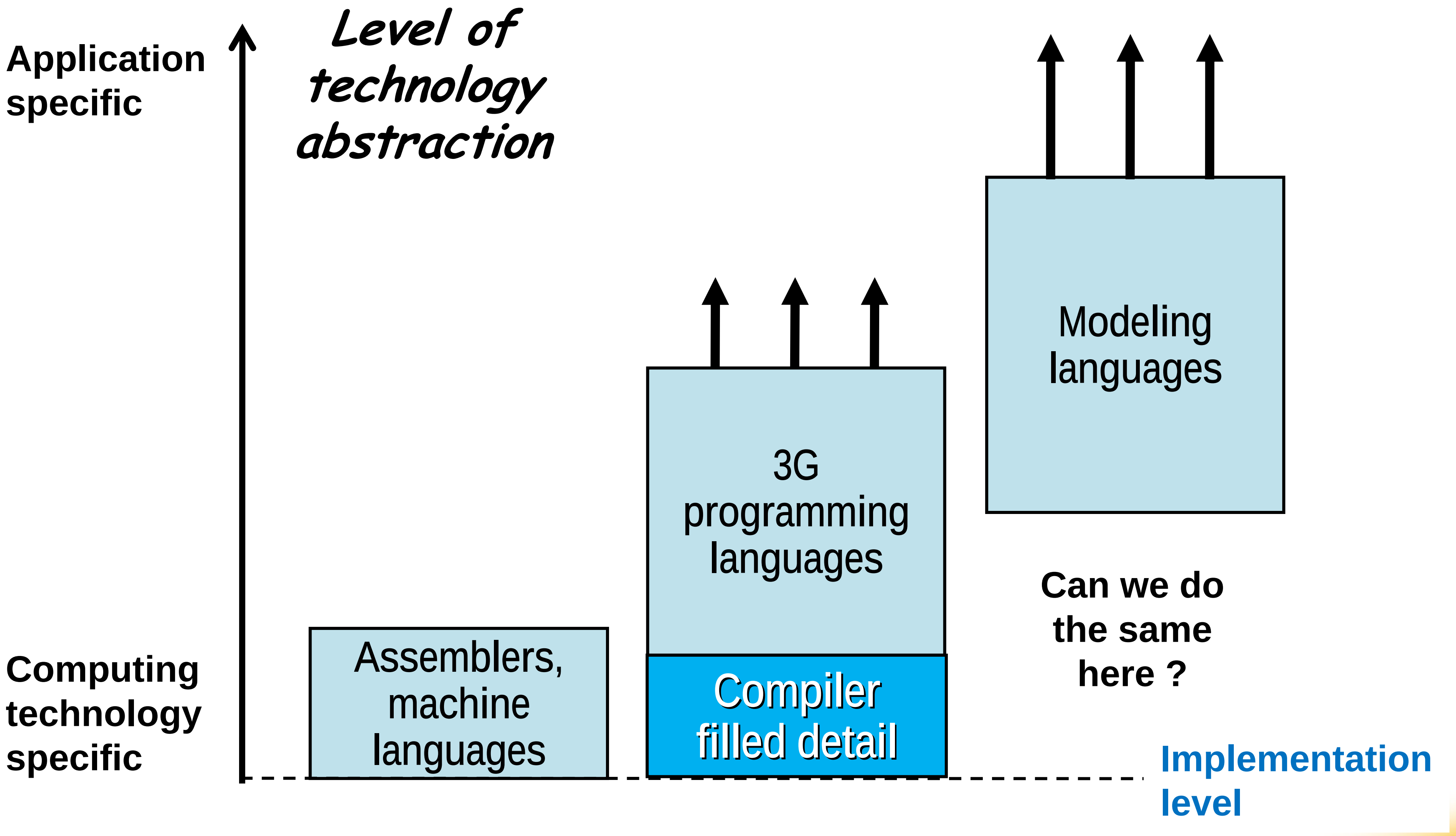
- **To focus on specific domain expertise while developing software system**
  - Domain Specific Languages

## Modeling vs. Programming Languages

- **The purpose of programming is implementation**
  - Implementation requires full precision and full detail
  - Takes precedence over clarity
- **The purpose of modeling is communication and understanding (“documentation”)**
  - These require clarity $\Rightarrow$ omission of “irrelevant” detail
- **However...**
  - Software is unique in that a program and its model share the same medium – the computer
  - The two can be formally linked to each other
  - The formal linkage can be realized by automated transformations

Credit to B. Selic

## Trends in Computer Languages


Application specific
Level of technology abstraction
Computing technology specific
Assemblers, machine languages
3G programming languages
Compiler filled detail
Modeling languages
Can we do the same here ?
Implementation level

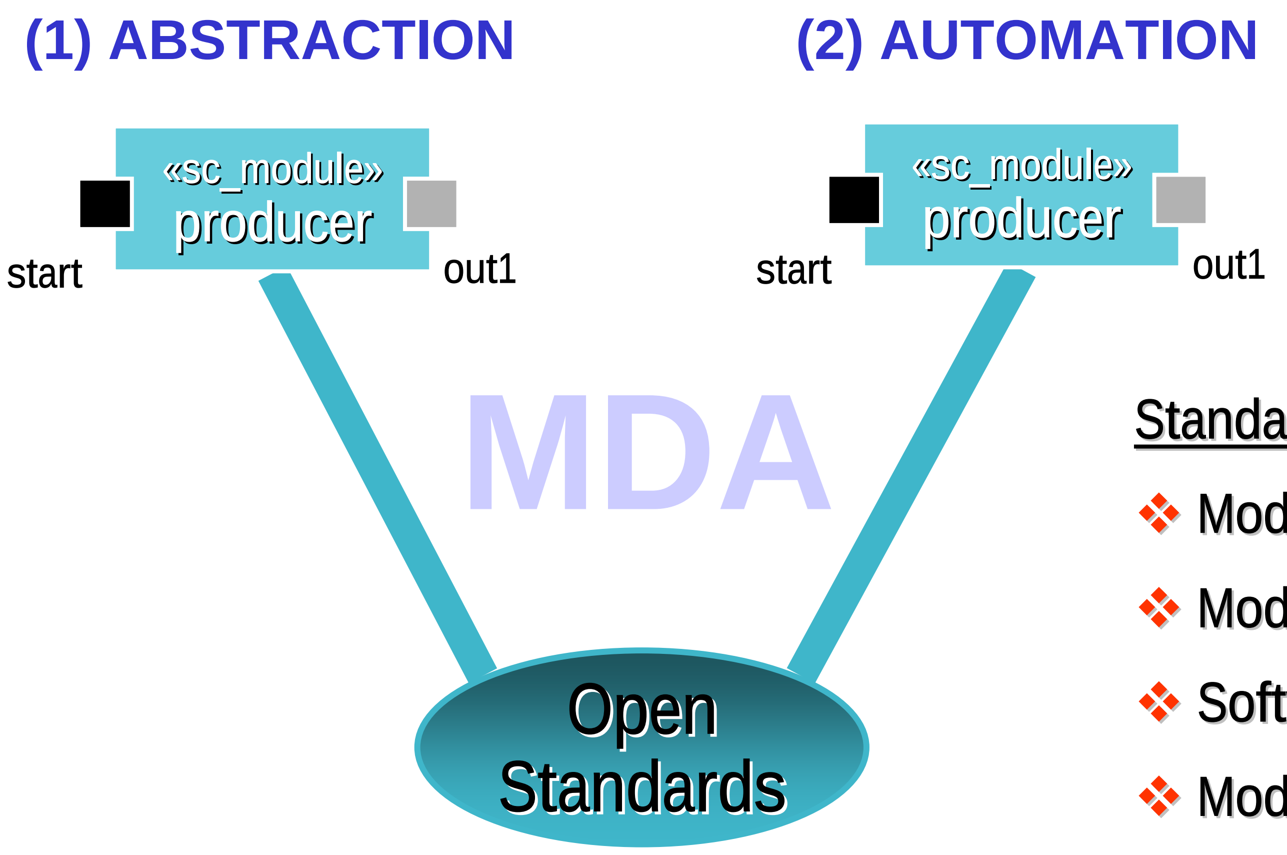
Credit to B. Selic

## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
  - Component-based Engineering,
  - Model-based Engineering,
  - **Standards for Component-based Modeling Engineering**
- **Part II: Some details on MARTE**
- **Part III: One typical usage example of MARTE**

## OMG's Model-Driven Architecture (MDA)

- **MDA (Model-Driven Architecture) = an OMG initiative**
  - A framework for a set of open standards in support of CBMDD

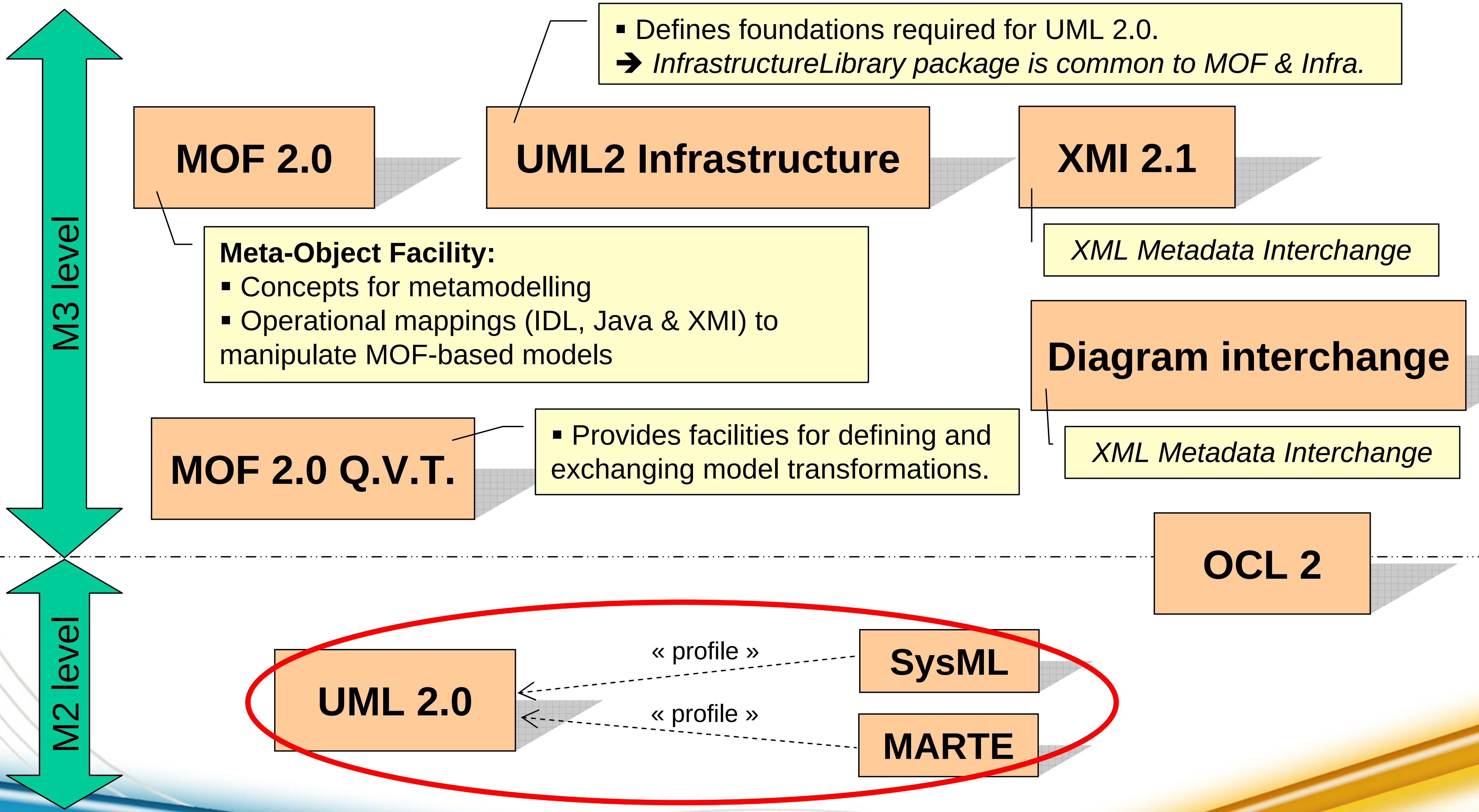


Standards for:

- Modeling languages
- Model transformations
- Software processes
- Model interchange

Extracted from B. Selic presentation during Summer School MDD For DRES 2004 (Brest, September 2004)

## Main OMG standards for MDSE



- Defines foundations required for UML 2.0.

➔ *InfrastructureLibrary package is common to MOF & Infra.*

**Meta-Object Facility:**
- Concepts for metamodelling
- Operational mappings (IDL, Java & XMI) to manipulate MOF-based models

*XML Metadata Interchange*

*XML Metadata Interchange*

- Provides facilities for defining and exchanging model transformations.

## Génèse de UML

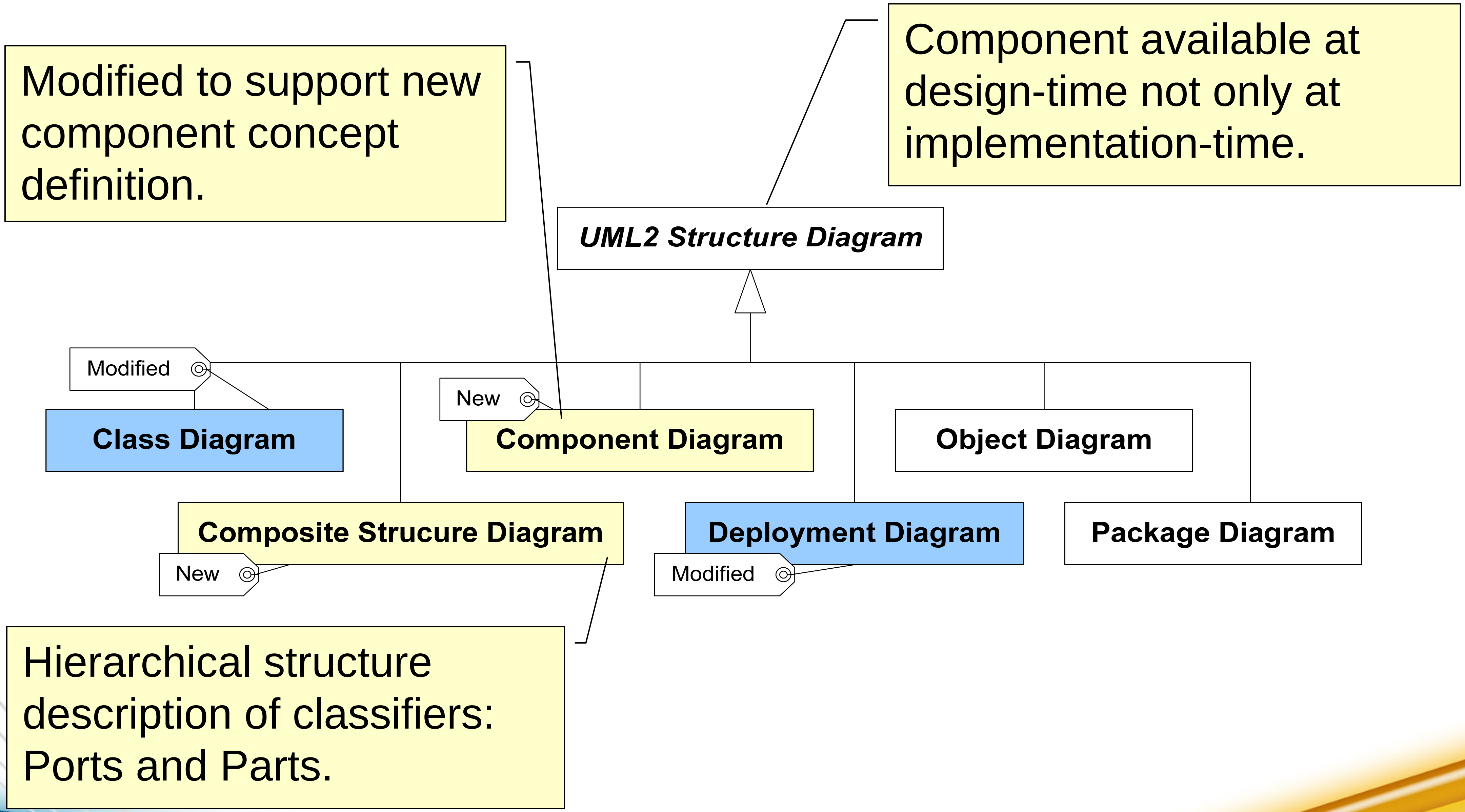
- **Unification des formalismes OO**
  - La guerre des méthodes ne faisait plus avancer la technologie des objets (fin des années 80)
  - Besoin d'un langage commun unique
    - Utilisable par toutes les méthodes
    - Adapté à toutes les phases du développement
    - Compatible avec toutes les techniques de réalisation
- **… sur plusieurs domaines d'applications**
  - Logiciels → Ingénierie des logiciels
  - Logiciels et matériels → Ingénierie des systèmes
  - Personnes → Ingénierie des affaires

## What is UML For?

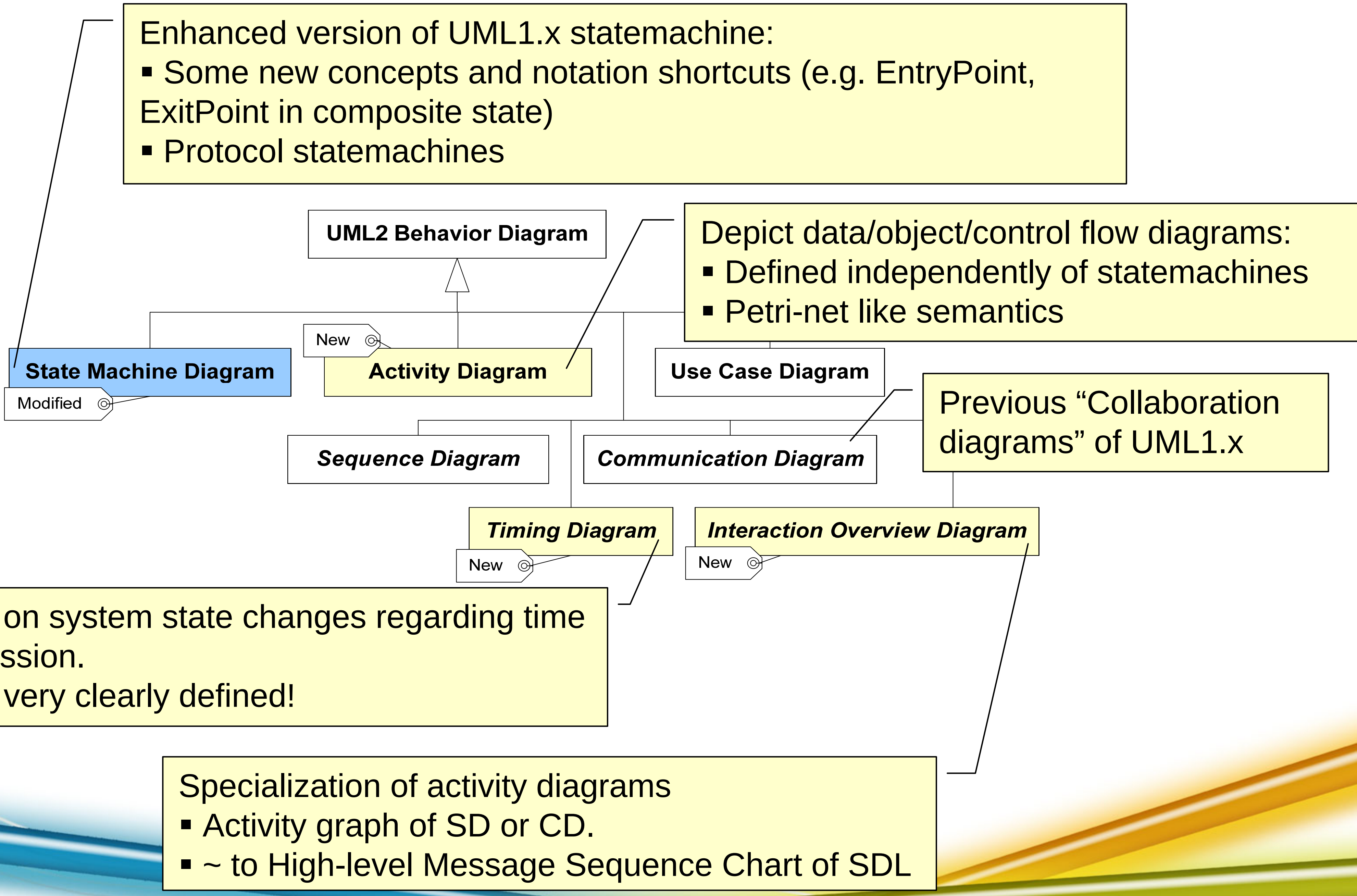
- **Originally intended for modeling software systems and**
  - UML models capture different views of a software system (information model, run-time structure/behavior, packaging, deployment, etc.)
  - Inspired primarily by the concepts from object-oriented languages (class, operation, object, etc.)

- **However, the general nature of these concepts makes UML suitable for extension to other domains**
  - E.g. systems engineering (SysML)
  - Motivation: to exploit advantages of a widely-adopted industry standard

## From UML 1.x to UML 2.0: static point of view


Modified to support new component concept definition.
Component available at design-time not only at implementation-time.
UML2 Structure Diagram
Modified
Class Diagram
New
Component Diagram
Object Diagram
Composite Strucure Diagram
New
Deployment Diagram
Modified
Package Diagram
Hierarchical structure description of classifiers: Ports and Parts.

# From UML 1.x to UML 2.0: dynamic point of view



Enhanced version of UML1.x statemachine:
- Some new concepts and notation shortcuts (e.g. EntryPoint, ExitPoint in composite state)
- Protocol statemachines

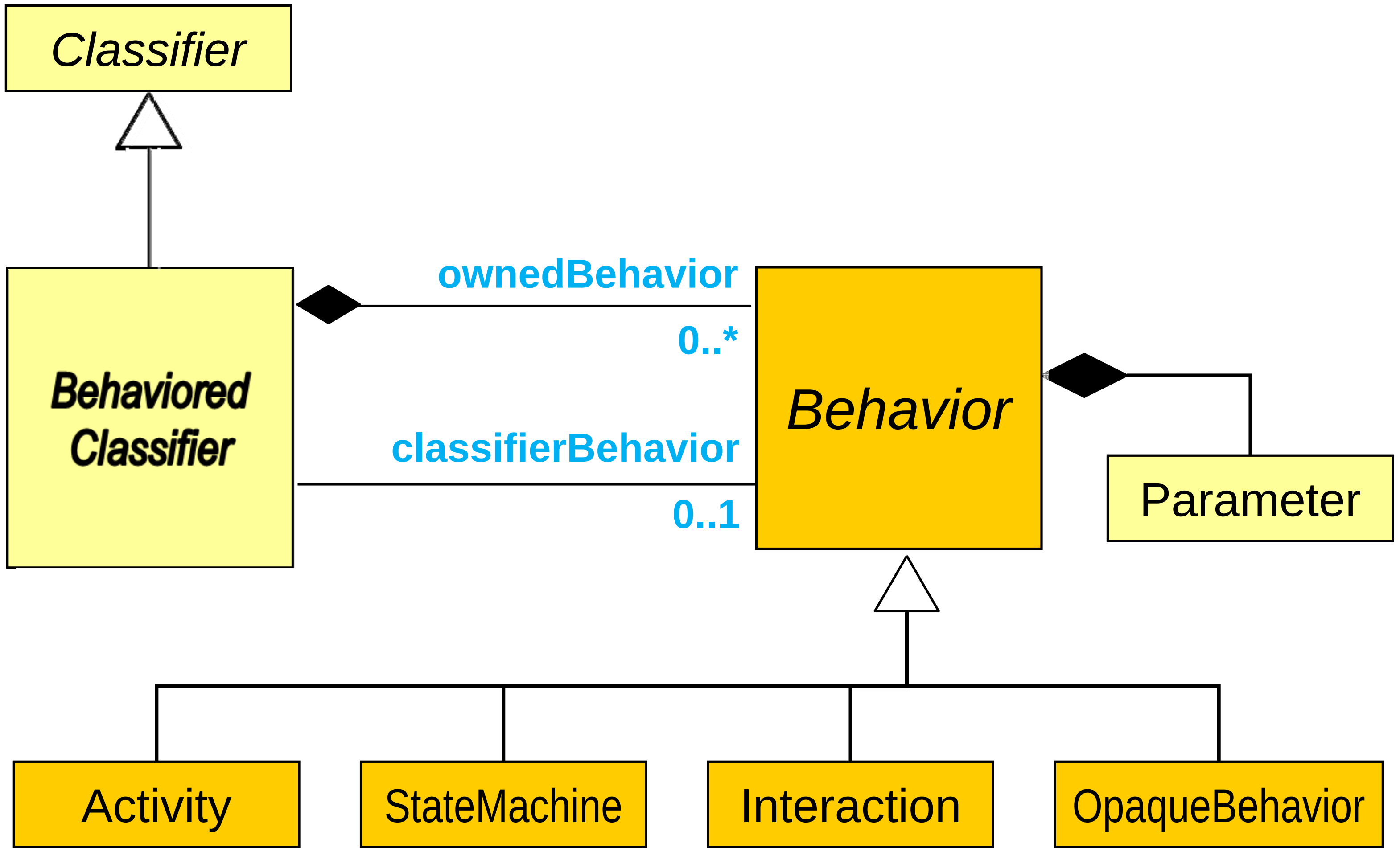
Depict data/object/control flow diagrams:
- Defined independently of statemachines
- Petri-net like semantics

Previous “Collaboration diagrams” of UML1.x

Focus on system state changes regarding time progression.
➔ not very clearly defined!

Specialization of activity diagrams
- Activity graph of SD or CD.
- ~ to High-level Message Sequence Chart of SDL

## Relationship between structure and behavior in UML



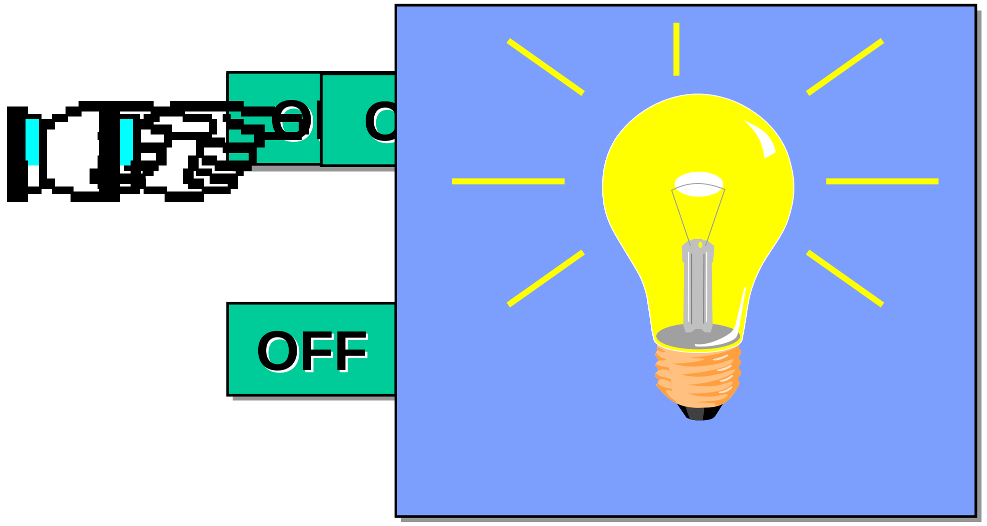
## Overview of State Machines package

- **Define the set of concepts required to model discrete behavior through finite state-transition systems**
- **Two kinds of state machine defined**
  - Behavioral state machine
  - Protocol state machine
- **The core constructs in statemachines include:**
  - StateMachine, States, Regions, Transitions and Events
- **Typical applications of statemachines include:**
  - Object lifecycle (i.e., states an “order” object might be in)
  - Event-driven behaviors of embedded controllers
  - UI controllers
  - ...

