UML MARTE

www.omgmarte.org

MARTE Tutorial

An OMG standard:
UML profile to develop Real-Time and Embedded systems

cea list THALES INRIA

Acknowledgment

- **This presentation reuses and extends material prepared by the ProMARTE partners for the OMG RTESS PTF meeting in San Diego, on March 28th 2007**

- **This tutorial has been designed in the context of CORTESS project within the CARROLL research program**
  - http://www.carroll-research.org/

- **Following persons have contributed to this tutorial**
  - CEA LIST (contact: sebastien.gerard@cea.fr)
    - Huascar Espinoza, Sébastien Gérard, Safouan Taha and Frédéric Thomas.
  - INRIA (contact: Robert.De_simone@sophia.inria.fr)
    - Charles André, Robert de Simone, Pierre Boulet
  - Thales TRT (contact: Laurent.rioux@thalesgroup.com)
    - Madeleine Faugère, Laurent Rioux, Sebastien Demathieu

How to read this tutorial

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

  - For UML Profile View Level

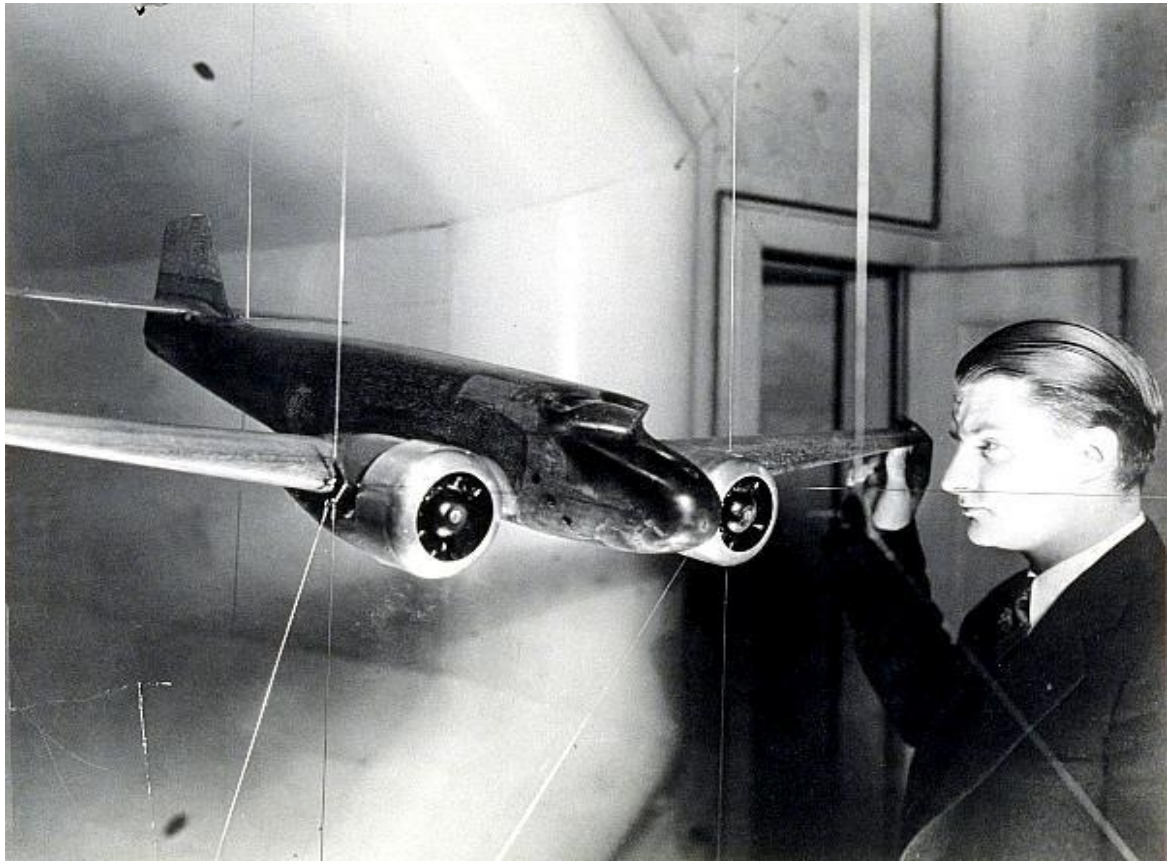
  - For User Model View Level

Agenda

- **Part 1**
  - **Introduction to MDD for RT/E systems & MARTE in a nutshell**
- **Part 2**
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- **Part 3**
  - The timing model
- **Part 4**
  - A component model for RT/E
- **Part 5**
  - Platform modeling
- **Part 6**
  - Repetitive structure modeling
- **Part 7**
  - Model-based analysis for RT/E
- **Part 8**
  - MARTE and AADL
- **Part 9**
  - Conclusions

Models in Traditional Engineering

- Probably as old as engineering



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

What is a Model in MDD

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

- Phil Bernstein, "A Vision for Management of Complex Systems".
  A model is a complex structure that represents a design artifact such as a relational schema, an interface definition (API), an XML schema, a semantic network, a UML model or a hypermedia document.
- OMG, "UML Superstructure".
  A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. This purpose determines what is included in the model and what is relevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at the appropriate level of detail.
- OMG, "MDA Guide".
  A formal specification of the function, structure and/or behavior of an application or system.
- Steve Mellor, et al., "UML Distilled"
  A model is a simplification of something so we can view, manipulate, and reason about it, and so help us understand the complexity inherent in the subject under study.
- Anneke Kleppe, et. al. "MDA Explained"
  A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.
- Chris Raistrick et al., "Model Driven Architecture with Executable UML"
  A formal representation of the function, behavior, and structure of the system we are considering, expressed in an unambiguous language.
- J. Bézivin & O. Gerbé, "Towards a Precise Definition of the OMG/MDA Framework"
  A simplification of a system built with an intended goal in mind; The model should be able to answer questions in place of the actual system.

One definition

- A reduced/abstract representation of some system that highlights the properties of interest from a given point of view.
- The point of view defines concern and scope of the model.

The Model

- Map is based on a legend (explicit or implicit)
  - Here the map of bicycle roads of Seattle
- As a map, the legend is defined in a graphical language, it means also the legend is declared with a similar formalism.
- If the Map is a Model, the legend is the meta-model defining the subset of graphical language used to build the model
- The Legend is necessary to interpret the map.
- If the legend is not shown, this mean we refer to a standard legend and implicit.

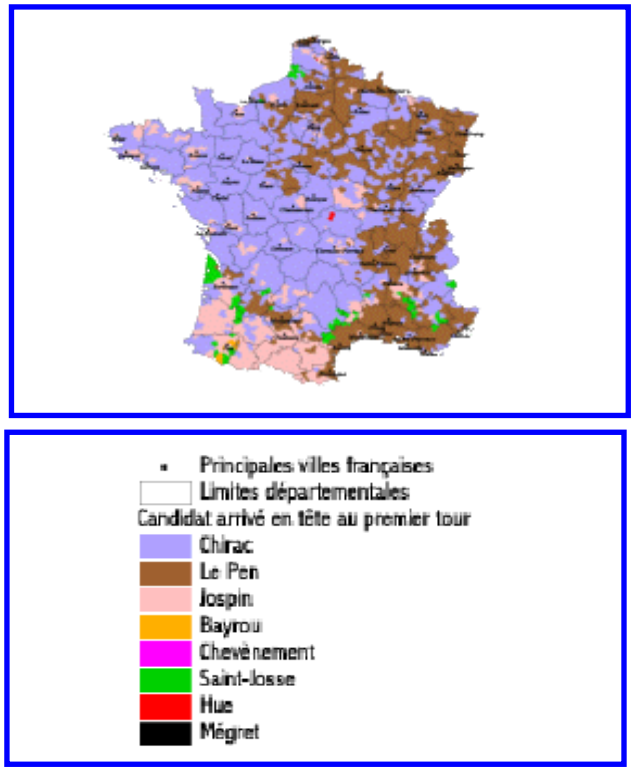
The Legend

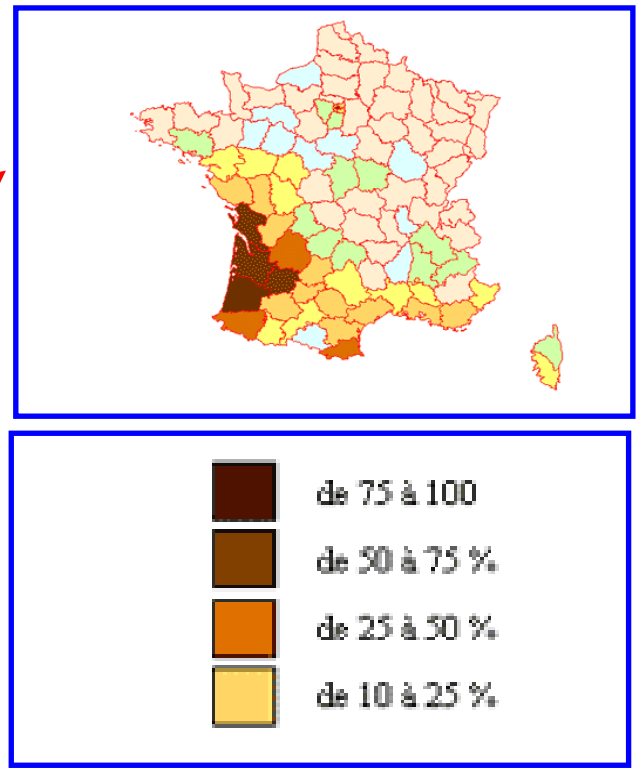
From J. Bezivin / INRIA

A Model without its meta-model has no meaning

Candidates at the Presidential election In France in 2002

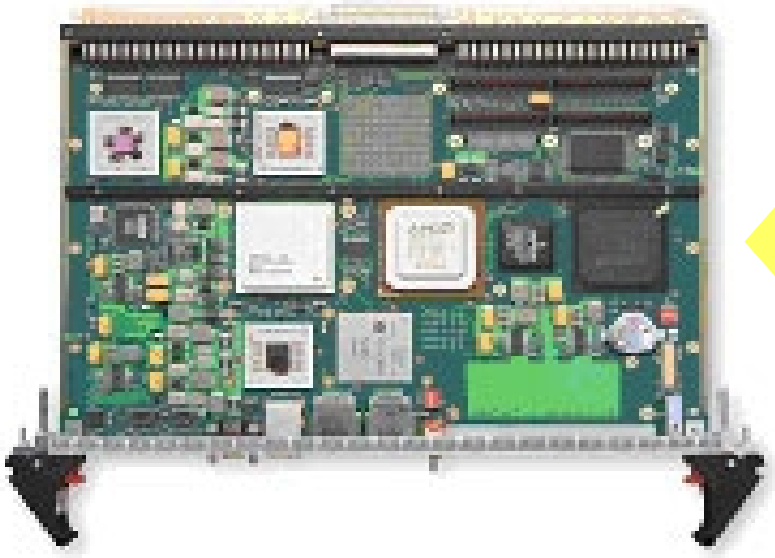
Percentage Of town infested of termites



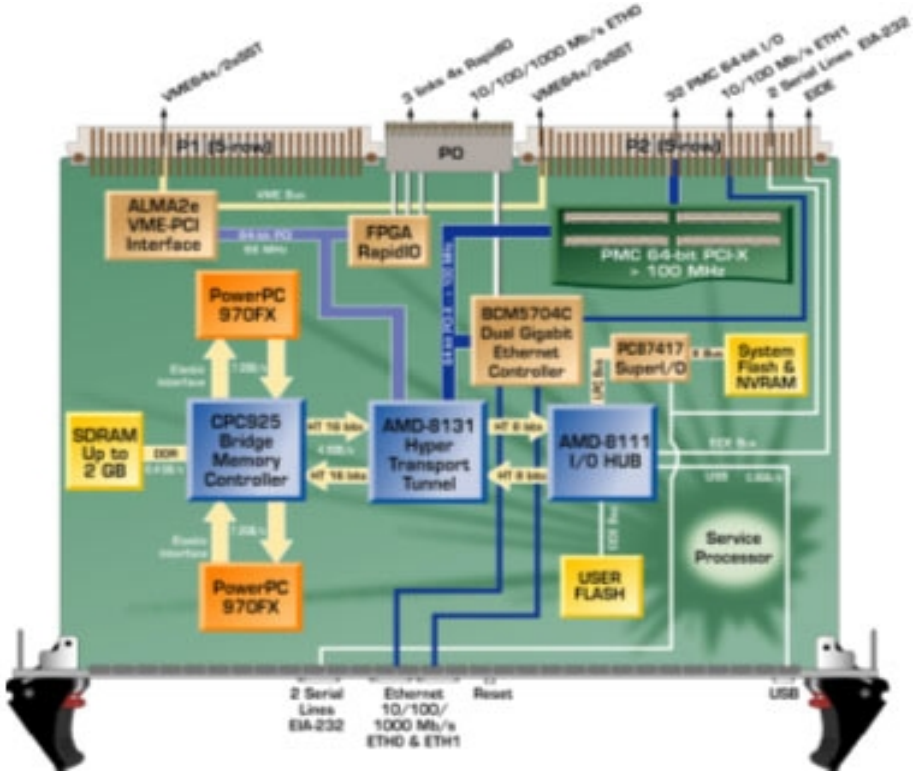


From J. Bezivin / INRIA

The Model help to understand the system



System to model



Functional model

- For Functional viewpoint and its design

Why Model Driven Engineering is Needed?

- **To deal with complexity of systems development**
  - Abstract a problem to focus on some particular points of interest
    ➔ improve understandability of a problem
  - Possible set of nearly independent views of a model
    - Separation of concerns (e.g. “Aspect Oriented Modeling”)
  - Iterative modeling may be expressed at different level of abstraction

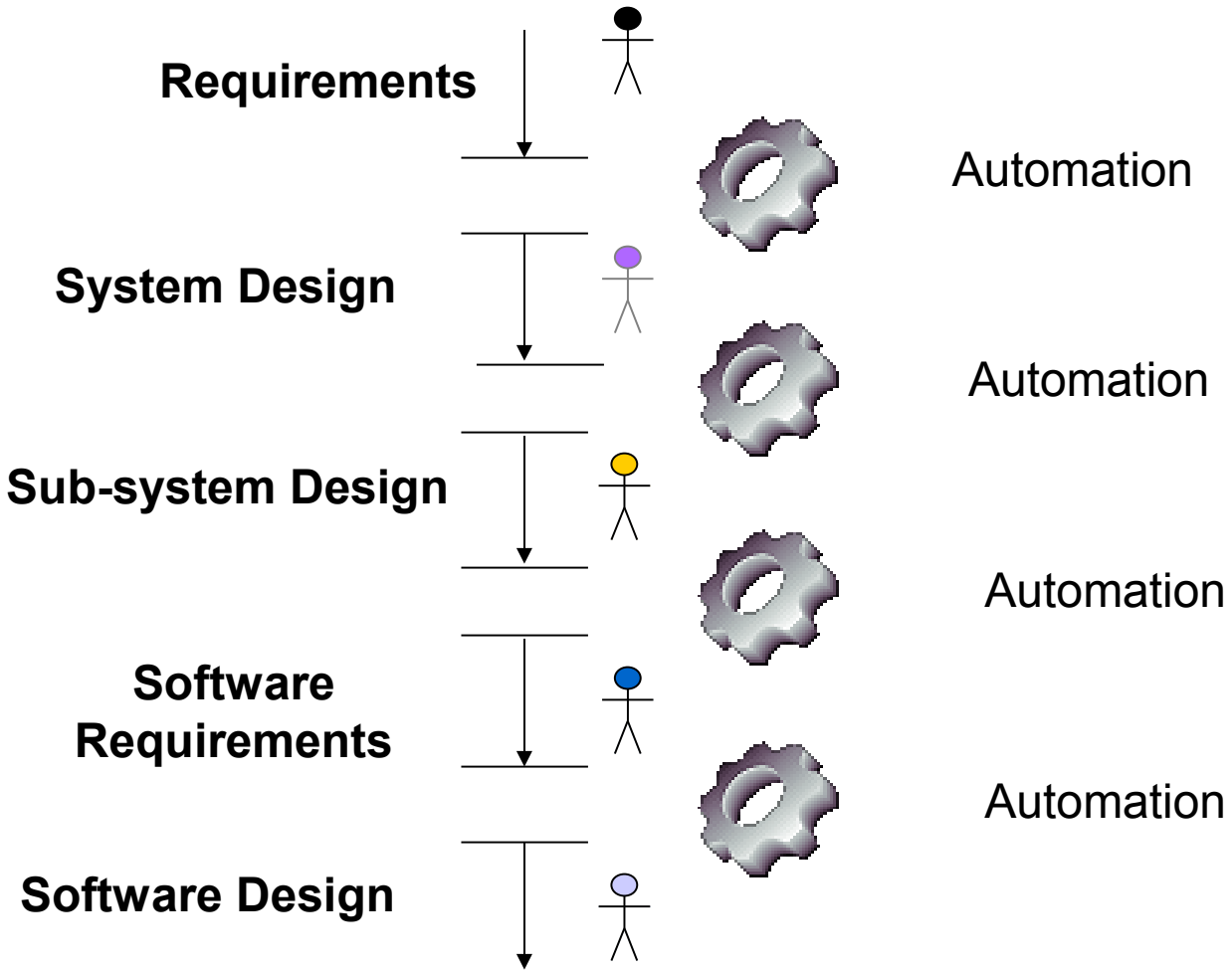
- **To minimize development risks**
  - Through analysis and experimentation performed early in the design cycle
  - Enable to investigate and compare alternative solutions

- **To improve communication ...**
- **... to foster information sharing and reuse!**
  - A good model is better than a long speech !

- **To reduce development flaws**
  - Automatic model transformation is less error-prone than building a specific compiler

Extracted from S.Gerard (ECRTS07)

Why: Provide Continuum in development process



Characteristics of Useful Models

- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Inexpensive**
  - Much cheaper to construct and study than the actual system

To be useful, engineering models must satisfy all of these characteristics!

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

A Bit of Modern Software…

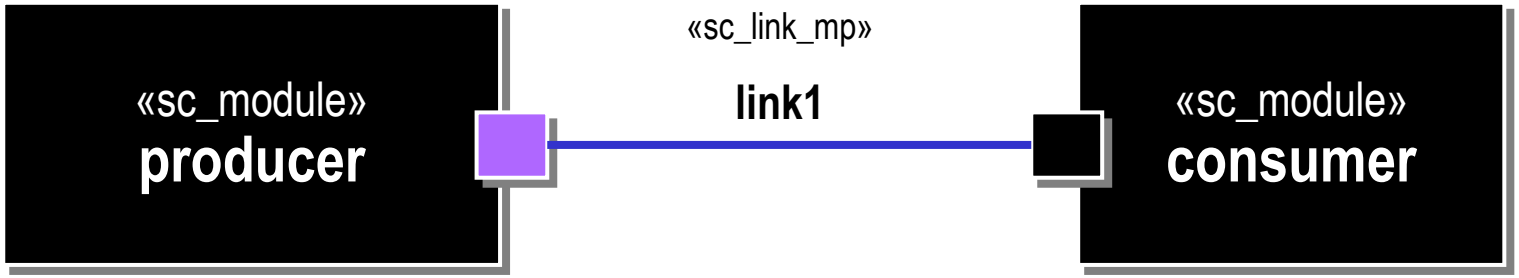
```
SC_MODULE(producer) {
  sc_outmaster<int> out1;
  sc_in<bool> start; // kick-start
  void generate_data () {
    for(int i =0; i <10; i++) {
      out1 =i ; //to invoke slave;
    }
  }
  SC_CTOR(producer) {
    SC_METHOD(generate_data);
    sensitive << start;
  }
};
SC_MODULE(top) { // container
  producer *A1;
  consumer *B1;
  sc_link_mp<int> link1;
  SC_CTOR(top) {
    A1 = new producer("A1");
    A1.out1(link1);
    B1 = new consumer("B1");
    B1.in1(link1);
  }
};
```

```
SC_MODULE(consumer) {
  sc_inslave<int> in1;
  int sum; // state variable
  void accumulate (){
    sum += in1;
    cout << "Sum = " << sum << endl;
  }
  SC_CTOR(consumer) {
    SC_SLAVE(accumulate, in1);
    sum = 0; // initialize
  }
};
```

Can you spot the architecture?

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

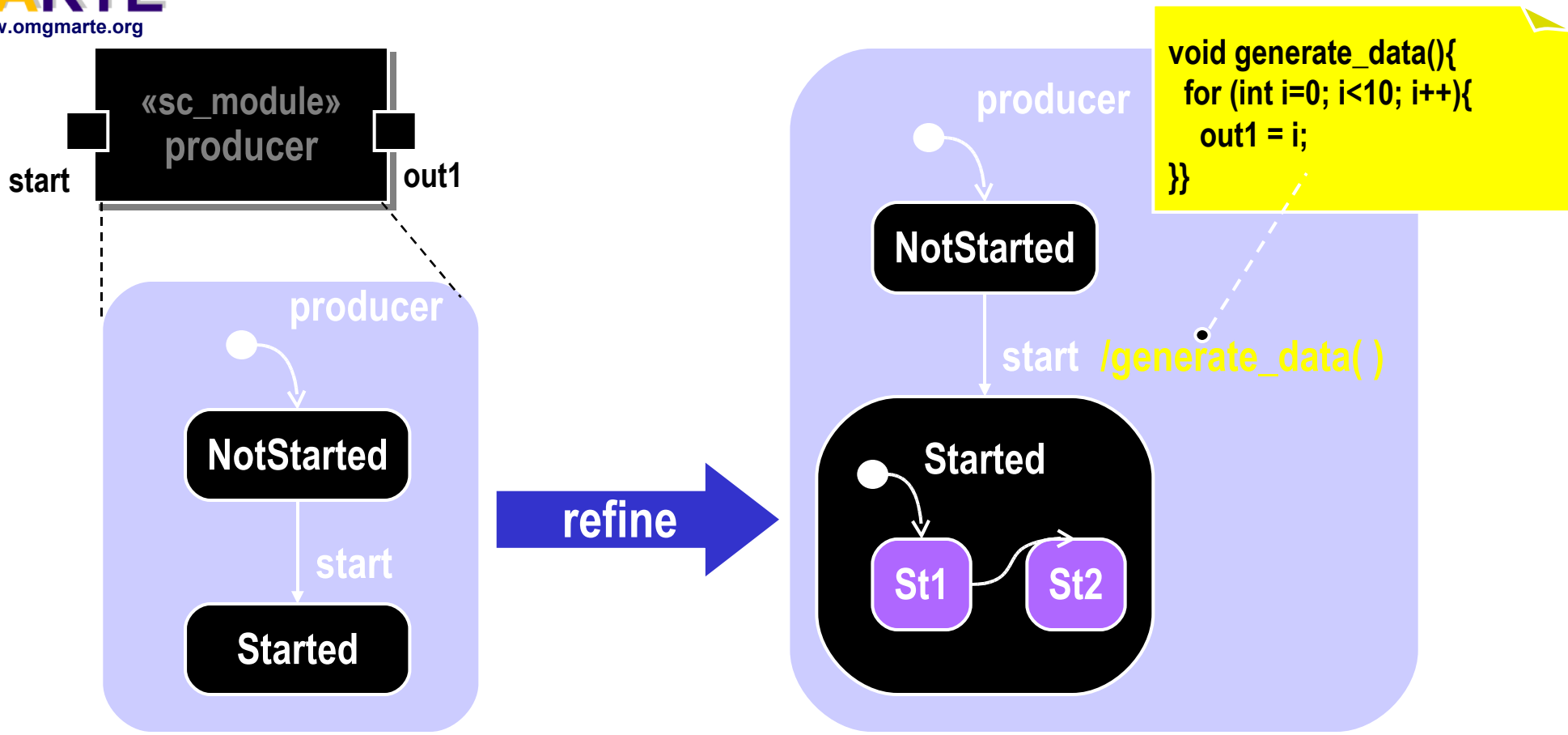
…and its UML Model



Can you spot the architecture?

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

Model Evolution: Refinement



- **Models can be refined continuously until the specification is complete**

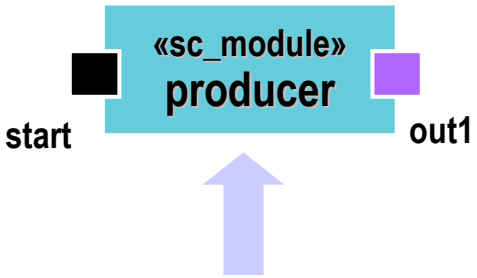
(Extracted from B. Selic presentation during Summer School MDD For DRES 2004 (Brest, September 2004)

Model-Driven Style of Development (MDD)

- An approach to develop systems and softwares in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on two time-proven methods

(1) ABSTRACTION

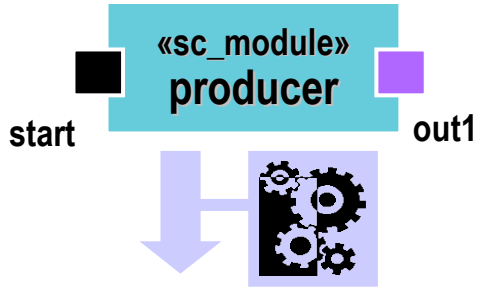
Realm of modeling languages




```
SC_MODULE(producer){
  sc_inslave<int> in1;
  int sum; //
  void accumulate () {
    sum += in1;
    cout <<   Sum =
             << sum <<
endl;}
```

(2) AUTOMATION

Realm of tools




```
SC_MODULE(producer) {
  sc_inslave<int> in1;
  int sum; //
  void accumulate (){
    sum += in1;
    cout <<   Sum =
             << sum <<
endl;}
```

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

Profiling 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 UML metamodel

Extracted from S.Gerard (ECRTS07)

UML2 Extension Mechanisms

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

- **Stereotypes**
  - Define how a specific metaclass may be extended
  - Provide additional semantics information, but only for:
    - Semantics 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 define specific notation
      - **E.g., new icons or shapes**
  - May have values that are usually referred to as tagged values

Extracted from S.Gerard (ECRTS07)

Profile Notation

- **Profile is a stereotyped package**

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

  - 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»**

Extracted from S.Gerard (ECRTS07)

The Profile Concept (cont.)