## Automata

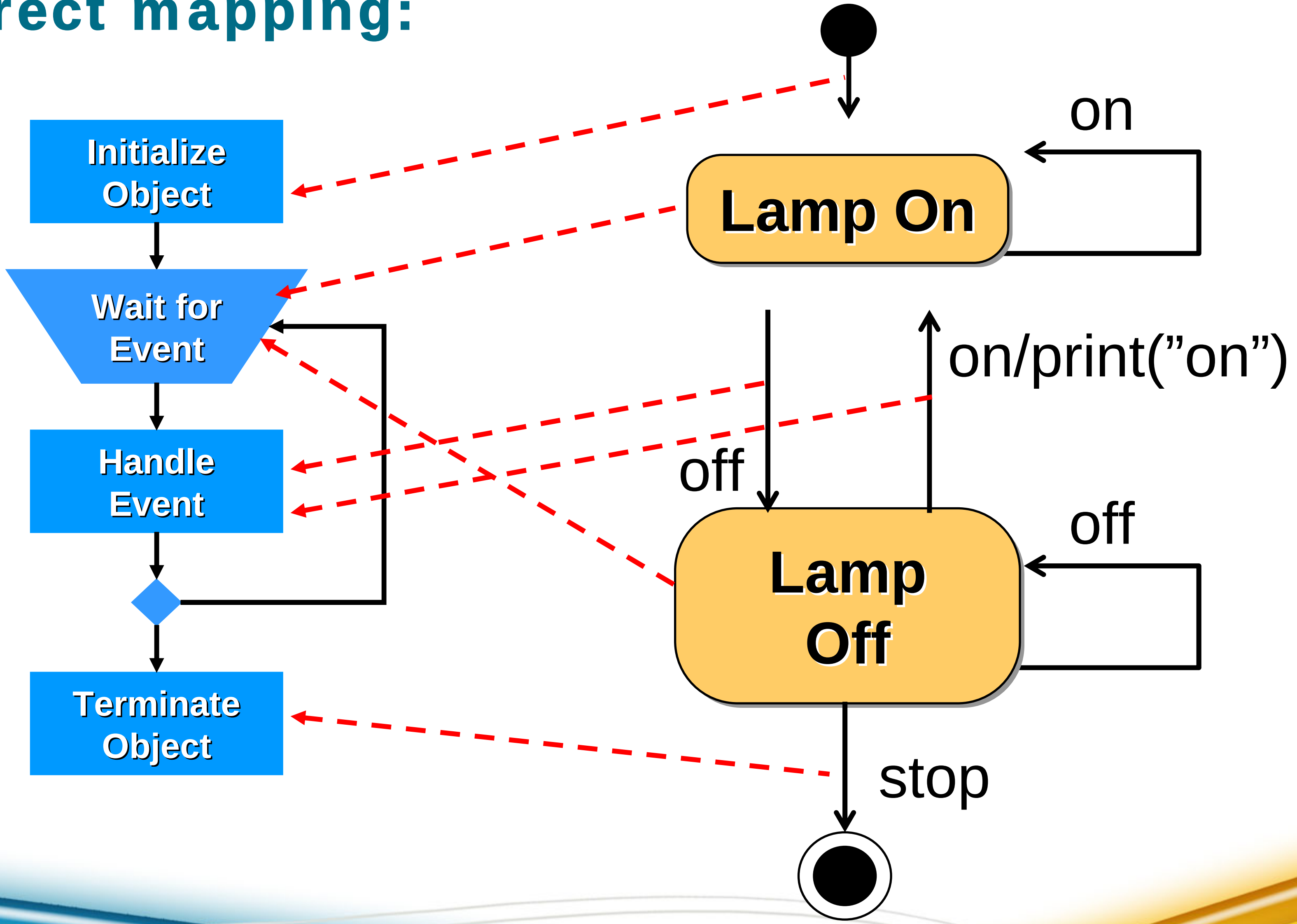
- A machine whose output behavior is not only a direct consequence of the current input, but of some past history of its inputs.

- Characterized by an internal state which represents this past experience.



## Object Behavior and State Machines

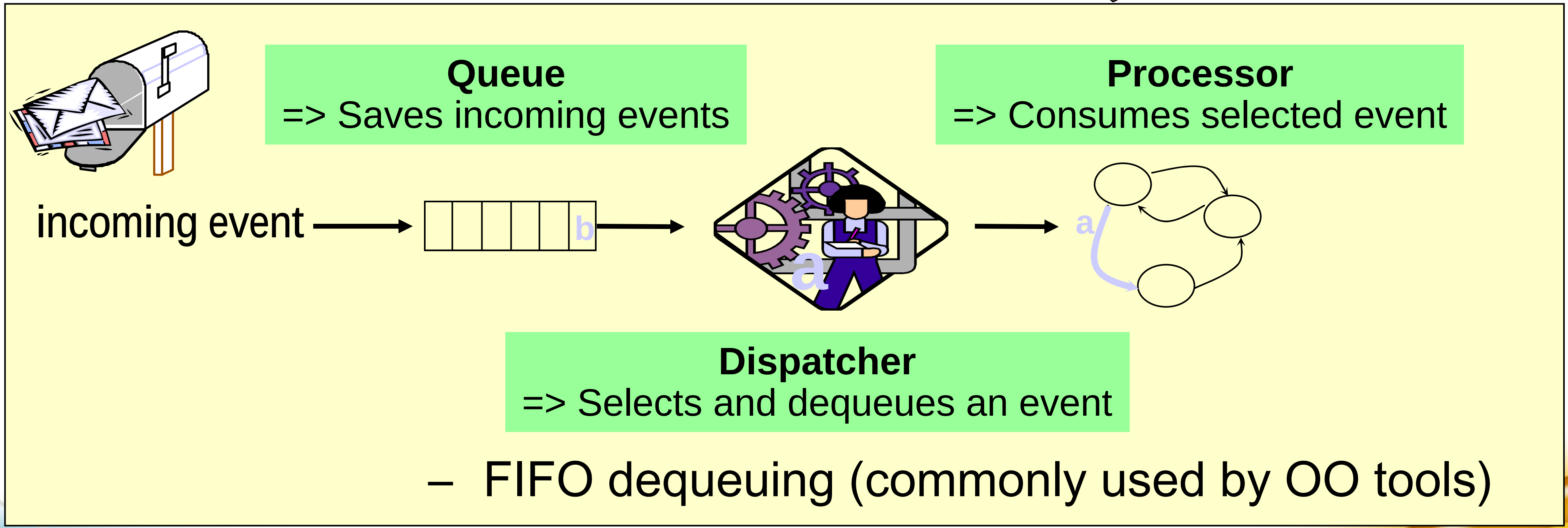
- **Direct mapping:**



## State machine semantics

- **UML state machines = hypothetical machine that:**
  - Queues event
  - Dispatches event
  - Processes event

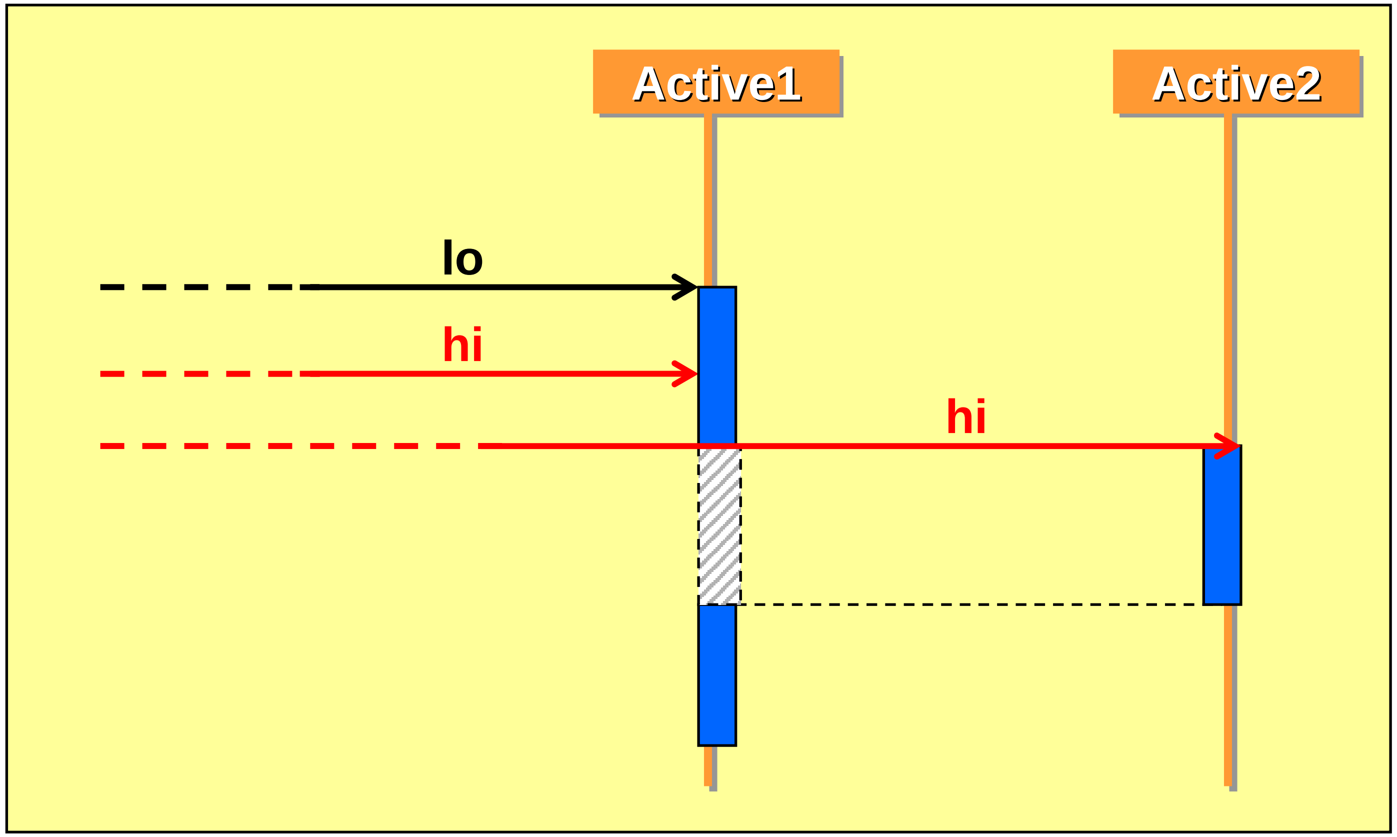
**Run-To-Completion:**
**only one event at a time**



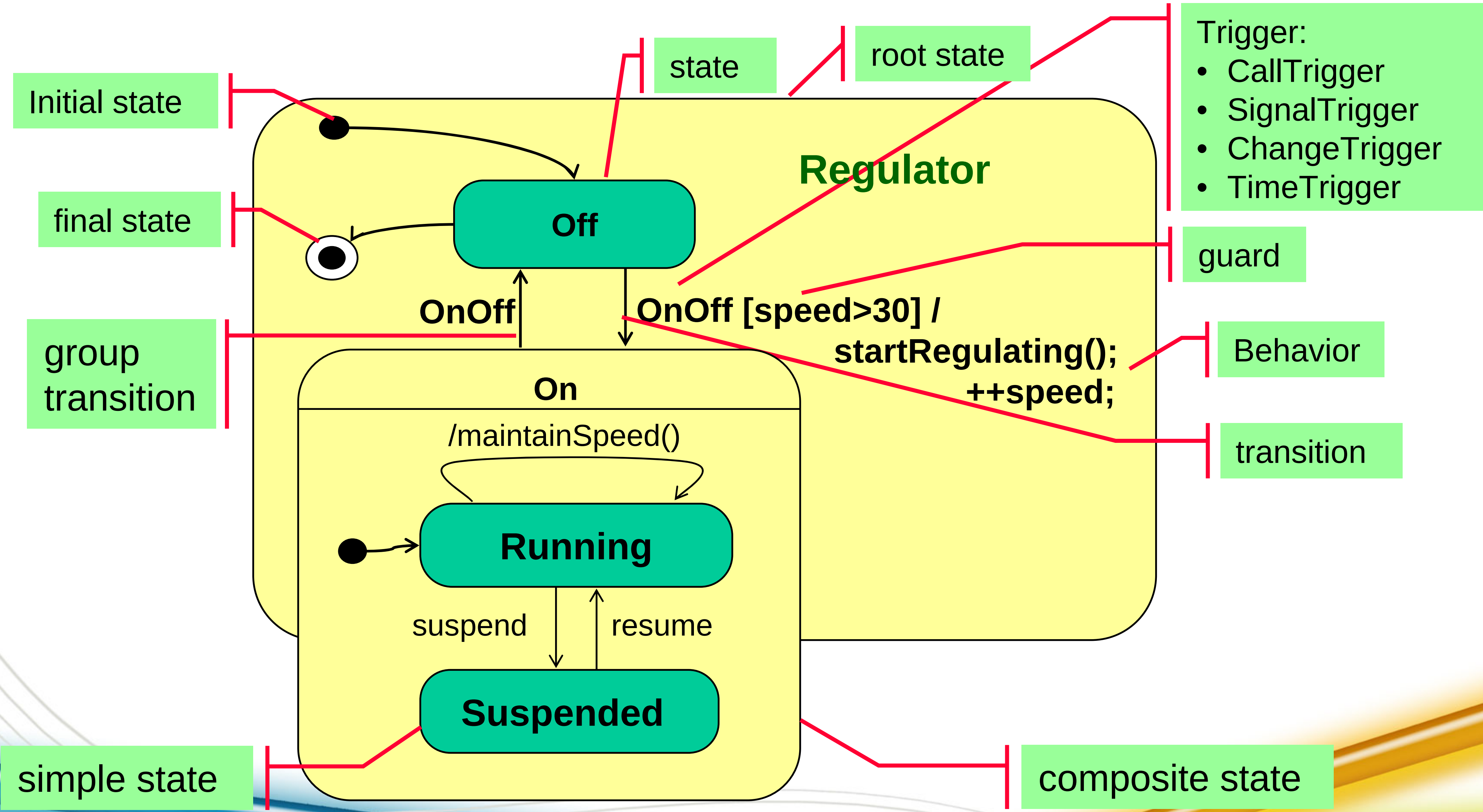
– FIFO dequeuing (commonly used by OO tools)

## The Run-to-Completion Model

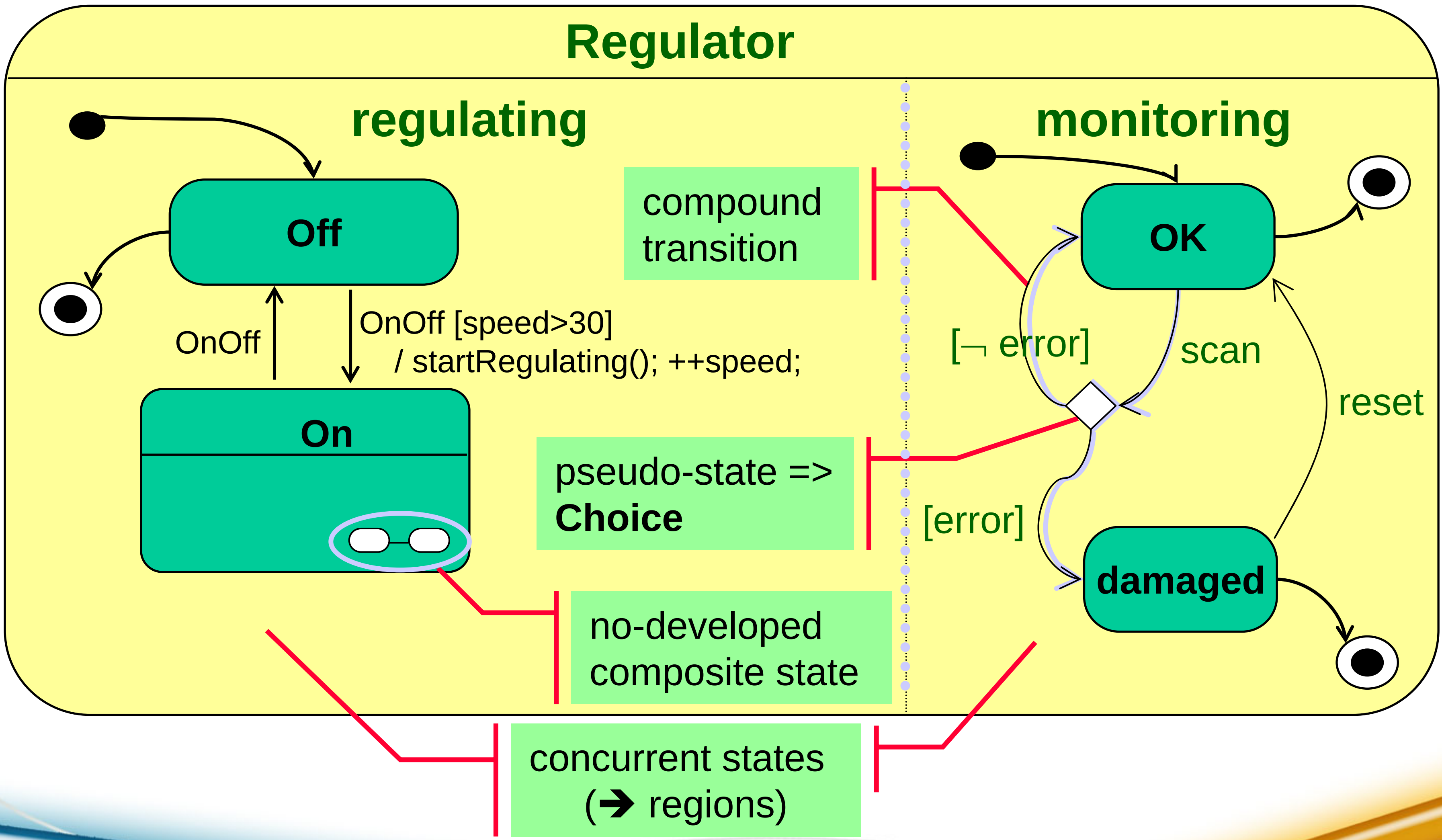
- A high priority event for (another) active object will preempt an active object that is handling a low-priority event



## State diagram



## State diagram (seq.)

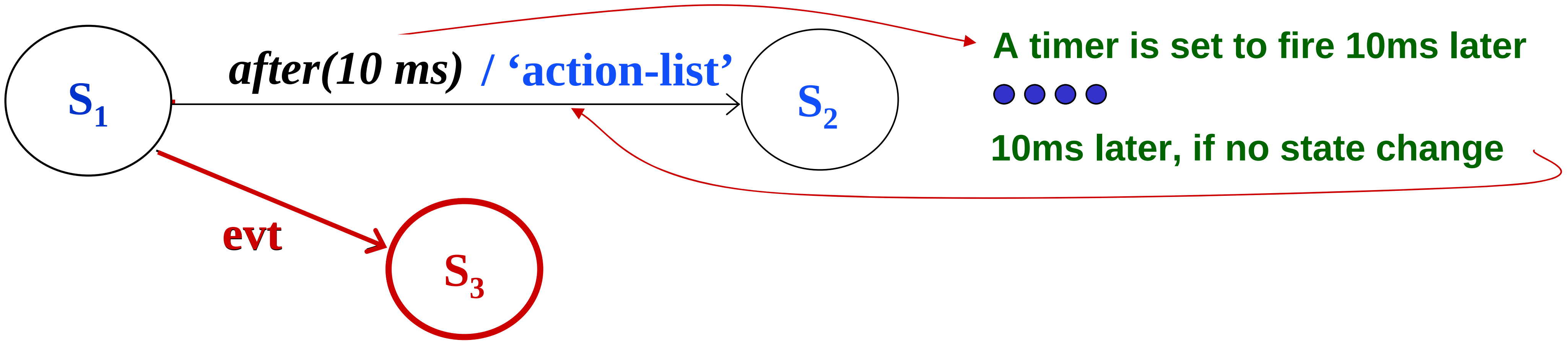

Regulator
regulating
monitoring
Off
On
OnOff
OnOff [speed>30]
/ startRegulating(); ++speed;
OK
damaged
[¬ error]
scan
[error]
reset
compound transition
pseudo-state => **Choice**
no-developed composite state
concurrent states (➔ regions)

## State machine semantics variation points

- **Event dequeuing policy has to be choosen**
  - Most widespread = FIFO
- **RTC granularity**
  - The only one defined = a single RTC for the whole state machine
  - But, it is possible to have several…
    - e.g.: one for each orthogonal region → but then « Just Do It !»

## Timing specification within state machine

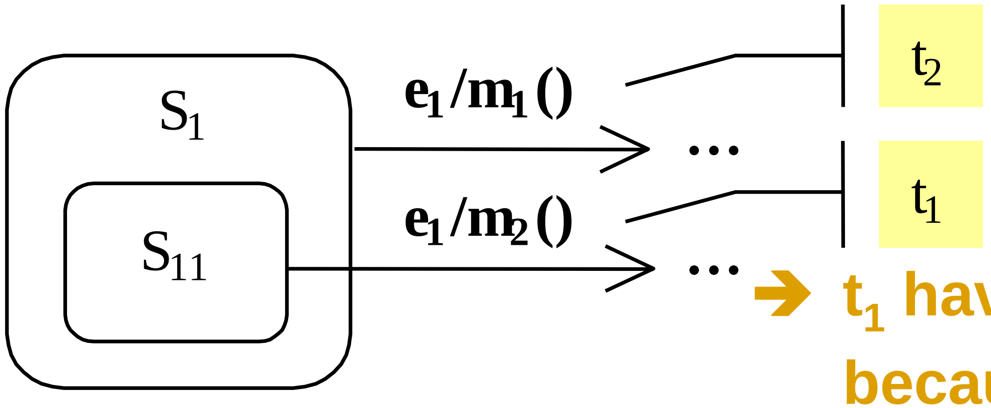
- **Timer setting on transitions of state machines**
  - TimeEvent
    - after ⇒ relative moment in time
    - when ⇒ absolute moment in time


$S_1$
*after(10 ms)* / 'action-list'
$S_2$
A timer is set to fire 10ms later
●●●●
10ms later, if no state change
evt
$S_3$

- **Drawbacks**
  - Specification is performed through implementation mechanisms
  - TimeEvent handled as other events ⇒ determinism issues !

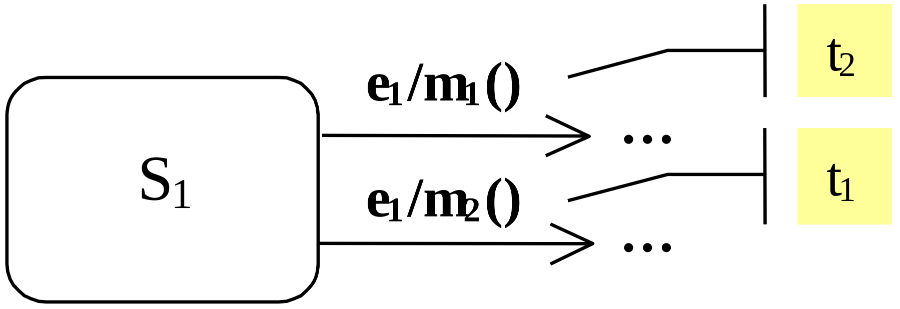
## Timing specification within state machine (seq.)

- Implicit priority rules between transitions …



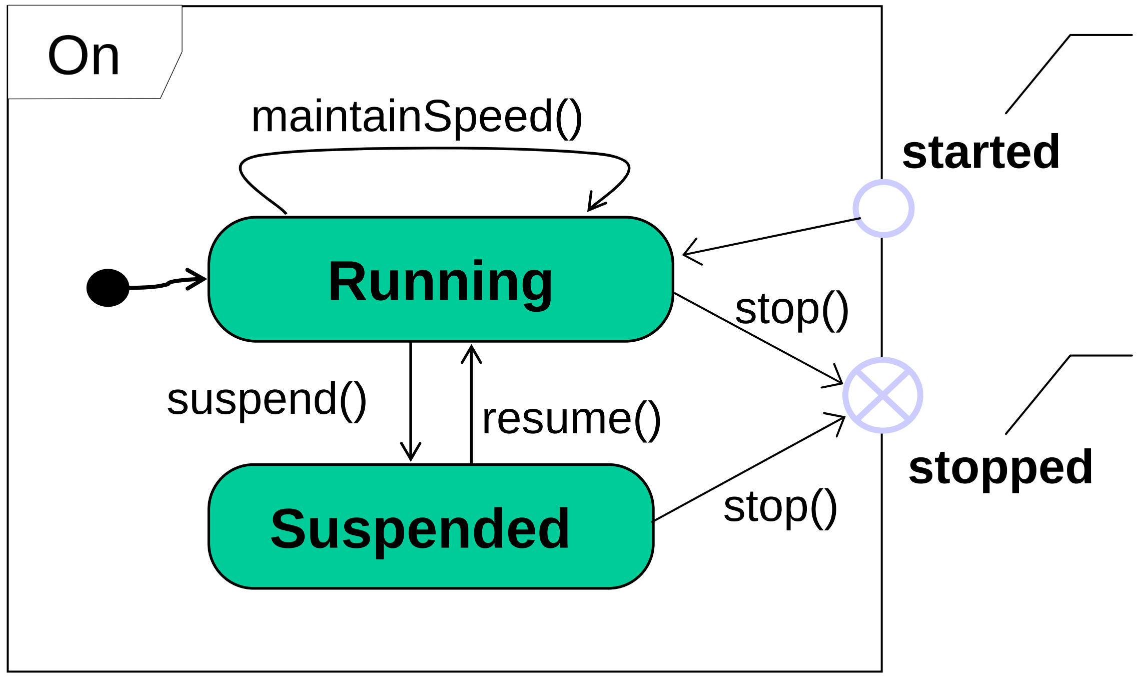
➔ $t_1$ have priority over $t_2$ because $S_{11}$ is a substate of $S_1$

- … but possible non determinism models



➔ non determinism situation !

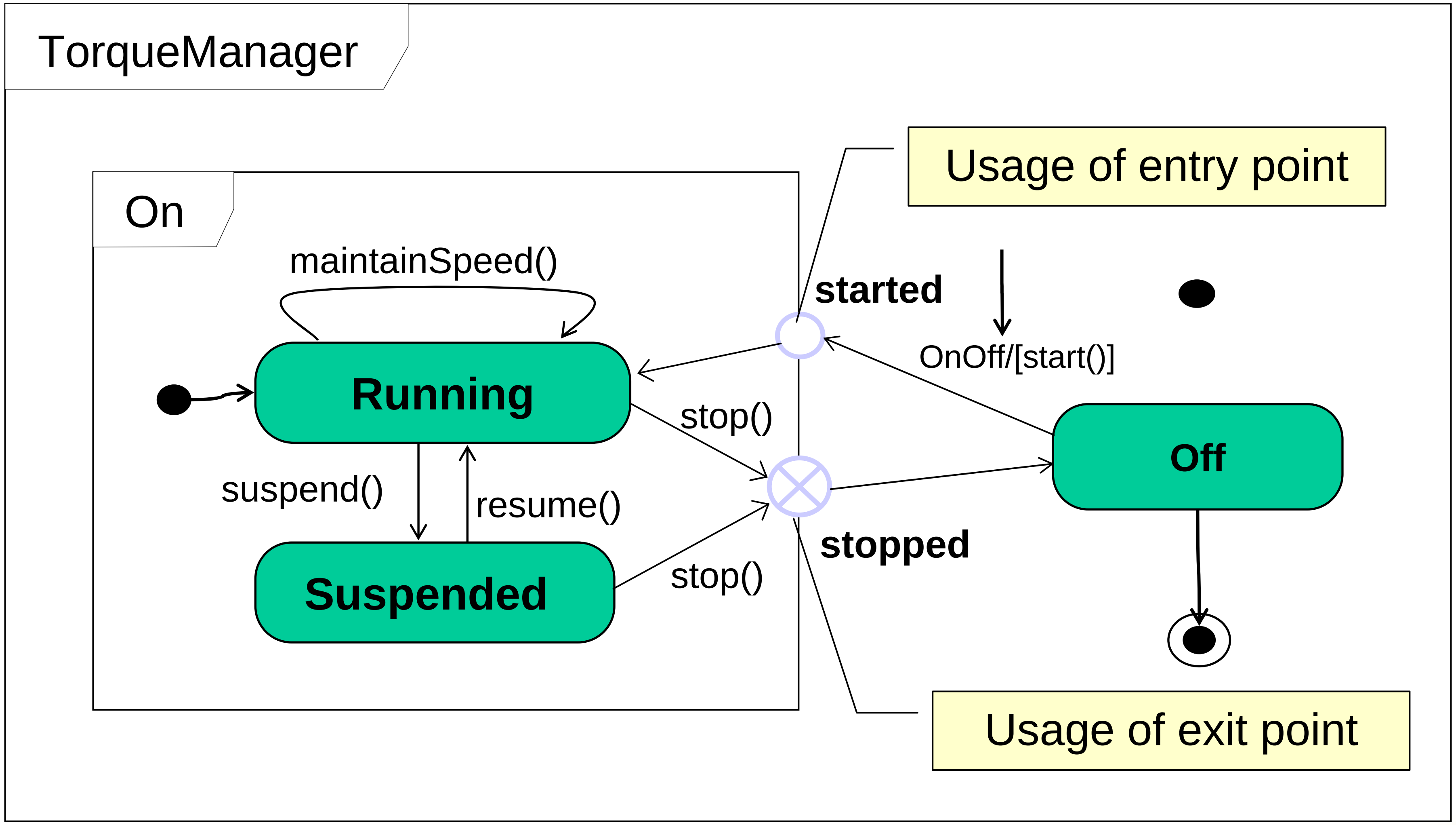
## Specification of Entry / Exit points on a state-machine



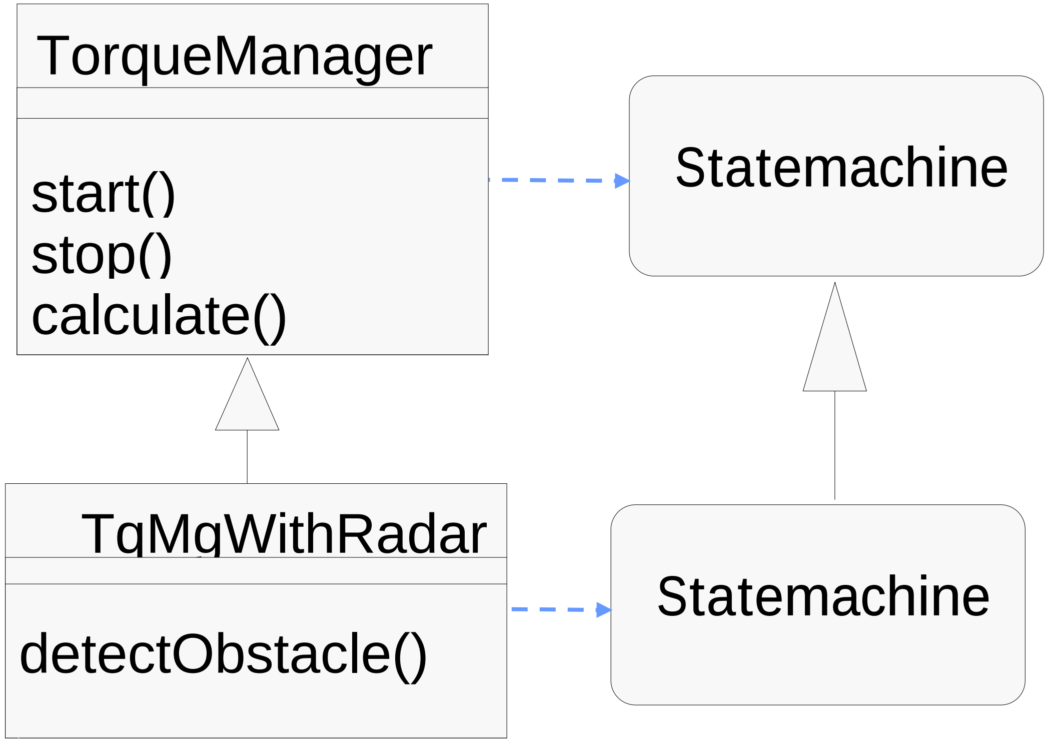
Specific entry point specification

Specific exit point specification

## Usage of Entry / Exit points



## Redefinition by Specialization

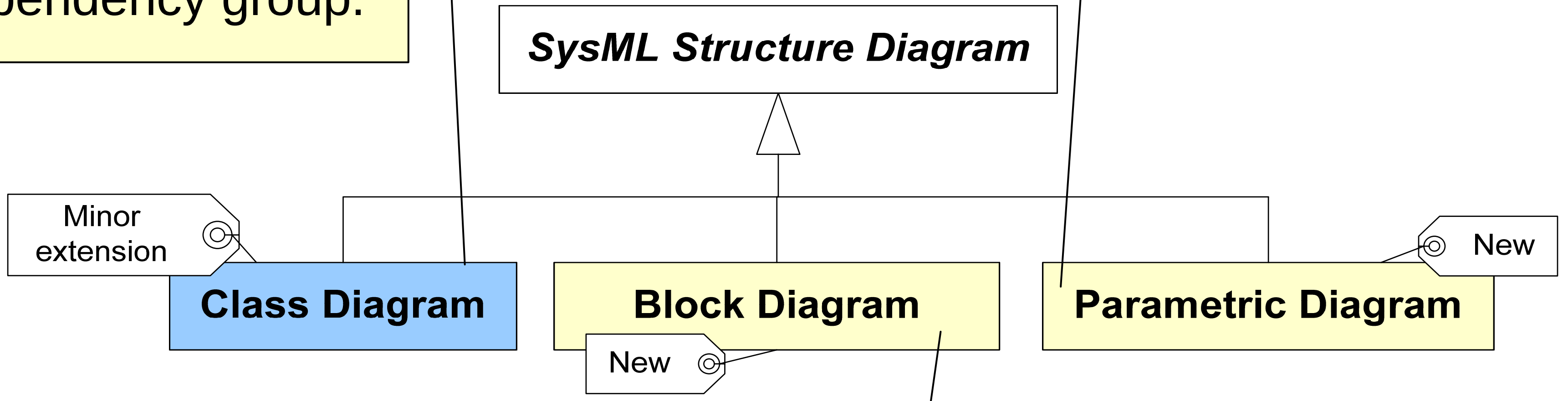


- States and transitions can be added
- States can be extended
- Regions can be added, and regions can be extended
- Submachine states can be replaced
- Transitions can be replaced or extended
  - Actions can be replaced
  - Guards can be replaced
  - Targets can be replaced

## Outlines of the SysML

- **Systems Modeling Language (www.SysML.org)**
  - General-purpose systems modeling language
    - Specification, analysis, design, verification and validation of a broad range of complex systems
- **UML-compatible systems modeling language**
  - For supporting the exchange of information using standardized notations and semantics that are understood in precise and consistent ways.
- **SysML will have to be customized to model domain specific applications**
  - Space, Automotive, Aerospace, Communications…
- **Aligned with the ISO AP-233 standard**

## From UML 2.0 to SysML: static point of view

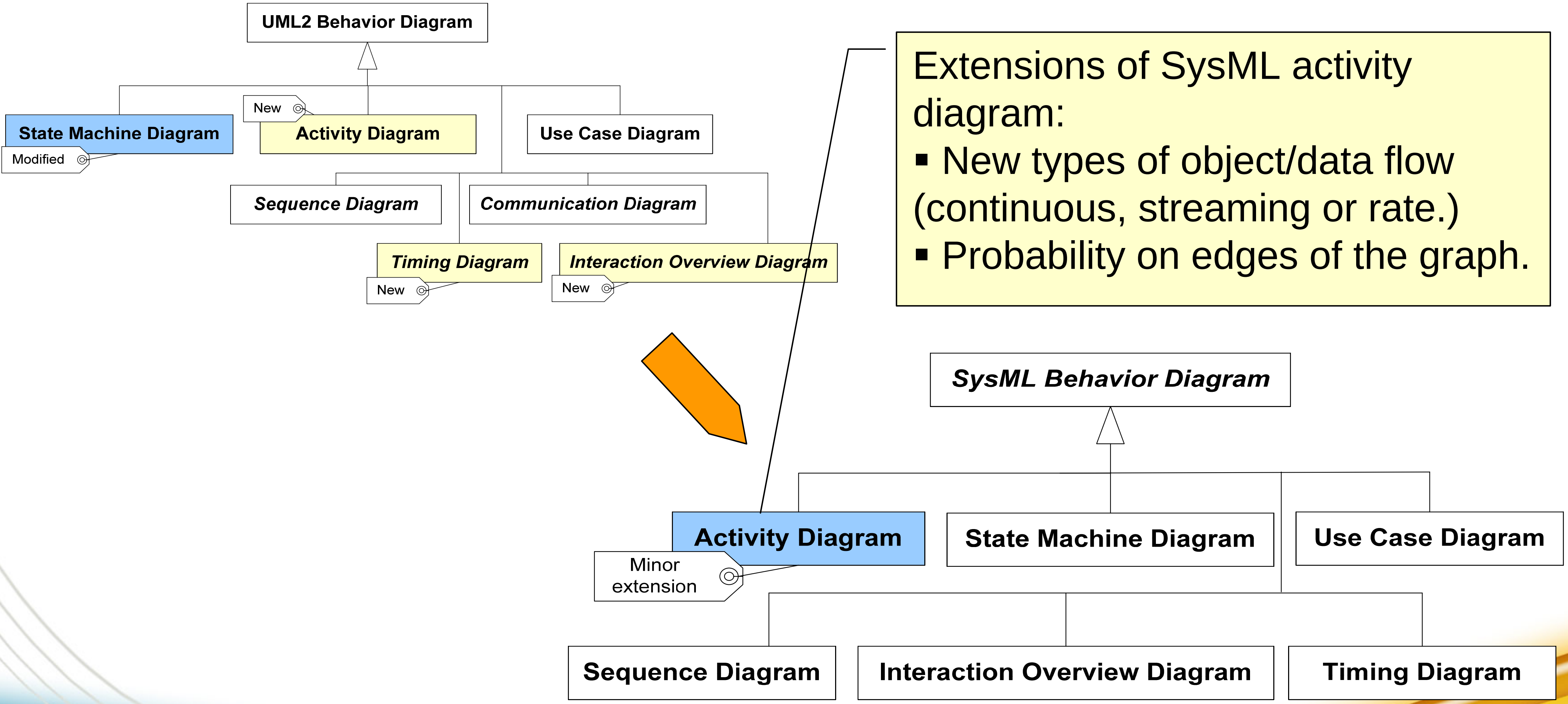


Minor extensions:
- Shortcut for inheritance modeling.
- Concepts of dependency group.

Define graphical notations for mathematical or logical constraints.

- Reused of UML2 structured classes.
- Provided more neutral concept for any kind of domain (not only preponderant software systems).

## From UML 2.0 to SysML: dynamic point of view



Extensions of SysML activity diagram:
- New types of object/data flow (continuous, streaming or rate.)
- Probability on edges of the graph.

## SysML transversal constructions

- **Allocation modeling**
  - Useful general modeling constructs
  - Need to be specialized for specific purposes

- **Requirements modeling**
  - Important contribution of SysML regarding UML2
  - Ensure seamless process between requirements and models engineering
  - Should be specialized for SSE in order to:
    - Introduce specific categories of requirements
    - Add new features on requirements categories

## UML Profiles for RTES

**SPT**

**(2005-adopted)**

- **SPT was the first OMG's UML profile for Real-Time Systems:**
  - Support for Schedulability Analysis with RMA-type techniques
  - Support for Performance Analysis with Queuing Theory and Petri Nets
  - A rich model for "metric" Time and Time Mechanisms
- **Several improvements were required:**
  - Modeling HW and SW platforms, Logical Time, MoCCs, CBSE…
  - Alignment to UML2, QoS&FT, MDA,…
  - SPT constructs were considered too abstract and hard to apply
  - …

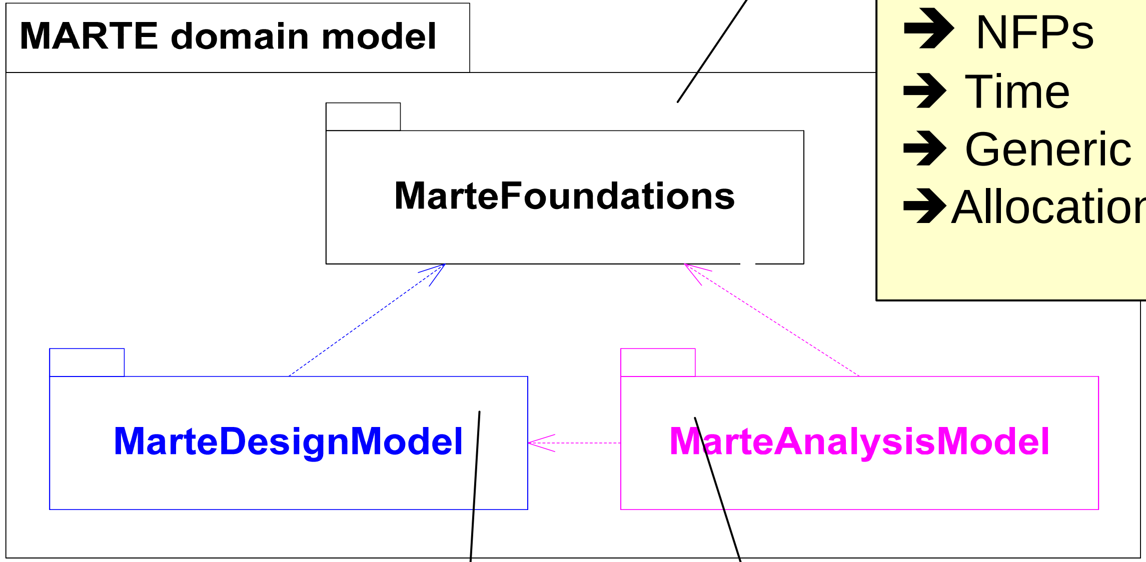
Hence, a Request For Proposal for a new profile was issued.

## The ProMARTE Team www.omgmarte.org

- **Industrials**
  - Alcatel*
  - Lockheed Martin*
  - Thales*
  - France-Telecom
- **Tool vendors**
  - ARTISAN Software Tools*
  - International Business Machines*
  - Mentor Graphics Corporation*
  - Softeam*
  - Telelogic AB (I-Logix*)
  - Tri-Pacific Software
  - France Telecom
  - No Magic
  - Mathworks
- **Academics**
  - Carleton University
  - Commissariat à l'Energie Atomique
  - ESEO
  - ENSIETA
  - INRIA
  - INSA from Lyon
  - Software Engineering Institute (Carnegie Mellon University)
  - Universidad de Cantabria

* Submitter to OMG UML Profile for MARTE RFP

## MARTE Architecture



Foundations for RT/E systems modeling and analysis:
➔ CoreElements
➔ NFPs
➔ Time
➔ Generic resource modeling
➔ Allocation

Specialization of MARTE foundations for modeling (e.g., specification and design):
➔ Generic component modeling
➔ High level modeling concepts for RTE
➔ Software resource modeling
➔ Hardware resource modeling

Specialization of foundations for annotating model for analysis:
➔ Generic quantitative analysis
➔ Schedulability analysis
➔ Performance analysis

## Some presentation conventions

- Within next slides, we may shown models at different levels of abstraction. We will clarify each level through following pictograms

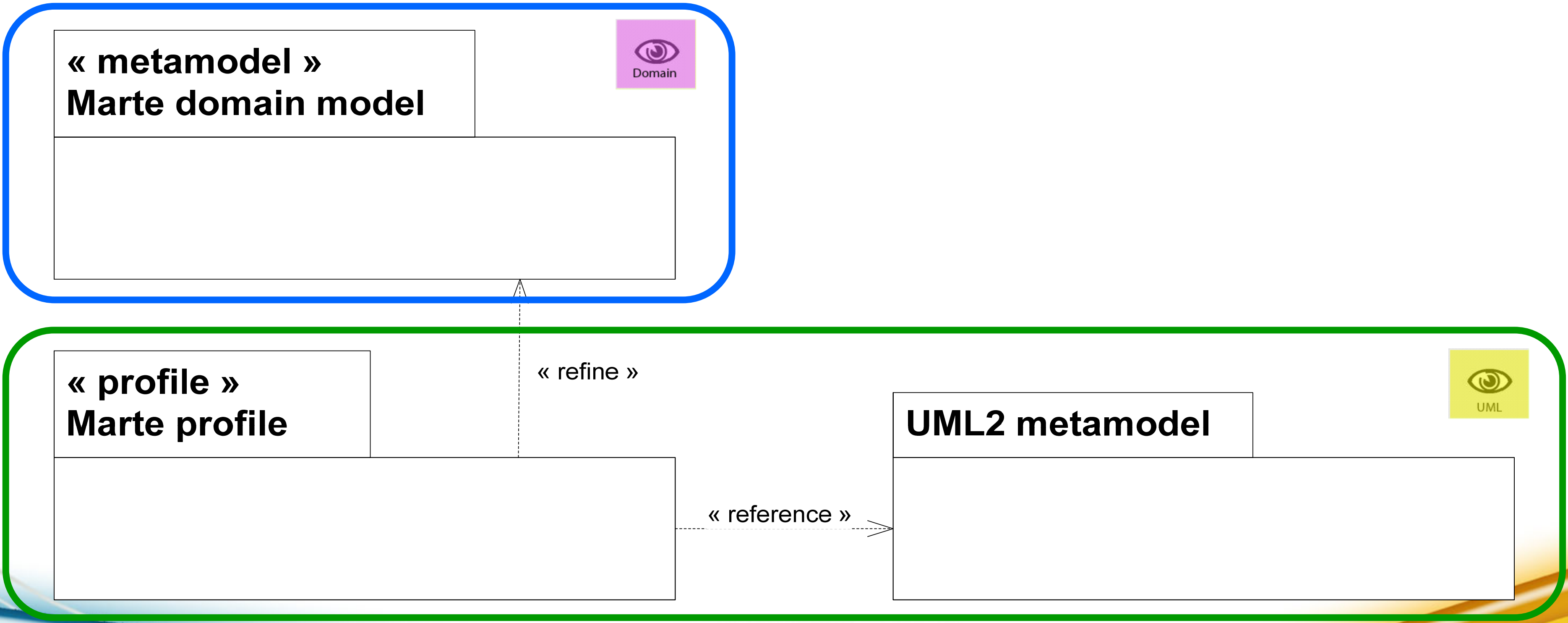
Domain — For Domain View level

UML — For UML Profile View Level

User — For User Model View Level

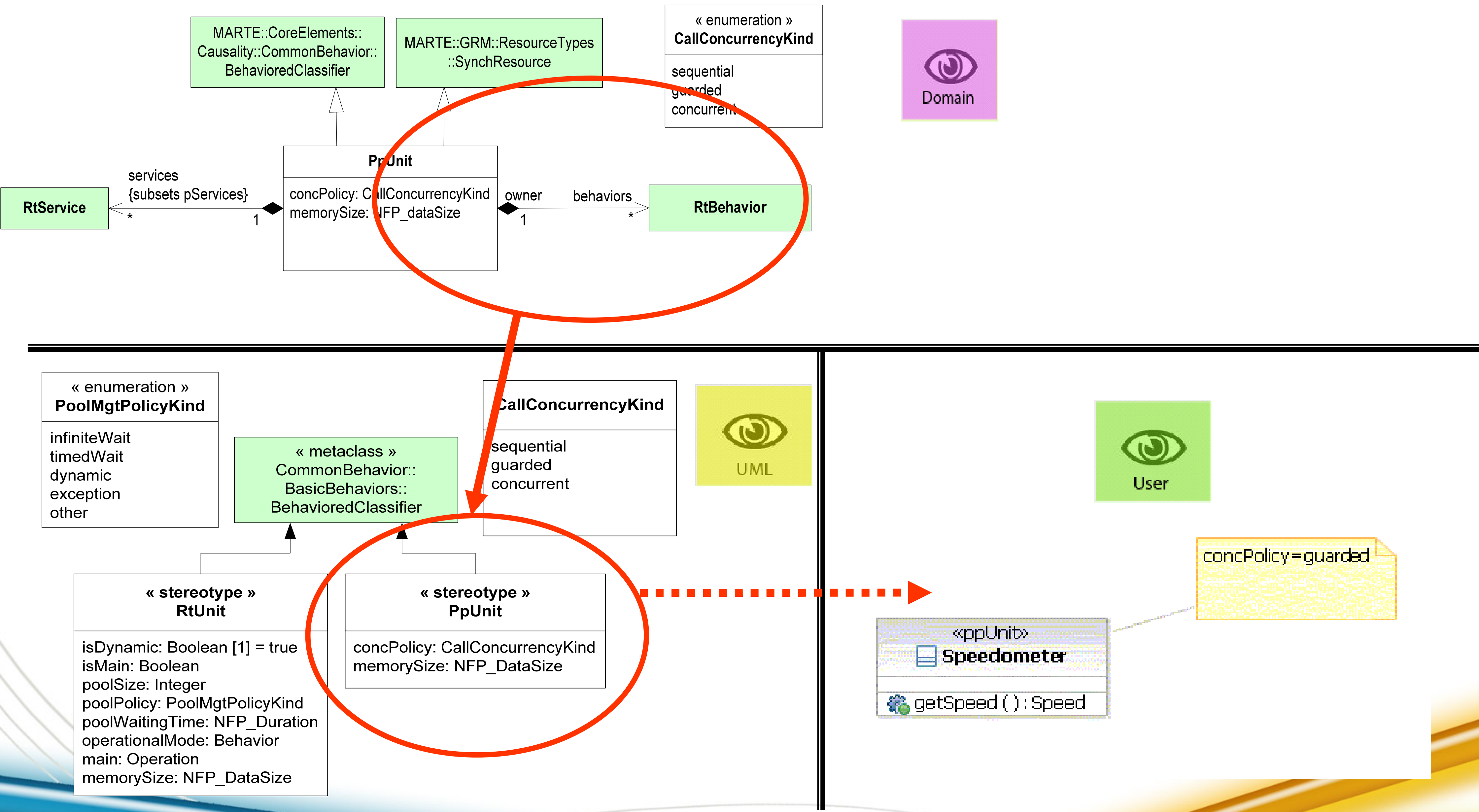
## Design Pattern Adopted for the MARTE Profile

- **Stage 1 → Description of MARTE domain models**
  **= Specification of the DSL**
  - Purpose: Formal description of the concepts required for MARTE
  - Techniques: Meta-modeling
- **Stage 2 → Mapping of MARTE domain models onto UML2 MM**
  **= UML design of the DSL**
  - Purpose: MARTE domain models design as a UML2 extensions
  - Techniques: UML2 profile



Extracted from S.Gerard (ECRTS07)

## Example: Domain model → Profile → Usage


MARTE::CoreElements::Causality::CommonBehavior::BehavioredClassifier
MARTE::GRM::ResourceTypes::SynchResource
« enumeration » CallConcurrencyKind
sequential
guarded
concurrent
Domain
RtService
services {subsets pServices}
*
1
PpUnit
concPolicy: CallConcurrencyKind
memorySize: NFP_dataSize
owner
1
behaviors
*
RtBehavior
« enumeration » PoolMgtPolicyKind
infiniteWait
timedWait
dynamic
exception
other
« metaclass » CommonBehavior::BasicBehaviors::BehavioredClassifier
CallConcurrencyKind
sequential
guarded
concurrent
UML
User
« stereotype » RtUnit
isDynamic: Boolean [1] = true
isMain: Boolean
poolSize: Integer
poolPolicy: PoolMgtPolicyKind
poolWaitingTime: NFP_Duration
operationalMode: Behavior
main: Operation
memorySize: NFP_DataSize
« stereotype » PpUnit
concPolicy: CallConcurrencyKind
memorySize: NFP_DataSize
concPolicy=guarded
«ppUnit»
Speedometer
getSpeed ( ) : Speed

## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
- **Part II: Some details on MARTE**
  - Non-functional Properties and Value Specification Language
  - CBME with GCM and HLAM
- **Part III: One typical usage example of MARTE**

## Examples of Non-Functional Specifications

- **Non Functional Requirements:**
  - Max. delay (deadline) of a control loop (sensor to actuator) = 10 ms
  - Max. deviation of a nominally periodic event (jitter) = 5.3 ms
  - Maximum average inter-session times = 20 minutes
  - Number of occurrences of an event in 50 ms interval time = 10 occurrences
- **And still…**
  - Ignition deadline according to the zero-position of a flywheel = 25 °CAM
  - Required confirmation delay: 95% < 500 ms
  - Required interval between frame displays = 99% < 30 ms
  - Hardware component MTBF = 90 %
  - Maximum processor utilization of P1 = 50 % of P2 utilization (math expressions, variables!)