- **A profile package may import external packages**
  - "Normal" packages
    - e.g. external pkgs defining specific types for a profile

  - "Profile" packages

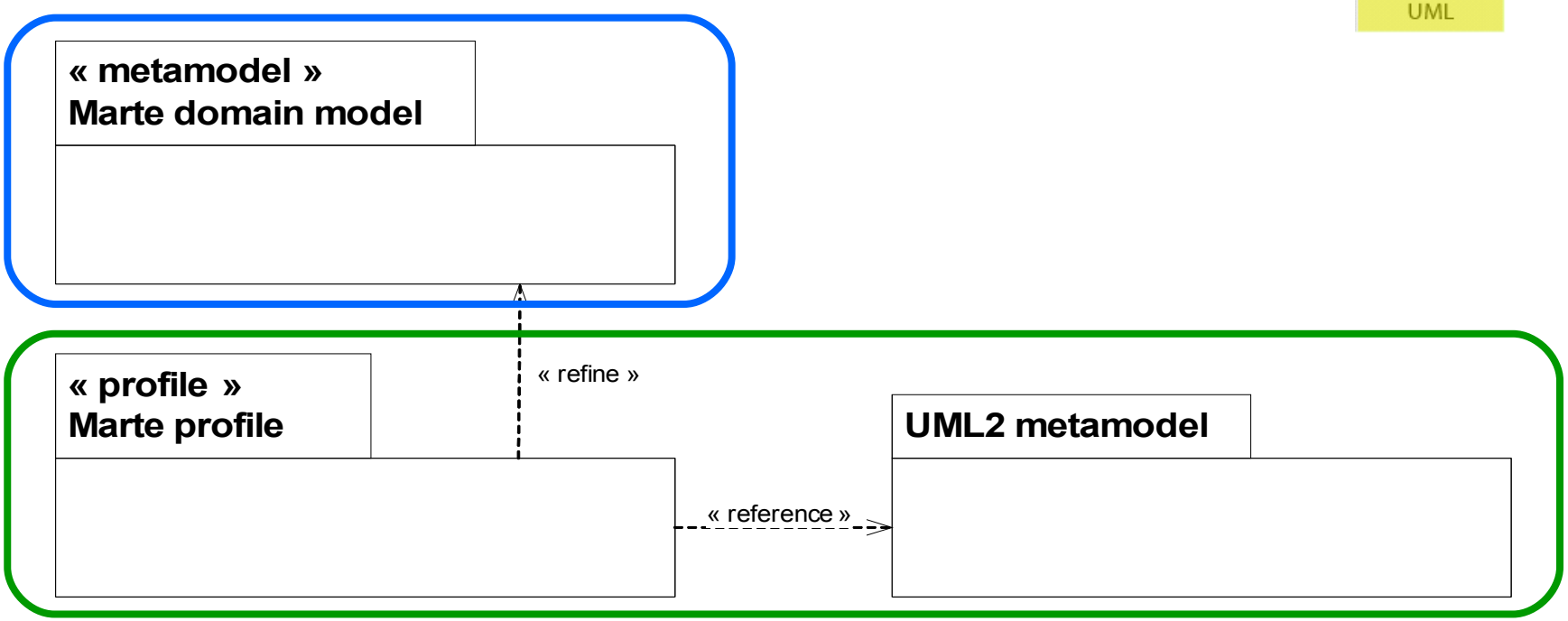
  - All imported elements may be used in pkgs applying the profile

Extracted from S.Gerard (ECRTS07)

Design Pattern Adopted for the MARTE Profile

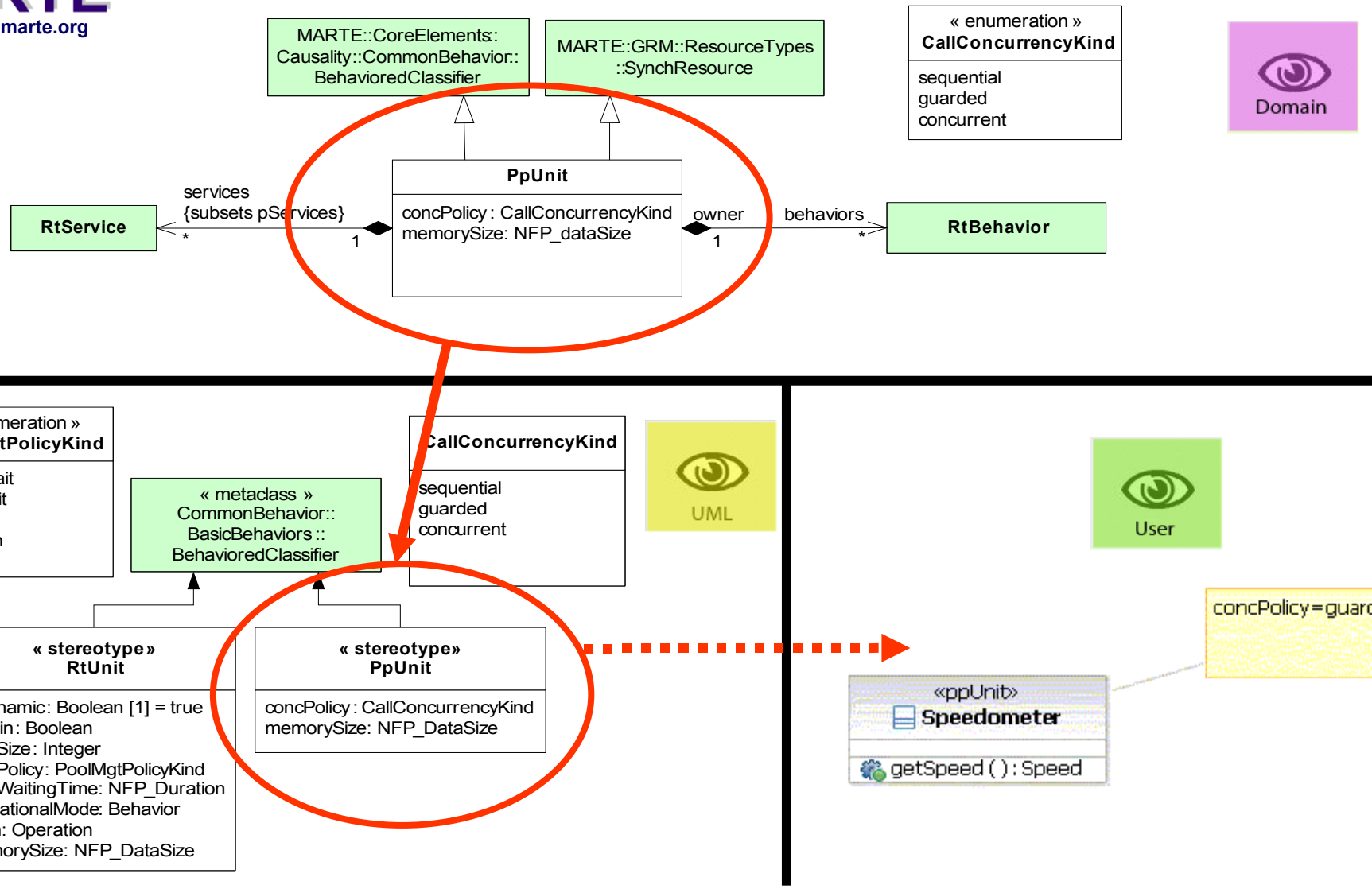
- **Stage 1 → Description of MARTE** domain models (Domain View)
  - Purpose: Formal description of the concepts required for MARTE
  - Techniques: Meta-modeling

- **Stage 2 → Mapping of MARTE domain models towards UML2:** (UML Representation)
  - Purpose: MARTE domain models design as a UML2 extensions
  - Techniques: UML2 profile



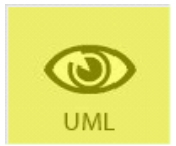
Extracted from S.Gerard (ECRTS07)

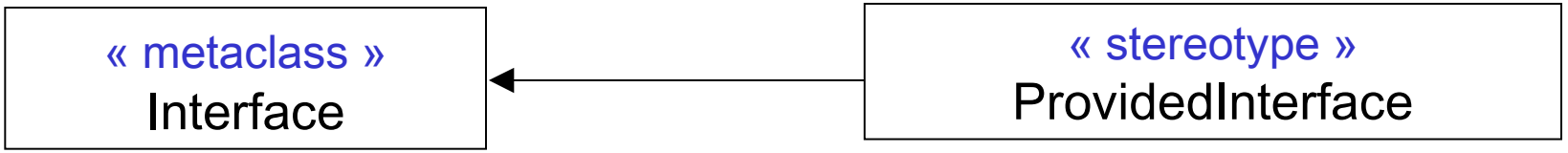
Example: Domain model → Profile → Usage



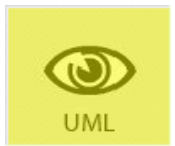
Notation for Stereotype Definition (Uml Representation)

- **Stereotype definition**

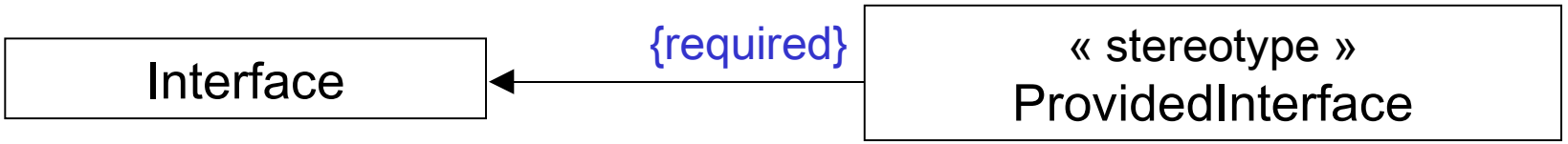




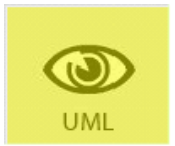
- **Required stereotype**

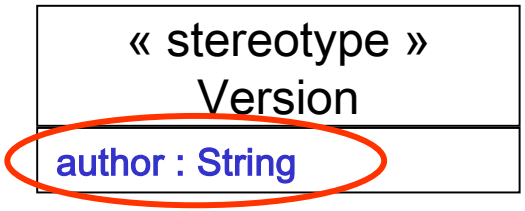


  - Extended meta-class may only be instantiated under its stereotyped form



- **Stereotype properties**





Extracted from S.Gerard (ECRTS07)

Notation for Stereotype Usage (user model-level)

- **Applying a stereotype**

« providedInterface »
MyInterface

- **Applying several stereotypes**

« providedInterface, version »
MyInterface

or

« providedInterface», «version »
MyInterface

- **Specifying values of a stereotype**

«version»
MyInterface

author = “myname”

- Use name of stereotypes when possible confusion

«version, status»
MyClass

«version»
author = “myname”
«status»
value = tested

Extracted from S.Gerard (ECRTS07)

UML Profiles for RTES

Extracted from S.Gérard (ECRTS07)

Tool Support

Time Analysis Support

- **SPT was the first OMG's UML profile for Real-Time Systems:**
  - Support for **S**chedulability Analysis with RMA-type techniques
  - Support for **P**erformance Analysis with Queuing Theory and Petri Nets
  - A rich model for "metric" **T**ime 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

- **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

Public website:

www.omgmarte.org

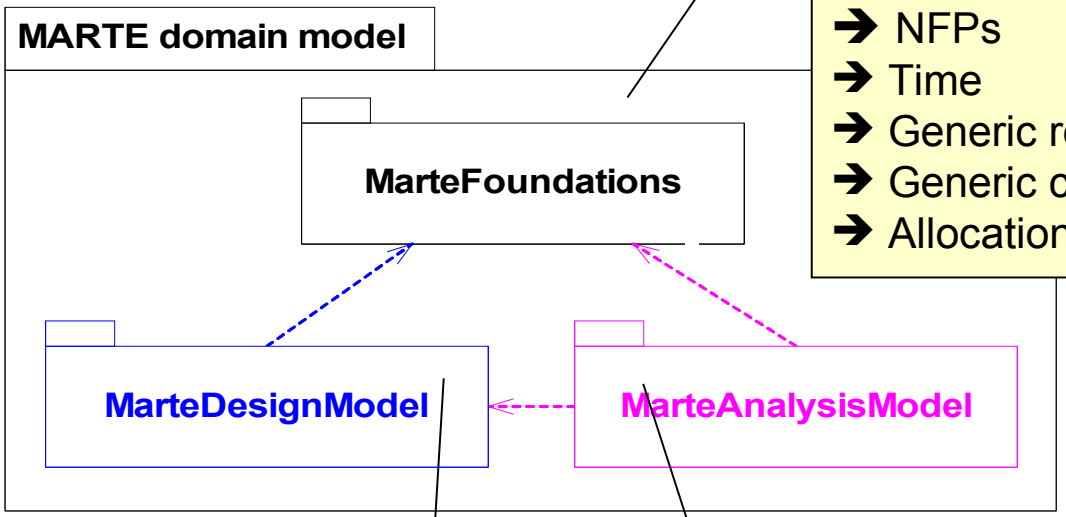
* Submitter to OMG UML Profile for MARTE RFP

Relationships with other OMG Standards

- **Relationships with generic OMG standards**
  - Profile the UML2 superstructure meta-model
  - Replace UML Profile for SPT (Scheduling, Performance and Time)
  - Use OCL2 (Object Constraints Language)

- **Relationships with RT&E specific OMG standards**
  - Existing standards
    - The UML profile for Modeling QoS and FT Characteristics and Mechanisms
      - **Addressed through MARTE NFP package (in a way detailed in the NFP presentation)**
    - The UML profile for SoC (System On Chip)
      - **More specific than MARTE purpose**
    - The Real-Time CORBA profile
      - **Real-Time CORBA based architecture can be annotated for analysis with Marte**
    - The UML profile for Systems Engineering (SysML)
      - **Specialization of SysML allocation concepts and reuse of flow-related concepts**
      - **Ongoing discussion to include VSL in next SysML version**
      - **Overlap of team members**

MARTE Overview



Domain

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

Specialization of MARTE foundations for modeling purpose (specification, design, …):
- ➔ RTE model of computation and communication
- ➔ Software resource modeling
- ➔ Hardware resource modeling

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

Extracted from S.Gerard (ECRTS07)

Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - **Non-functional properties modeling**
  - **Outline of the Value Specification Language (VSL)**
- Part 3
  - The timing model
- Part 4
  - A component model for RT/E
- Part 5
  - Platform modeling
- Part 6
  - Repetitive structure modeling
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

Non-Functional Properties (NFPs)

Non-functional properties describe the "fitness" of systems behavior (E.g., performance, memory usage, power consumption)

- **Nature of NFPs**
  - Quantitative: magnitude + unit (E.g., energy, data size, duration)
  - Qualitative (E.g., periodic or sporadic event arrival patterns)
- **NFP values need to be qualified**
  - E.g. source, statistical measure, precision,…
- **NFPs need to be parametric and derivable**
  - Variables: placeholders for unknown values
  - Expressions: math. and time expressions
- **NFPs need clear semantics**
  - Predefined NFPs (E.g., end-to-end latency, processor utilization)
  - User-specific NFPs (but still unambiguously interpreted!)

Introduction to the MARTE's NFPs Framework

- UML lacks modeling capabilities for NFPs !!

- Value qualifiers?
- Measures?
- NFP Libraries?
- Annotation mechanism?

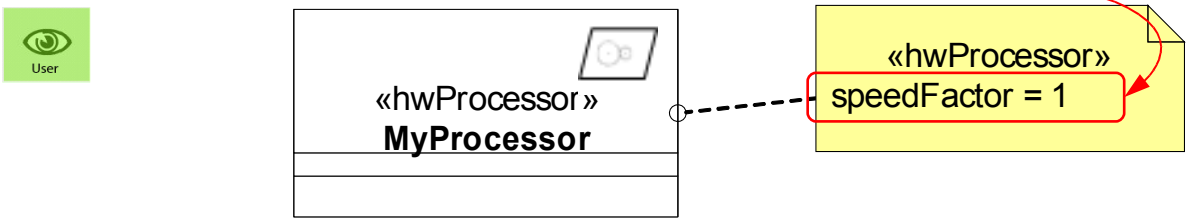
- And UML expression syntax is also not sufficient!!

- Variables?
- **Structured Values?**
- Data Type System?
- Complex time expressions?

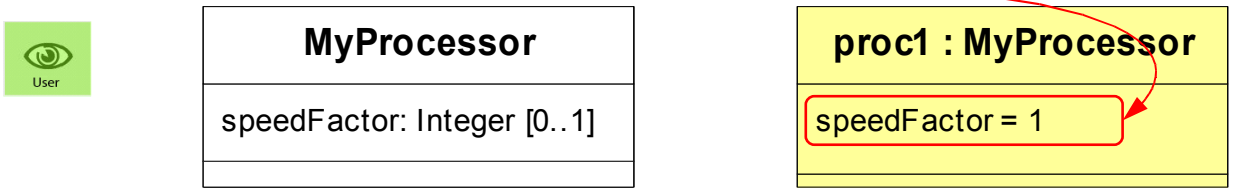
The MARTE's NFP sub-profile

UML Profile for NFPs

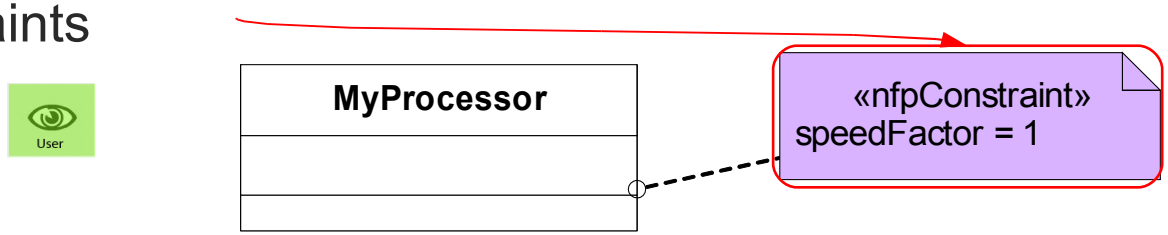
- **Three mechanisms to annotate UML models:**
  - Values of stereotype properties



  - Slot values of classifier instances



  - Constraints



Annotating NFPs in Tagged Values

UML MARTE
www.omgmarte.org

MARTE pre-defined

1) Declare NFP types

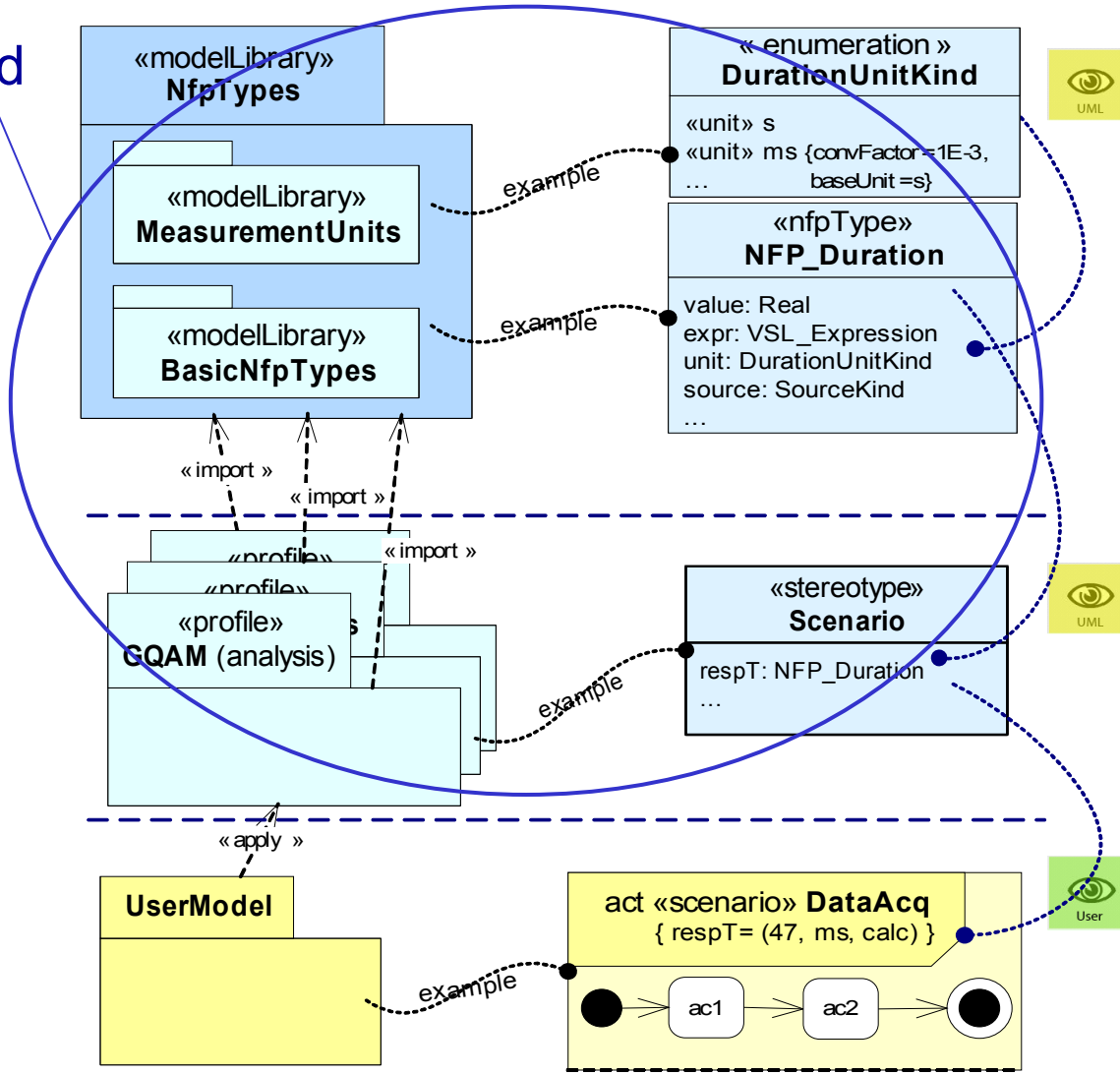
- Define measurement units and conversion parameters
- Define NFP types with qualifiers

2) Define NFP-like extensions

- Define stereotypes and their attributes using NFP types

3) Specify NFP values

- Apply stereotypes and specify their tag values using VSL



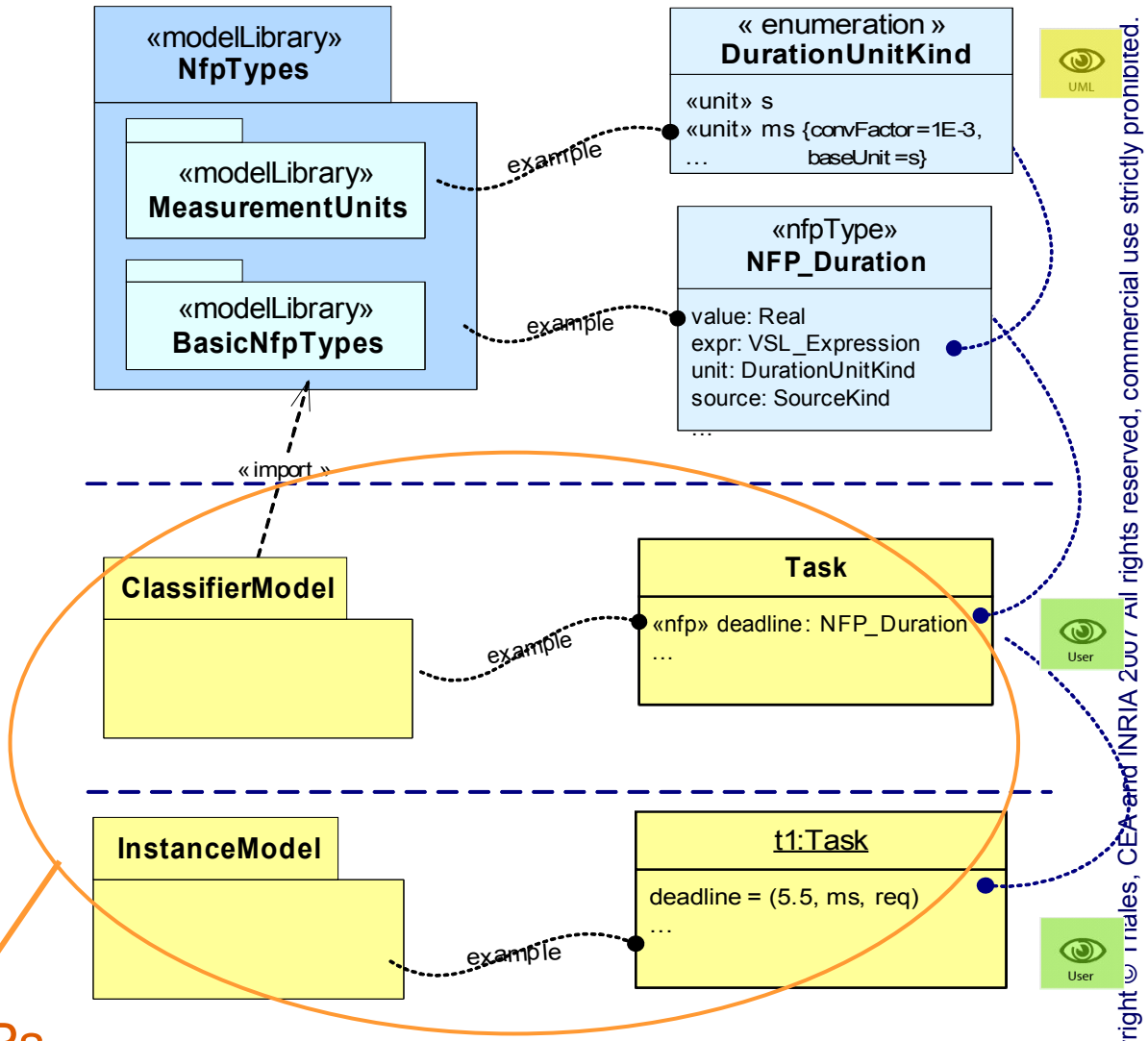
Annotating NFPs in Slots

UML MARTE
www.omgmarte.org

1) Declare NFP types
- Define measurement units and conversion parameters
- Define NFP types with qualifiers

2) Declare NFPs in user models
- Define classifiers and their attributes using NFP types
- Such attributes are tagged as «nfp»

3) Specify NFP values
- Instantiate classifiers and specify their slot values using VSL



Model-specific NFPs

cea list THALES INRIA

Annotating NFPs in Constraints

1) Declare NFP types

- Define measurement units and conversion parameters
- Define NFP types with qualifiers