## Examples of Non-Functional Specifications (seq.)

- **Non Functional Attributes:**
  - Estimated video server processing per frame = 25 ms
  - Assumed network delay distribution: exponential with mean = 10 ms
  - Measured packets per frame (LAN) = 65
  - Number of concurrent readers = $N_readers (parameters!)
  - Execution time of an application function by operational mode = 20 ms in nominal mode & 50 ms in degraded mode
  - Delay between the Nth and the Nth + 10 occurrence of an event = 125 ms iff generated data is grater than 10 KB
  - Dynamic power consumption of an electronic component = 12 mW

## Outline of the MARTE NFP Modeling Framework

- **NFPs sub-profile:**
  - Measurements: magnitude + unit (e.g., energy, data size, duration)
  - Value Qualifiers: Value source, statistical measure, precision…
- **Value Specification Language (VSL):**
  - Mathematical expressions (arithmetic, logical…)
  - Variables: placeholders for unknown analysis parameters.
  - Extended system of data types: tuples, collections, intervals…
  - Time expressions (delays, periods, trigger conditions…)

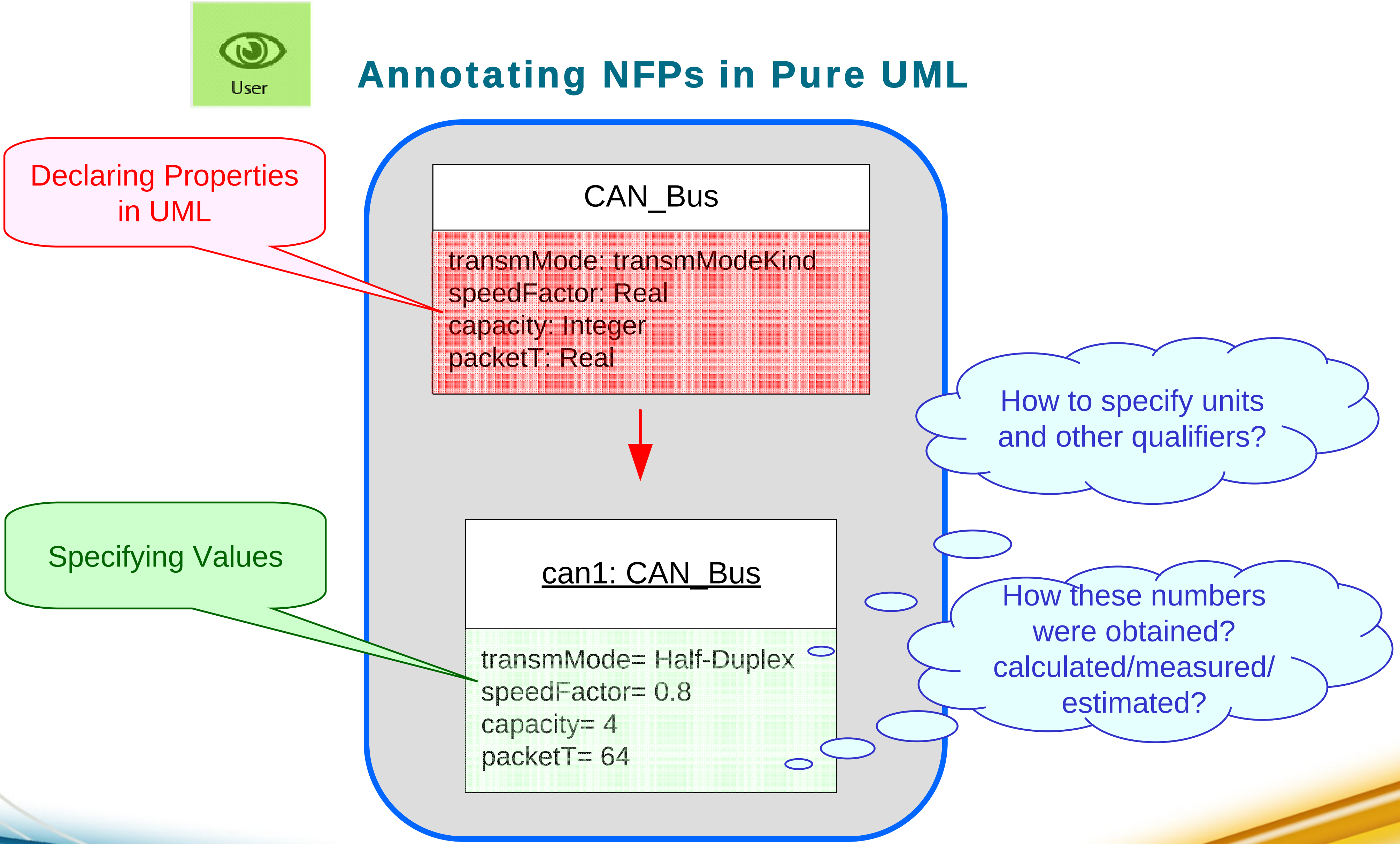
- **Why we need this level of formalization?**
  - Ability to support automated V&V.
  - Unambiguous understanding by stakeholders

## NFPs Sub-profile Landscape

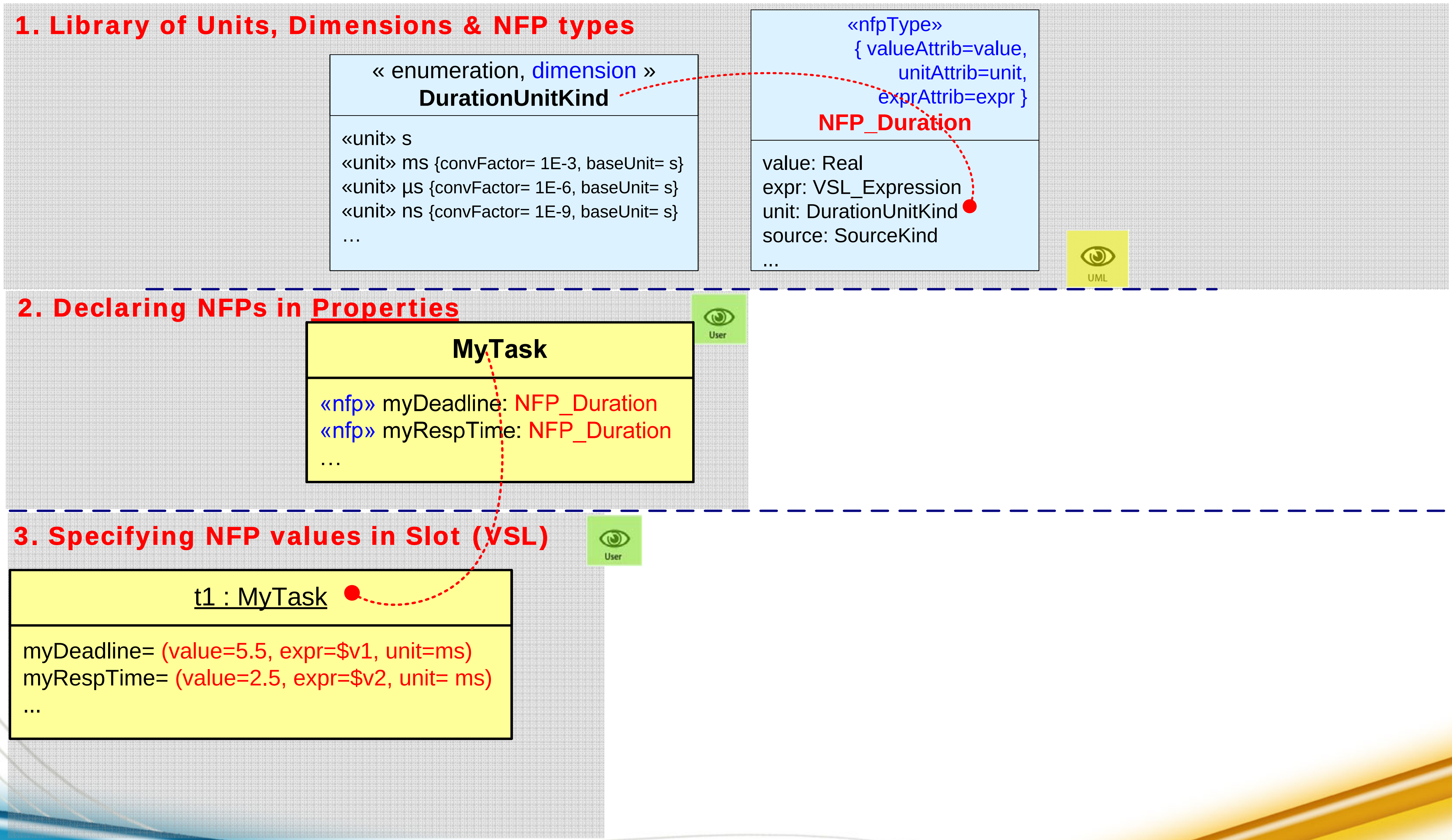
- **The NFPs Sub-profile: a set of extensions to specify semantically rich non-functional annotations**
- **Five stereotypes**
  - Nfp, NfpType, NfpConstraint, Unit, Dimension
- **A predefined library of Units, Dimensions and NfpTypes**
  - Power, Frequency, DataSize, DataTxRate, Duration, BoundDuration,…
  - A set of generic qualifiers for these NFP Types
- **Three mechanism to specify NFP values**
  - UML::ValueSpecification of InstanceSpecification Slots (M1)
  - Stereotype attributes (M2)
  - Constraints (M3)

## Specifying NFPs in Instance Slots: UML

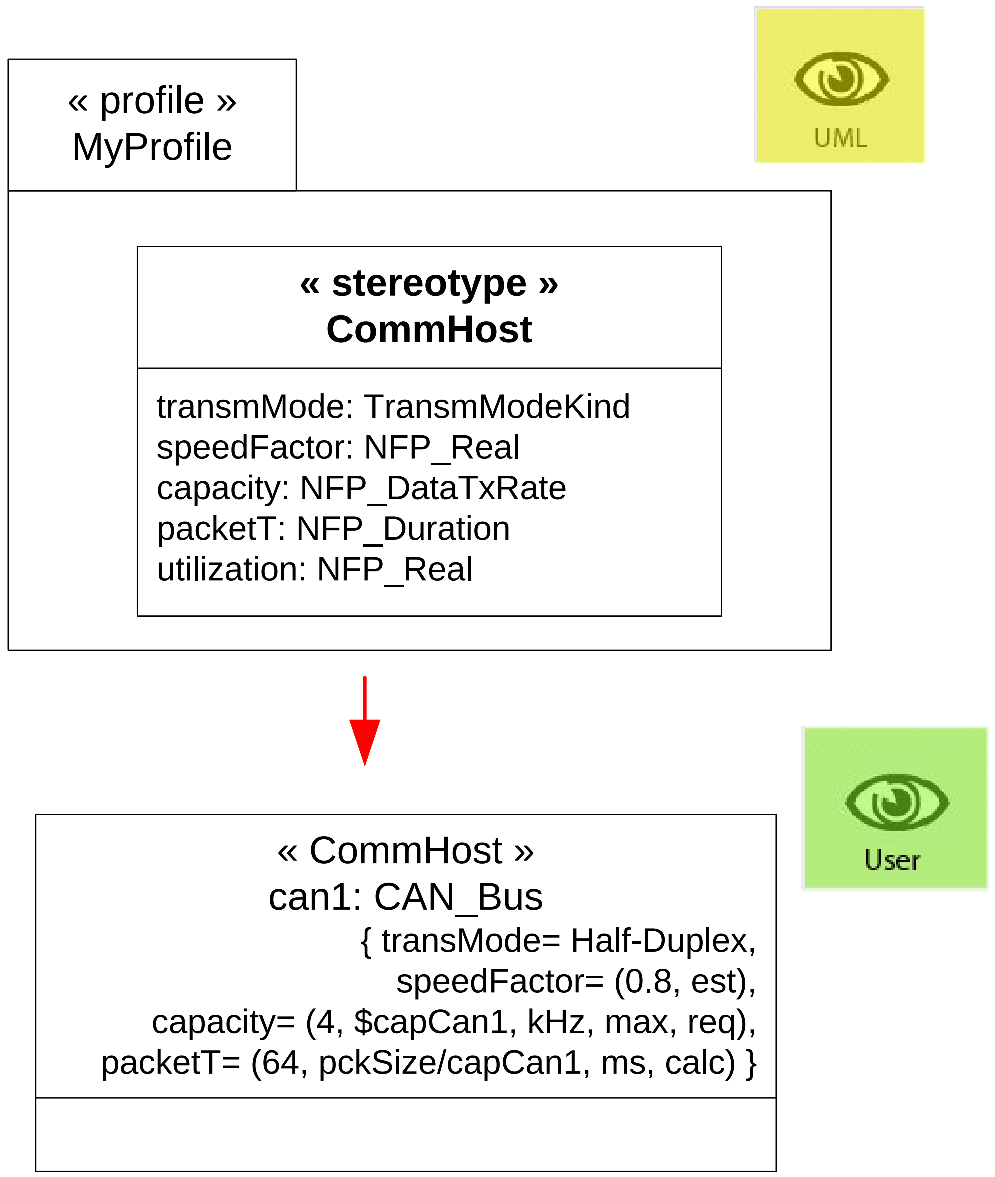
### Annotating NFPs in Pure UML


User
Declaring Properties in UML
CAN_Bus
transmMode: transmModeKind
speedFactor: Real
capacity: Integer
packetT: Real
Specifying Values
can1: CAN_Bus
transmMode= Half-Duplex
speedFactor= 0.8
capacity= 4
packetT= 64
How to specify units and other qualifiers?
How these numbers were obtained? calculated/measured/ estimated?

## Specifying NFPs in Instance Slots: MARTE

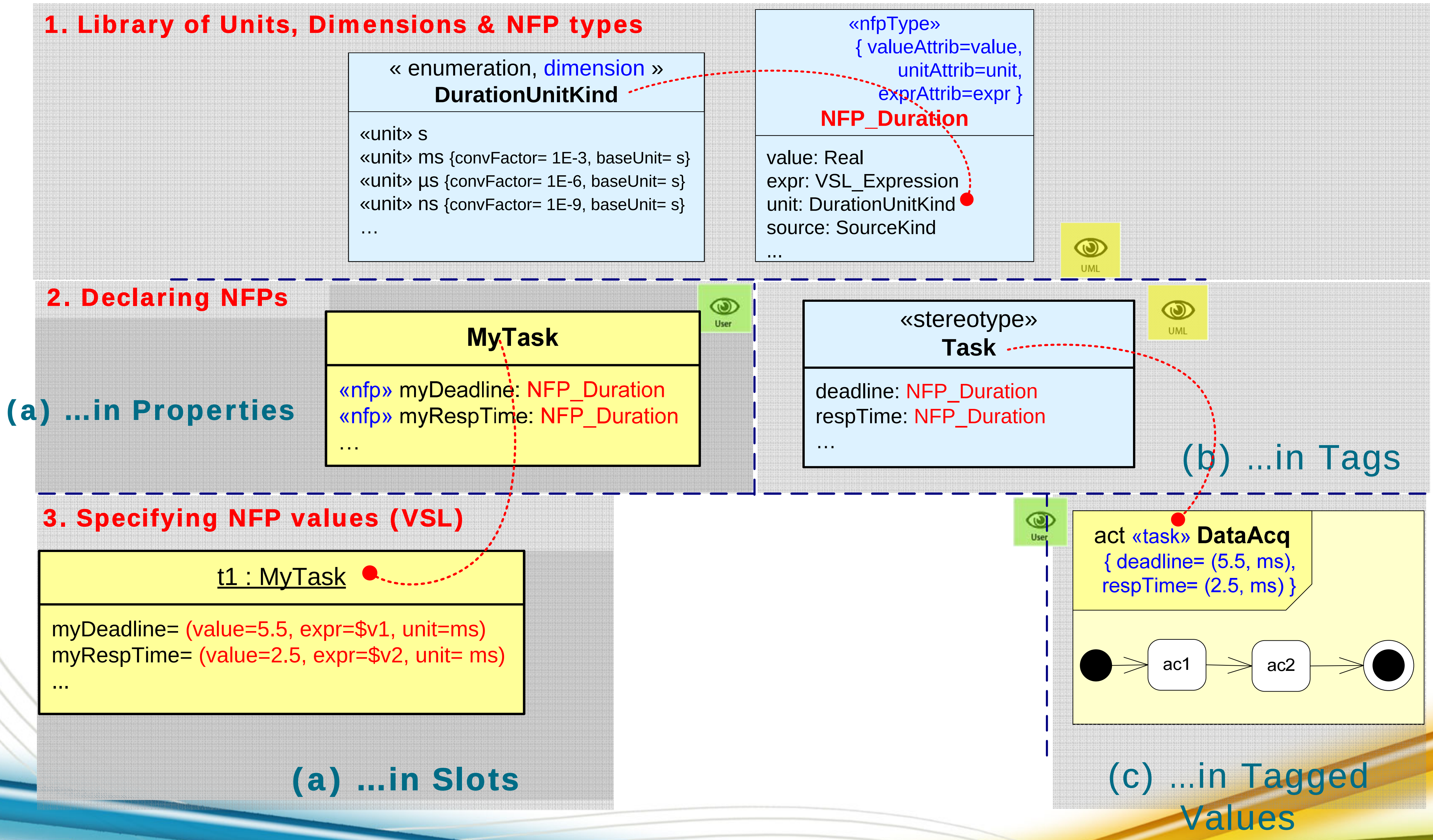

1. Library of Units, Dimensions & NFP types
« enumeration, dimension »
DurationUnitKind
«unit» s
«unit» ms {convFactor= 1E-3, baseUnit= s}
«unit» µs {convFactor= 1E-6, baseUnit= s}
«unit» ns {convFactor= 1E-9, baseUnit= s}
…
«nfpType»
{ valueAttrib=value,
unitAttrib=unit,
exprAttrib=expr }
NFP_Duration
value: Real
expr: VSL_Expression
unit: DurationUnitKind
source: SourceKind
...
UML
2. Declaring NFPs in Properties
User
MyTask
«nfp» myDeadline: NFP_Duration
«nfp» myRespTime: NFP_Duration
…
3. Specifying NFP values in Slot (VSL)
User
t1 : MyTask
myDeadline= (value=5.5, expr=$v1, unit=ms)
myRespTime= (value=2.5, expr=$v2, unit= ms)
...

## Specifying NFPs in Stereotype Attribute Values: MARTE



**CAN Bus annotated with Stereotypes**

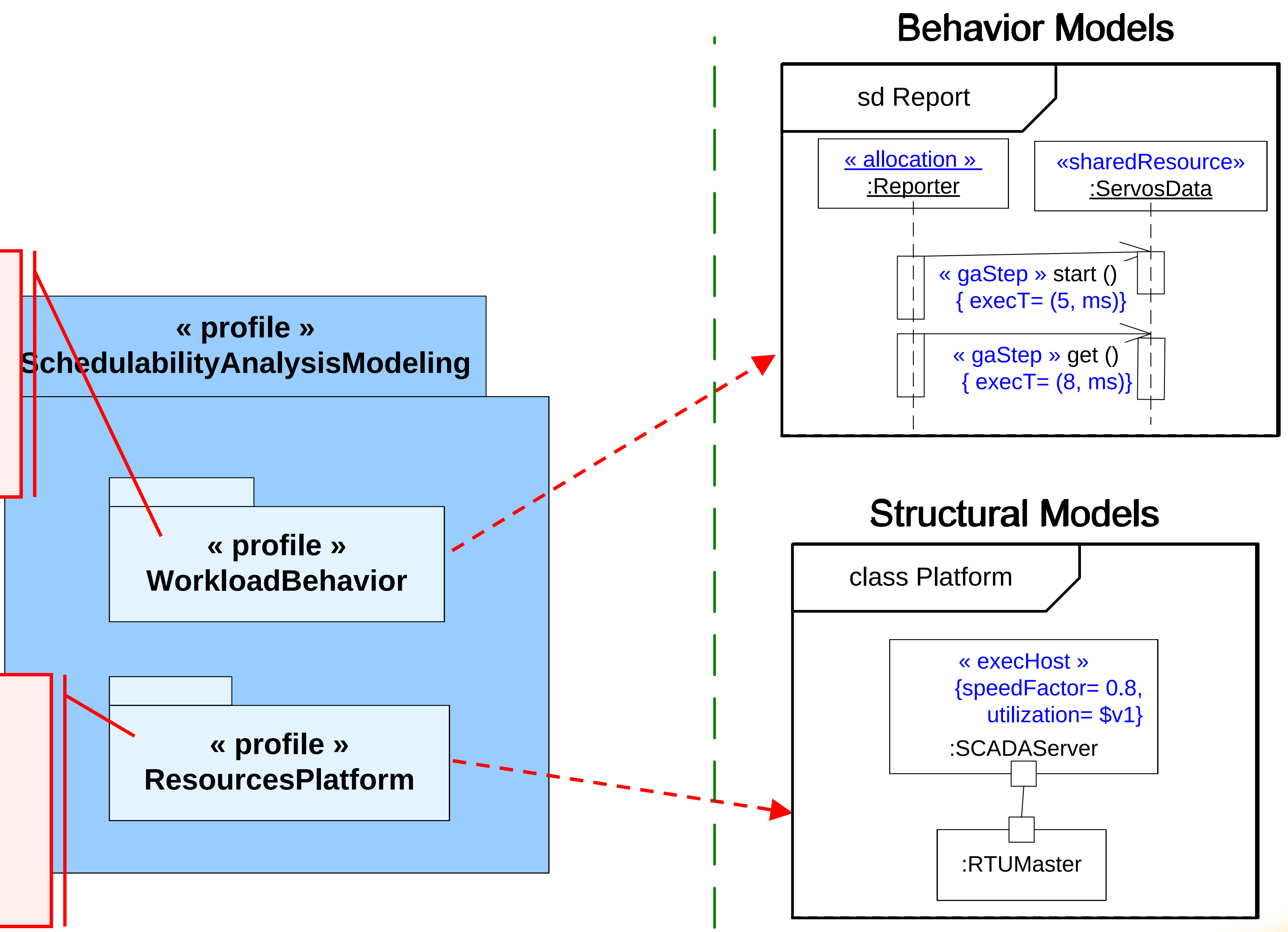
## M2. Specifying NFPs in Stereotype Attribute Values: MARTE



## Using NFPs in the other MARTE Sub-profiles

**Stereotype attributes**:
- •Stimuli models
- •Execution times
- •Transmission Delays
- •Constraints

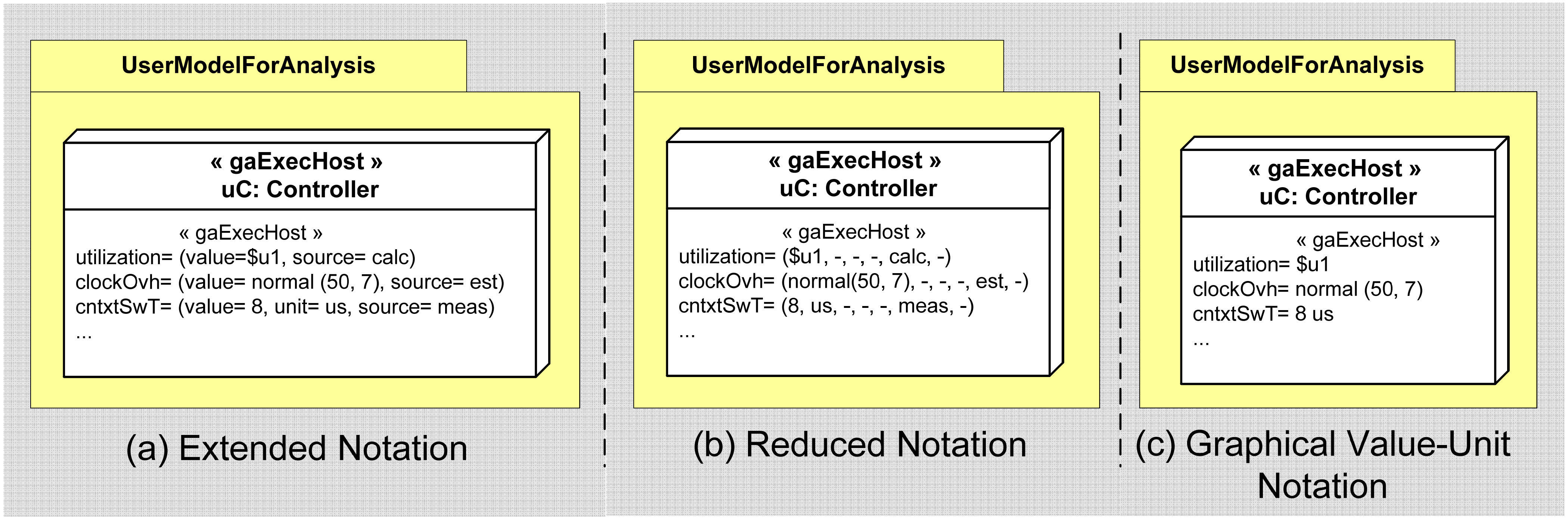
**Stereotype attributes:**
- •Capacity
- •Overheads
- •Scheduling schemes
- •Resources access



analysis language construct (UML profile)

UML user models

## Syntax of NFP Values



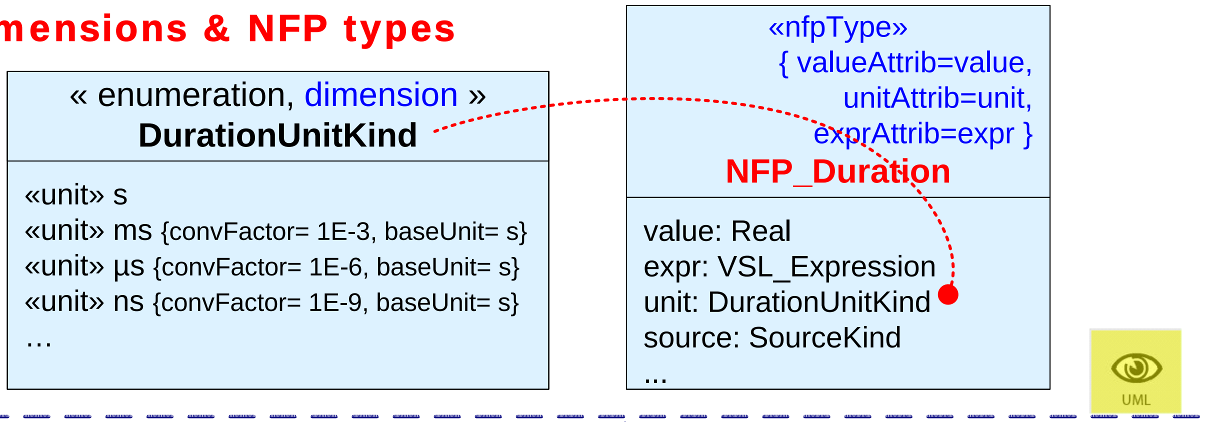
(a) Extended Notation

(b) Reduced Notation

(c) Graphical Value-Unit Notation

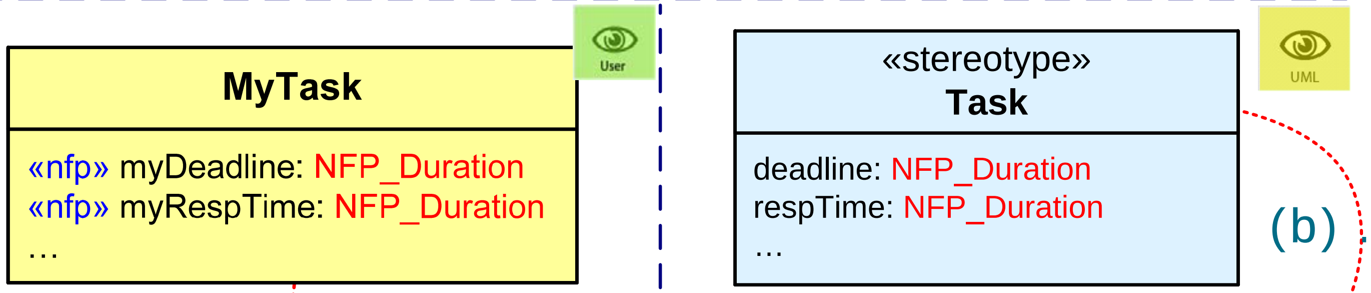
## Specifying NFPs in UML::Constraints: MARTE

**1. Library of Units, Dimensions & NFP types**



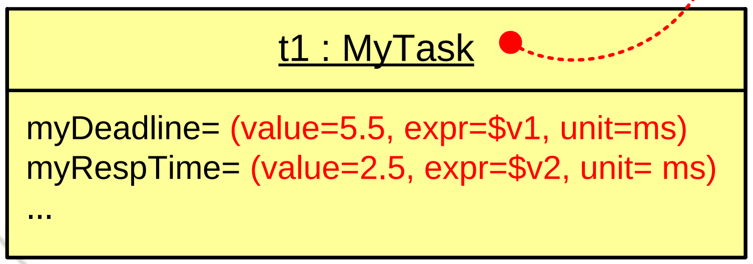
**2. Declaring NFPs**

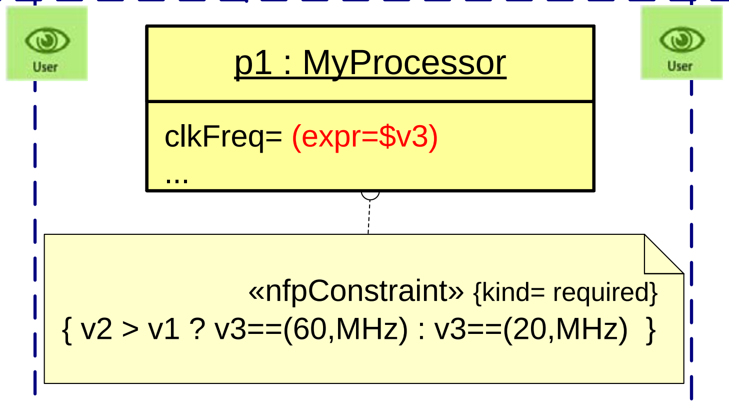
(a) ...in Properties

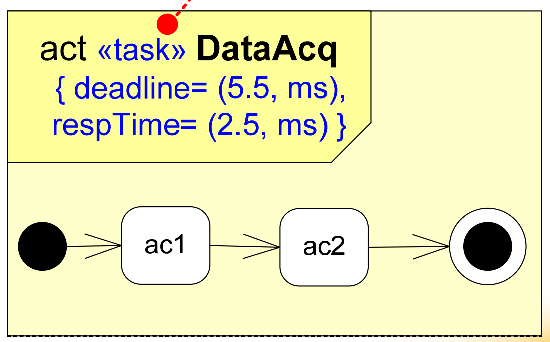


(b) ...in Tags

**3. Specifying NFP values (VSL)**


t1 : MyTask
myDeadline= (value=5.5, expr=$v1, unit=ms)
myRespTime= (value=2.5, expr=$v2, unit= ms)
...


User
p1 : MyProcessor
clkFreq= (expr=$v3)
...
«nfpConstraint» {kind= required}
{ v2 > v1 ? v3==(60,MHz) : v3==(20,MHz) }
User



(a) ...in Slots (b) ...in Constraints (c) ...in Tagged Values

## The Value Specification Language (VSL)

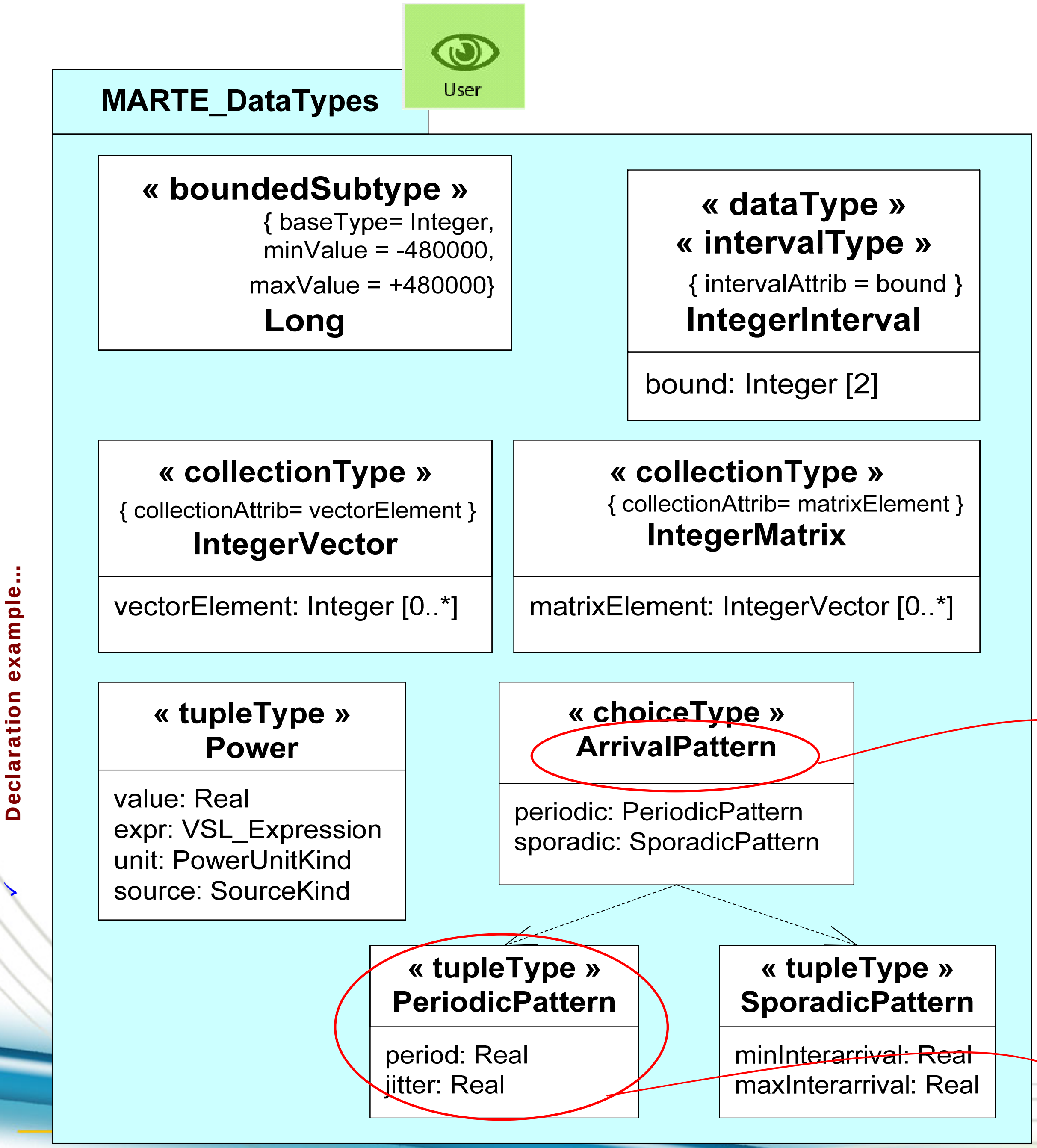
The expression language for values, functions, variables, and (basic) time expressions

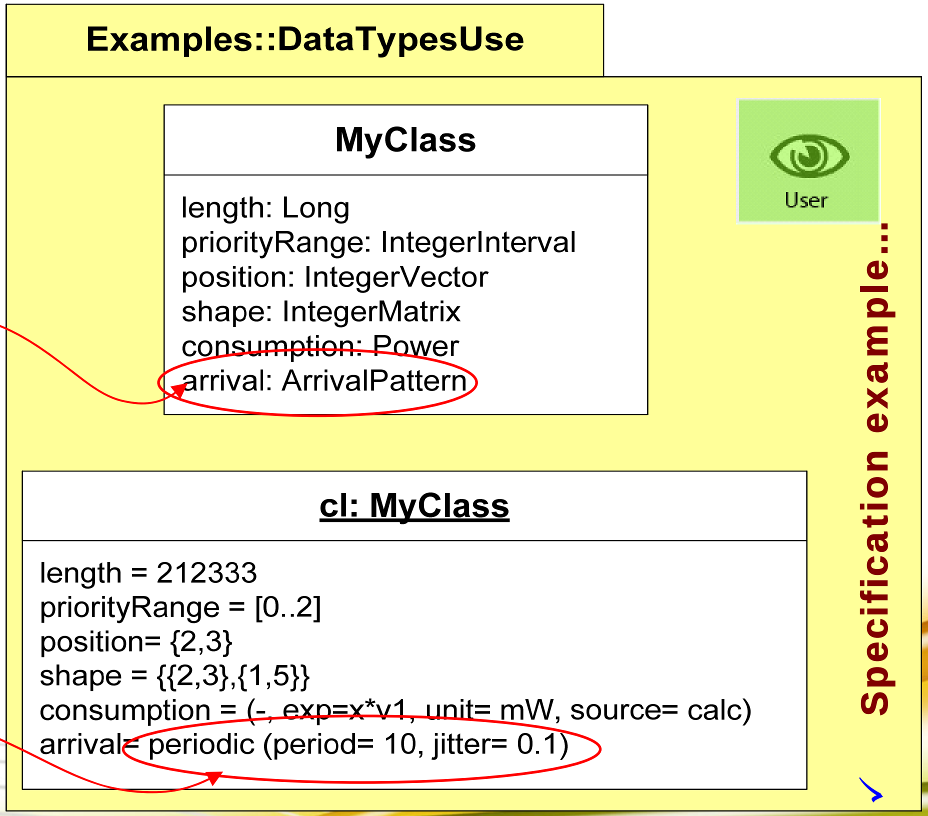
## VSL Landscape

- **The Value Specification Language (VSL)**
  - The expression language for values, functions, variables, and (basic) time expressions
- **Formally defines:**
  - A Set of Stereotypes extending UML::DataTypes
  - A Grammar (EBNF) define the VSL textual syntax.
- **A extended system of data types:**
  - Composite types: Tuples, Collection, Choice, Interval types
  - Subtypes: bounded subtype
- **A extended language for expressions:**
  - Primitives, Composites, Expression & Variables, Time Expressions.

# VSL Extended Data Types

- BoundedSubtype
- IntervalType
- CollectionType
- TupleType
- ChoiceType





## Basic Textual Expressions in VSL

- **Extended Primitive Values**
- **Extended Composite Values**
- **Extended Expressions**

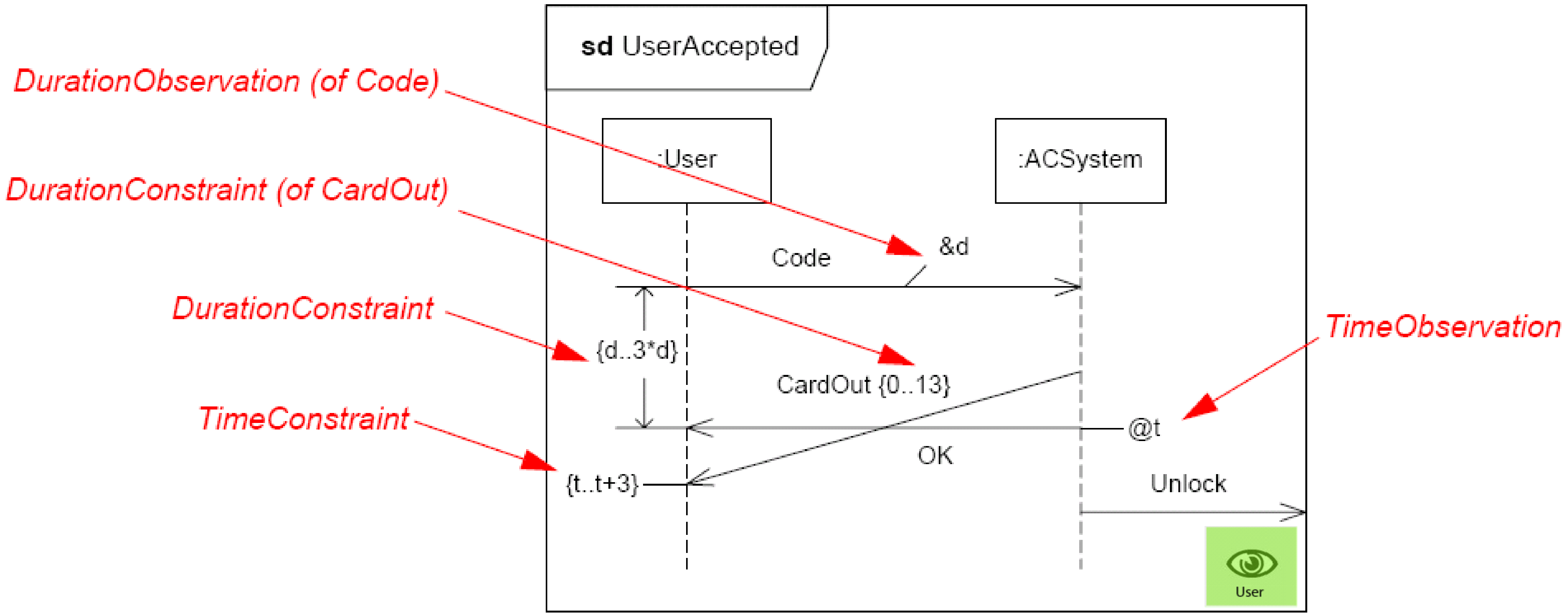
| Value Spec. | Examples |
|---|---|
| *Real Number* | `1.2E-3        //scientific notation` |
| *DateTime* | `#12/01/06 12:00:00#   //calendar date time` |
| *Collection* | `{1, 2, 88, 5, 2}  //sequence, bag, ordered set..`<br>`{{1,2,3}, {3,2}} //collection of collections` |
| *Tuple and choice* | `(value=2.0, unit= ms)    //duration tuple value`<br>`periodic(period=2.0, jitter=3.3) //arrival pattern` |
| *Interval* | `[1..251[  //upper opened interval between integers`<br>`[$A1..$A2]      //interval between variables` |
| *Variable declaration & Call* | `io$var1      //input/output variable declaration`<br>`var1       //variable call expression.` |
| *Arithmetic Operation Call* | `+(5.0,var1) //”add” operation on Real datatypes`<br>`5.0+var1    //infix operator notation` |
| *Conditional Expression* | `((var1<6.0)?(10^6):1)  //if true return 10 exp 6,else 1` |

## UML (native) Observation Concept

An time observation is a reference to a time instant during execution.

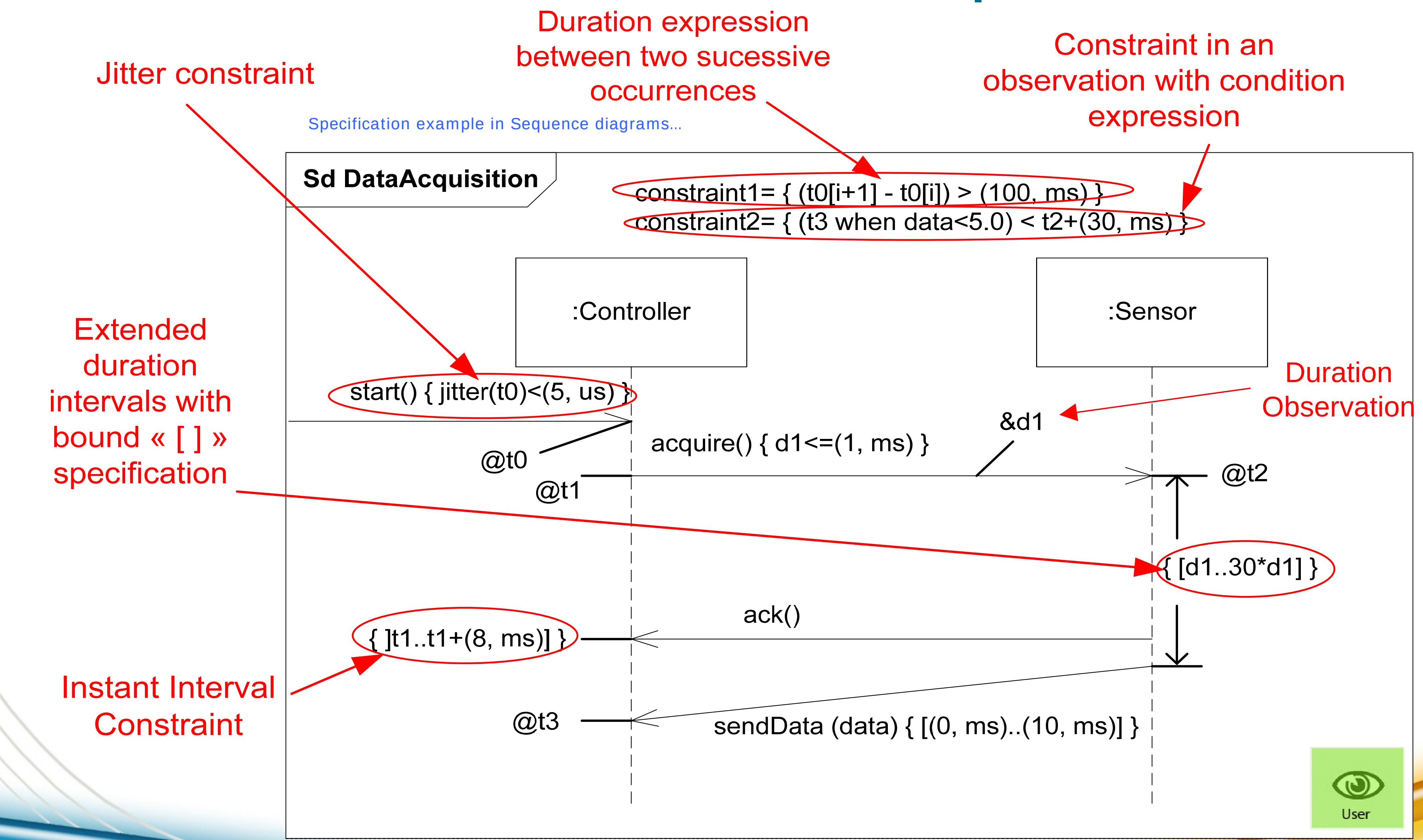
An duration observation is a reference to a time interval during execution.

Specification example in Sequence diagrams…

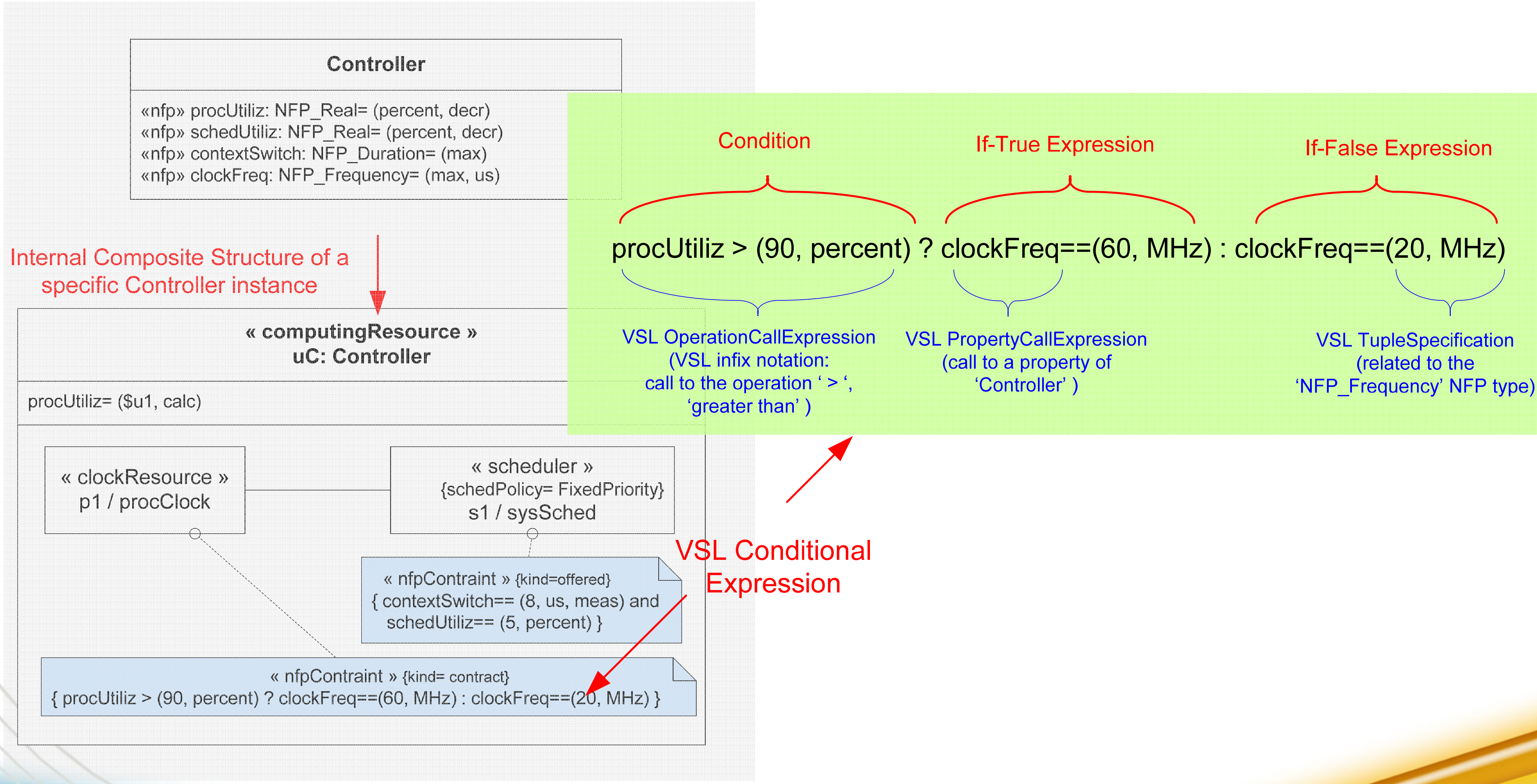


## Time Expressions with VSL

Specification example in Sequence diagrams…


Jitter constraint
Duration expression between two sucessive occurrences
Constraint in an observation with condition expression
Sd DataAcquisition
constraint1= { (t0[i+1] - t0[i]) > (100, ms) }
constraint2= { (t3 when data<5.0) < t2+(30, ms) }
:Controller
:Sensor
Extended duration intervals with bound « [ ] » specification
start() { jitter(t0)<(5, us) }
Duration Observation
&d1
acquire() { d1<=(1, ms) }
@t0
@t1
@t2
{ [d1..30*d1] }
ack()
{ ]t1..t1+(8, ms)] }
Instant Interval Constraint
@t3
sendData (data) { [(0, ms)..(10, ms)] }
User

# Integrating VSL Expressions in *NfpConstraints*


Controller
«nfp» procUtiliz: NFP_Real= (percent, decr)
«nfp» schedUtiliz: NFP_Real= (percent, decr)
«nfp» contextSwitch: NFP_Duration= (max)
«nfp» clockFreq: NFP_Frequency= (max, us)
Internal Composite Structure of a specific Controller instance
« computingResource »
uC: Controller
procUtiliz= ($u1, calc)
« clockResource »
p1 / procClock
« scheduler »
{schedPolicy= FixedPriority}
s1 / sysSched
« nfpContraint » {kind=offered}
{ contextSwitch== (8, us, meas) and schedUtiliz== (5, percent) }
« nfpContraint » {kind= contract}
{ procUtiliz > (90, percent) ? clockFreq==(60, MHz) : clockFreq==(20, MHz) }
Condition
If-True Expression
If-False Expression
procUtiliz > (90, percent) ? clockFreq==(60, MHz) : clockFreq==(20, MHz)
VSL OperationCallExpression (VSL infix notation: call to the operation ' > ', 'greater than' )
VSL PropertyCallExpression (call to a property of 'Controller' )
VSL TupleSpecification (related to the 'NFP_Frequency' NFP type)
VSL Conditional Expression

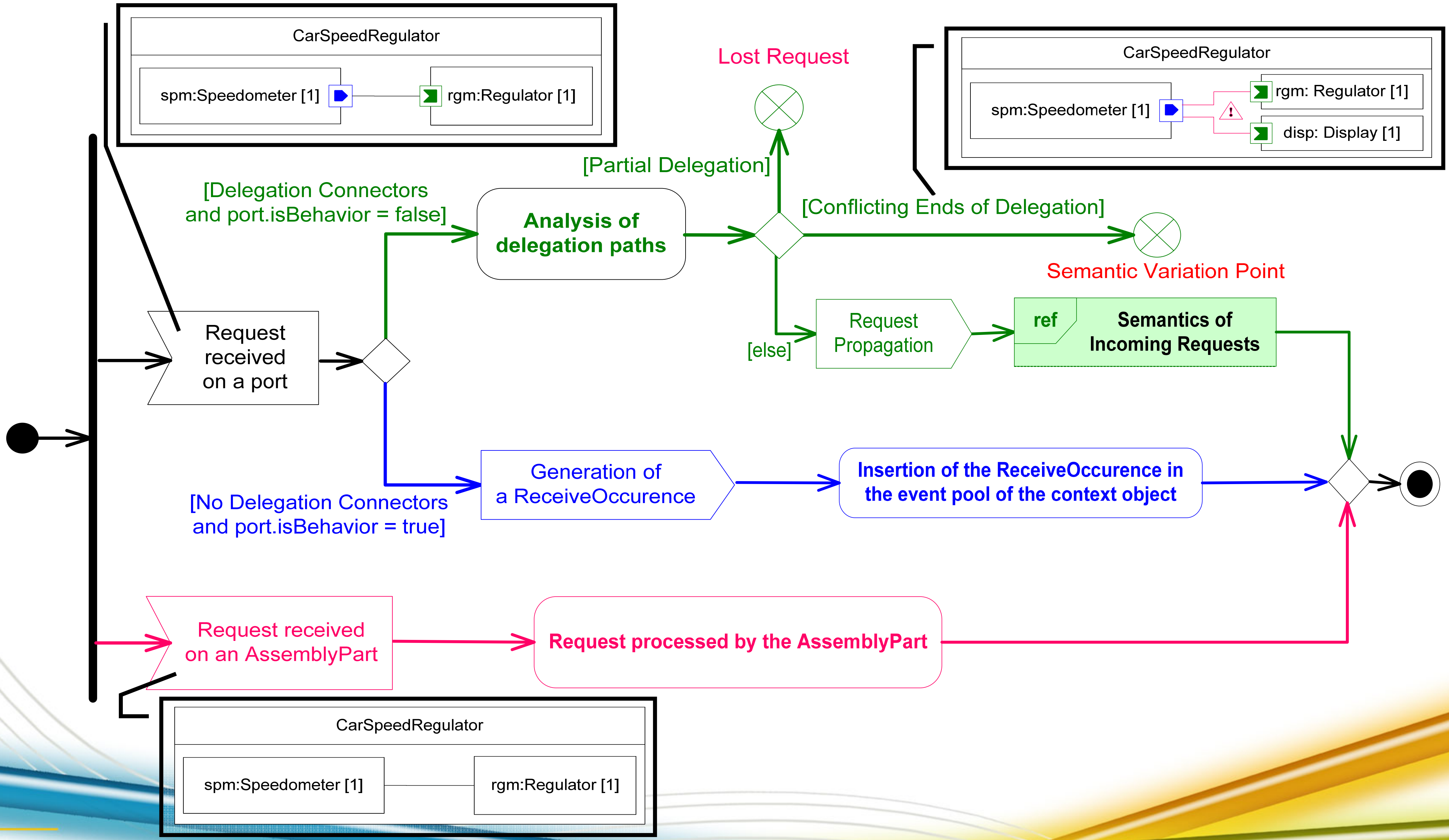
## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
- **Part II: Some details on MARTE**
  - Non-functional Properties and Value Specification Language
  - **CBME with GCM and HLAM**
- **Part III: One typical usage example of MARTE**