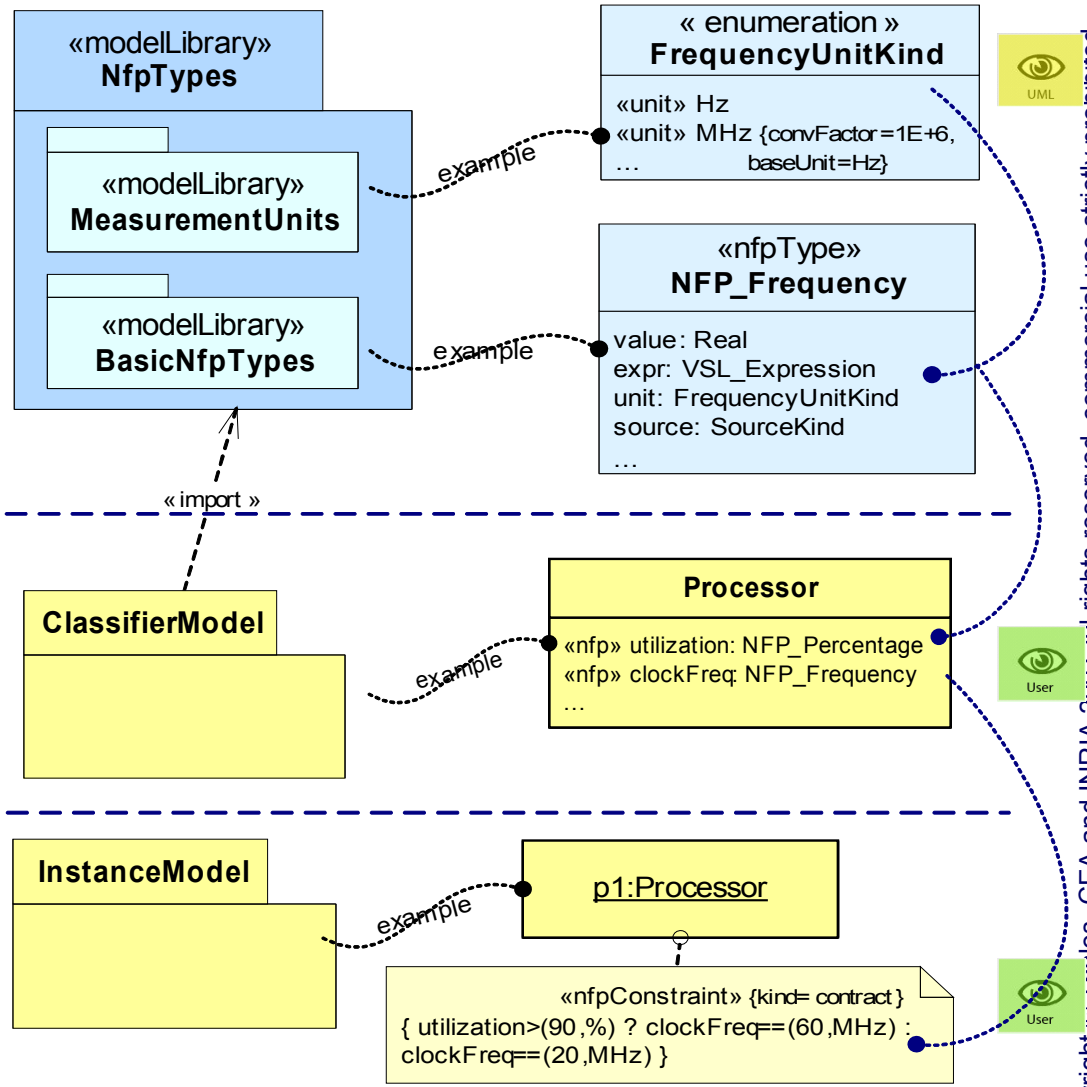
2) Declare NFPs

-Define classifiers and their attributes using NFP types

3) Specify NFP values

-Create Constraints to define assertions on NFP values using VSL

-«nfpConstraint» is a *required*, *offered*, or *contract* constraint of NFPs



The MARTE's NFP Modeling Framework

- Three main language extensions to UML syntax
  - Grammar for extended expressions
  - Stereotypes for extended data types
  - Complex time expressions

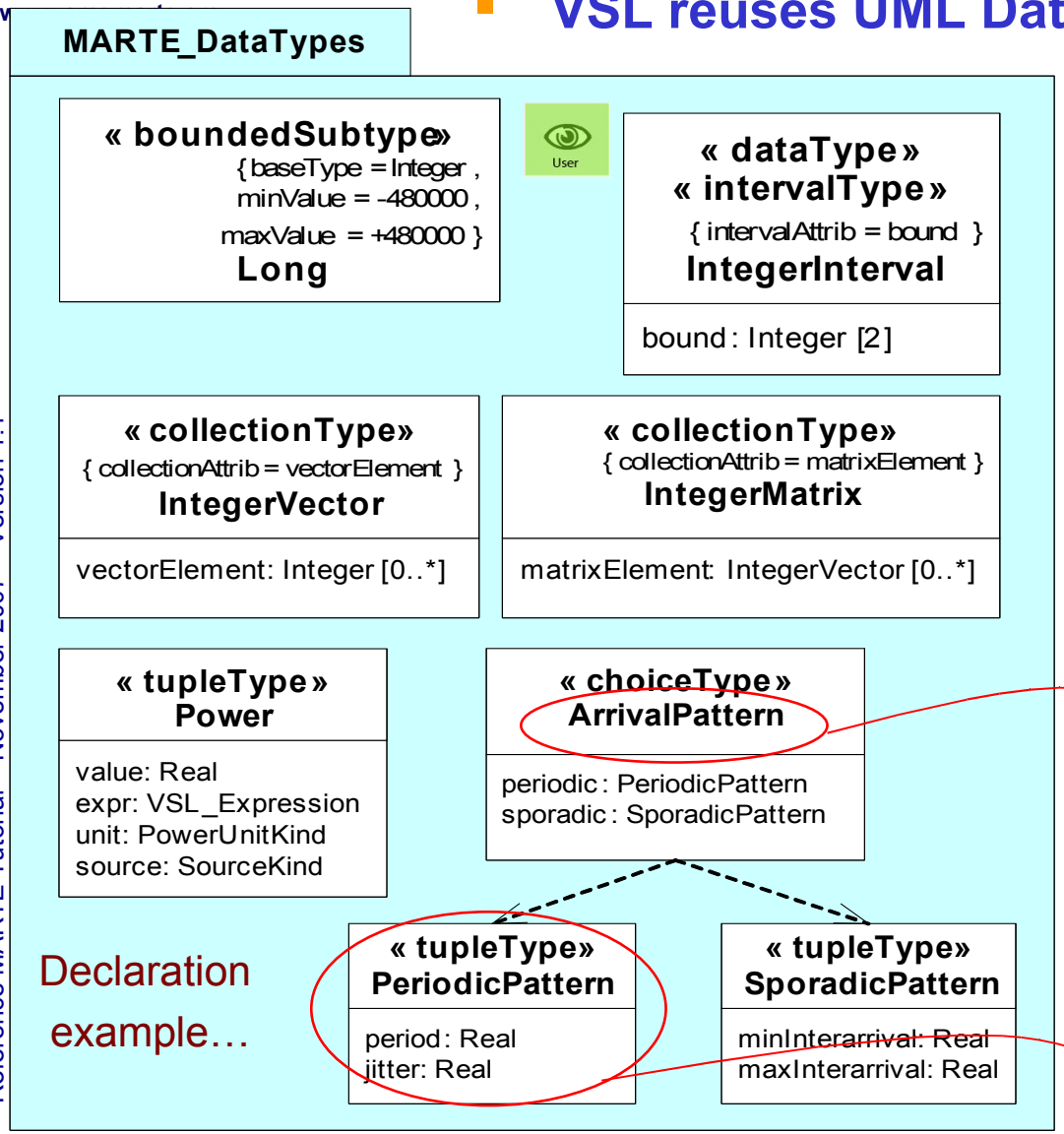
Basic Textual Expressions in VSL

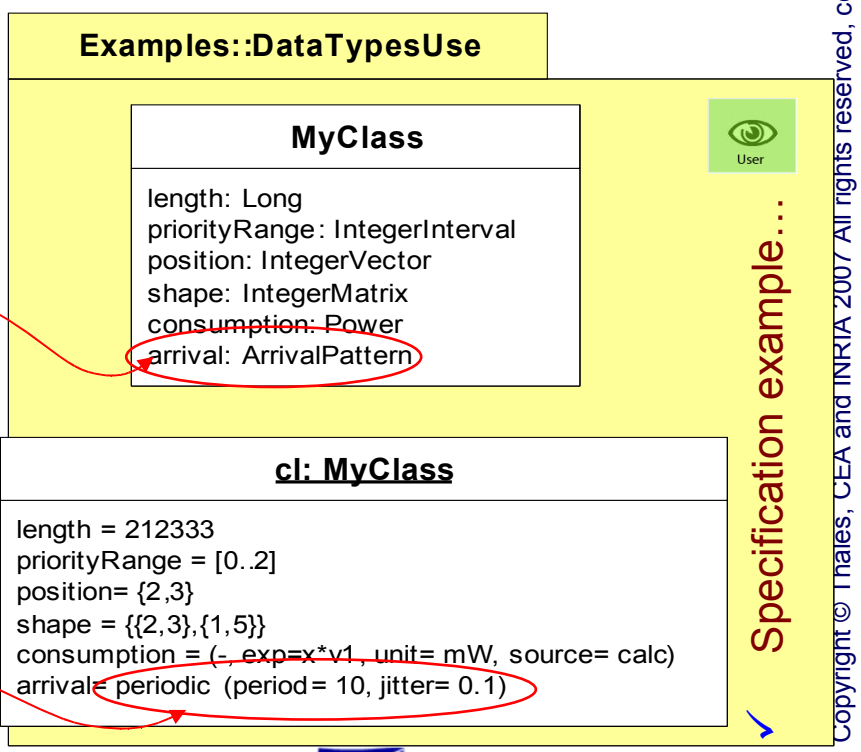
- Scope of the proposed extensions
  - Extended Primitive Values
  - Extended Composite Values
  - Extended Expressions

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..` `{{1,2,3}, {3,2}} //collection of collections`
Tuple and choice	`(value=2.0, unit= ms) //duration tuple value` `periodic(period=2.0, jitter=3.3) //arrival pattern`
Interval	`[1..251[ //upper opened interval between integers` `[$A1..$A2] //interval between variables`
Variable declaration & Call	`io$var1 //input/output variable declaration` `var1 //variable call expression.`
Arithmetic Operation Call	`+(5.0,var1) //"add" operation on Real datatypes` `5.0+var1 //infix operator notation`
Conditional Expression	`((var1<6.0)?(10^6):1) //if true return 10 exp 6,else 1`

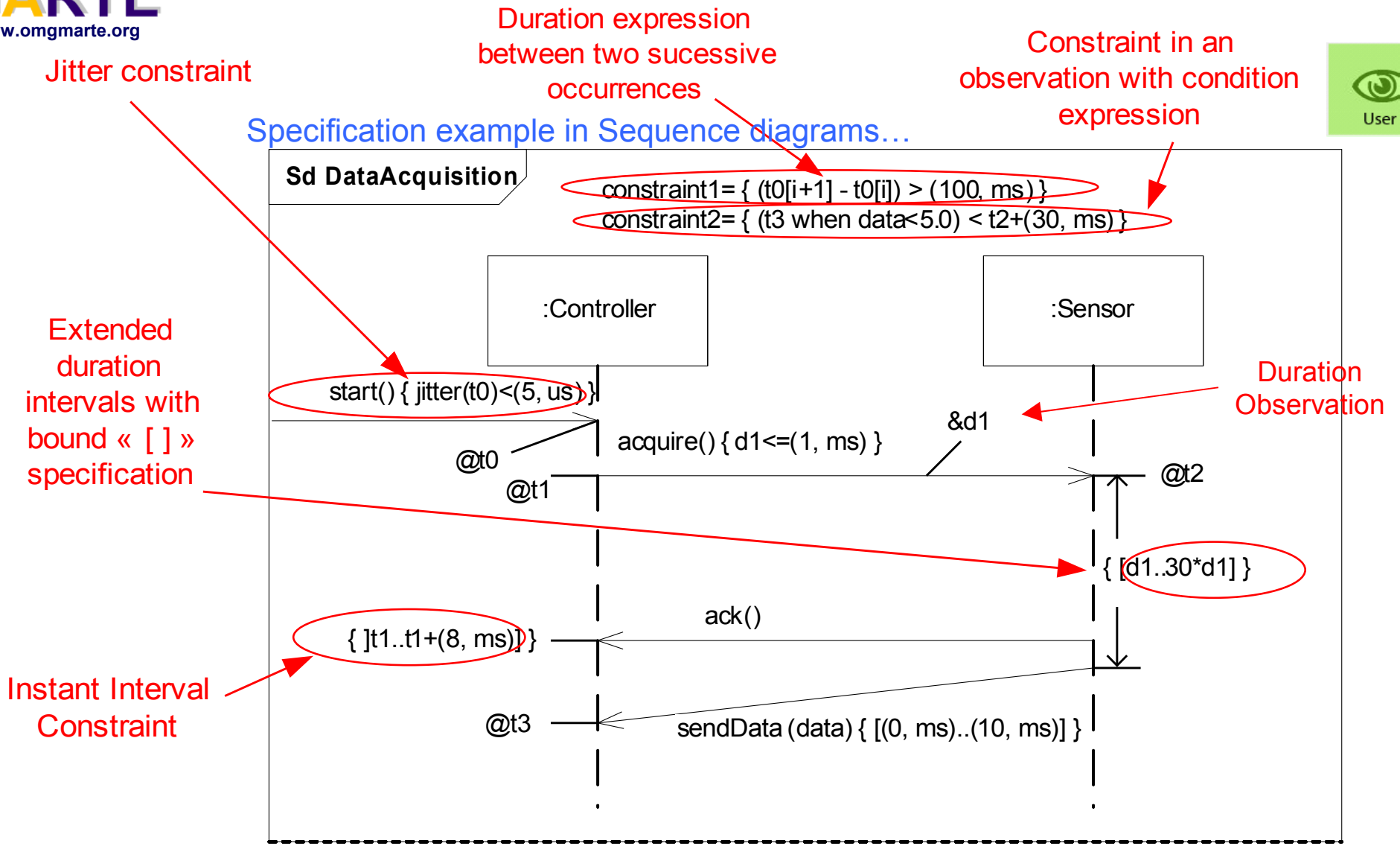
VSL Extended Data Types

- VSL reuses UML DataType constructs, but adds…
  - BoundedSubtype
  - IntervalType
  - CollectionType
  - TupleType
  - ChoiceType





Examples of Time Expressions with VSL



Conclusions on MARTE::NFPs

- **Synthesis of best modeling practices…**
  - OCL: full constraint language, but hard to use and not real-time oriented
  - SPT Profile: built-in TVL language is simpler, but not flexible
  - QoS&FT Profile: annotation mechanism is flexible, but complex

  ➔ NFP & VSL reuse selected modeling features, while still providing simplicity and flexibility

- **Foundations…**
  - Reuse OCL constructs: grammar for values and expressions
  - Generic data type system: (based on ISO's General-Purpose Datatypes)
  - VSL extends UML Simple Time model (e.g. occurrence index, jitters)
  - Formally defined by abstract and concrete syntaxes (grammar)

Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- **Part 3**
  - **The timing model**
- Part 4
  - A component model for RT/E
- Part 5
  - Platform modeling
- Part 6
  - Repetitive structure modeling
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

Time in MARTE Overview

- **SPT, UML 2 and Time**
  - UML::CommonBehaviors::SimpleTime
- **the MARTE Time domain view**
  - a.k.a. the MARTE Time meta-model
  - Concepts and relationships
- **the MARTE Time sub-profile**
  - a.k.a. UML view
- **Usage of the Time sub-profile**

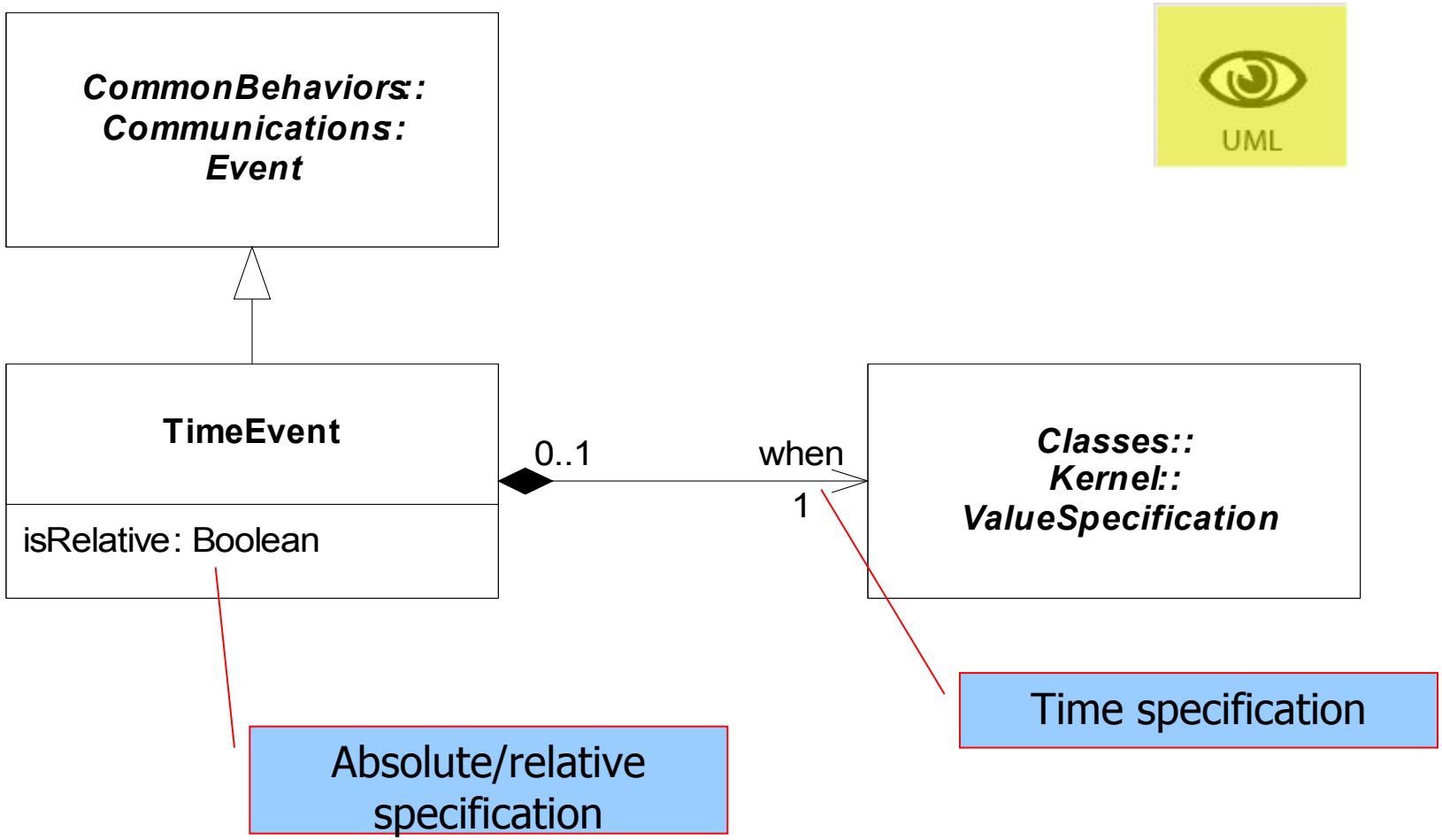
- OMG UML profile formal/05-01-02 (v1.1)
- Based on UML 1.4 — To be aligned to UML 2
- Dealing with time and resources
- Quantitative time information — Metric time
- Concepts
  - Instant, duration
  - Event bound to time, stimuli
- Timing mechanisms & services

UML::CommonBehaviors::SimpleTime

- **UML2 adds new metaclasses to represent**
  - Time
  - Duration
  - Observation (of time passing)
  - Some forms of time constraints
- **Simple (even simplistic) model of time**
- **Advice: *Use a more sophisticated model of time provided by an appropriate profile*, if needed. [UML superstructure, chapter 13]**

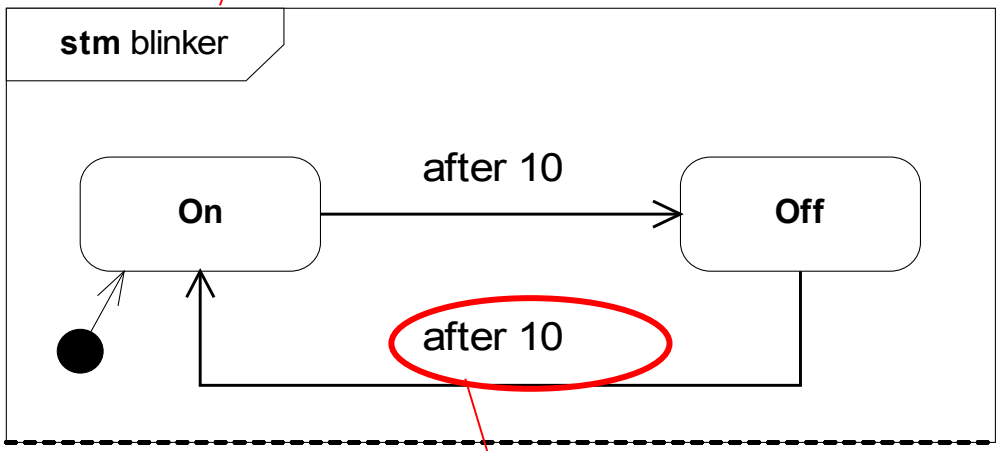
e.g., MARTE

SimpleTime::TimeEvent



TimeEvent – usage (1)

UML state machine = behavior

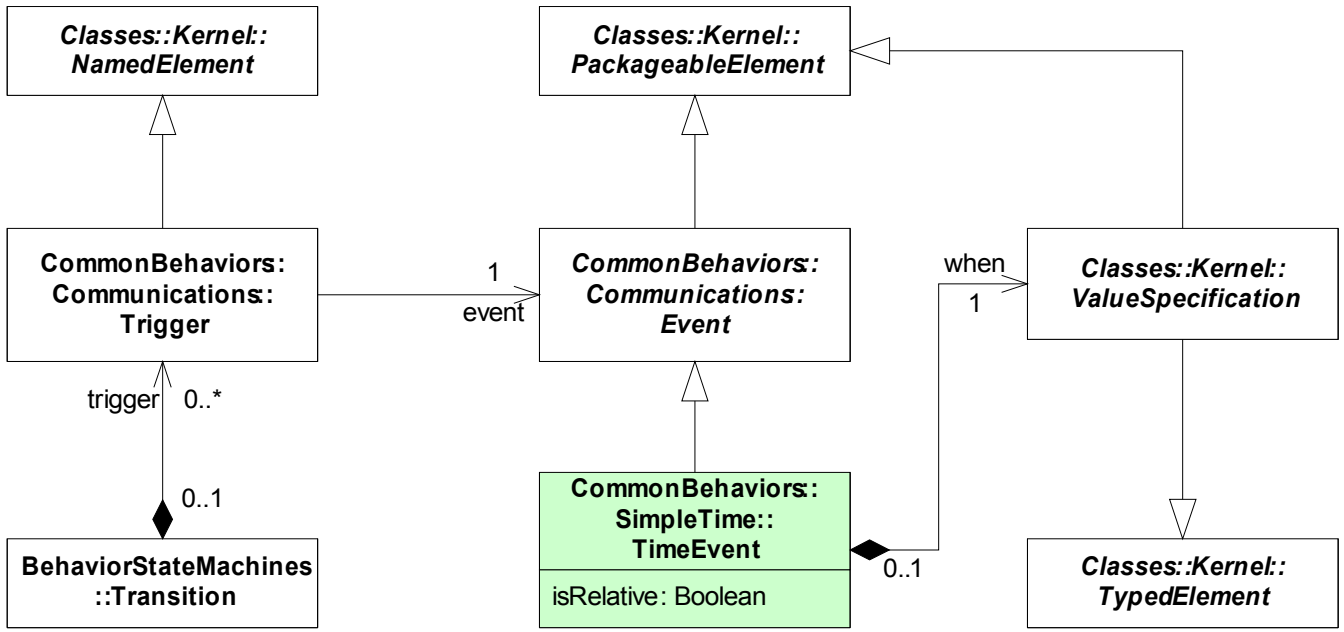


Specification of a time-trigger

Informal semantics

TimeEvent – usage (2)

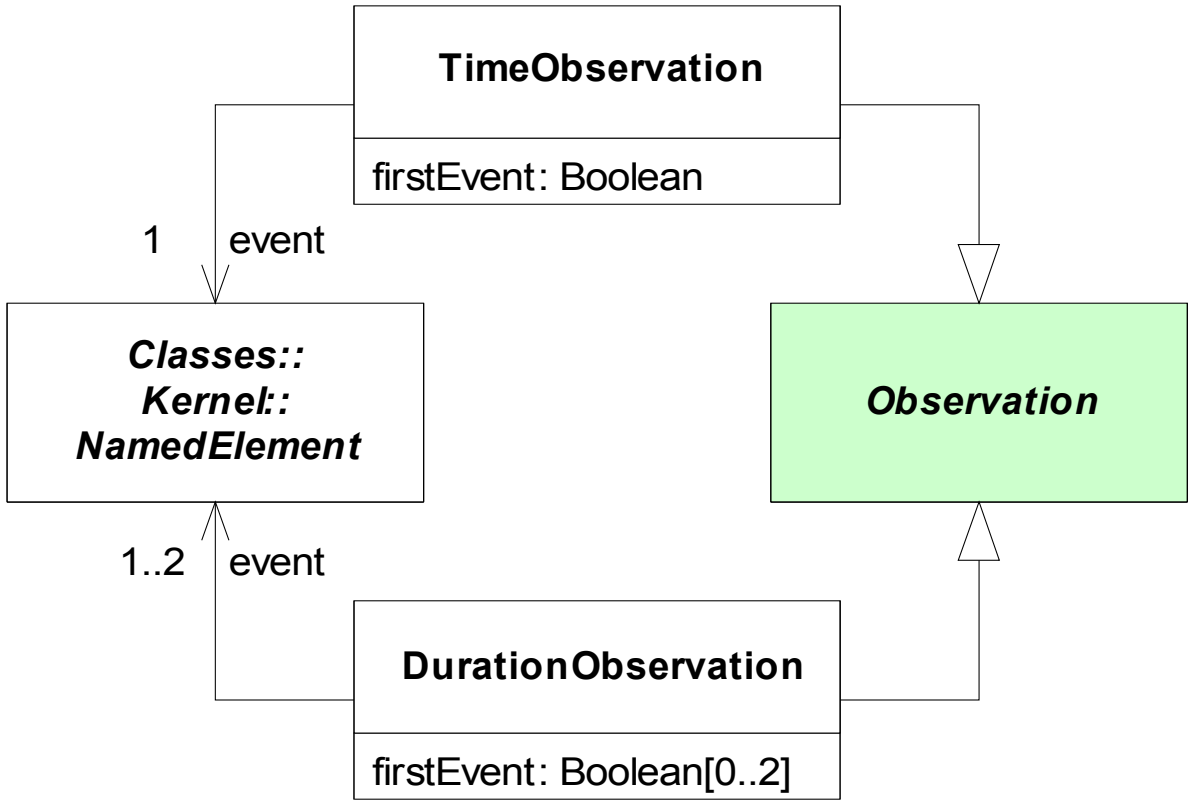
Meaning of "after 10"