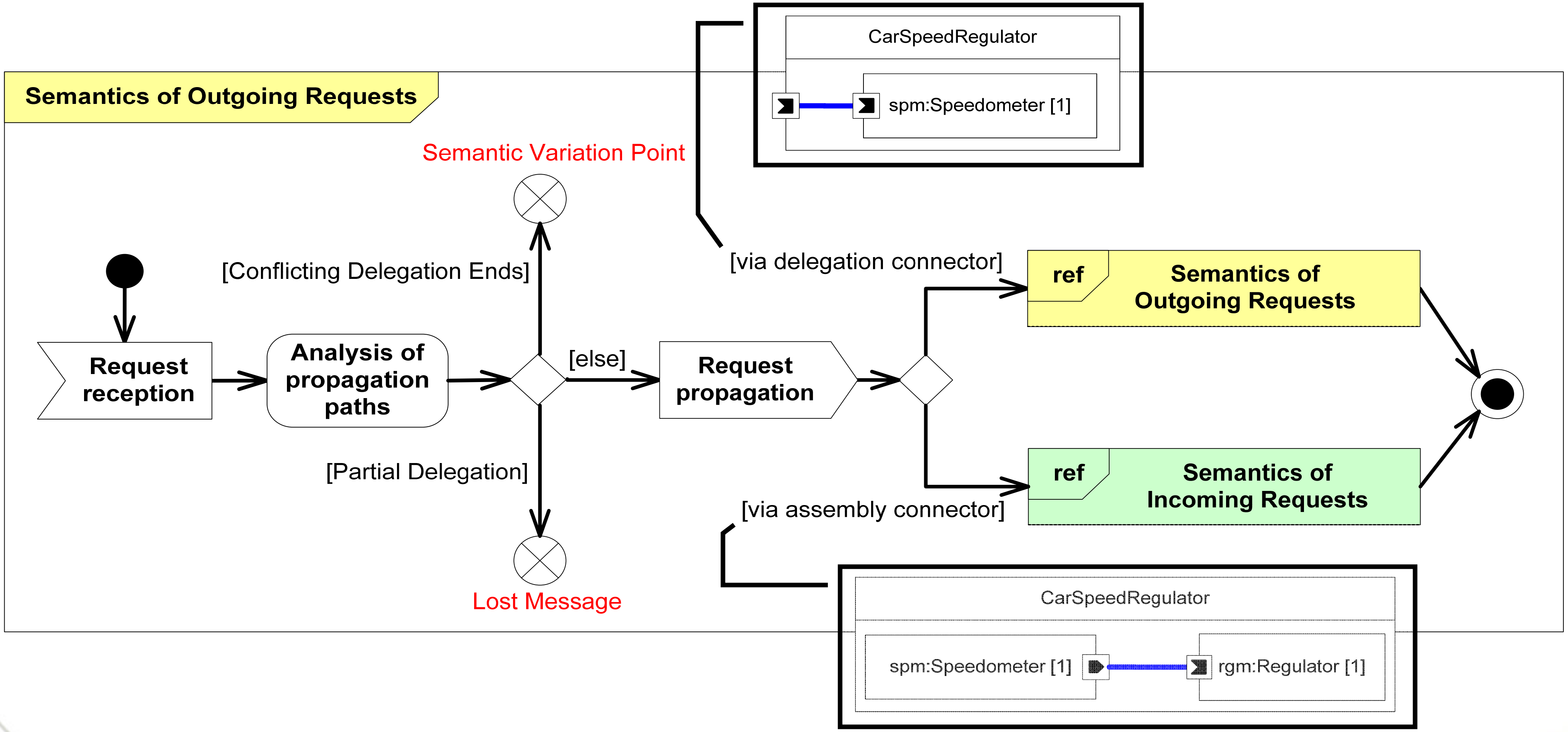
## MARTE’s GCM in a nutshell

- **Introduced to cope with various RTE component models**
  - AADL, Autosar, EAST-ADL2, Lightweight-CCM, XP-ACT and SysML.
- **Precise semantics supporting various models of computation and communication**
  - Clarification of the relationships between structural and behavioral aspects.
- **Relies on UML structured classes, and only extend UML’s Port**

# MARTE's GCM causality model: incoming requests

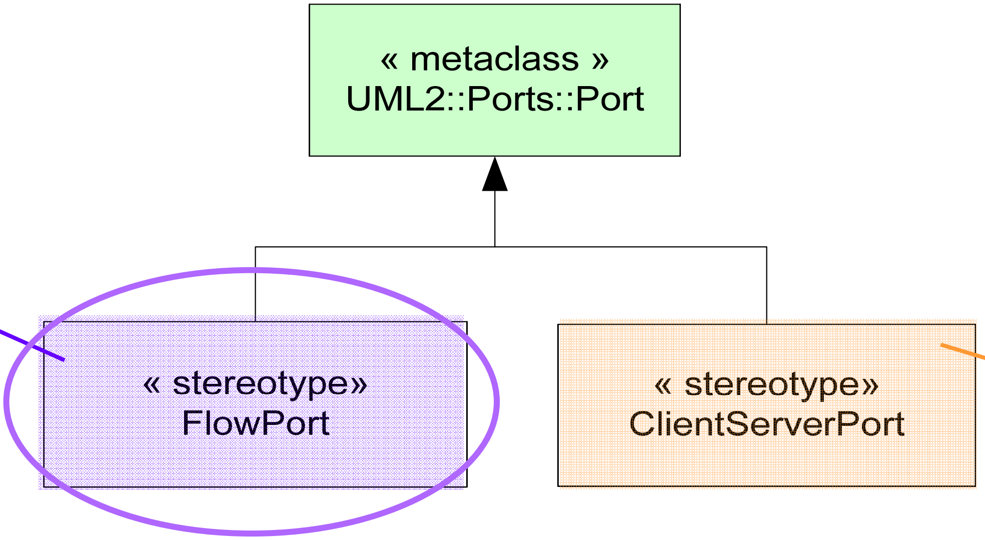

CarSpeedRegulator
spm:Speedometer [1]
rgm:Regulator [1]
Lost Request
[Partial Delegation]
CarSpeedRegulator
spm:Speedometer [1]
rgm: Regulator [1]
disp: Display [1]
[Delegation Connectors and port.isBehavior = false]
Analysis of delegation paths
[Conflicting Ends of Delegation]
Semantic Variation Point
Request received on a port
[else]
Request Propagation
ref
Semantics of Incoming Requests
Generation of a ReceiveOccurence
Insertion of the ReceiveOccurence in the event pool of the context object
[No Delegation Connectors and port.isBehavior = true]
Request received on an AssemblyPart
Request processed by the AssemblyPart
CarSpeedRegulator
spm:Speedometer [1]
rgm:Regulator [1]

## MARTE’s GCM causality model: outgoing requests



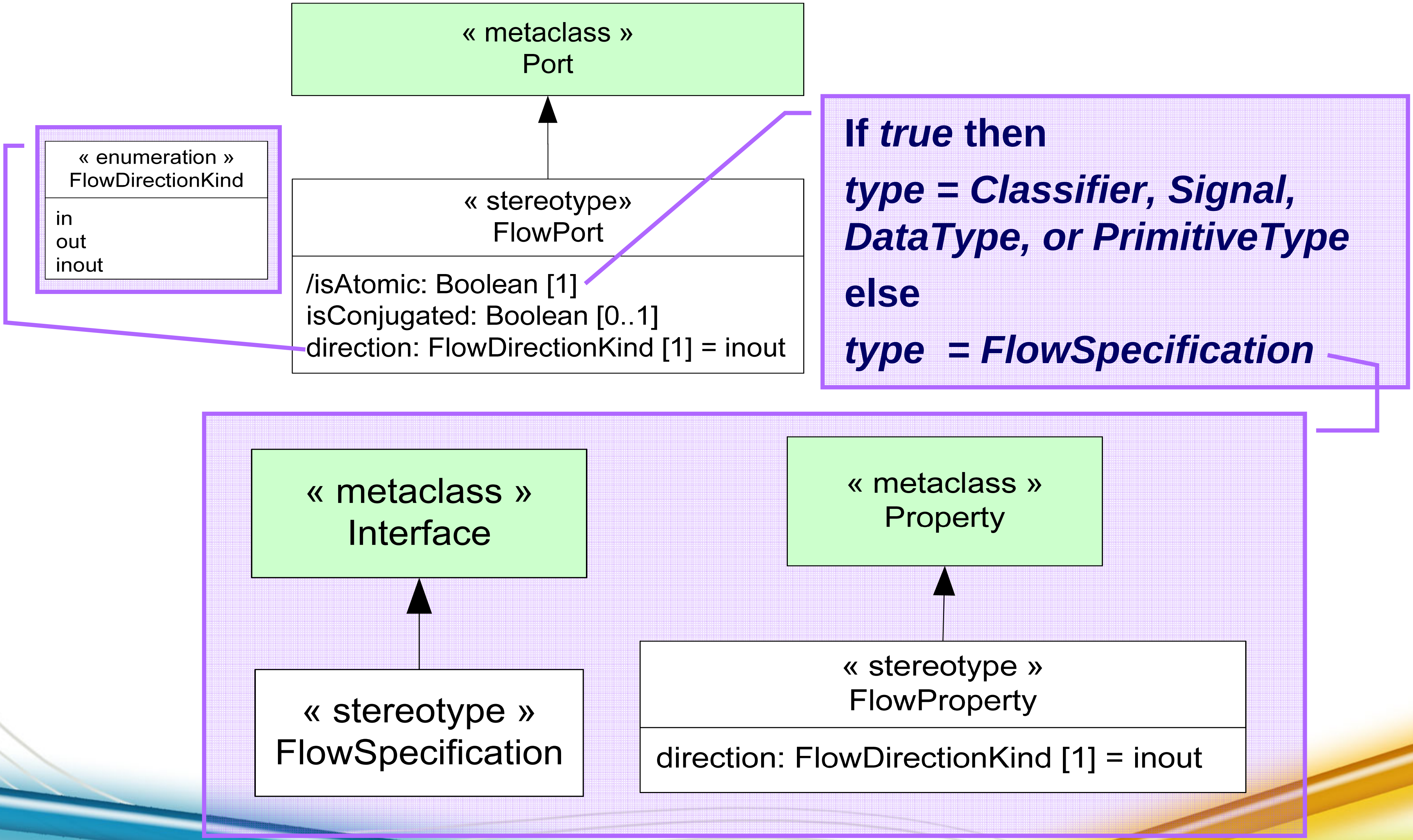
## MARTE extensions to UML Ports

Support for data-based communication schema between components (similar to SysML).



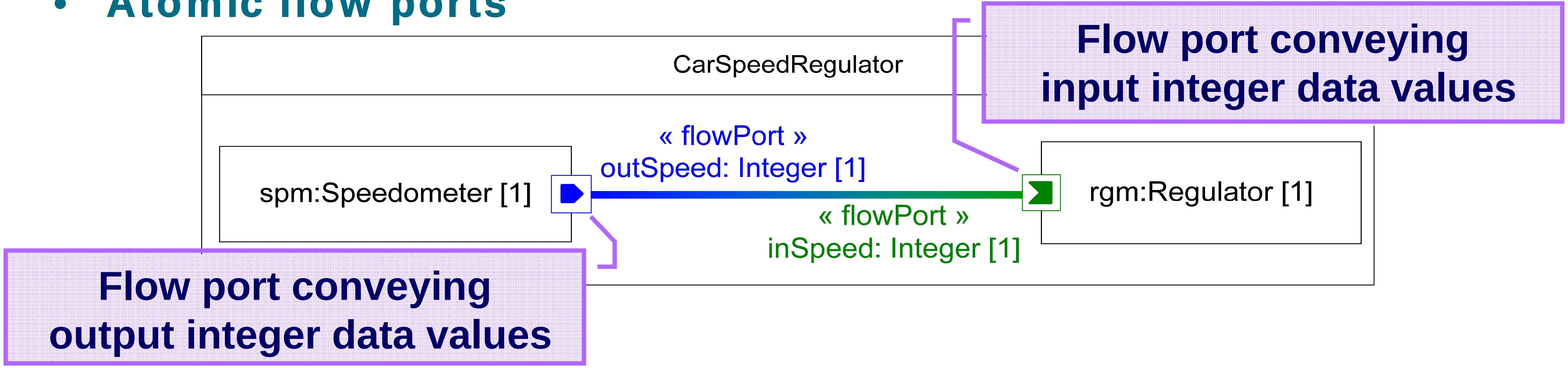
Support for “classic” OO message-based communication schema (including UML signal-based communication).

## MARTE flow port: syntactical view

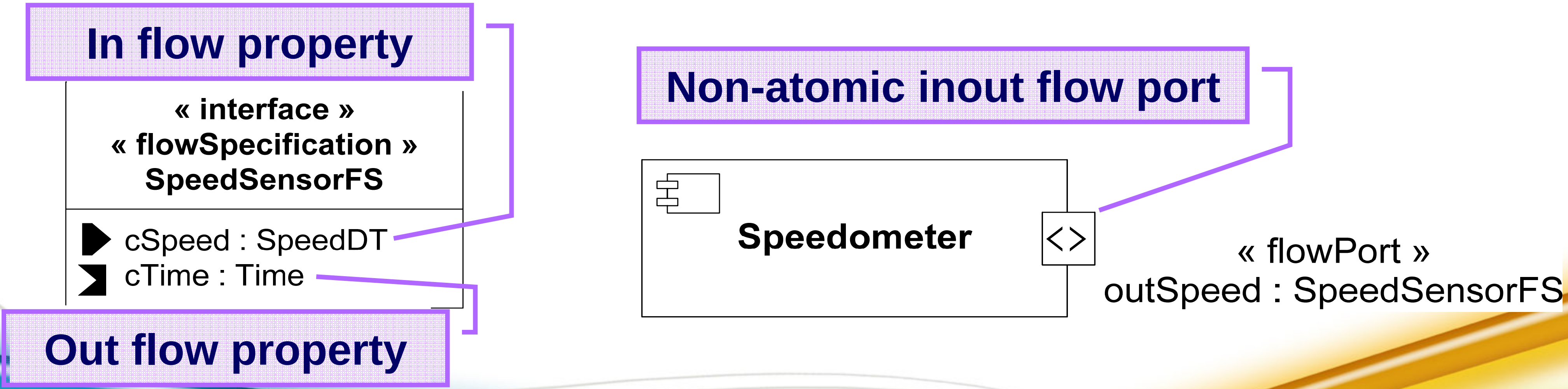


## Some examples of flow ports

- Atomic flow ports



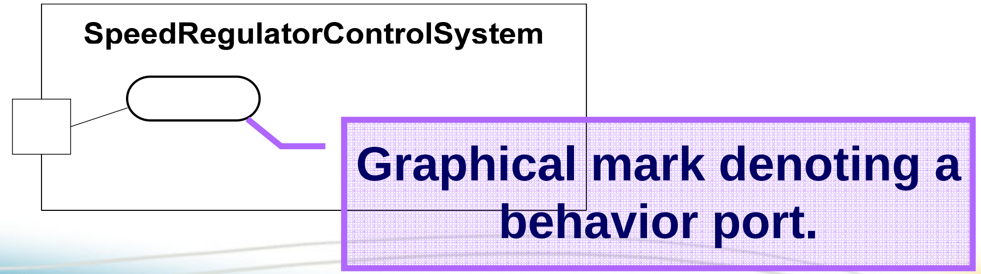
- Non-atomic flow ports ➔ typed by flow specification!



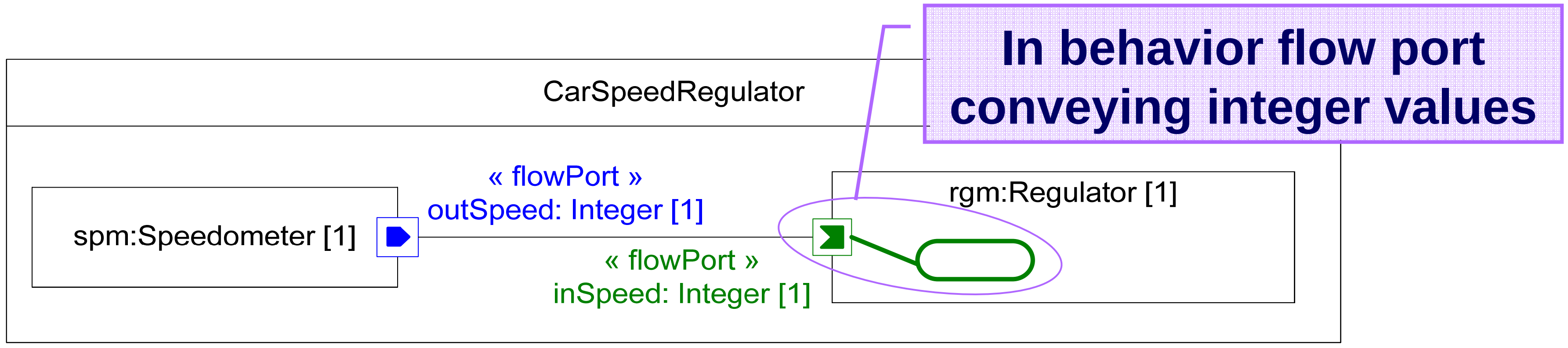
## Two reminders on UML

- **UML::BehavioredClassifier**
  - ownedBehavior: Behavior [0..*]
    - References behavior specifications owned by a classifier.
  - / classifierBehavior: Behavior [0..1]
    - A behavior specification that specifies the behavior of the classifier itself. (Subsets BehavioredClassifier::ownedBehavior)

- **UML::Port**
  - isBehavior: Boolean [0..1] = false
    - If true, requests arriving at this port are sent to the classifier behavior of this classifier. They are referred to as behavior ports.
  - Notation


SpeedRegulatorControlSystem
Graphical mark denoting a behavior port.

## Semantics view of MARTE's flow port: Push semantics



### Reception Semantics

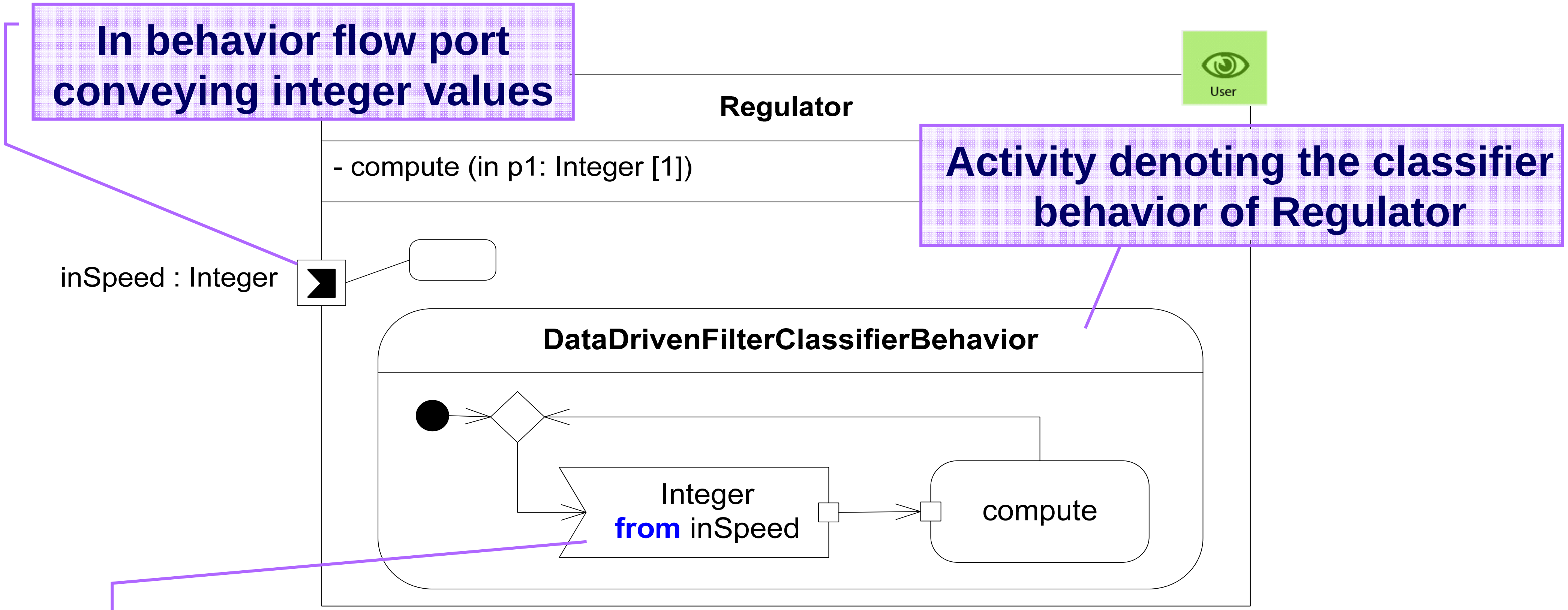
When a data is received on the port, a DataEvent is raised and stored in the event pool of the receiving instance.

### Consumption Semantics

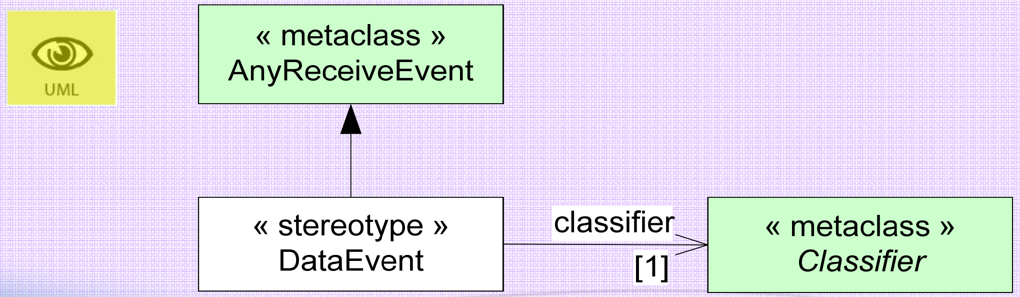
Standard UML semantics for Event apply also to MARTE's DataEvent.

MARTE modeling rule: A model owning in or inout behavior flow ports with delegation connectors is considered to be ill-formed.

## Semantics view of MARTE's flow port: Push semantics (seq.)

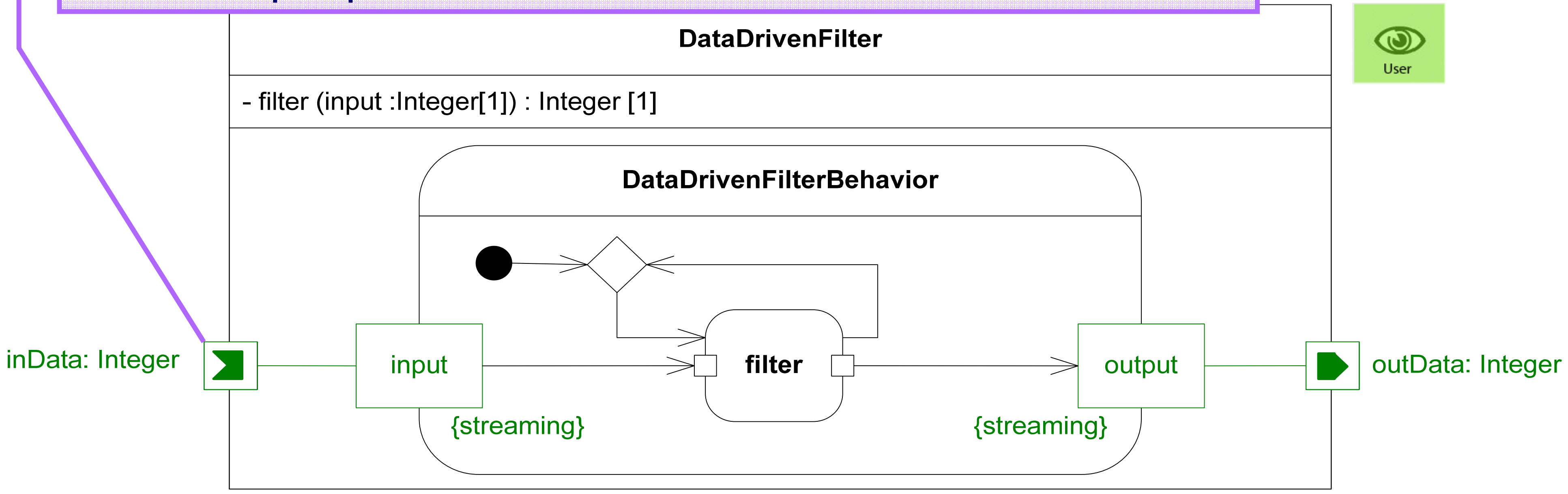

In behavior flow port conveying integer values
User
Regulator
- compute (in p1: Integer [1])
Activity denoting the classifier behavior of Regulator
inSpeed : Integer
DataDrivenFilterClassifierBehavior
Integer
from inSpeed
compute

**AcceptEventAction with a trigger based on a « DataEvent ».**



## Semantics view of MARTE's flow port: Push semantics (seq.)

In non-behavior flow ports owning a delegation connector with an input parameter of the classifier behavior.



### Reception Semantics

Data received on such port is made available as a data token on the activity parameter via the delegation connector.

### Consumption Semantics

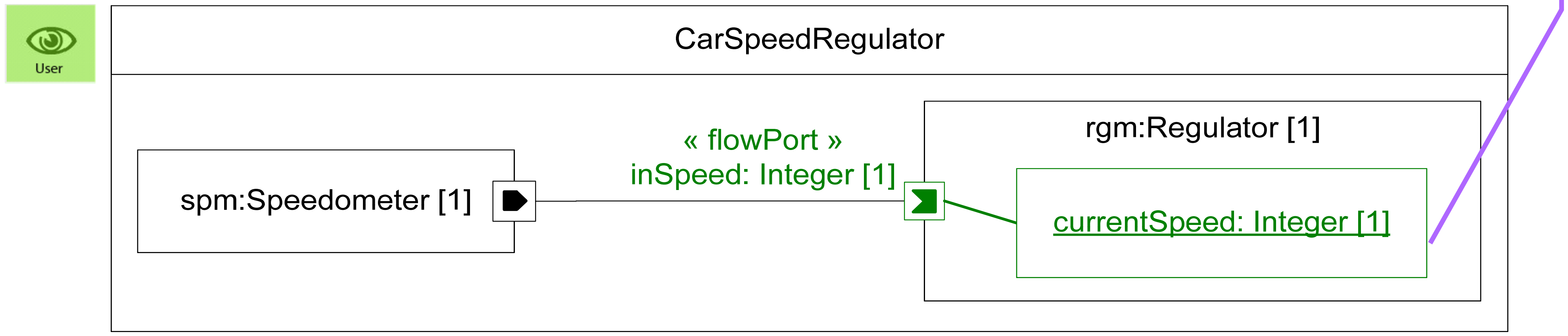
The semantics of token consumption is those of UML 2 activities.

Note: if streaming parameter ➔ activity execution can accept & produce data in a pipeline manner.

## Semantics view of MARTE’s flow port: Pull semantics

- **Pull semantics is relying on a specific modelling pattern defined in:**
  - Bock, C., "UML 2 Activity and Action Models Part 4: Object Nodes", in Journal of Object Technology, vol.3, no.1, pp.27-41.