Metaclasses involved in the modeling of a transition triggered by a TimeEvent

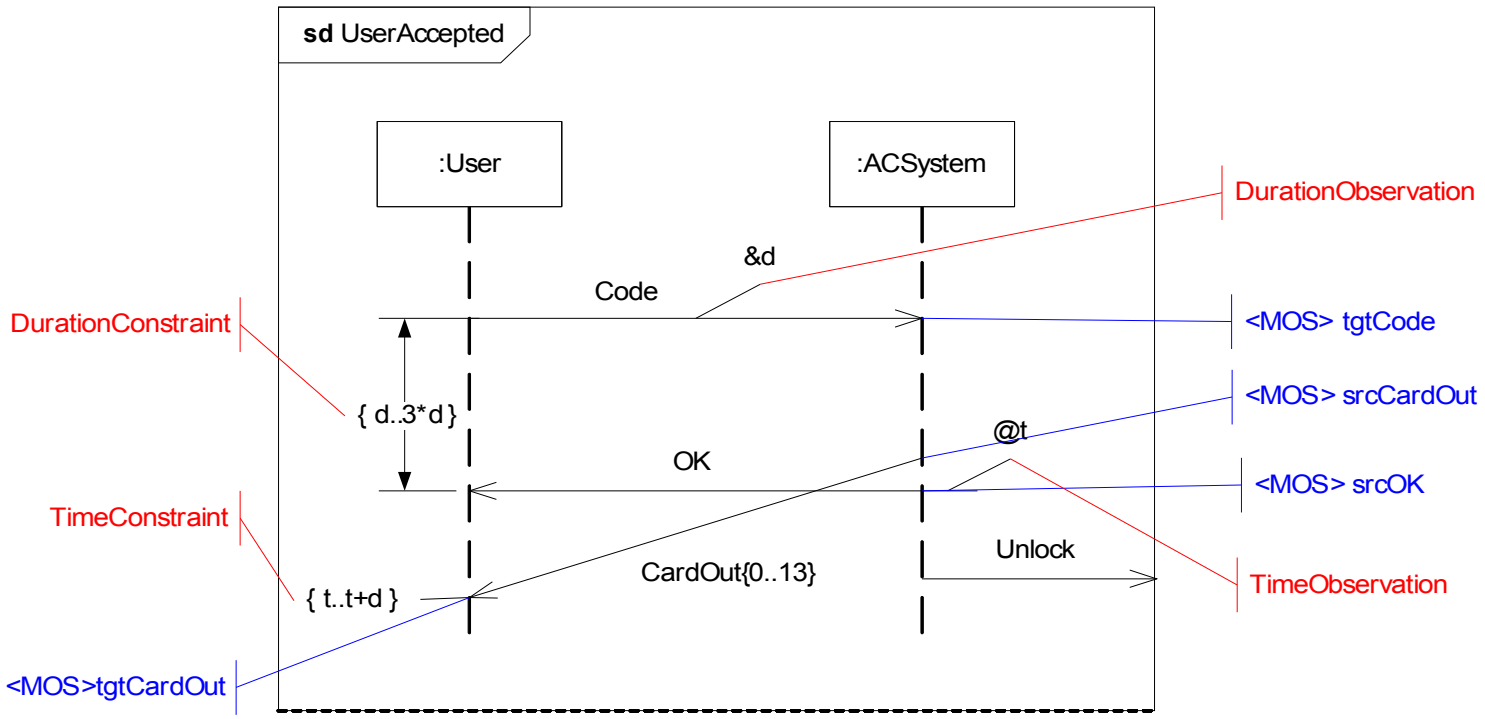
Simple annotation → complex implied structure

SimpleTime::Observation



Observation – usage (1)

Example of sequence diagram

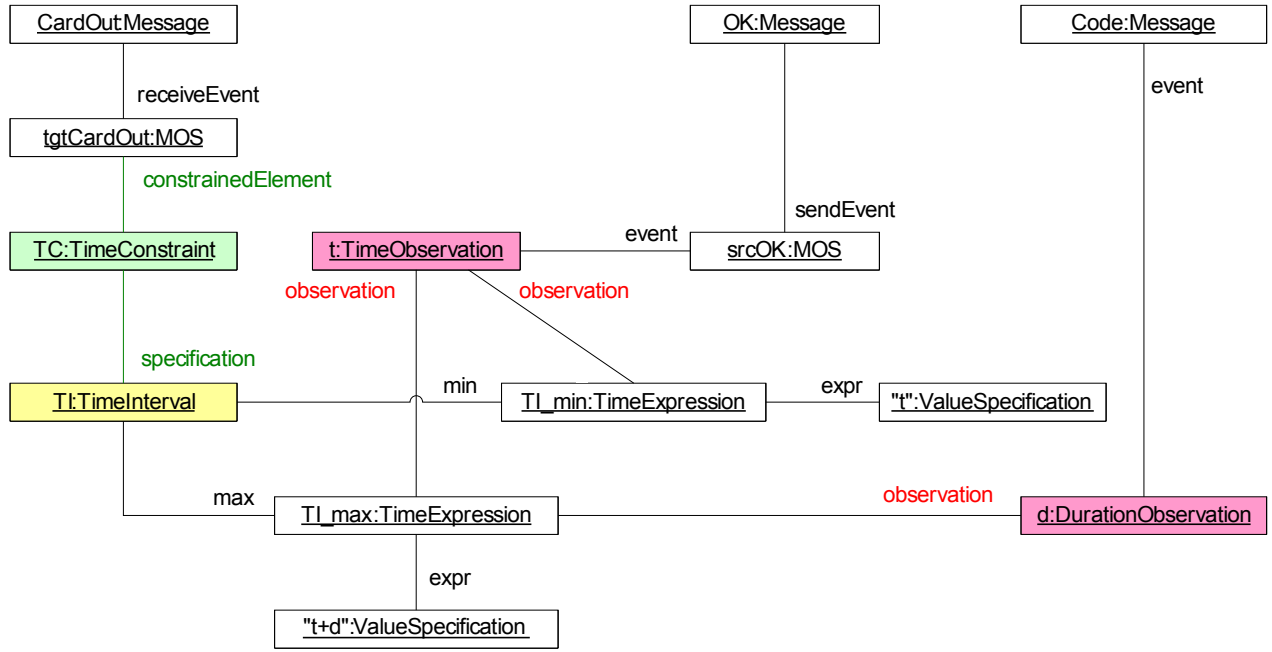


MOS stands for MessageOccurrenceSpecification

Note that red and blue annotations are not part of the UML notation.

Observation – usage (2)

Instance model of the time constraint: receive CardOut in {t .. t+d}



Simple annotation → complex implied structure

Overview

- SPT, UML 2 and Time
  - UML::CommonBehaviors::SimpleTime
- **the MARTE Time domain view**
  - **a.k.a. the MARTE Time meta-model**
  - **Concepts and relationships**
- the MARTE Time sub-profile
  - a.k.a. UML view
- Usage of the Time sub-profile

Concepts in MARTE::Time (1/2)

- **Time structure =**

  set of time bases + time structure relations

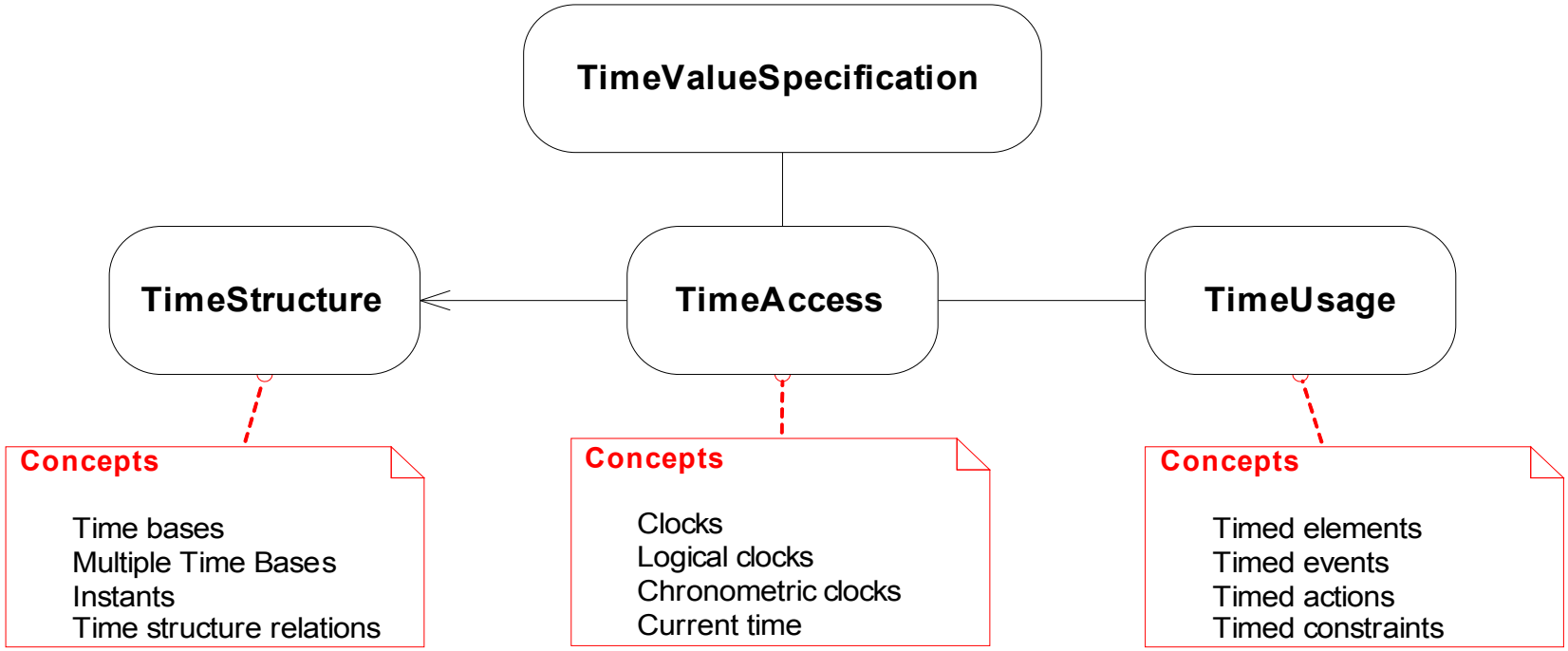
  → Partially ordered set of instants

- **Access to time = Clock**

- **Principle:** associate Clocks with model elements
  - Behavioral elements → TimedEvent, TimedProcessing
  - Constraints → TimedConstraint
  - Data types and values → TimedValue

Concepts in MARTE::Time (2/2)

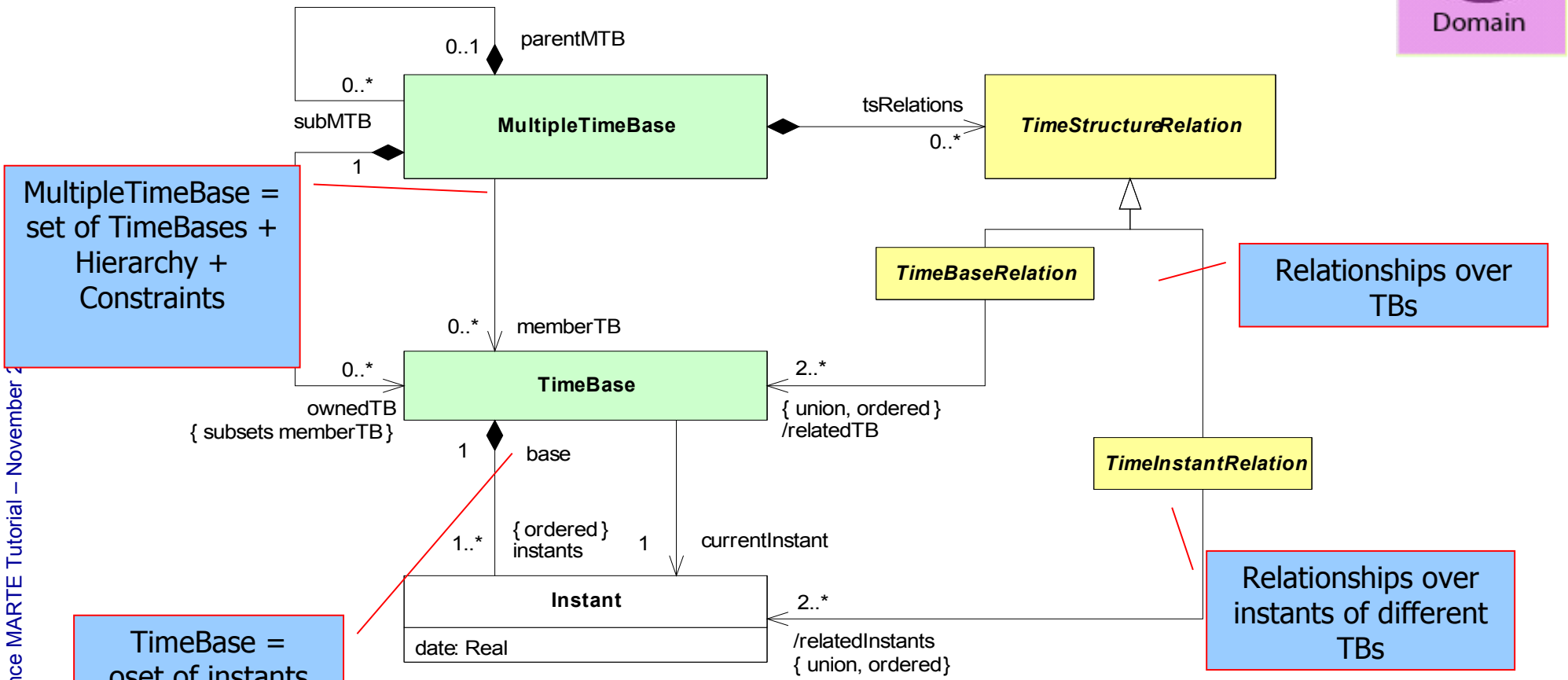
Main concepts introduced in Time modeling



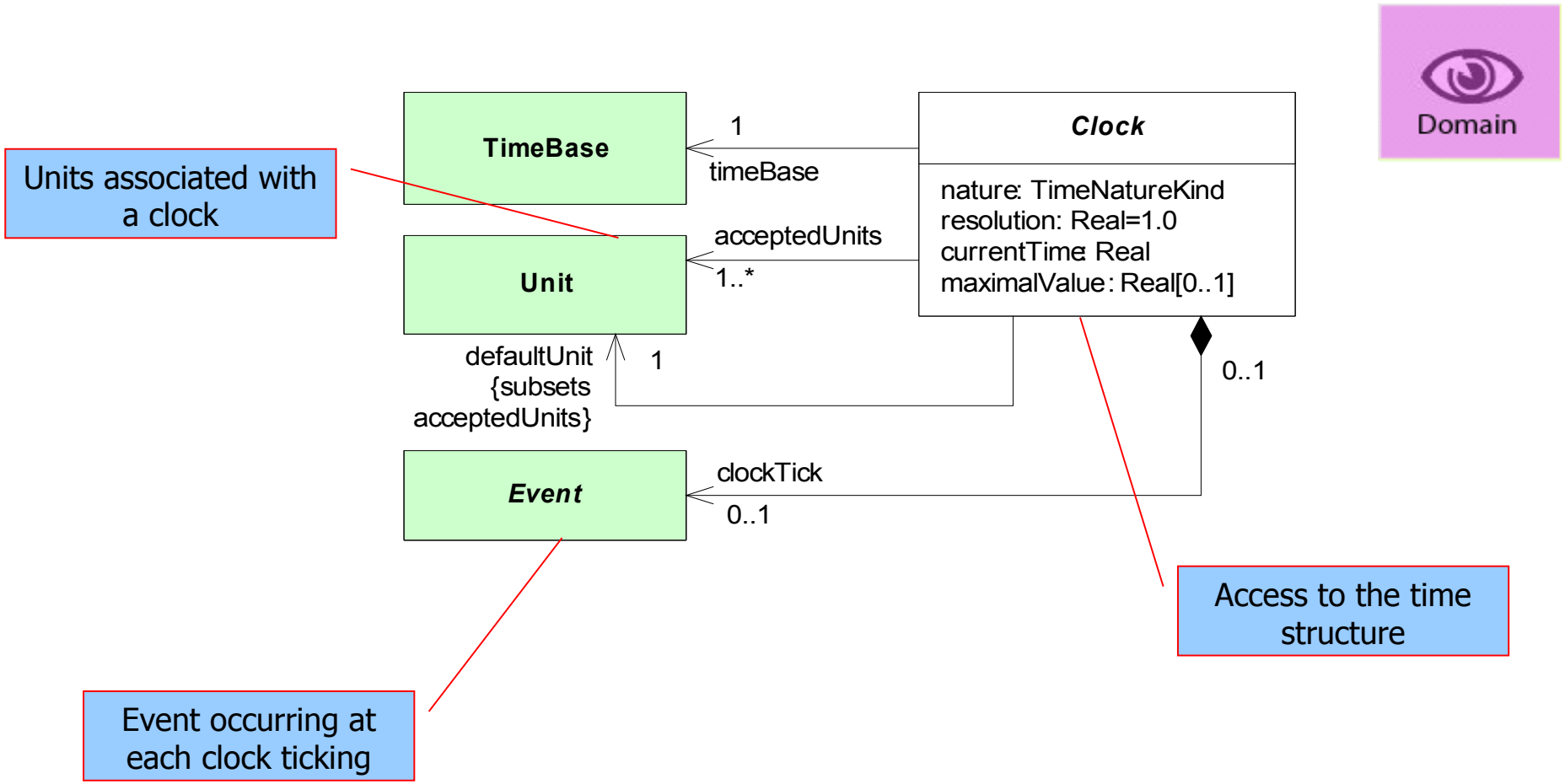
Not a UML diagram!

Time Structure

Domain



Access to Time: Clock

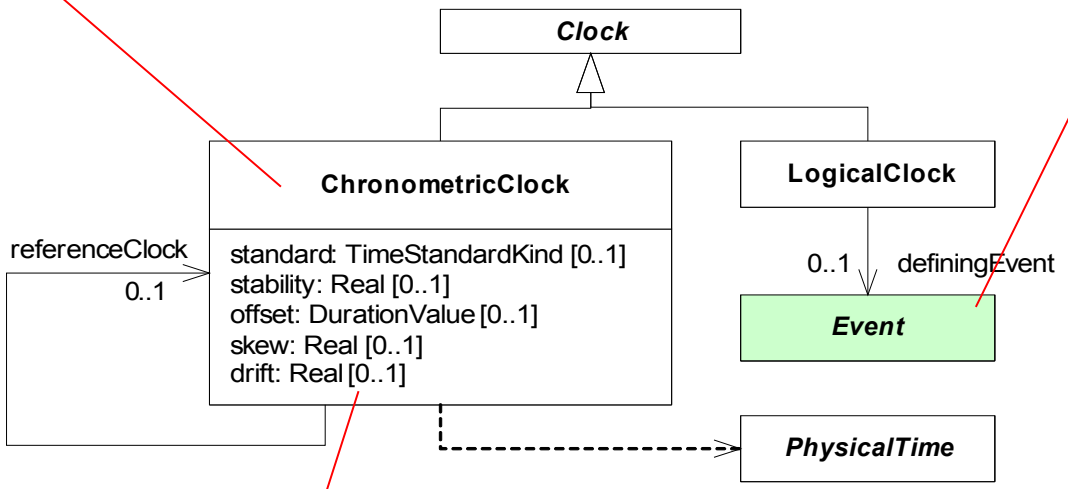


Chronometric/Logical Clocks

Two kinds of clocks

Domain

Implicit reference to physical time

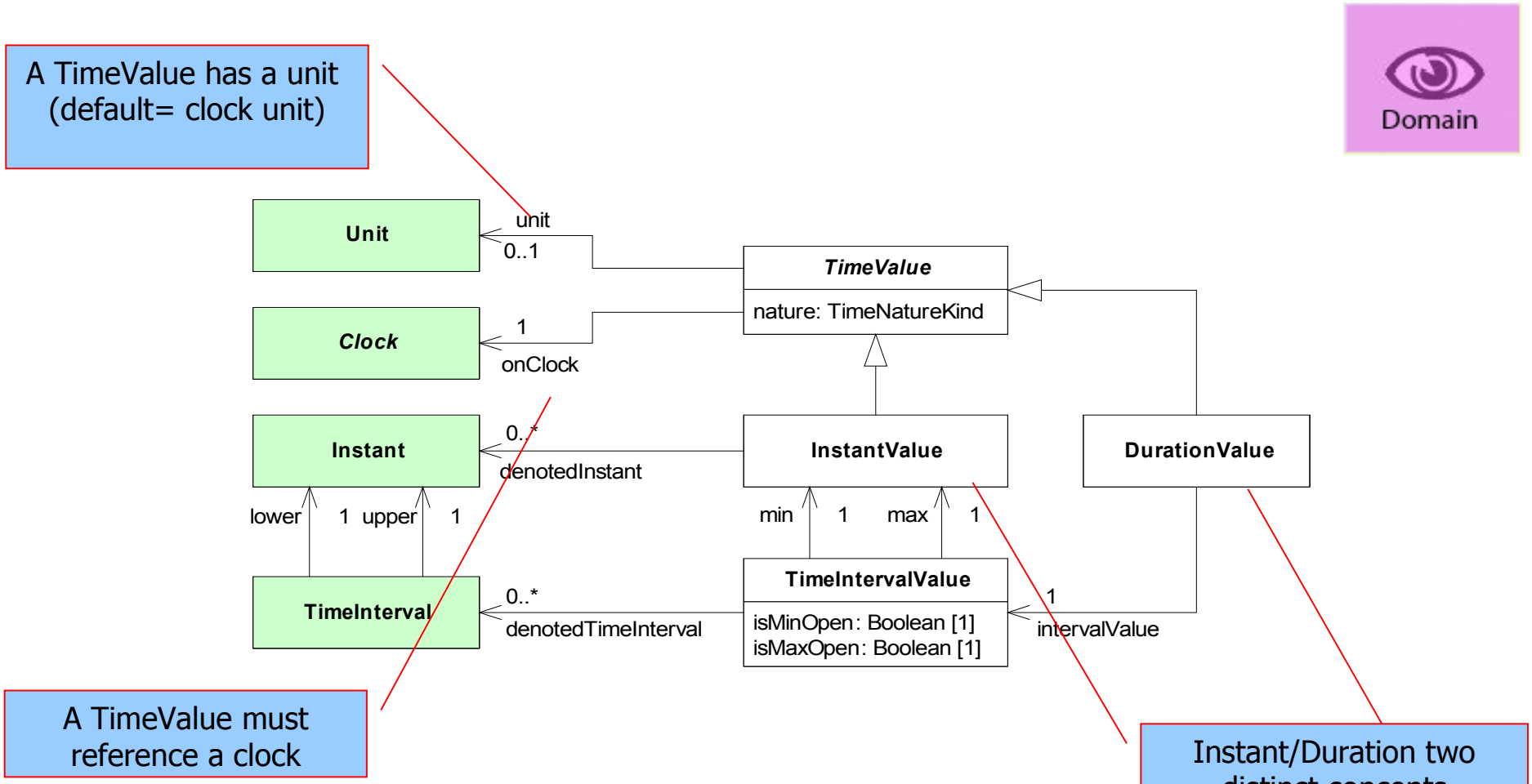


Possible reference to a repetitive event

NFPs measured against a reference clock

Time Values

A TimeValue has a unit (default= clock unit)

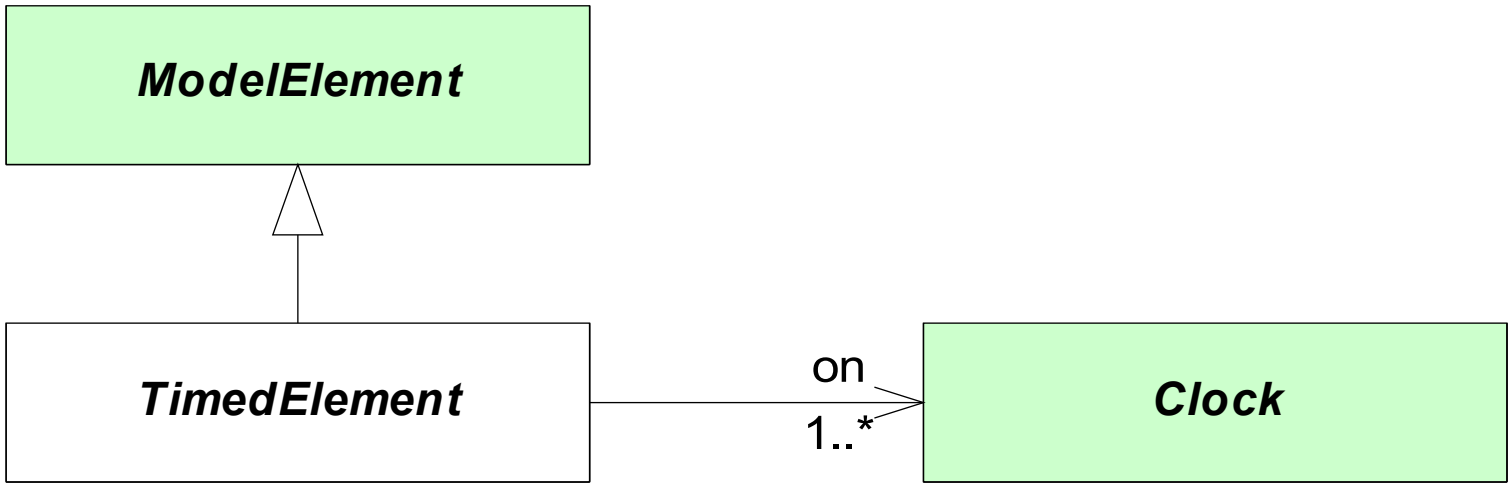


A TimeValue must reference a clock

Instant/Duration two distinct concepts

Timed Entities: TimedElement

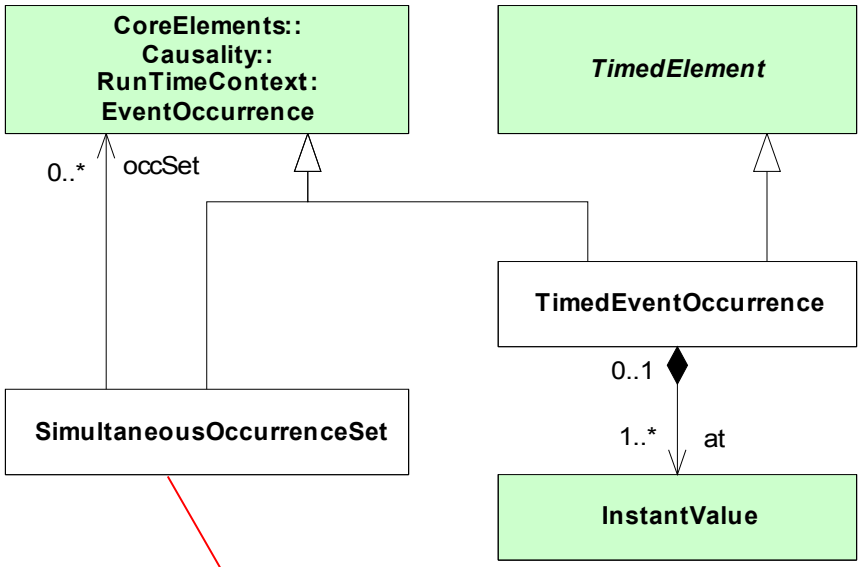
The unifying concept: a TimedElement = a ModelElement + a Clock



Timed Entities: TimedEvent

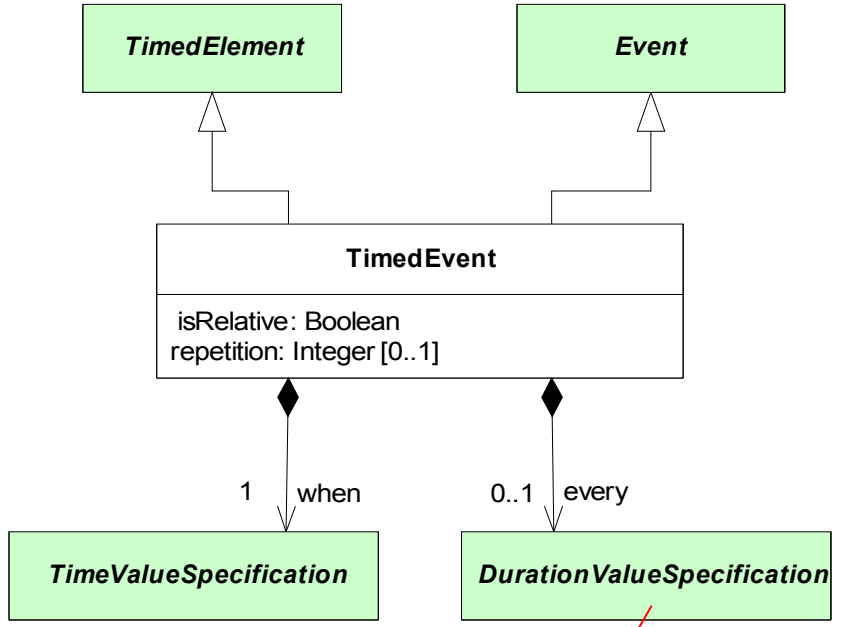
Domain

occurrences



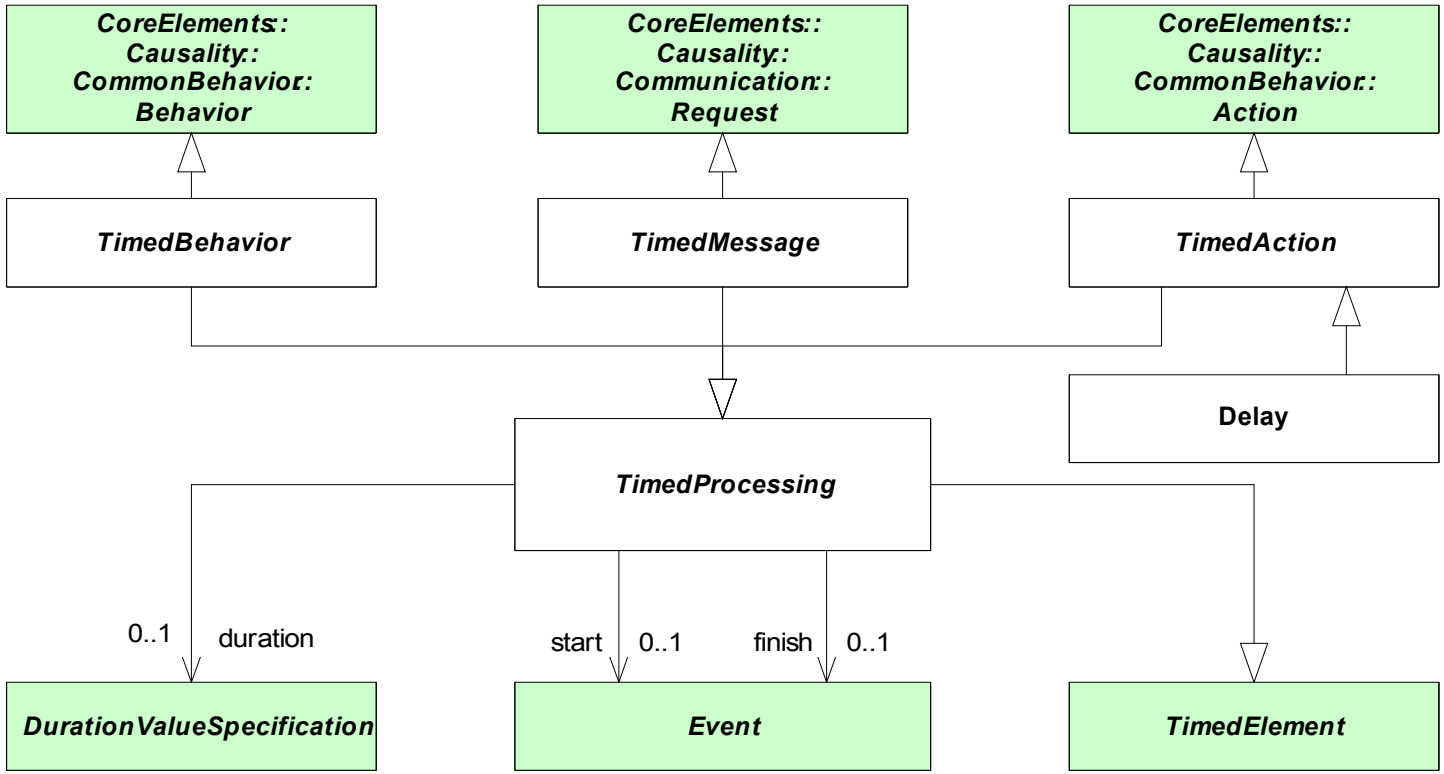
Provision for simultaneity

events

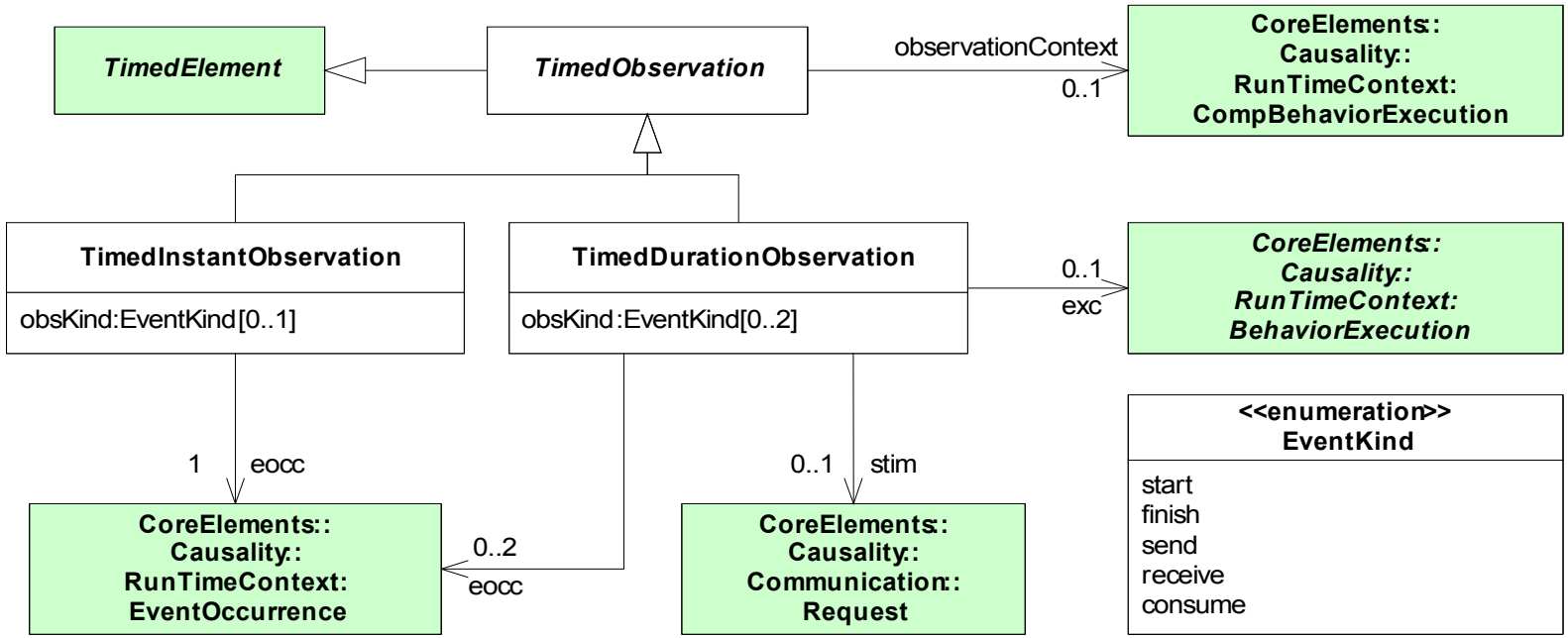


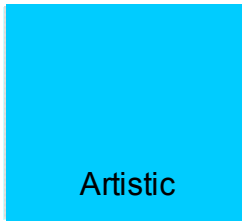
Facility to specify multiple occurrences

Timed Entities: TimedProcessing



Timed Entities:TimedObservation



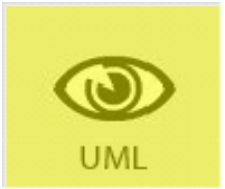


See:

http://en.wikipedia.org/wiki/Image:The_Persistence_of_Memory.jpg

Overview

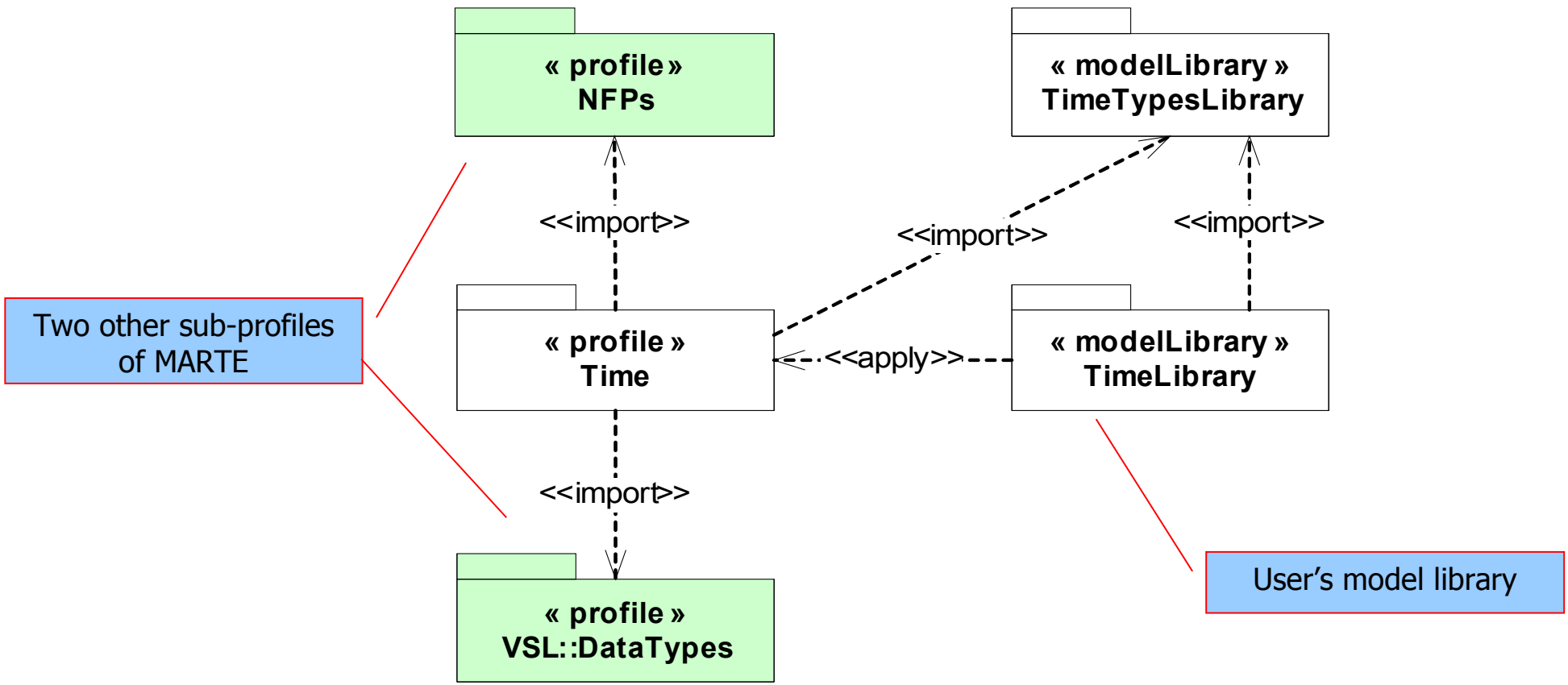
- SPT, UML 2 and Time
  - UML::CommonBehaviors::SimpleTime
- the MARTE Time domain view
  - a.k.a. the MARTE Time meta-model
  - Concepts and relationships
- **the MARTE Time sub-profile**
  - **a.k.a. UML view**
- Usage of the Time sub-profile



Providing extensions to UML

- **Through a UML profile**
  - New Stereotypes
- **Facilities**
  - Model libraries
  - Dedicated languages (especially for expressions)

Dependencies



Central stereotypes: ClockType & Clock

Chronometric clock → "physical " time; units ∈{s,ms,us,…}

Logical clock → any repetitive event; units ∈ {tick} U PhysicalUnits

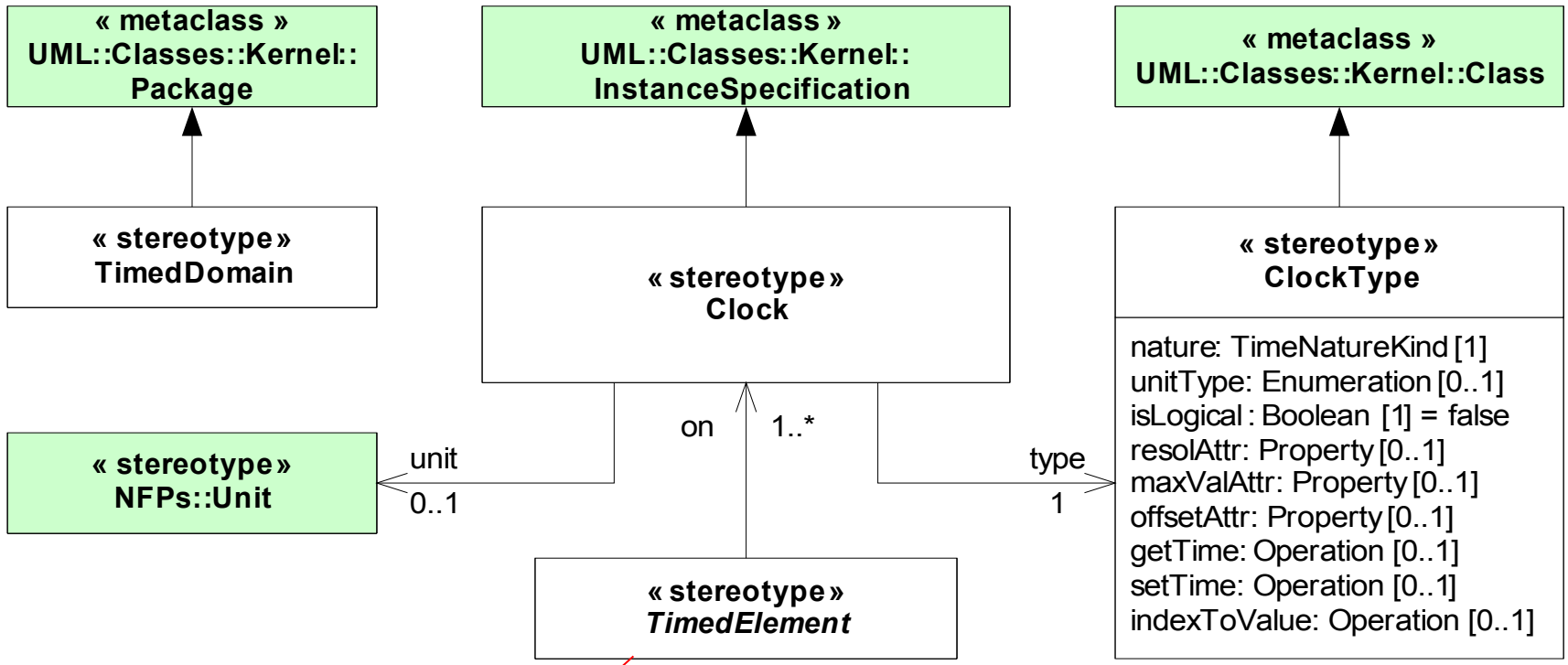
- **Accepted units**
- **Default unit**

Stereotype properties :
Special semantics

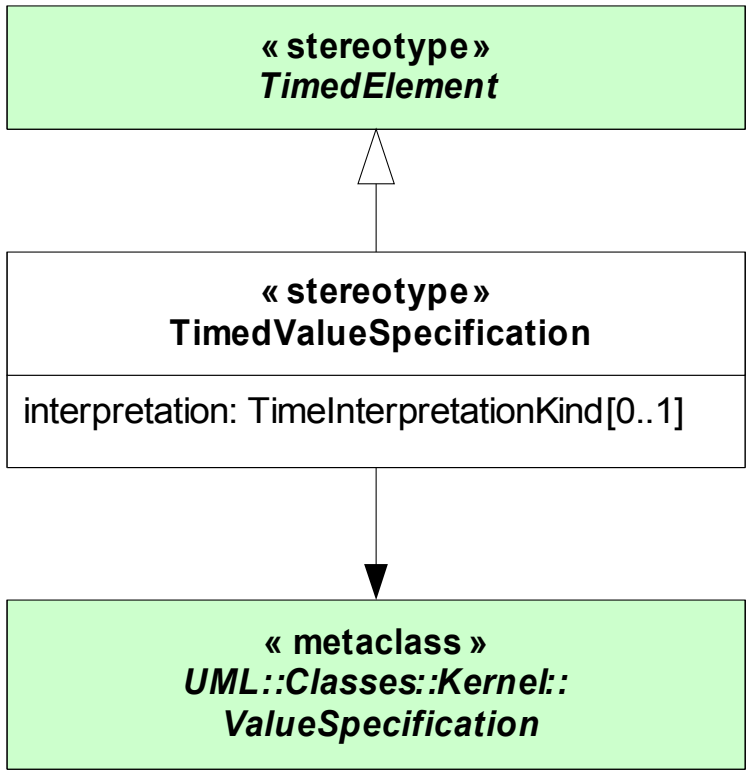
+ optional
- **set of properties**
- **set of operations**

nature / isLogical	discrete	dense
true	**Logical clock**	Not used
false	**Chronometric clock**	
	discrete	dense