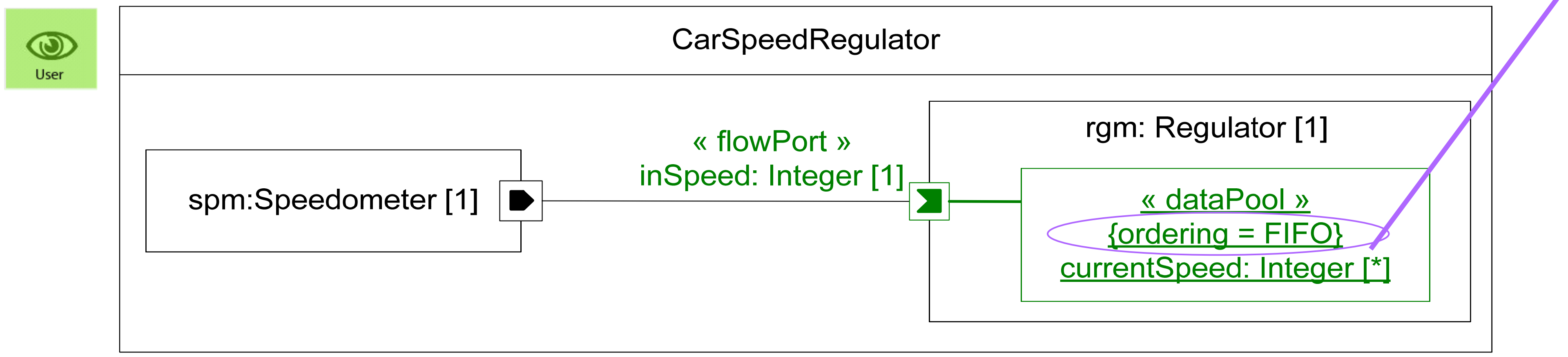
Data arriving on the in flow port via the delegation connector are stored in the property: by default, overload policy.



## Semantics view of MARTE's flow port: Pull semantics (seq.)

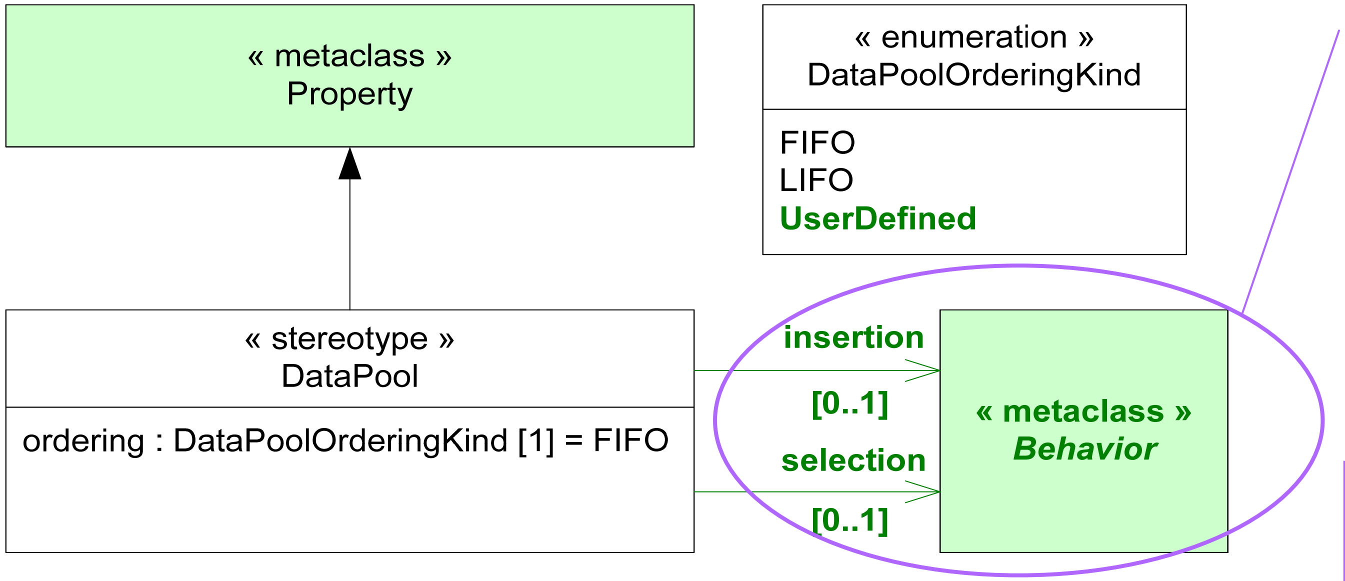
- **Possible other standard storing policies: « DataPool »**

**Two standard policies defined via the property ordering: FIFO and LIFO**



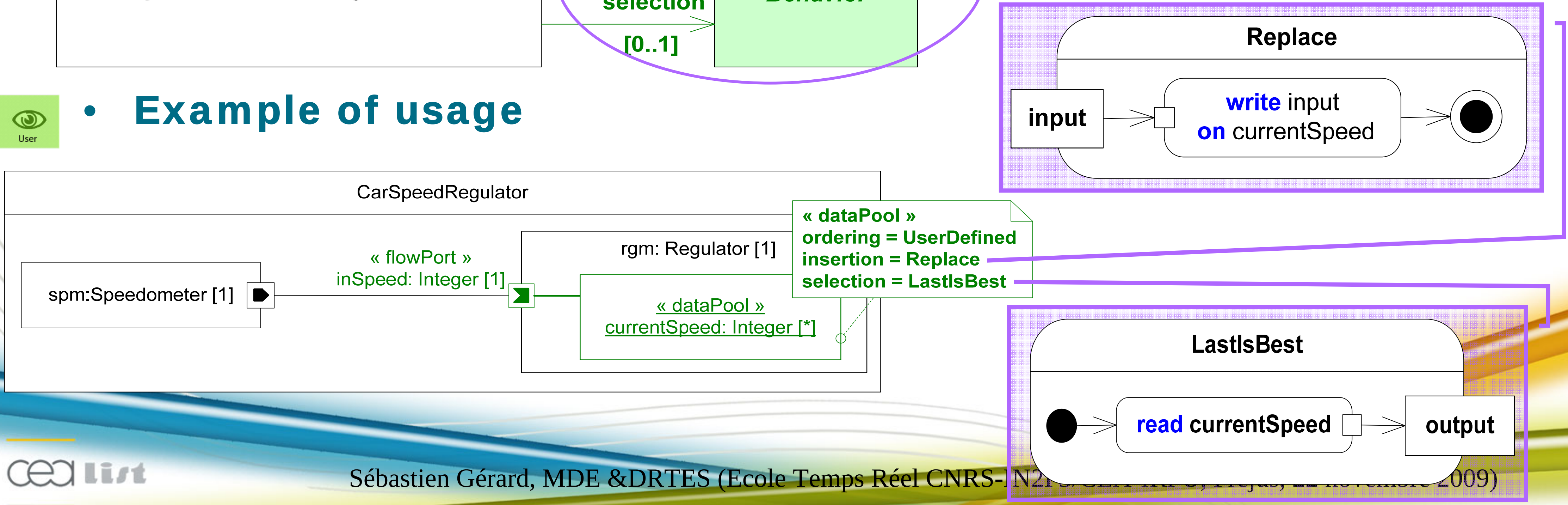
## Semantics view of MARTE's flow port: Pull semantics (seq.)

- **Details of the stereotype « DataPool »**



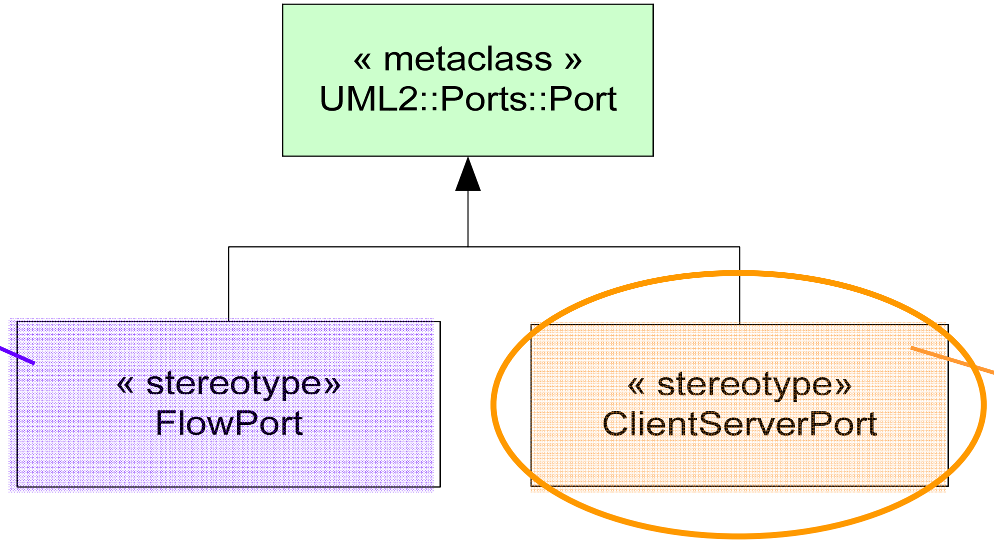
**Used to specify explicit user-defined description of how data should be inserted and selected from the pool.**

- **Example of usage**



## MARTE extensions to UML Ports

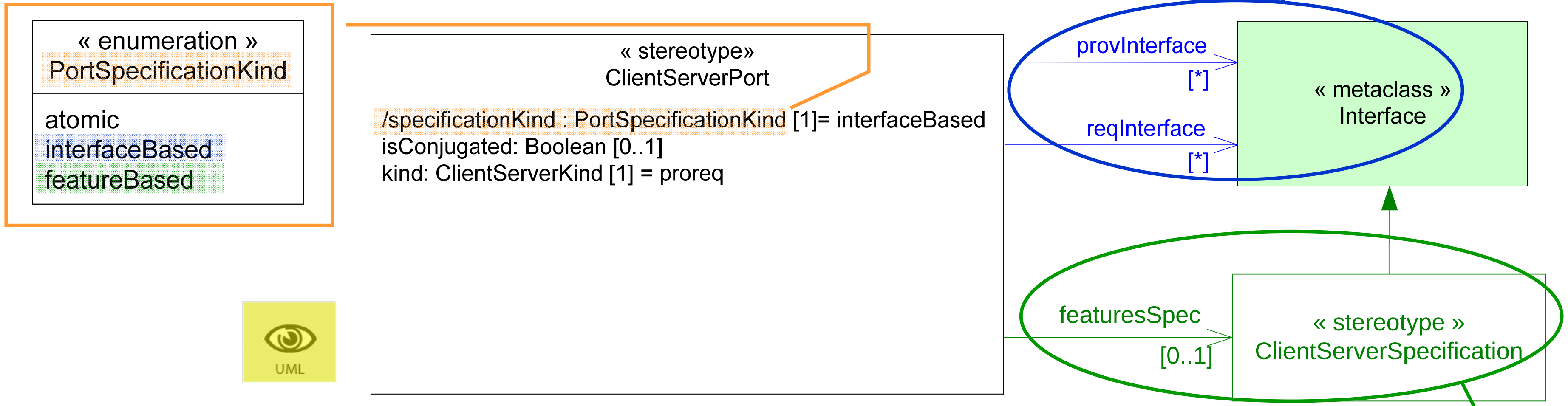
Support for data-flow communication schema between components (similar to SysML).



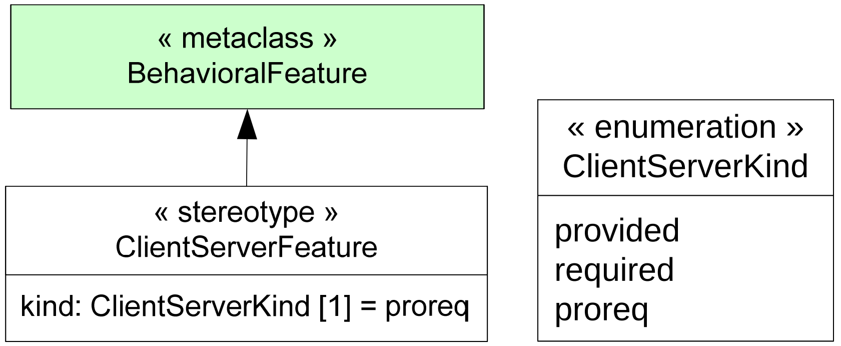
Support for “usual” OO message-based communication schema (including UML signal-based communication).

## MARTE client-server port: syntactical view

**<u>Interface-based specification:</u> each provided and required interfaces are explicitly specified.**

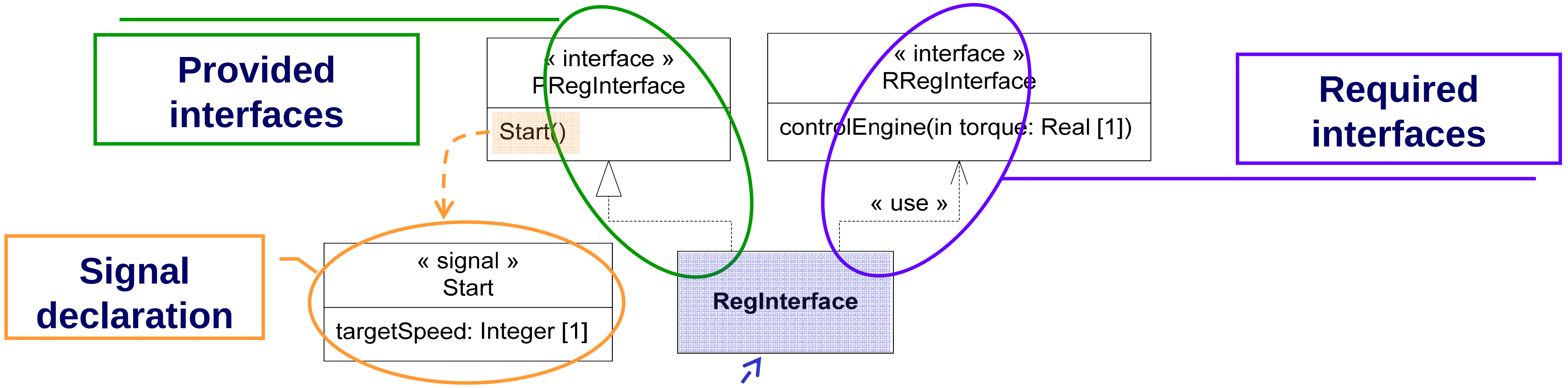


**<u>Feature-based specification:</u> the port is typed by a client-server interface.**

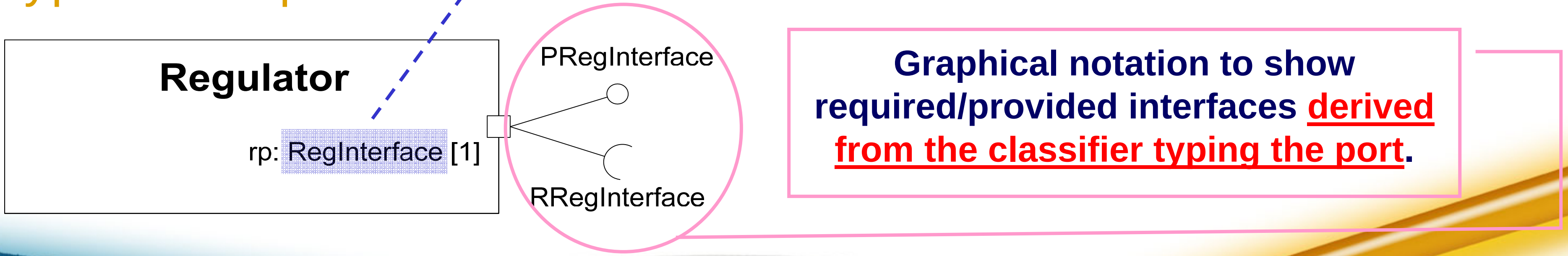


## Modeling provided and required interfaces with plain UML

- **Step 1 – Define the required interface and related properties**



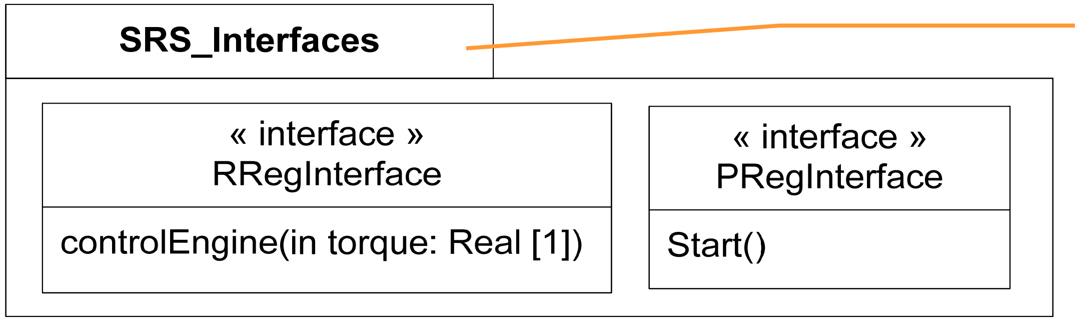
- **Step 2 – Define the component with its port and type it with aforementioned interfaces**
  - Provided and required interfaces are then calculating from the type of the port.



**Graphical notation to show required/provided interfaces derived from the classifier typing the port.**

## Modeling provided and required interfaces with MARTE's interface-based port

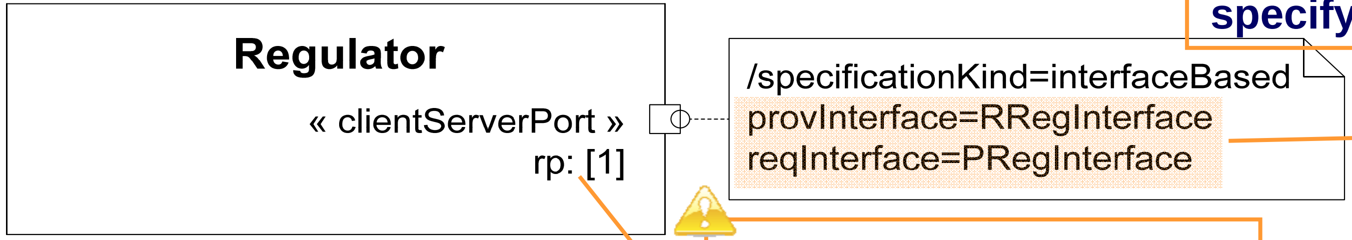
- **Step 1: Define the interface and their properties**



**Declaration of the UML "normal" interfaces of my system.**

- **Step 2: Apply « clientServerPort » and set up both properties provInterface and reqInterface.**

**Direct use of the interfaces for specifying the features of the ports.**


Regulator
« clientServerPort »
rp: [1]
/specificationKind=interfaceBased
provInterface=RRegInterface
reqInterface=PRegInterface

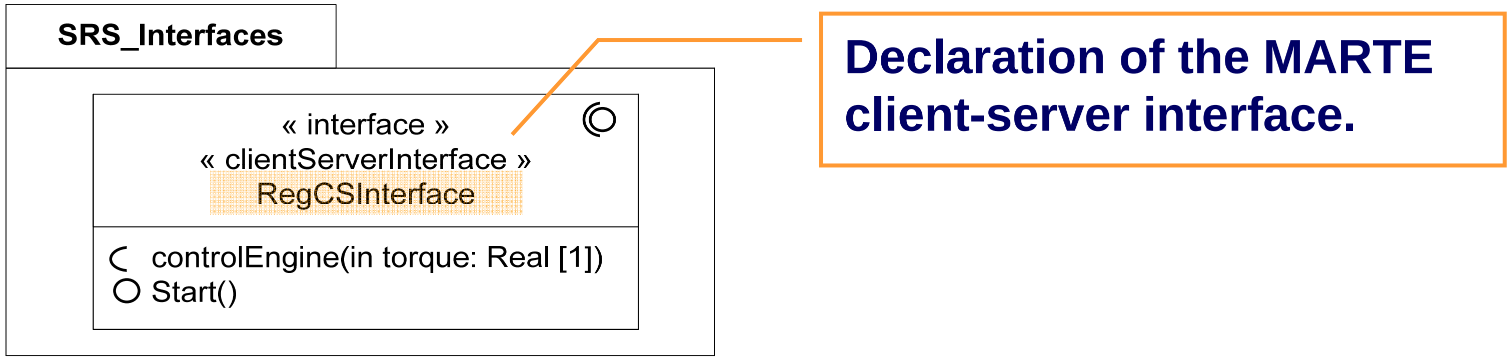
**Such ports are not typed!**

👍 Based on "normal" UML interfaces and easy to use.

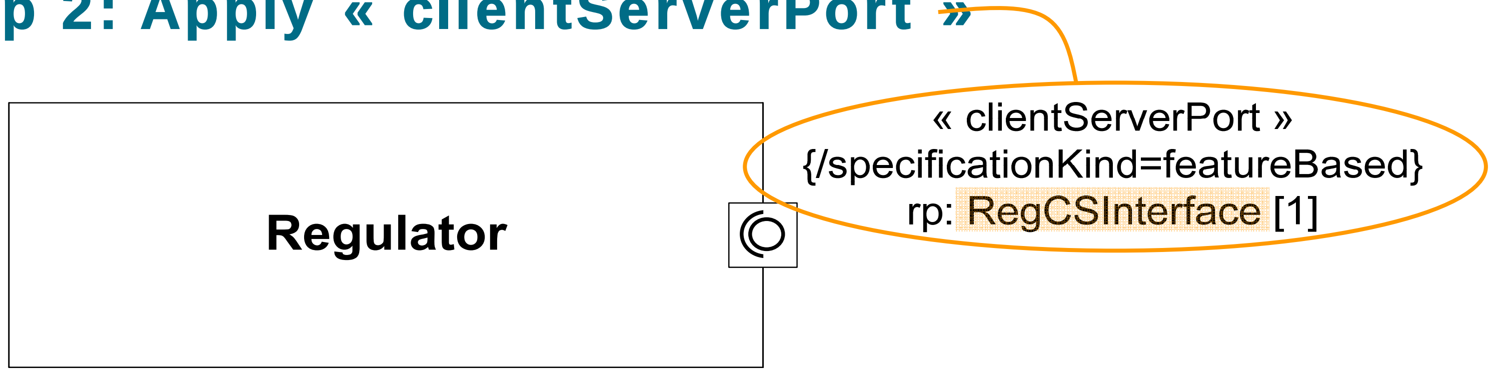
👎 Lost of type checking possiblity.

## Modeling provided and required interfaces with MARTE's feature-based port

- **Step 1: Define the client-server interfaces and their properties**


SRS_Interfaces
« interface »
« clientServerInterface »
RegCSInterface
controlEngine(in torque: Real [1])
Start()
Declaration of the MARTE client-server interface.

- **Step 2: Apply « clientServerPort »**



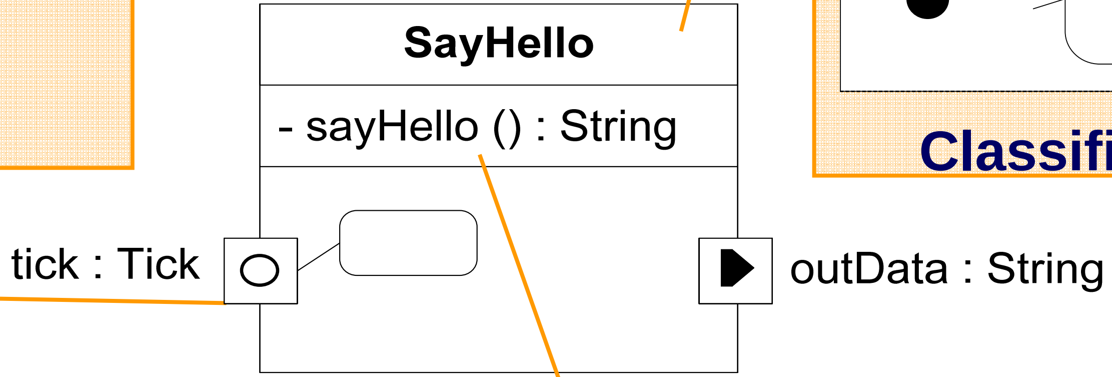
👍 Possible validation by type checking and easy to use.

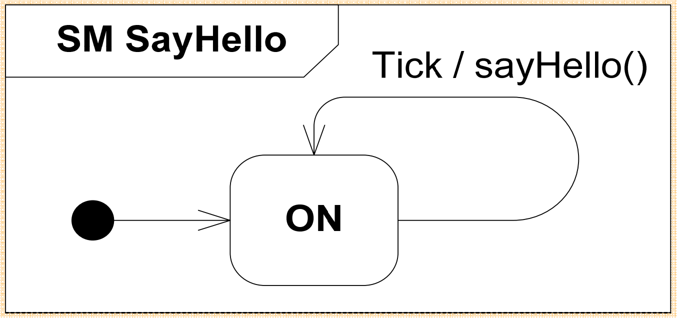
👎 Based on specific interfaces: ClientServerInterface.