Clock and TimedElement

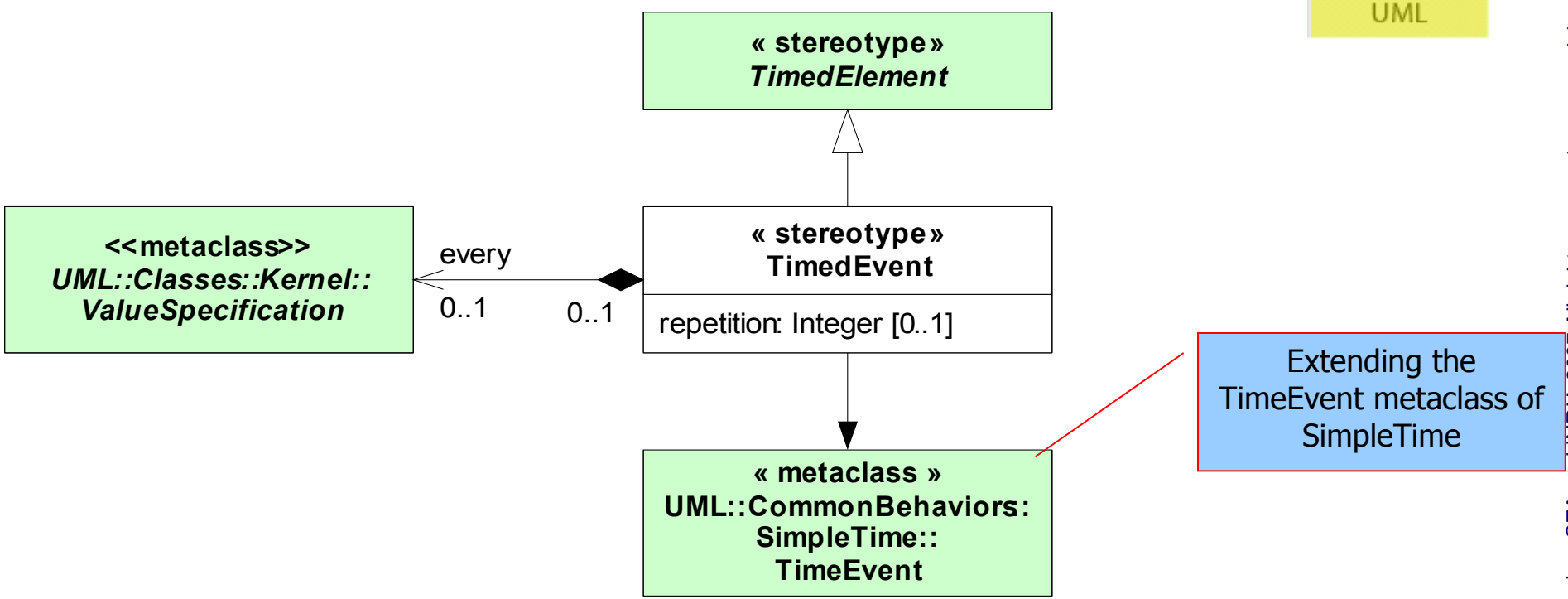


TimedValueSpecification

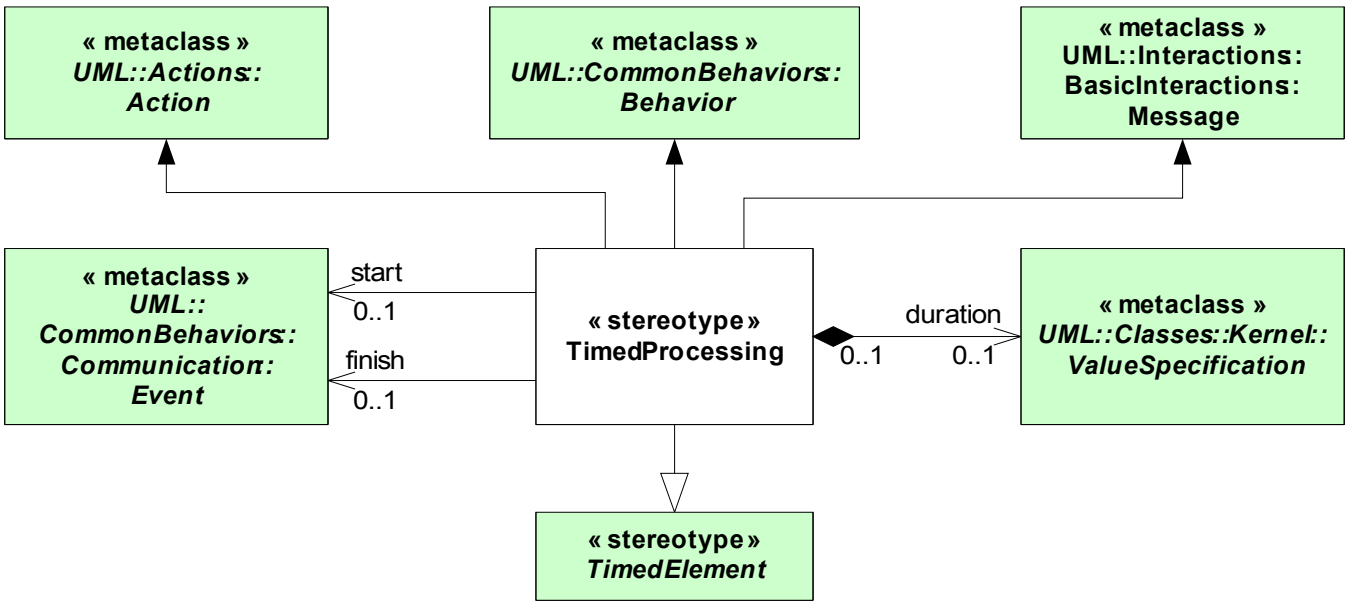


- either Instant
- or Duration

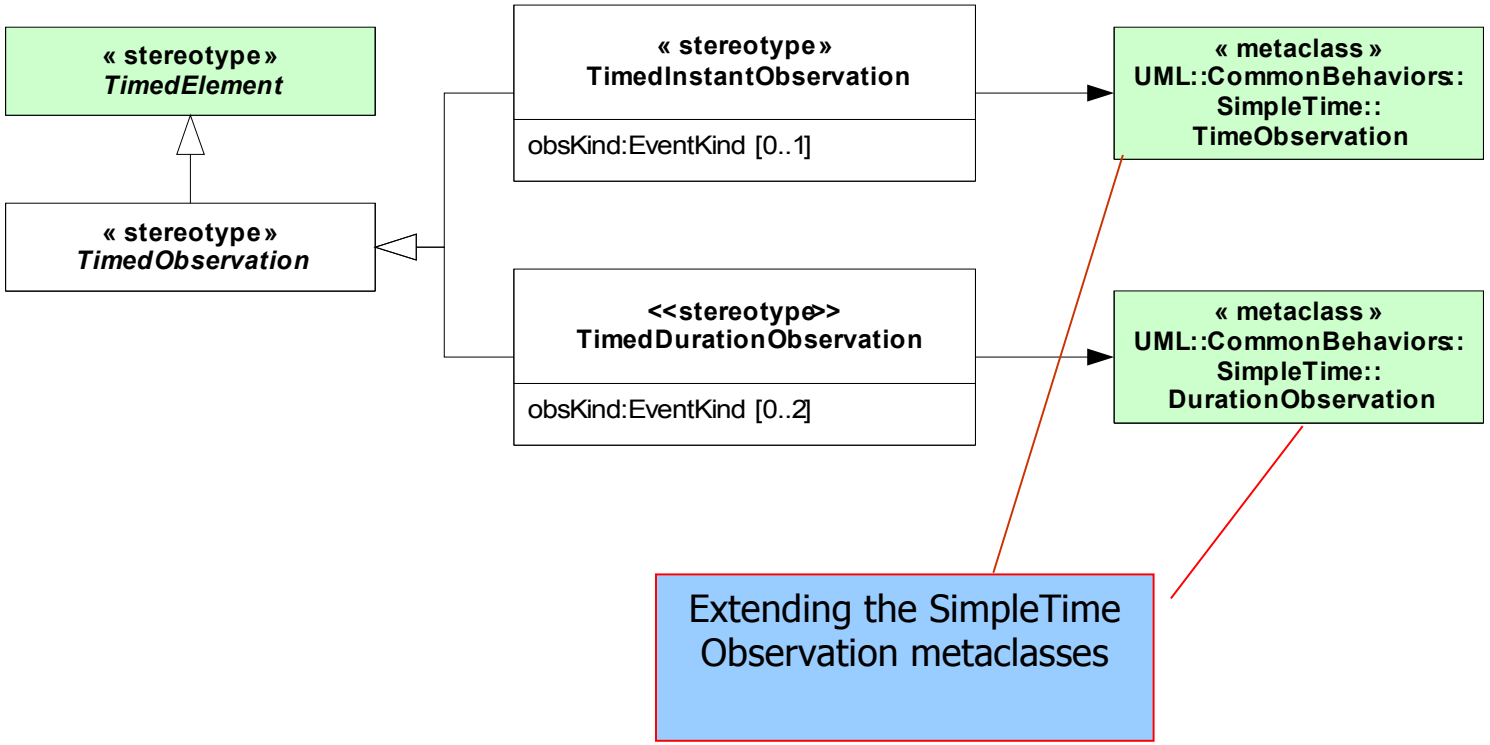
TimedEvent



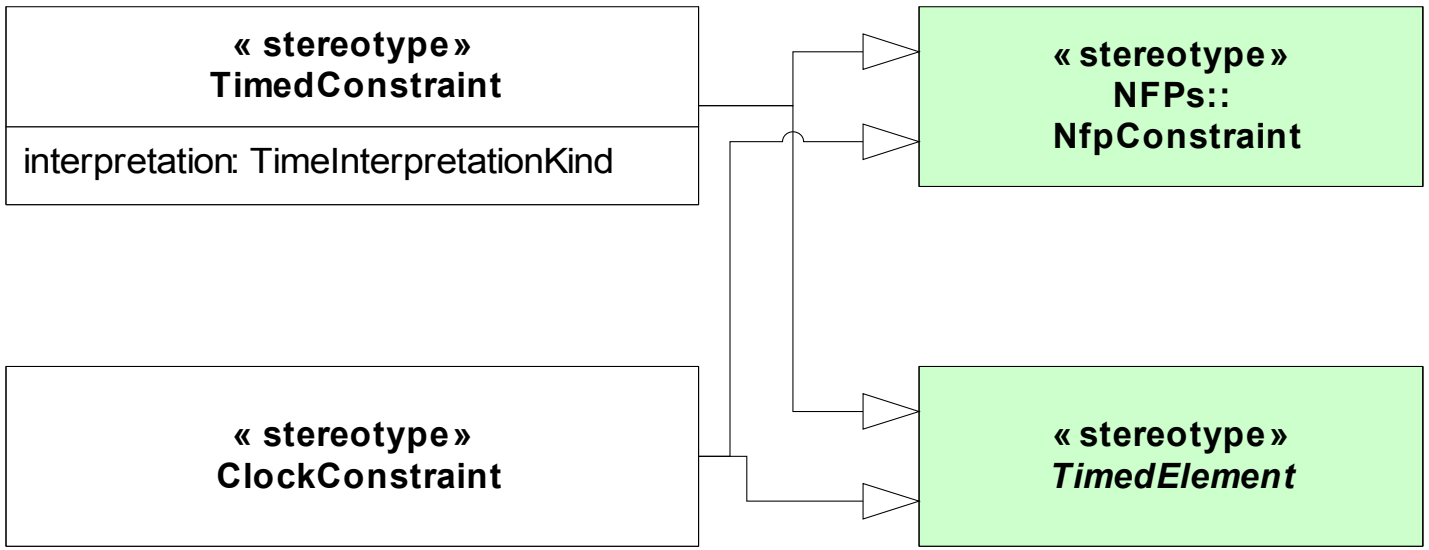
TimedProcessing



TimedObservation

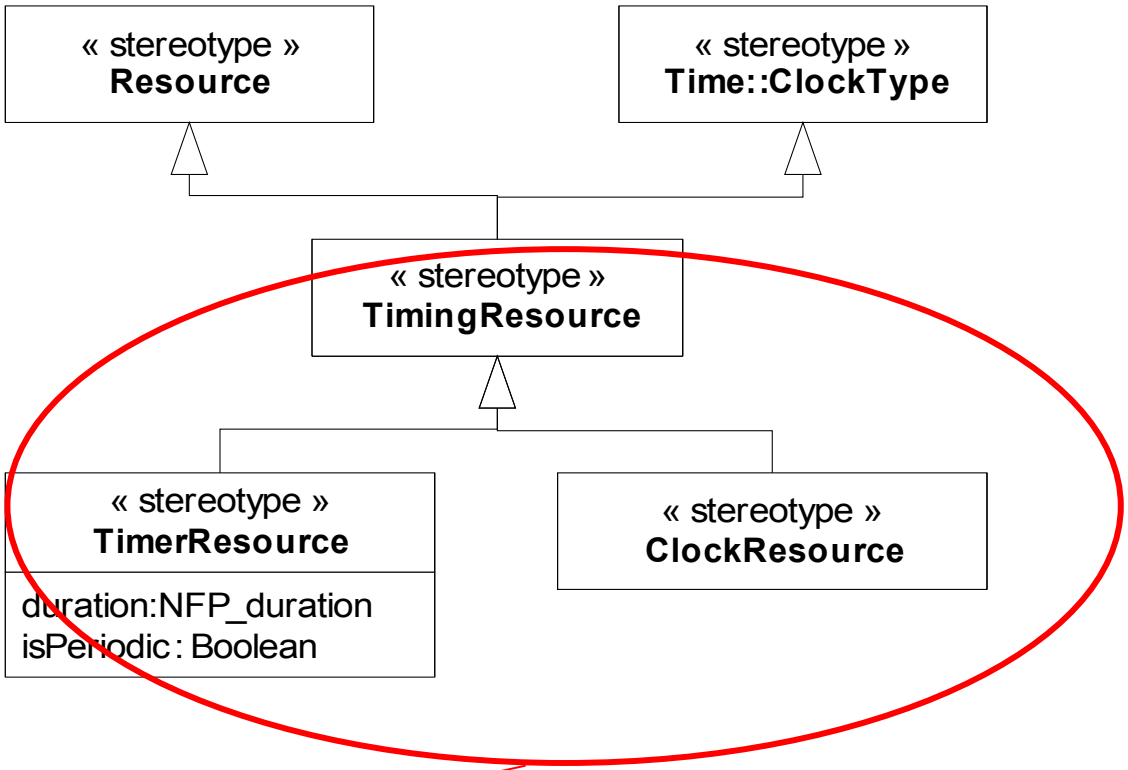


TimedConstraint & ClockConstraint



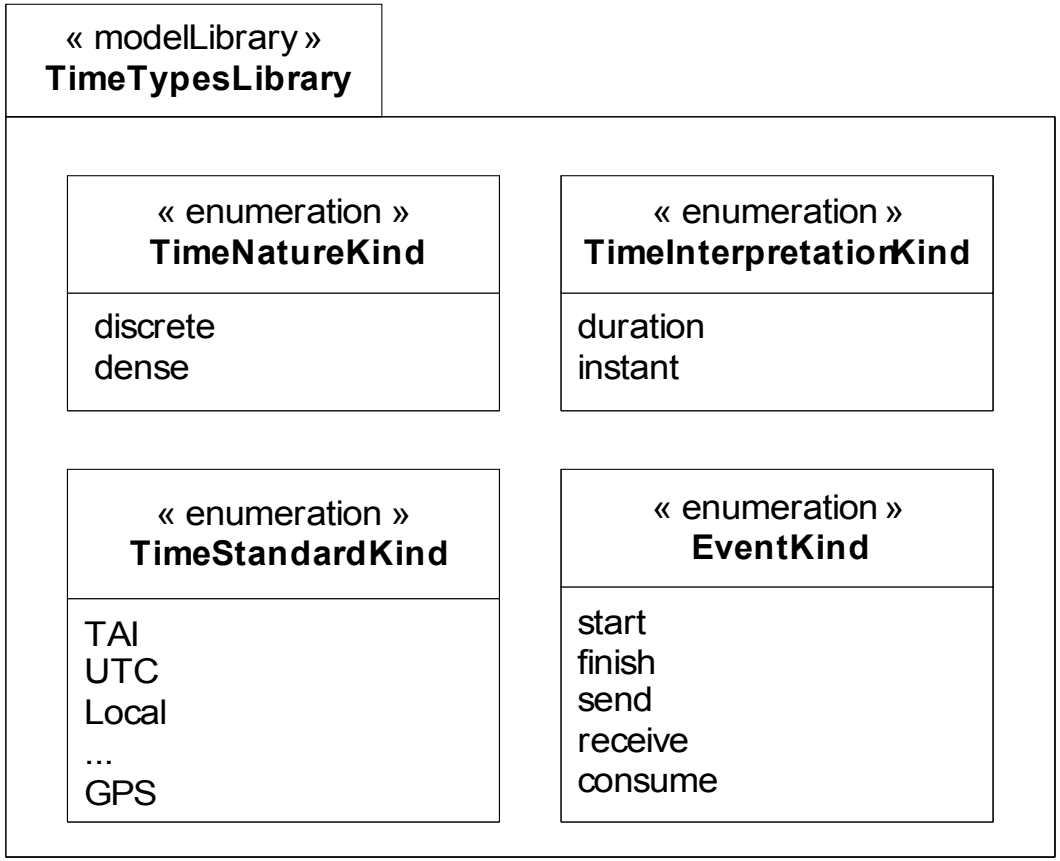
Time-related GRM stereotypes

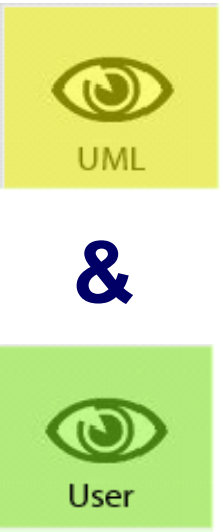
Sterotypes defined in the Generic Resource Modeling sub-profile



Resources for time management

Time-related libraries: TimeTypesLibrary





Time-related libraries: TimeLibrary

Two usual sets of Time Units

« modelLibrary »
TimeLibrary

« enumeration »
TimeUnitKind

«unit» s
«unit» ms {baseUnit=s, convFactor=0.001}
«unit» us {baseUnit=ms, convFactor=0.001}
«unit» ns {baseUnit=us, convFactor=0.001}
«unit» min {baseUnit=s, convFactor=60}
«unit» hrs {baseUnit=min, convFactor=60}
«unit» dys {baseUnit=hrs, convFactor=24}
…

« enumeration »
LogicalTimeUnitKind

<<unit>> tick

« clock »
{ unit = s }
idealClk:IdealClock

« primitive »
ClockedValueSpecification

TUK

« tupleType »
TimedValueType

value: Real
expr: ClockedValueSpecification
unit: TUK
onClock: String

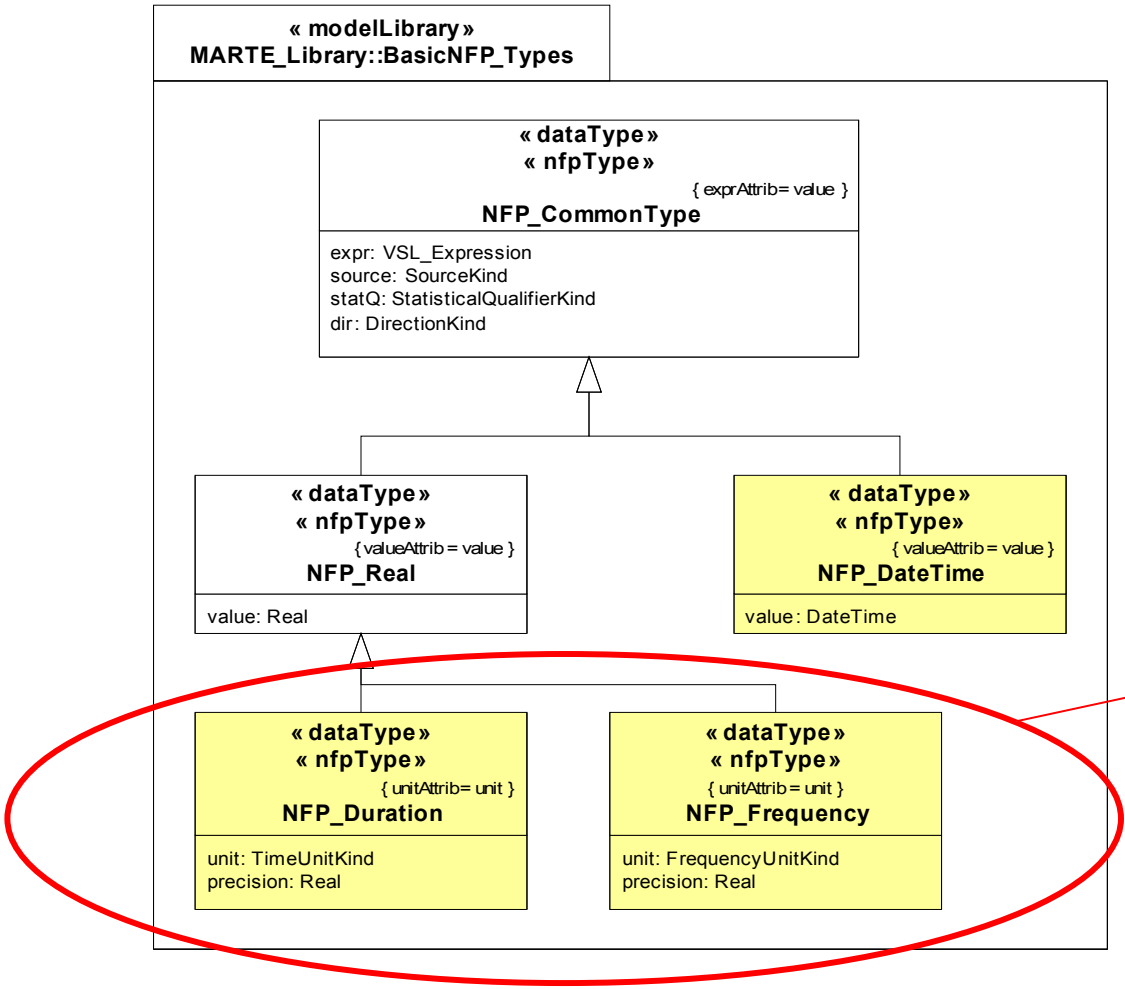
<<clockType>>
{ nature = dense, unitType = TimeUnitKind,
getTime = currentTime }
IdealClock

currentTime( ): Real

Model of ideal "physical time"

Templated DataType

Time-related NFP types

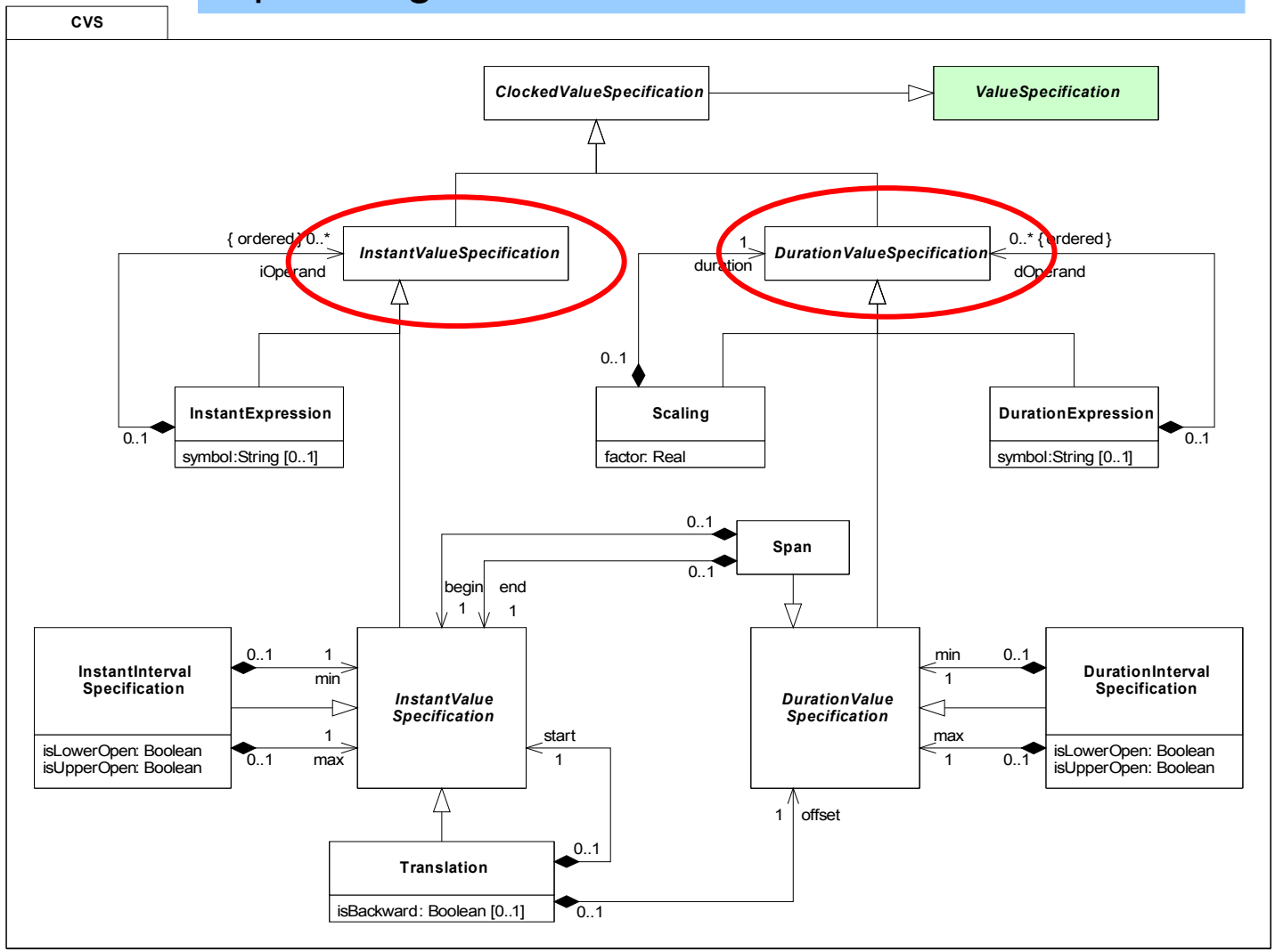


User

Time-related types.
Often used.

Time specific languages:Clocked Value Specification

Expressing time values with EXPLICIT clocks

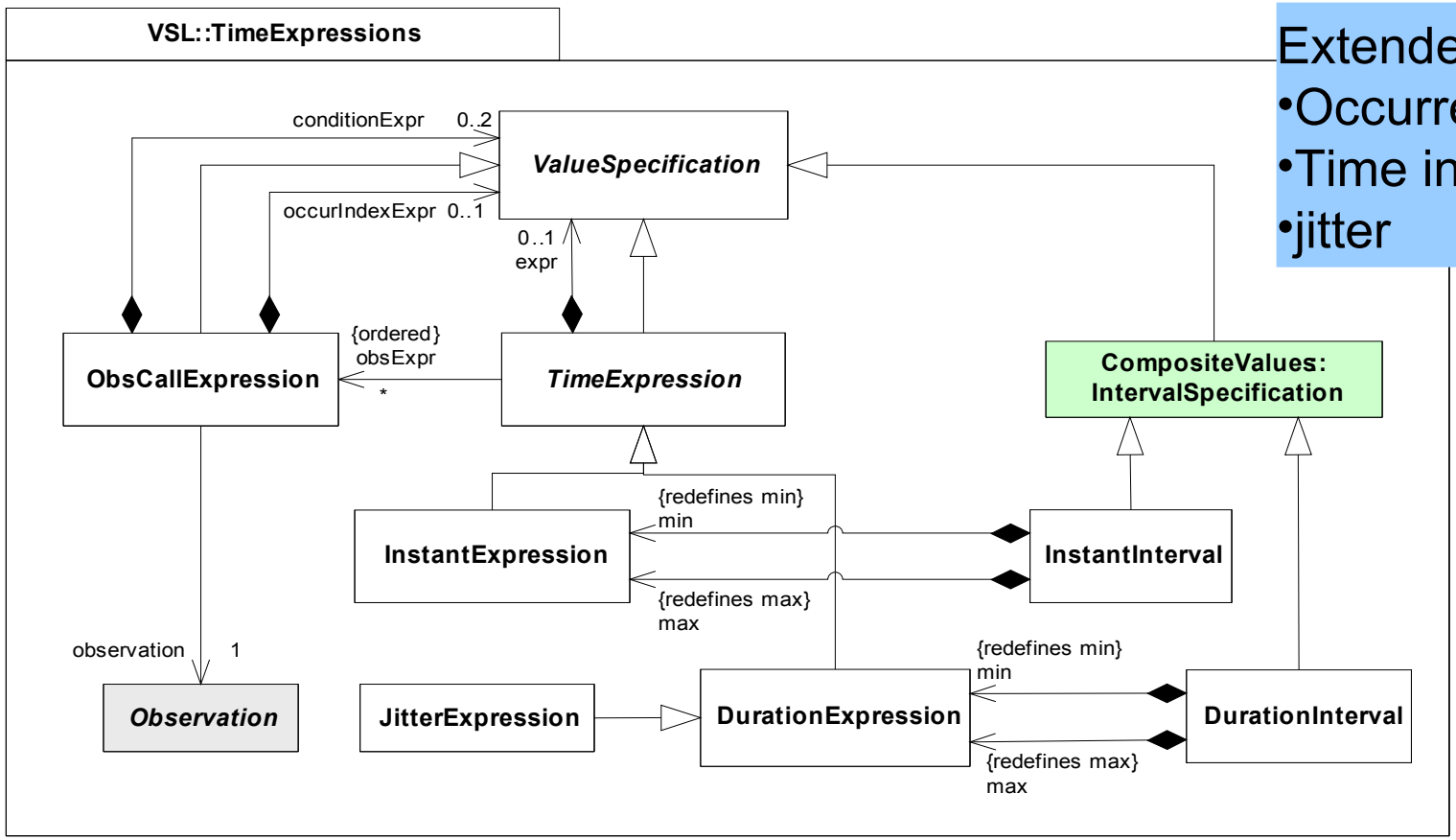


Domain

Instant ≠ Duration

Time specific languages:VSL Time Expressions

Expressing time values with EXPLICIT clocks



Extended capabilities:
- Occurrence index
- Time intervals
- jitter

Domain

Time Values: Concrete syntax

Examples of Clocked value expressions

- **Simple time values**
  (value=3.5, unit=ms, onClock='idealClk');
  3.5 ms on idealClk;

tuple, *a la* VSL

short form

- **Homogemeous expressions**
  (value=1.5, unit=ms, onClock='idealClk') +
  (value=150, unit=us, onClock='idealClk');
  → (value=1650, unit=us, onClock='idealClk')

Can be evaluated, because convFactor between units

- **Heterogeneous expressions**
  min (15 tick on prClk, 5 ms on idealClk);

Clock relation between prClk and idealClk must be provided

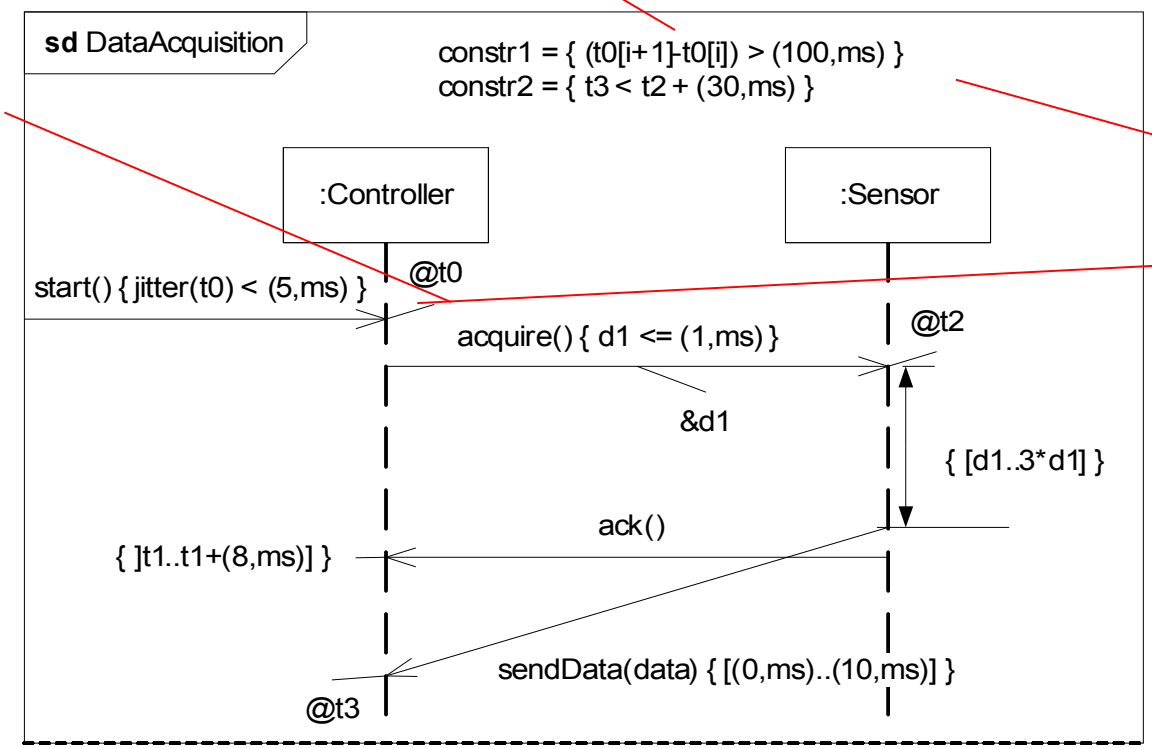
- **Additional capabilities with VSL**
  - Occurrence number, jitter,…
  - but implicitly on idealClk