## Dynamical view of the client-server port: behavior port

**Behavior client-server port enables to receive signal instance of:**

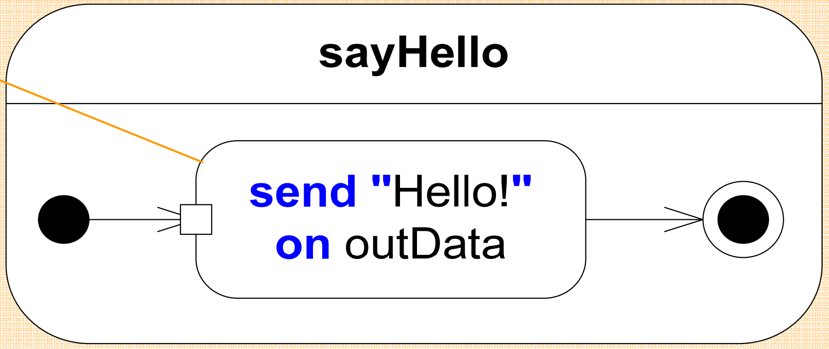
**« Signal »**
**Tick**





**Classifier behavior**

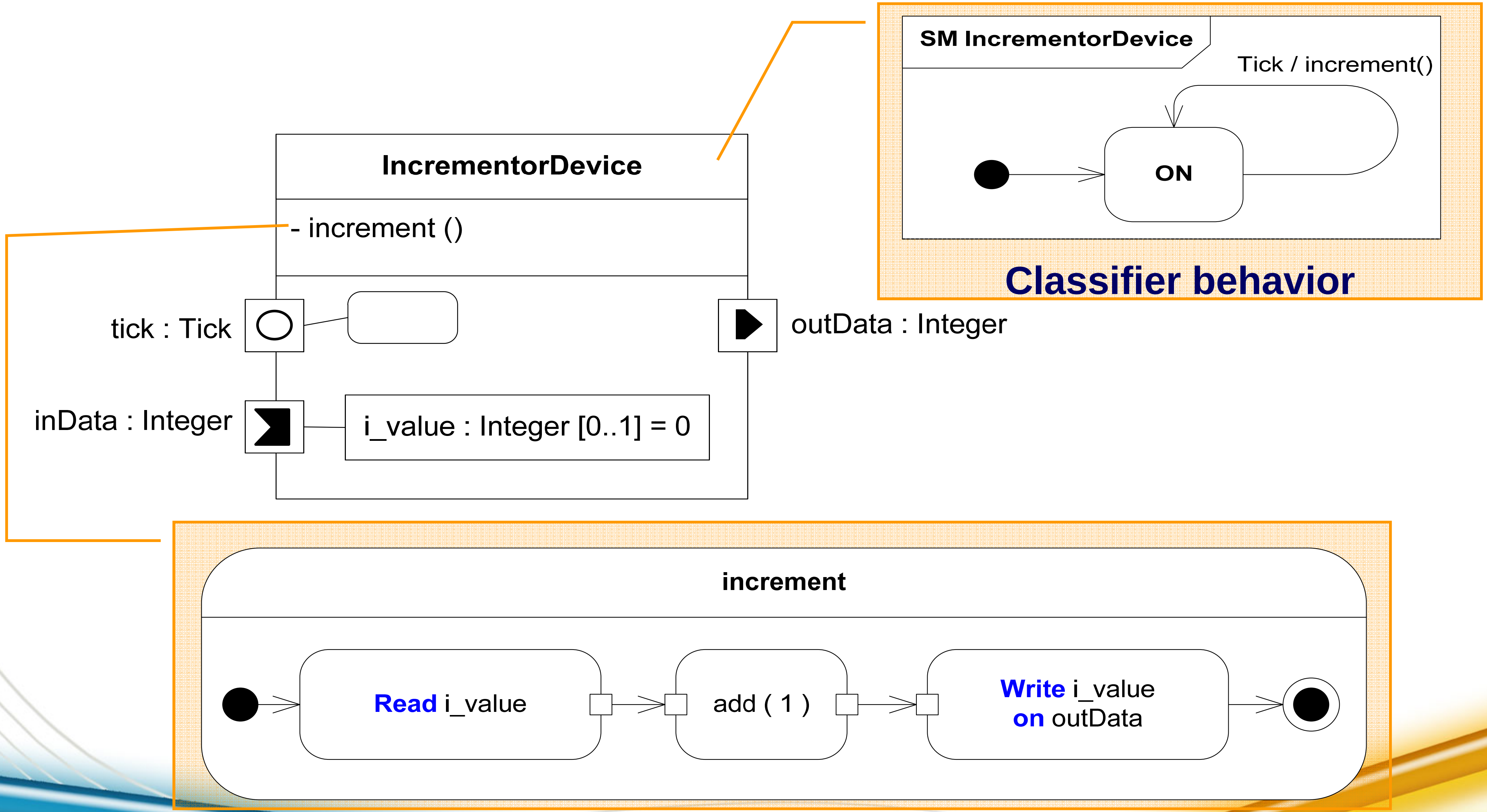
*Note: This is a SendObjectAction of UML used to send the string value "Hello!" on the out flow port outData.*



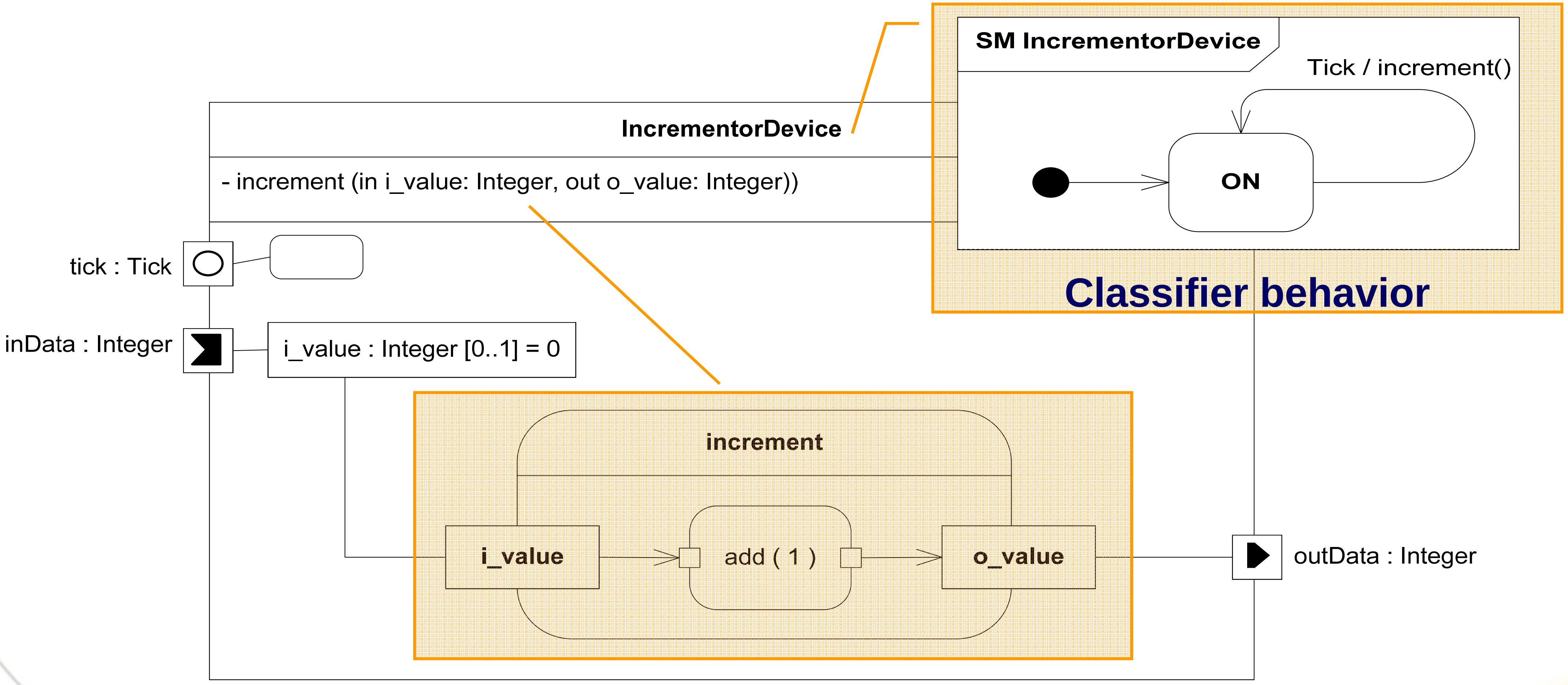
**Activity behavior implementing the operation sayHello()**

MARTE modeling rule: A model owning provided behavior client-server ports with delegation connectors is considered to be ill-formed.

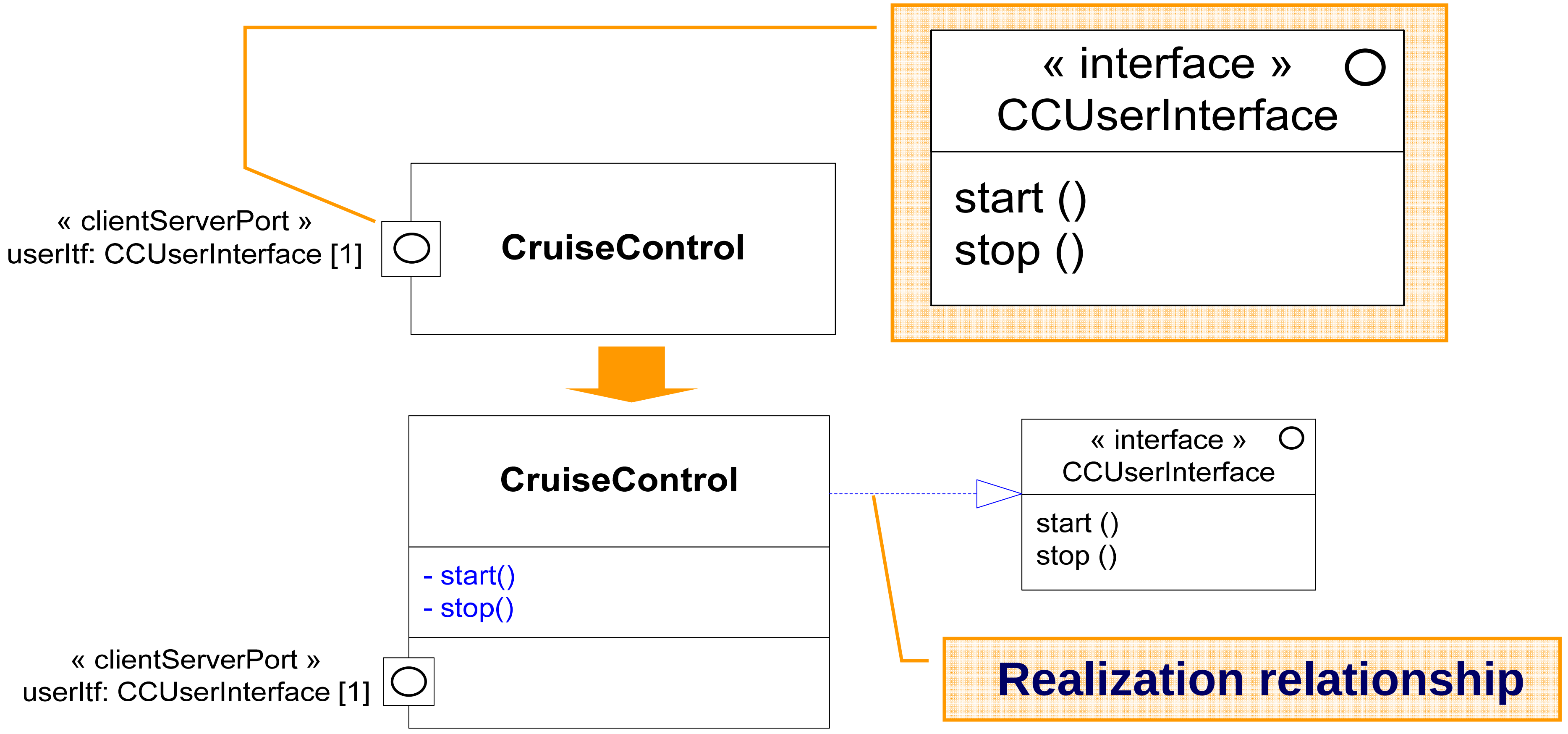
## Dynamical view of the client-server port: behavior port (seq.)



## Dynamical view of the client-server port: behavior port (seq.)

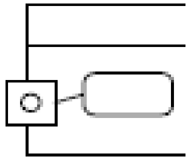
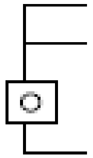
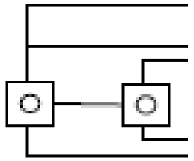


## Dynamical view of the client-server port: non-behavior port



MARTE modeling rule: A model element owning provided non-behavior client-server ports has to realize the interfaces provided by these latter. In other case, the message reveived on the port are lost.

## Summary of the dynamical view of the client-server Port

| | Delegation Connector | Reception Semantics |
|---|---|---|
| Behavioral ClientServerPort | NO  | A MessageEvent (i.e., a CallEvent or a SignalEvent) is raised and stored in the event pool of the receiving instance. |
| Behavioral ClientServerPort | YES | In MARTE, we consider such model to be ill-formed. |
| Non-Behavioral ClientServerPort | NO  | If the classifier of the receiving instance directly realizes the behavioral feature, the reception of the message directly triggers a call to this behavioral feature (and no MessageEvent is generated).<br>If the behavioral feature is not realized by the classifier of the receiving instance, the message is lost. |
| Non-Behavioral ClientServerPort | YES  | The received message follows one of the available delegation connectors, so that the message is handled by the delegation target.<br>If multiple connectors can be followed (i.e., multiple targeted elements are able to handle the message), the choice of the connector to be followed is a semantic variation point (for more details, let's see the "Conflicting End" semantic variation point, in section 12.2.2 on page 153 on the causality model of the MARTE GCM). |

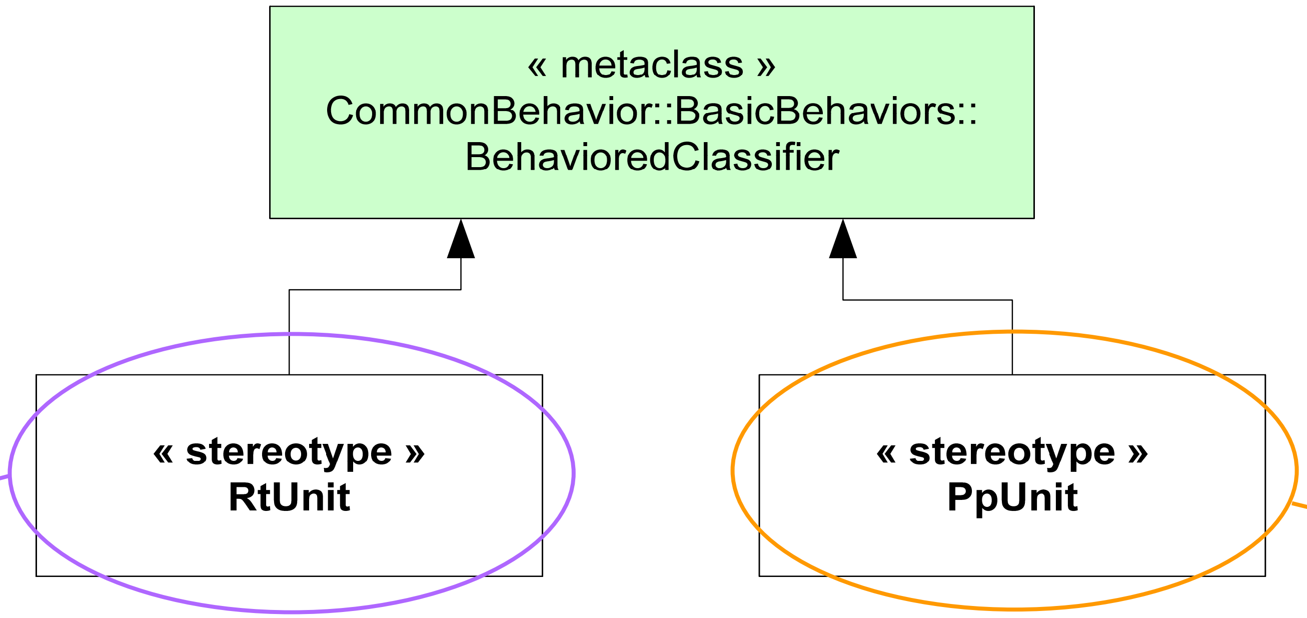
## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
- **Part II: Some details on MARTE**
  - The foundations: NFP, Time, and Allocation modeling.
  - **CBME with GCM and HLAM**
- **Part III: One typical usage example of MARTE**

## Outlines of HLAM

- **High-level modeling concepts for RT/E design**
  - Qualitative aspects
    - E.g. concurrency and behavior
  - Quantitative aspects as real-time feature
    - E.g. deadline or period
- **Allows expressing real-time constraints on component interfaces and connectors**
  - Applicable whether component are active or passive
- **For active components, introduces specific models of computation**
  - "Common" active objects MoCC
  - Alternative MoCC can be defined

## Details of the « RtUnit » and « PpUnit »



- Generalization of the UML 2 active object concept.
- Owns at least one schedulable resource.
- Resources are managed either statically (pool) or dynamically.
- May have operational mode description.

- Generalization of the Passive Objects of the UML2
- Requires schedulable resources to be executed
- Supports different concurrency policies: sequential, guarded or concurrent.

## Details of the real-time units

**Semantics of the RtUnit queue:**

« ennumeration »
MARTE_Library::GRM_BasicTypes::
**SchedPolicyKind**

EarliestDeadlineFirst
LIFO
FixedPriority
LeastLaxityFirst
RoundRobin
TimeTableDriven
Undef
Other

**« stereotype »**
**RtUnit**

isDynamic: Boolean [1] = true
srPoolSize: Integer
srPoolPolicy: PoolMgtPolicyKind
srPoolWaitingTime: NFP_Duration
queueSchedPolicy: SchedPolicyKind
queueSize: Integer
msgMaxSize: NFP_DataSize
operationalMode: Behavior

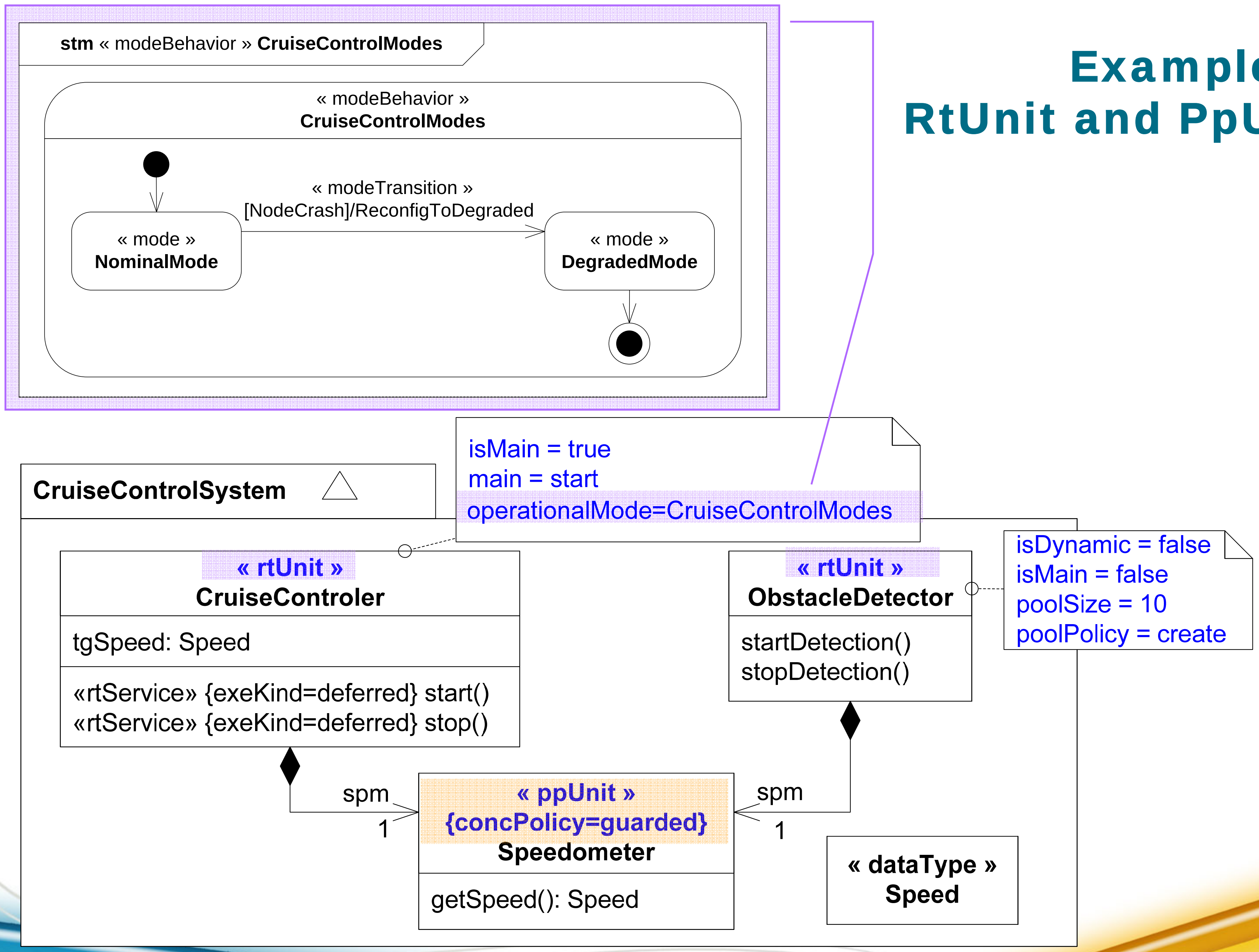
**Non-dynamic RT units owns a pool of computing resources:**

« enumeration »
**PoolMgtPolicyKind**

infiniteWait
timedWait
dynamic
exception
other

**Refers to a behavior denoting an possible operational modes and their related transitions.**

## Example of RtUnit and PpUnit



## Agenda

- **Part I: Introduction to MBE for real-time and embedded**
- **Part II: Some details on MARTE**
- **Part III: MBE in action, examples**
  - Model-based analysis: e.g., performance and scheduling analysis
  - Modeling, model transformation and code generation
    - cf. afternoon TP.

## MARTE SAM & PAM profiles

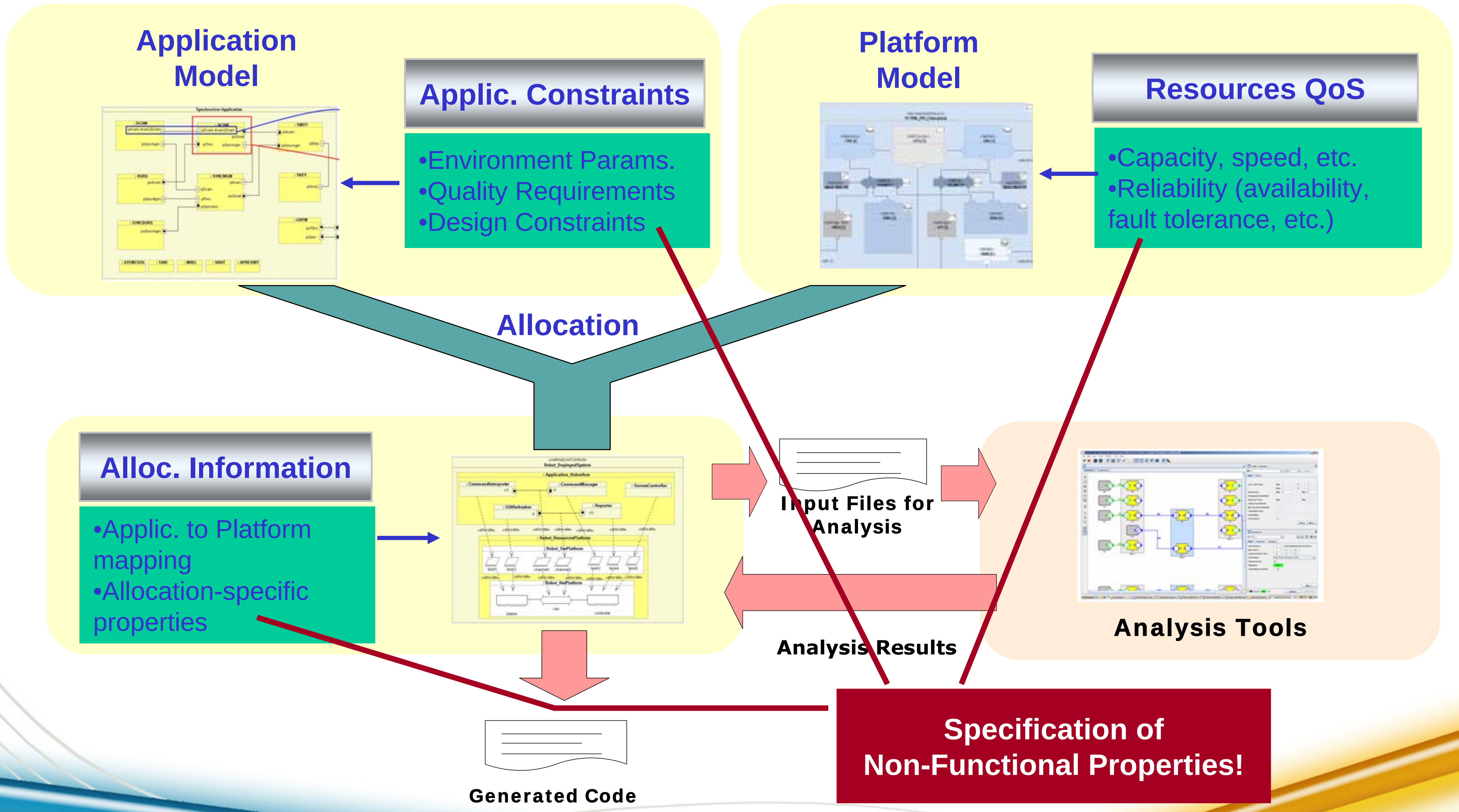
- **Schedulability Analysis Modeling (SAM):**
  ***Modeling for analysis techniques taking into account scheduling aspects***
  - Provides high-level analysis constructs
    - Sensitivity analysis, parametric analysis
    - Observers for time constraints and time predictions at analysis context level
  - Supports most common scheduling analysis techniques
    - RMA-based, holistic techniques and modular techniques

- **Performance Analysis Modeling (PAM)**
  ***Supports most common performance analysis techniques***
  - Queuing Networks and extensions, Petri Nets, simulation

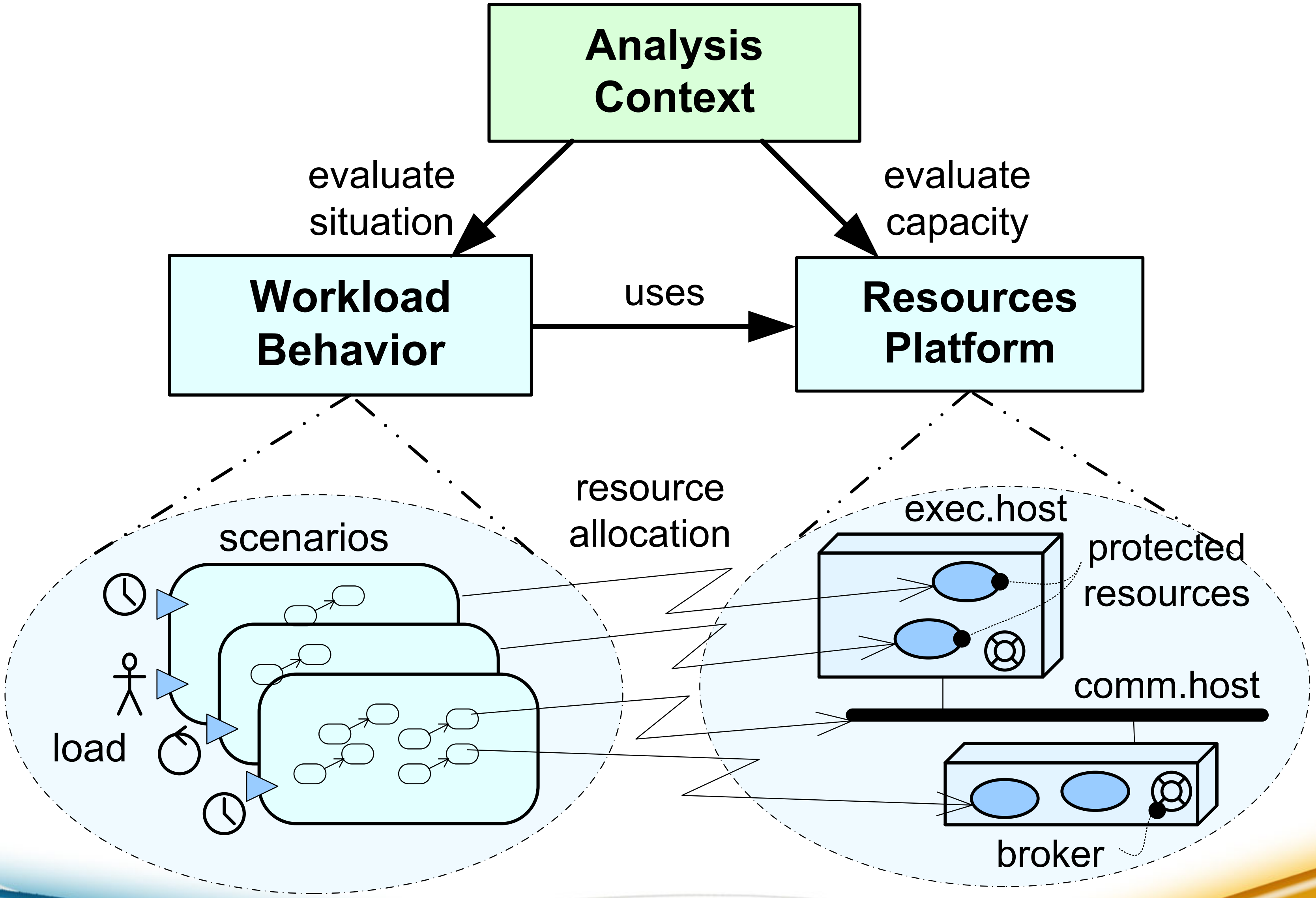
## What’s Provided by MARTE for Analysis, What’s Not

- **It provides specification means:**
  - Common patterns for analysis e.g. end-to-end flows
  - Non-functional annotations e.g. throughput, response time
- **It doesn’t preclude:**
  - Specific execution models or implementation technologies
  - Modelling styles
- **It doesn’t define:**
  - A methodological framework

# “Y-Chart” Approach for Model-Based Analysis



## Generic modeling-framework for model-based analysis



## Démo