Time specific languages:VSL Time Constraints

t0[i] denotes the i-th occurrence of

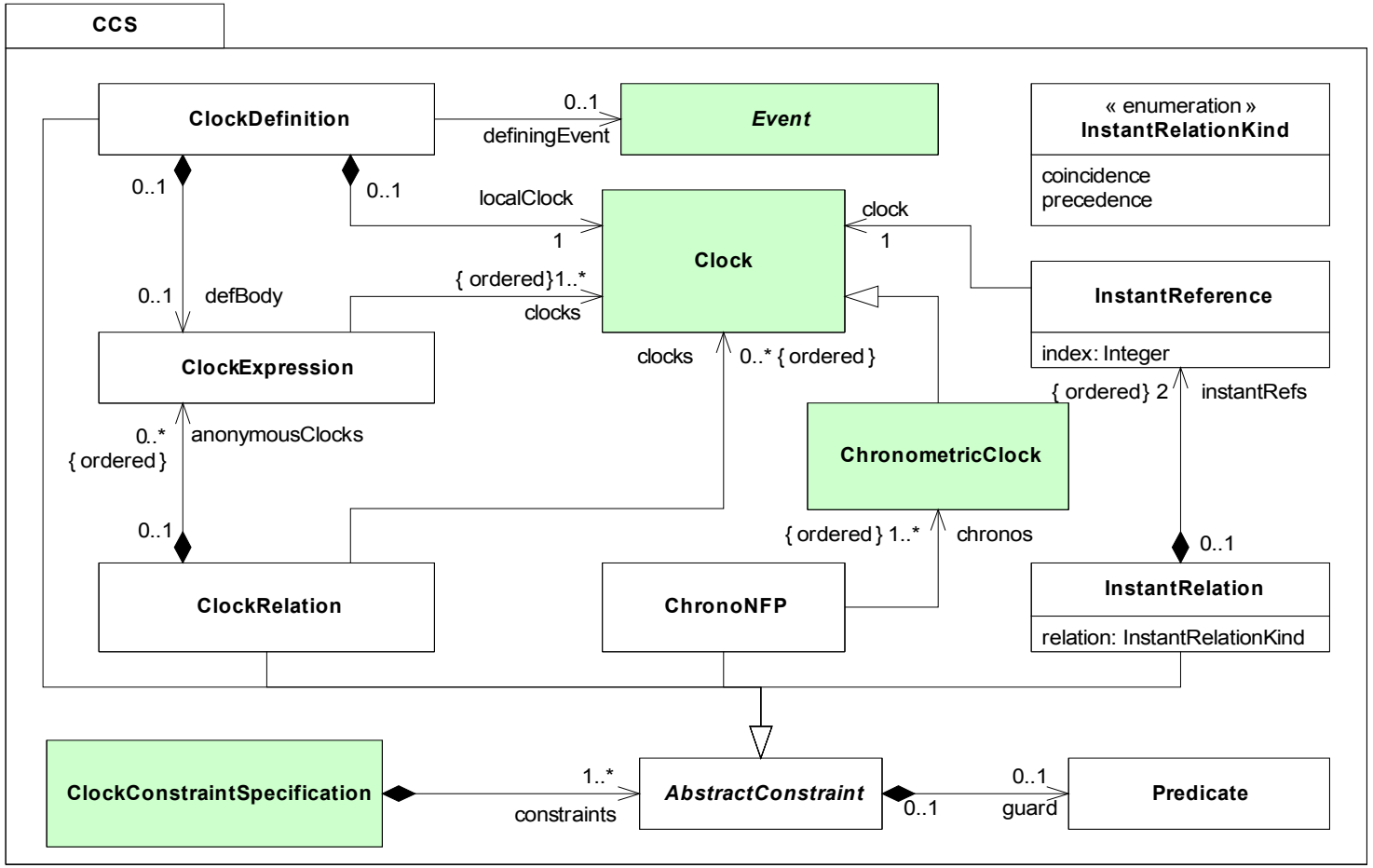
t0: observation of the message: start

t0 is periodic, period 100ms with a jitter less than 5ms



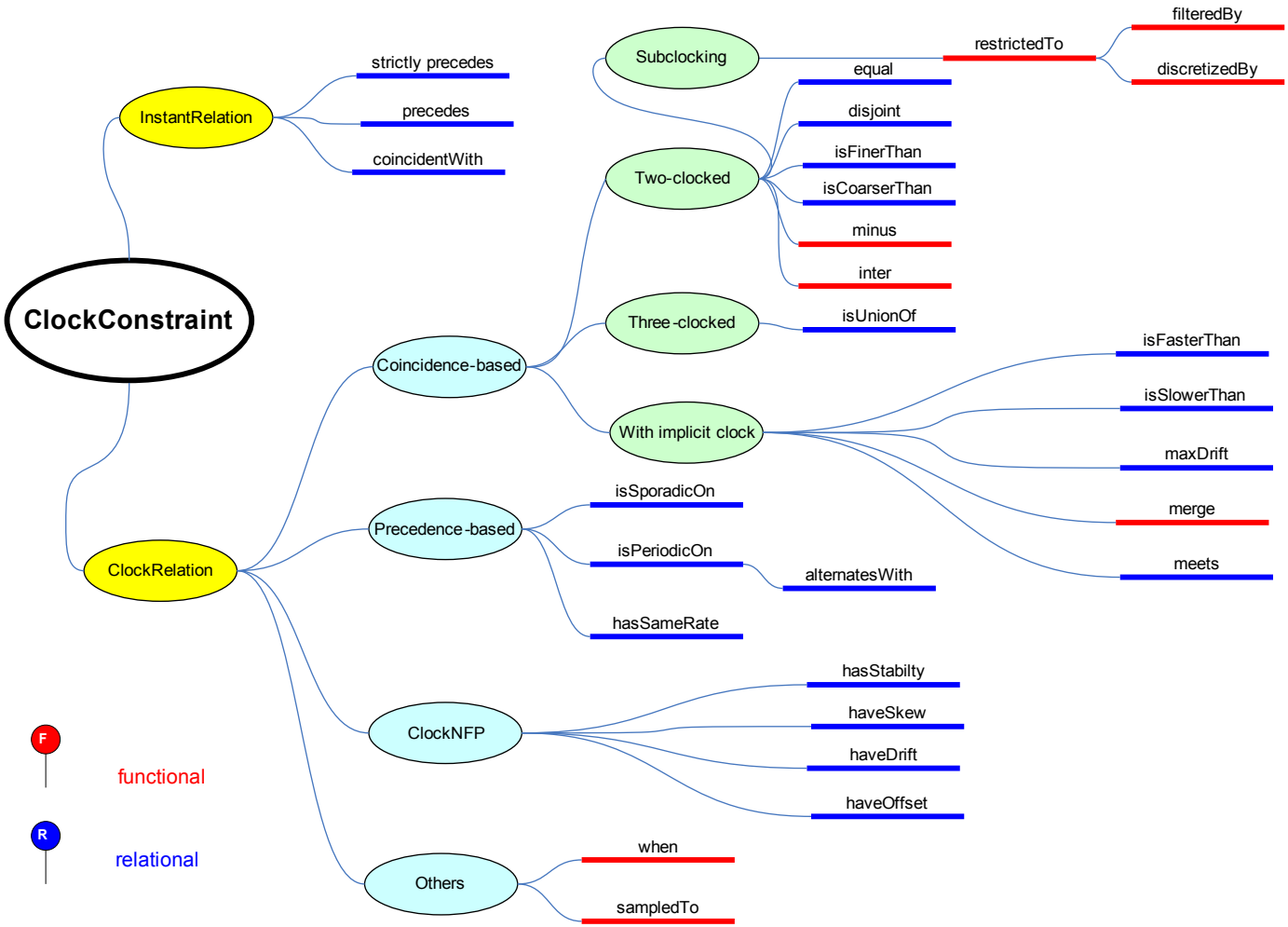
Time specific languages:Clock Constraint Specification

Domain



Expression of Clock dependencies

Clock Constraint Specification



Pre-defined Clock Constraints

Each relation has a mathematical specification

Overview

- SPT, UML 2 and Time
  - UML::CommonBehaviors::SimpleTime
- the MARTE Time domain view
  - a.k.a. the MARTE Time meta-model
  - Concepts and relationships
- the MARTE Time sub-profile
  - a.k.a. UML view
- **Usage of the Time sub-profile**

Chronometric Clocks (1)

How to specify chronometric clocks

« clockType »
{ nature = discrete, unitType = TimeUnitKind, resolAttr=resolution, getTime = currentTime }
Chronometric

resolution: Real {readOnly}

currentTime( ): Real

« clockType »
{ nature = dense, unitType = TimeUnitKind, getTime = currentTime }
IdealClock

currentTime( ): Real

Imported from MARTE::TimeLibrary

An user's defined ClockType

Chronometric Clocks (2)

Specifying NFP of (non ideal) chronometric clocks

« TimedDomain »
ApplicationTimeDomain

« clock »
{ unit = s, standard = UTC }
cc1:Chronometric
resolution = 0.01

« clock »
{ unit = s, standard = UTC }
cc2:Chronometric
resolution = 0.01

« clock »
{ unit = s }
idealClk:IdealClock

« clockConstraint » { kind = required }
{ **Clock** c **is** idealClk **discretizedBy** 0.001;
cc1 **isPeriodicOn** c **period** 10;
cc2 **isPeriodicOn** c **period** 10;
cc1 **hasStability** 1E-4;
cc2 **hasStability** 1E-4;
cc1,cc2 **haveOffset** [0..5] ms **wrt** idealClk;
}

Two instances

c: local ideal discrete clock – 1kHz

Exists d such that for all k:
$c[d+10*(k-1)] < cc1[k] \leq c[d+10*k]$
$\Rightarrow 0 < cc1[k+1]-cc1[k] < 20$ **ms**

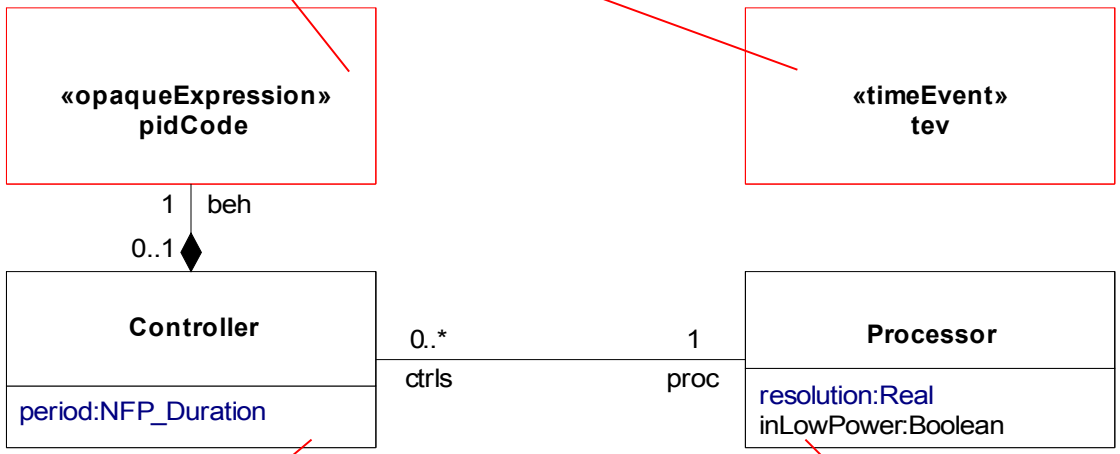
A Non-functional property:**Stability**
$10-0.001 \leq cc1[k+1]-cc1[k] \leq 10+0.001$ in **ms**

Another Non-functional property:**Offset**

Logical Clocks (1/4)

How to specify logical clocks:
1) Start with a standard UML class diagram

Explicit model elements not usual in Class Diagrams



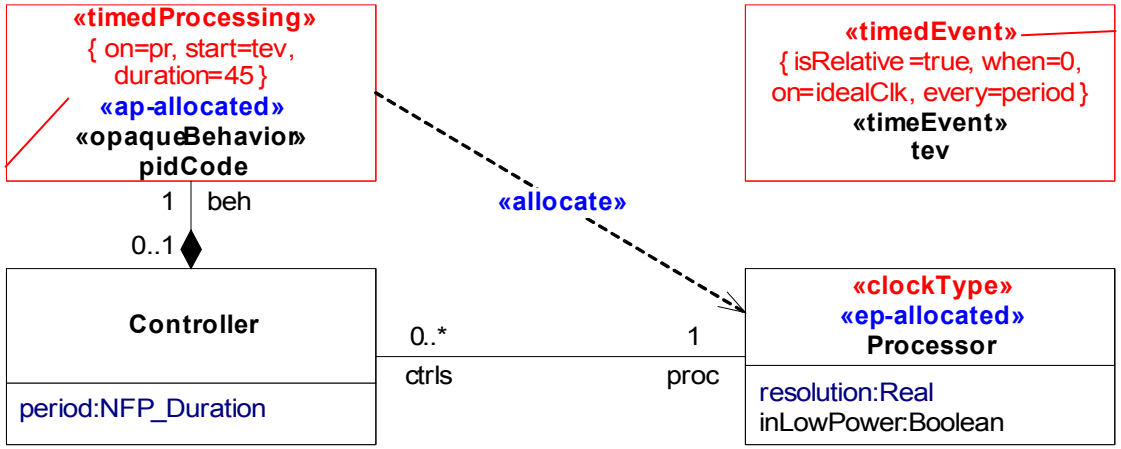
Period of the PID controller: uses a NFP-type

A Voltage-Scaling processor. Assume 2 frequencies for simplicity

Logical Clocks (2/4)

2) Apply MARTE stereotypes

User



The pid code is triggered by tev and takes 45 cycles of Processor

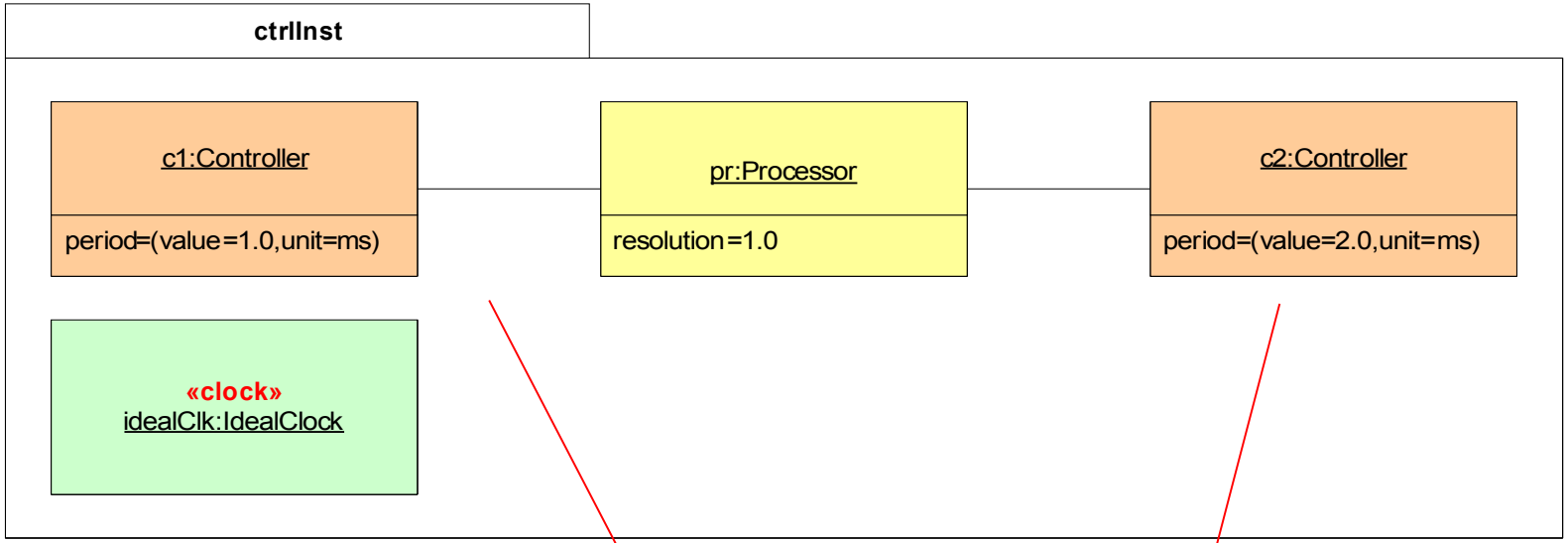
Event tev is periodic on idealClock, the period is the value of the controller's attribute

The class Processor is stereotyped by ClockType

Logical Clocks (3/4)

3) Instantiate user's model elements

An instance of the system with an instance of Processor supporting two instances of Controller



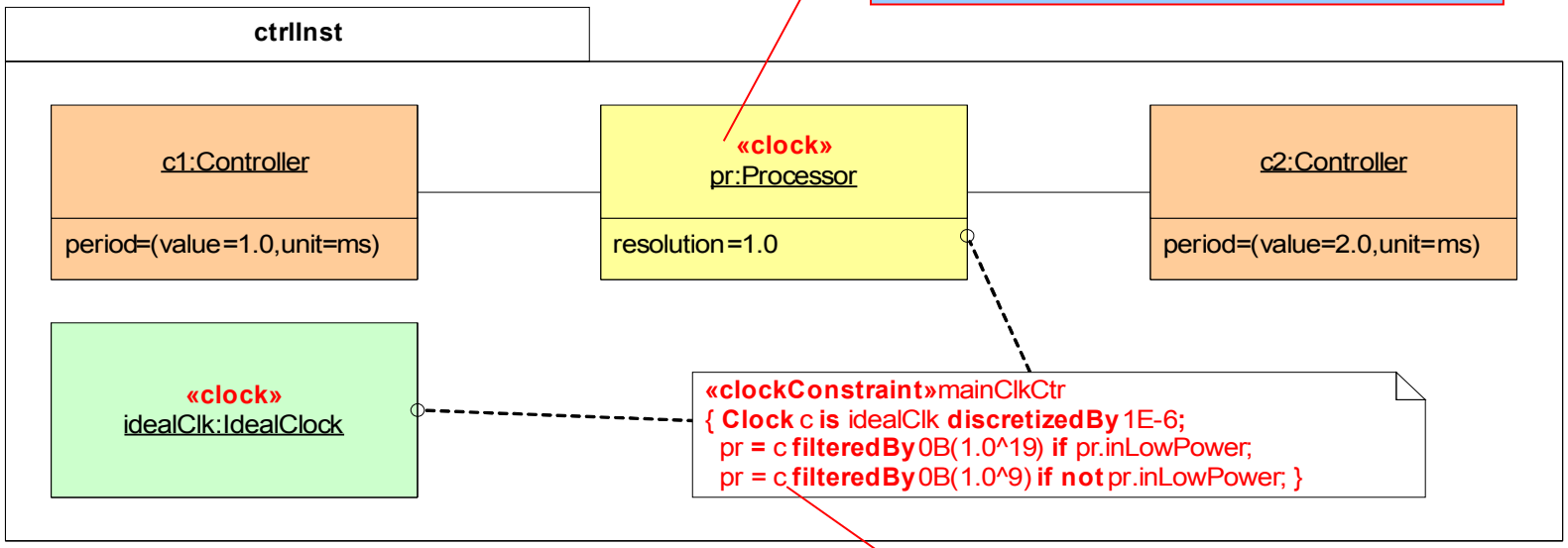
Each controller instance has its own period

Logical Clocks (4/4)

4) Introduce clock (by stereotyping)

User

This instance of Processor is used as a logical clock



This clock constraint binds processor clock cycle to physical time, taking account of the power mode

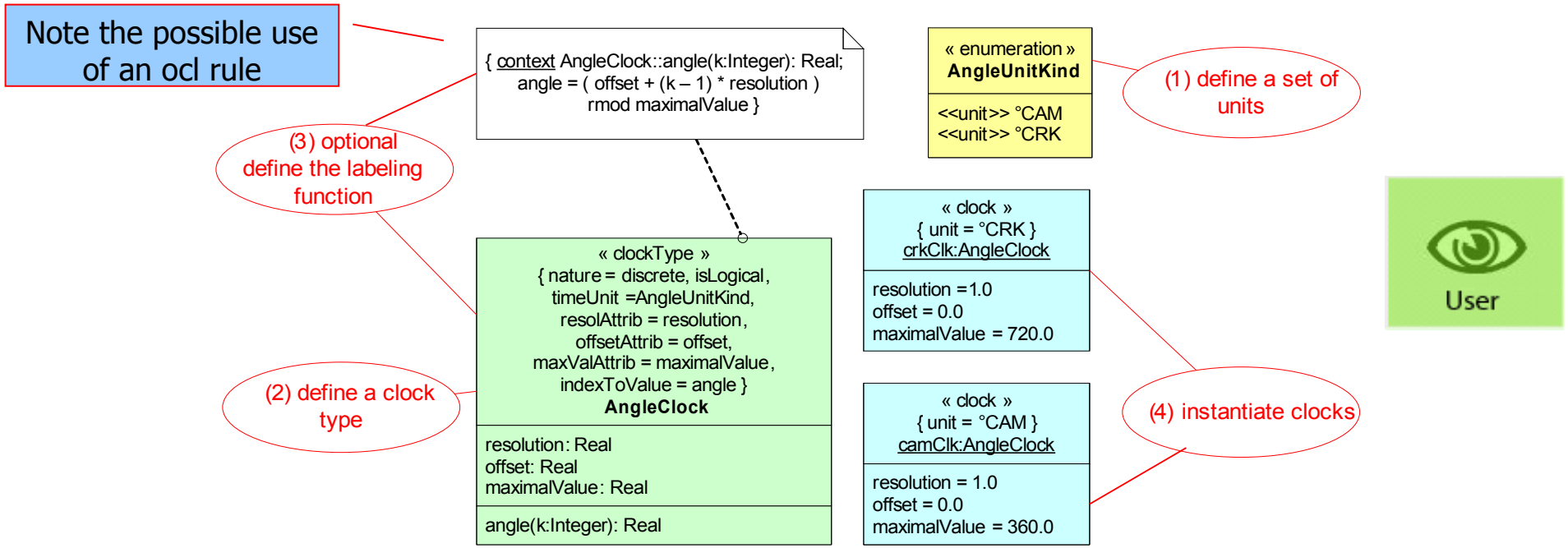
Multiform time (1/4)

Another example of logical clocks

Automotive application

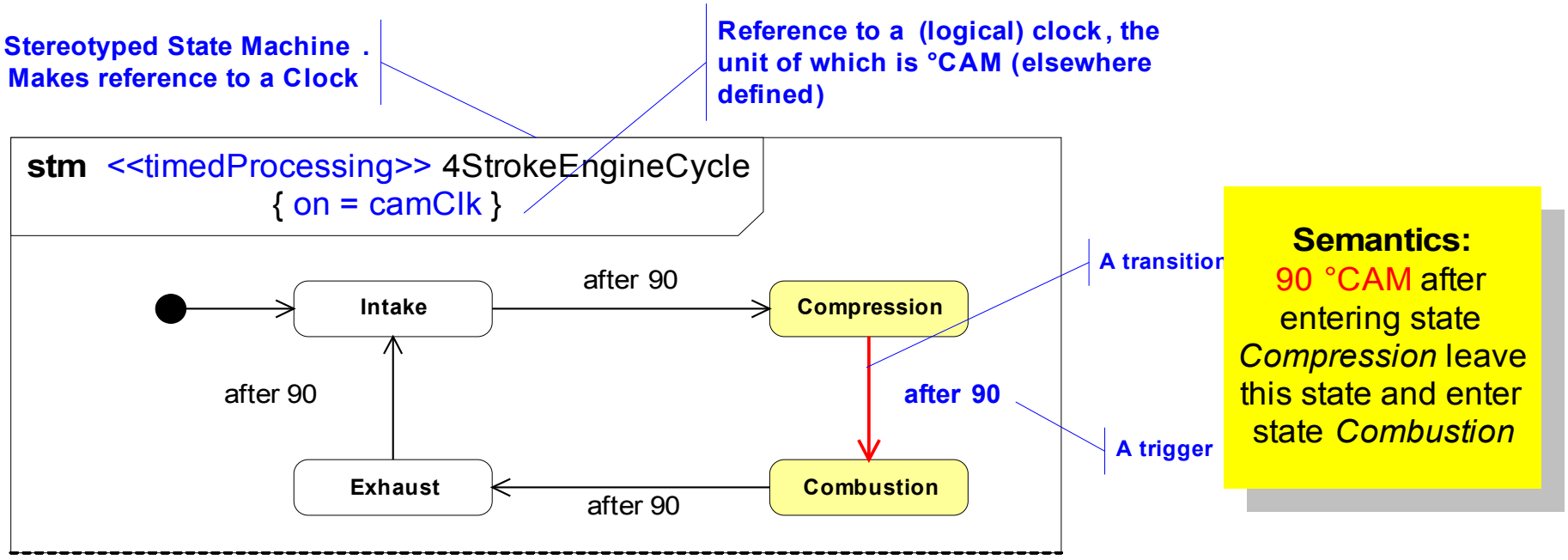
For ignition and injection, the position of the camshaft or the crankshaft is a "natural" reference frame for events and behaviors.

=> Define logical clocks dealing with angular positions.



Multiform time (2/4)

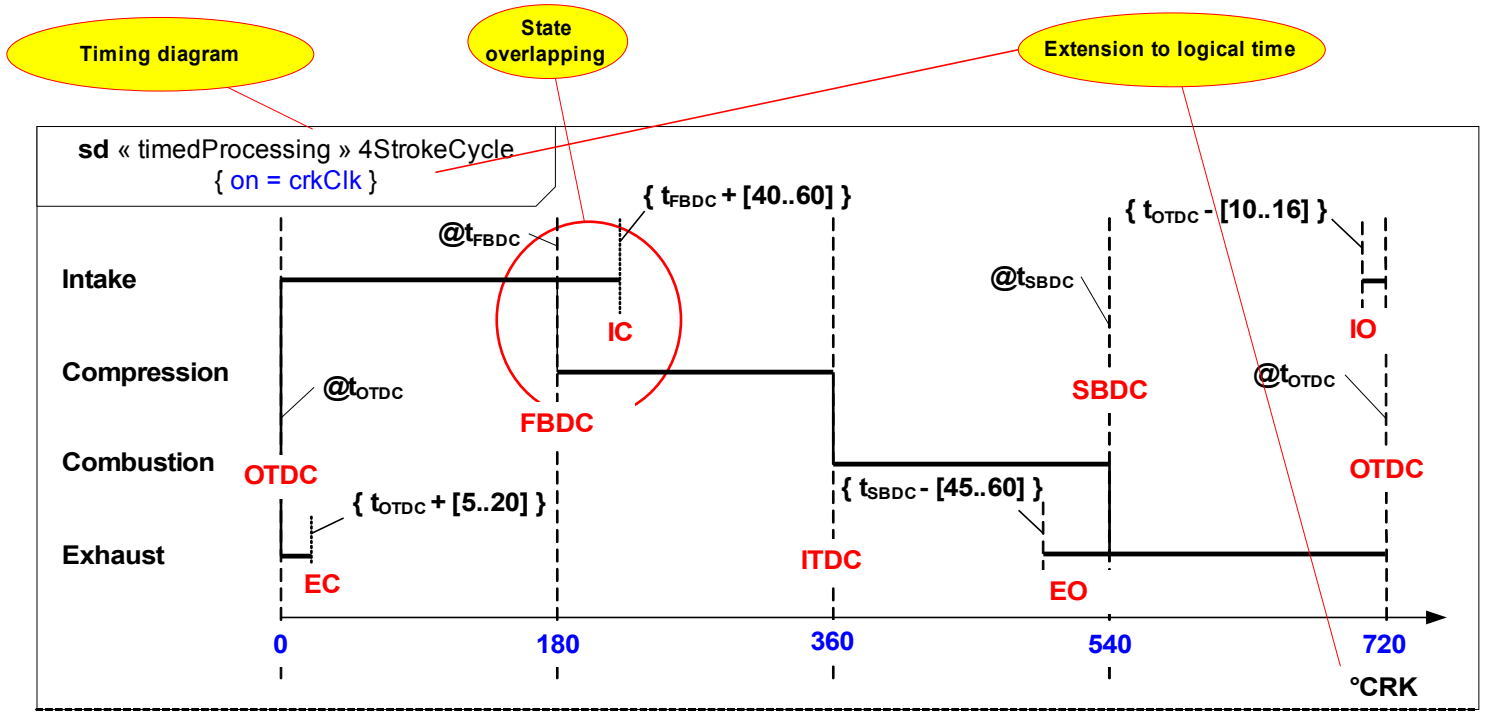
Example of usage of an “AngleClock”



Multiform time (3/4)

Another example of usage of an “AngleClock”:
Enhanced timing diagram used in specification

User



crkClk: crankshaft Clock
°CRK: degree crank

TDC: Top Dead Center
ITDC: Ignition TDC
OTDC: Overlap TDC

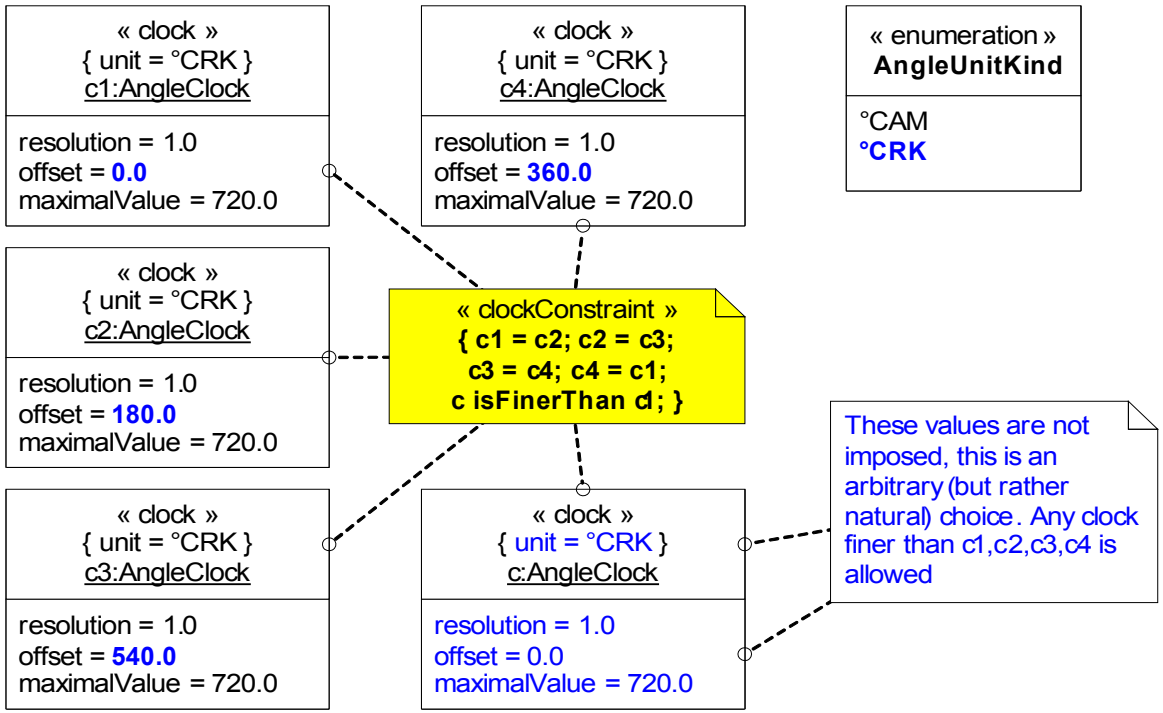
BDC: Bottom Dead Center
FBDC: First BDC
SBDC: Second BDC

IC: Intake closes
IO: Intake opens
EC: Exhaust closes
EO: Exhaust opens

Multiform time (4/4)

Combining logical clocks:
ck is an AngleClock used to specify the ignition of a cylinder
c is the clock used to specify ignitions in a 4-cylinder engine

User



Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- Part 3
  - The timing model
- Part 4
  - **A component model for RT/E**
- Part 5
  - Platform modeling
- Part 6
  - Repetitive structure modeling
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

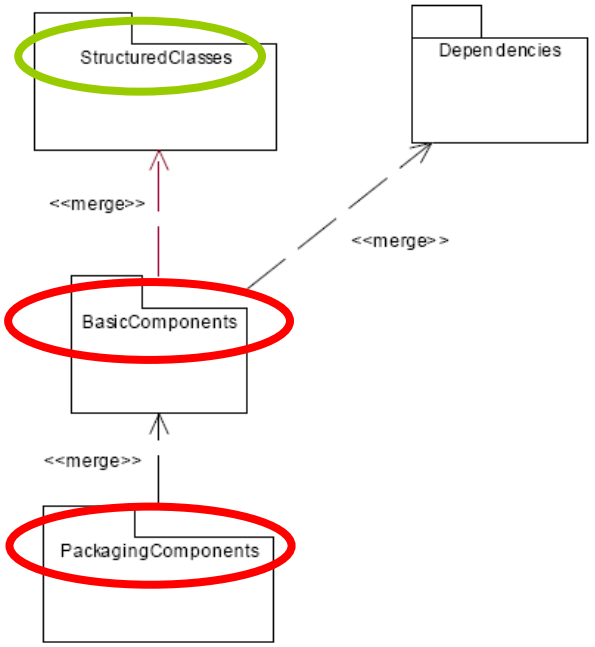
Component-based paradigms in the RTE domain

- **Component architectures are increasingly used in RTE execution platforms**
  - Need for manageable and reusable pieces of software
  - Key examples: Lightweight-CCM, SCA, Autosar
- **Concept of component also used to structure System / Software engineering processes**
  - Entities under analysis/design broken down into a series of components
  - Applicable at different stages of the process
  - Different kind: active vs. passive (e.g., UML active classes)
  - Examples of related languages: SysML, AADL

There is a need to provide modeling constructs to support these concepts at different levels of abstraction

What is a component in UML?

- **UML distinguishes the notions of structured class and component**
  - The kernel of the language defines *Class* and *Interface*
  - *StructuredClasses* defines *Port* and *Connector* and provide the ability to describe a *Class* as an assembly of parts
  - *Basic* and *PackagingComponent* define the notion of component realization and adds packaging capabilities

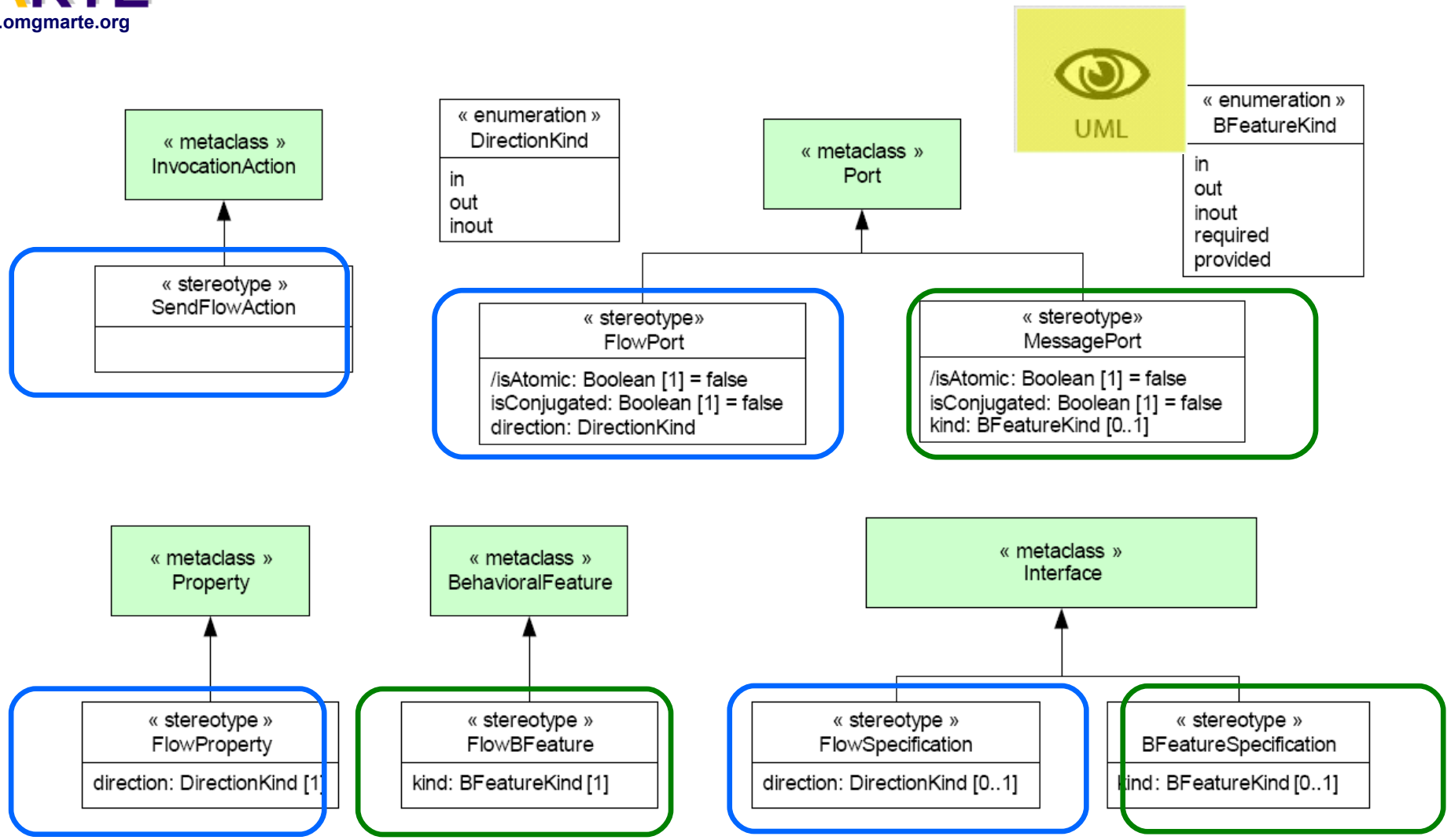


- In any case, no support for flow-oriented communications

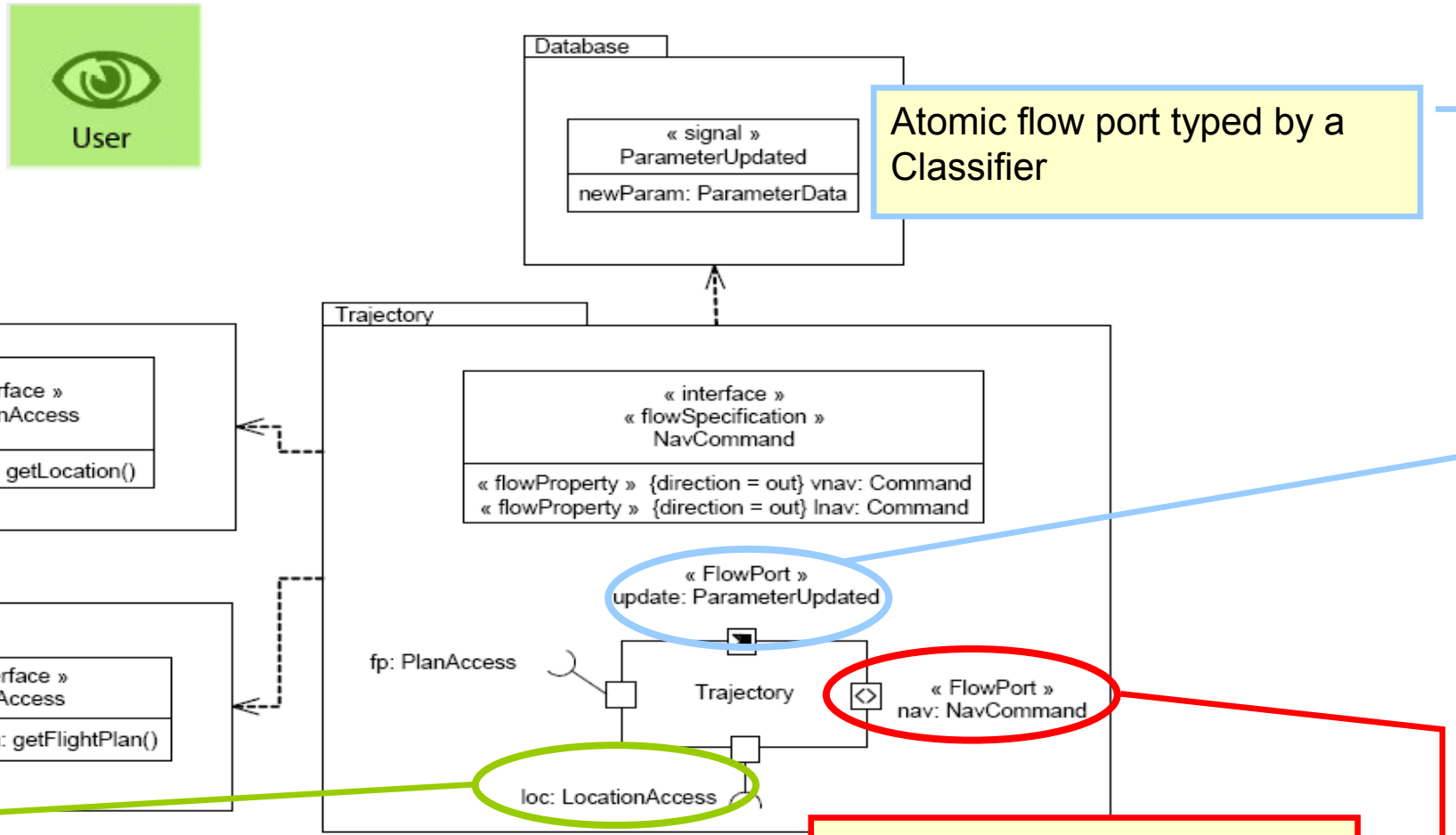
General Component Model

- **Introduced to cope with various component-based models**
  - SysML, Spirit, AADL, Lightweight-CCM, EAST-ADL2, Autosar
- **Does not imply any specific model of computation**
- **Relies mainly on UML structured classes, on top of which a support for SysML blocks has been added**
  - Atomic and non-atomic flow ports
  - Flow properties and flow specifications
- **But also providing a support for Lightweight-CCM, AADL and EAST-ADL2, Spirit and Autosar**

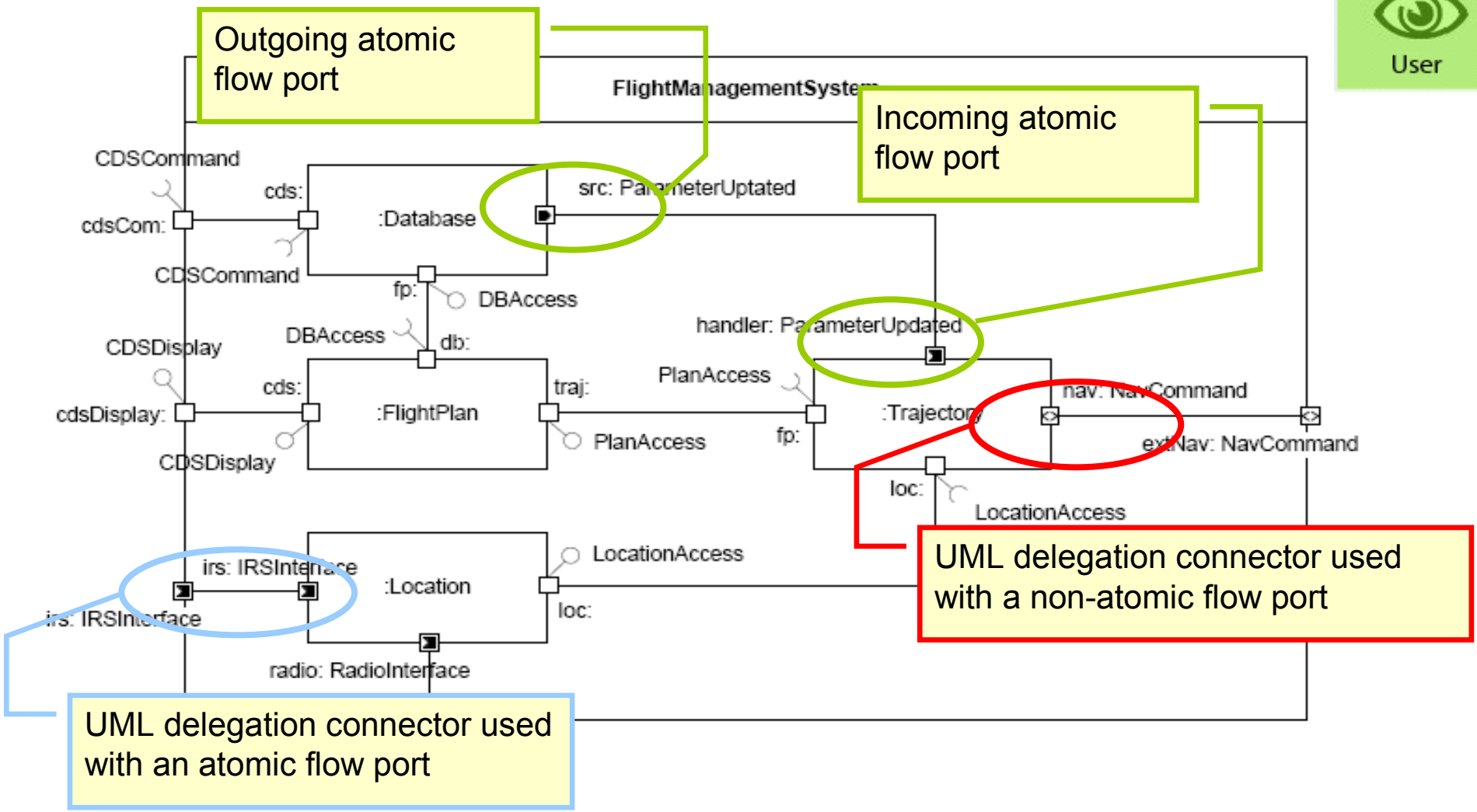
The MARTE GCM subprofile



Example of component definition



Example of component usage



RTE Model of Computation and Communication

- **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**
  - Currently, active objects (e.g. Rhapsody, Rose RT, ACCORD)
  - Alternative MoCC can be defined using the MARTE foundations

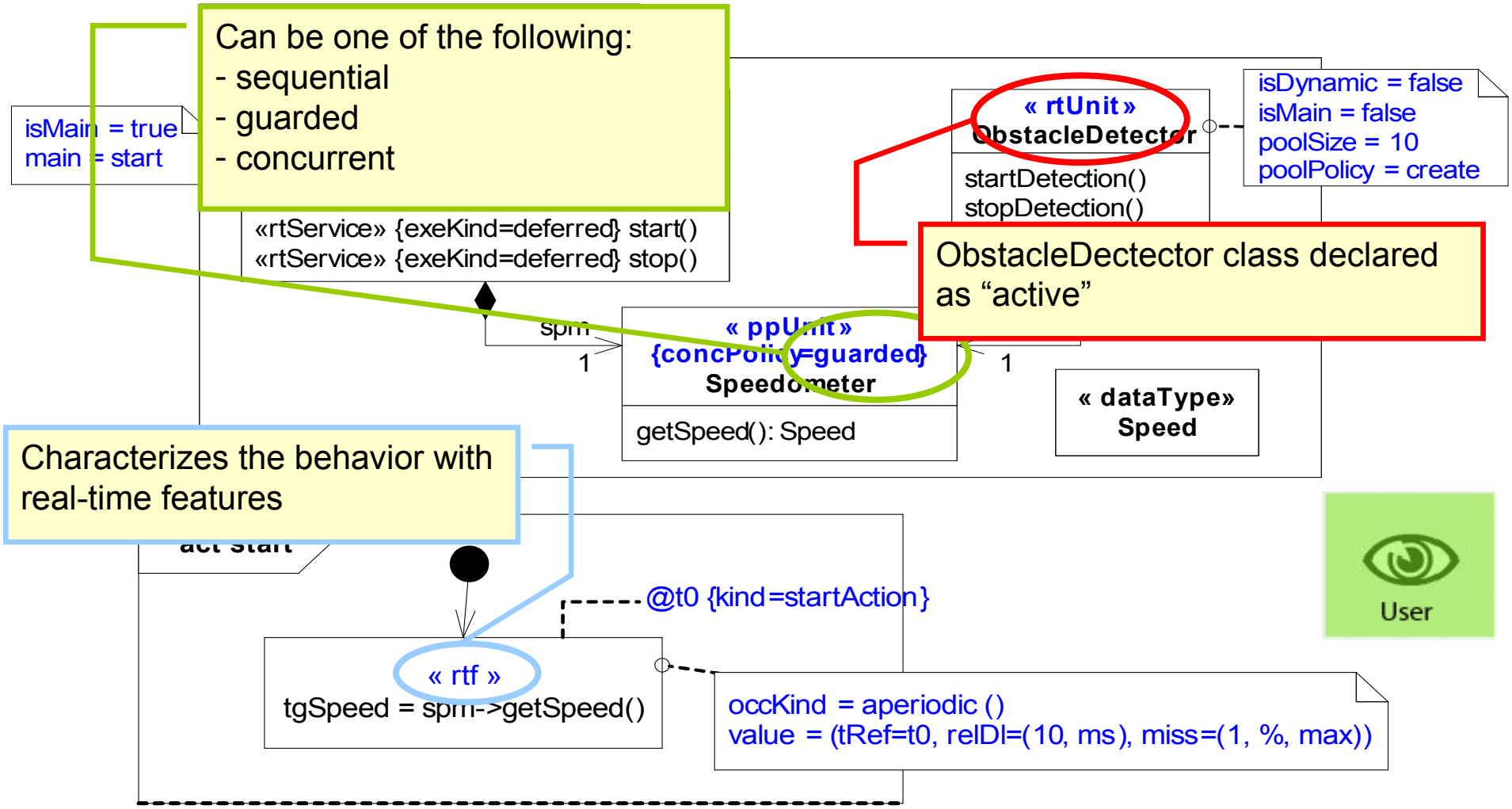
RTE Model of Computation and Communication

- **Provides high-level concepts for modeling qualitative real-time features on classes / structured classes / components**
  - Real-Time Unit (RTUnit)
    - Generalization of the Active Objects of the UML 2
    - Owns at least one schedulable resource
    - Resources are managed either statically (pool) or dynamically
    - May have operational mode description (similar to AADL modes)
  - Protected Passive Unit (PPUnit)
    - Generalization of the Passive Objects of the UML2
    - Requires schedulable resources to be executed
    - Supports different concurrency policies (e.g. sequential, guarded)
    - Policies are specified either locally or globally
    - Execution is either immediateRemote or deferred

RTE Model of Computation and Communication (cont'd)

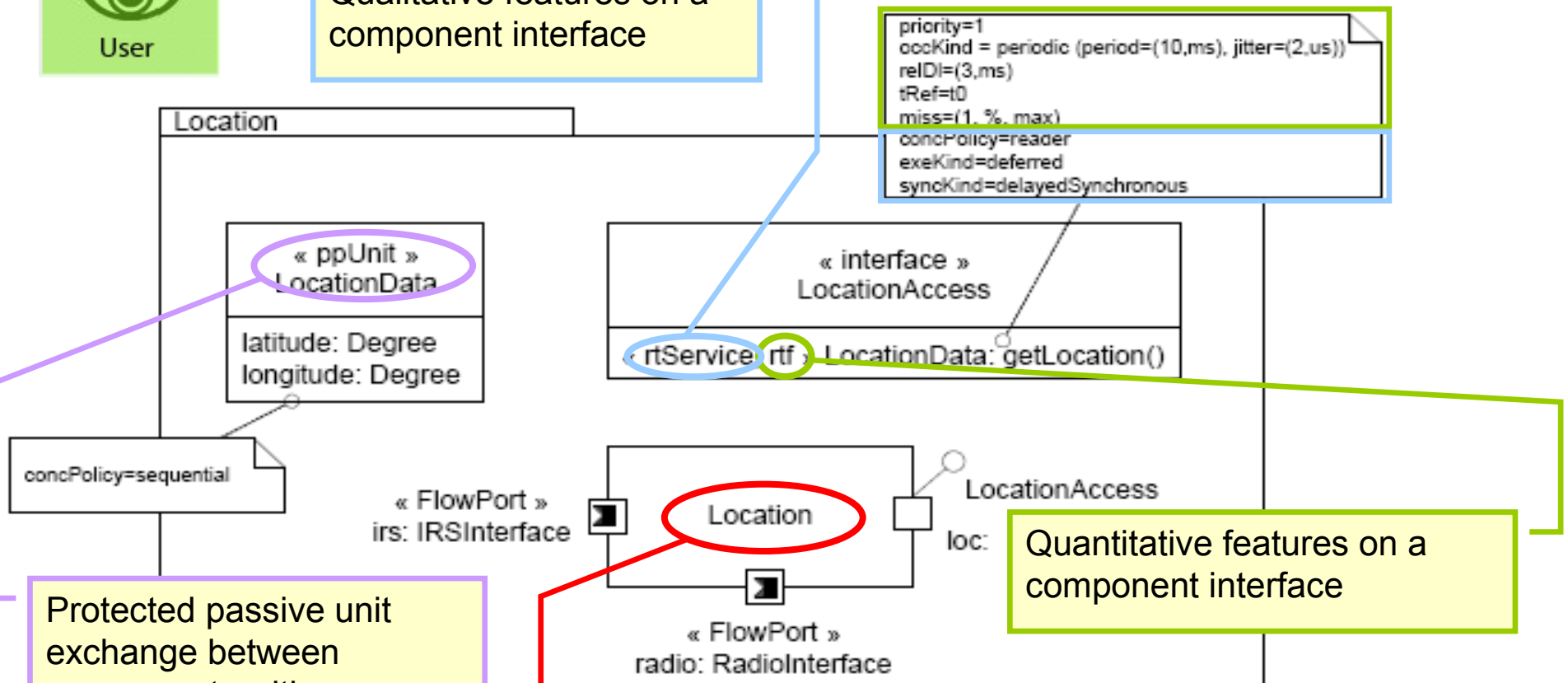
- **Provides high-level concepts for modeling quantitative real-time features on classes / structured classes / components**
  - Real-Time Behavior (RtBehavior)
    - Message Queue size and policy bound to a provided behavior
  - Real-Time Feature (RTF)
    - Extends UML Action, Message, Signal, BehavioralFeature
    - Relative/absolute/bound deadlines, ready time and miss ratio
  - Real-Time Connector (RteConnector)
    - Extends UML Connector
    - Throughput, transmission mode and max blocking/packet Tx time

Usage examples of the RTEMoCC extensions



Modeling real-time features of components

Qualitative features on a component interface

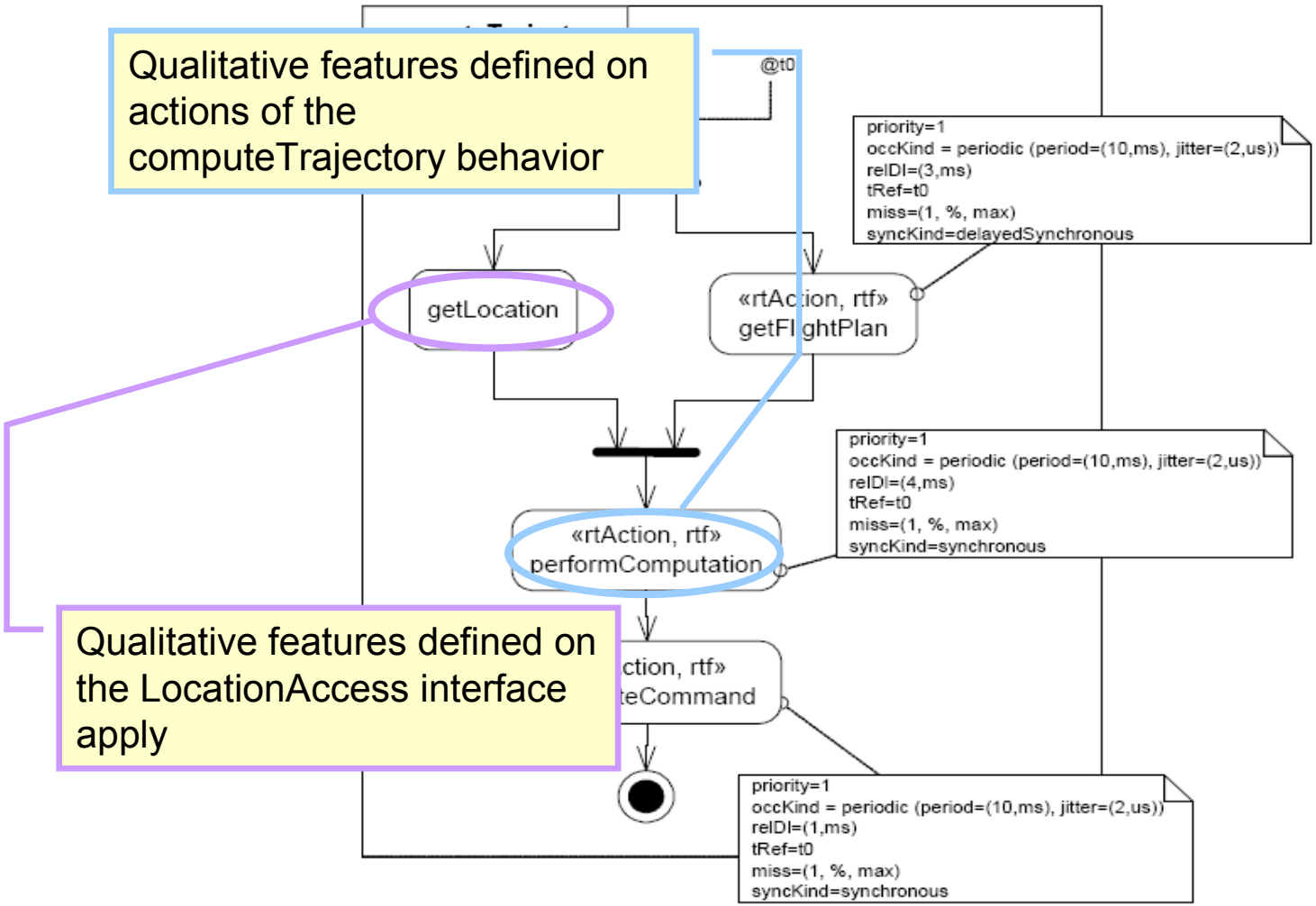


Quantitative features on a component interface

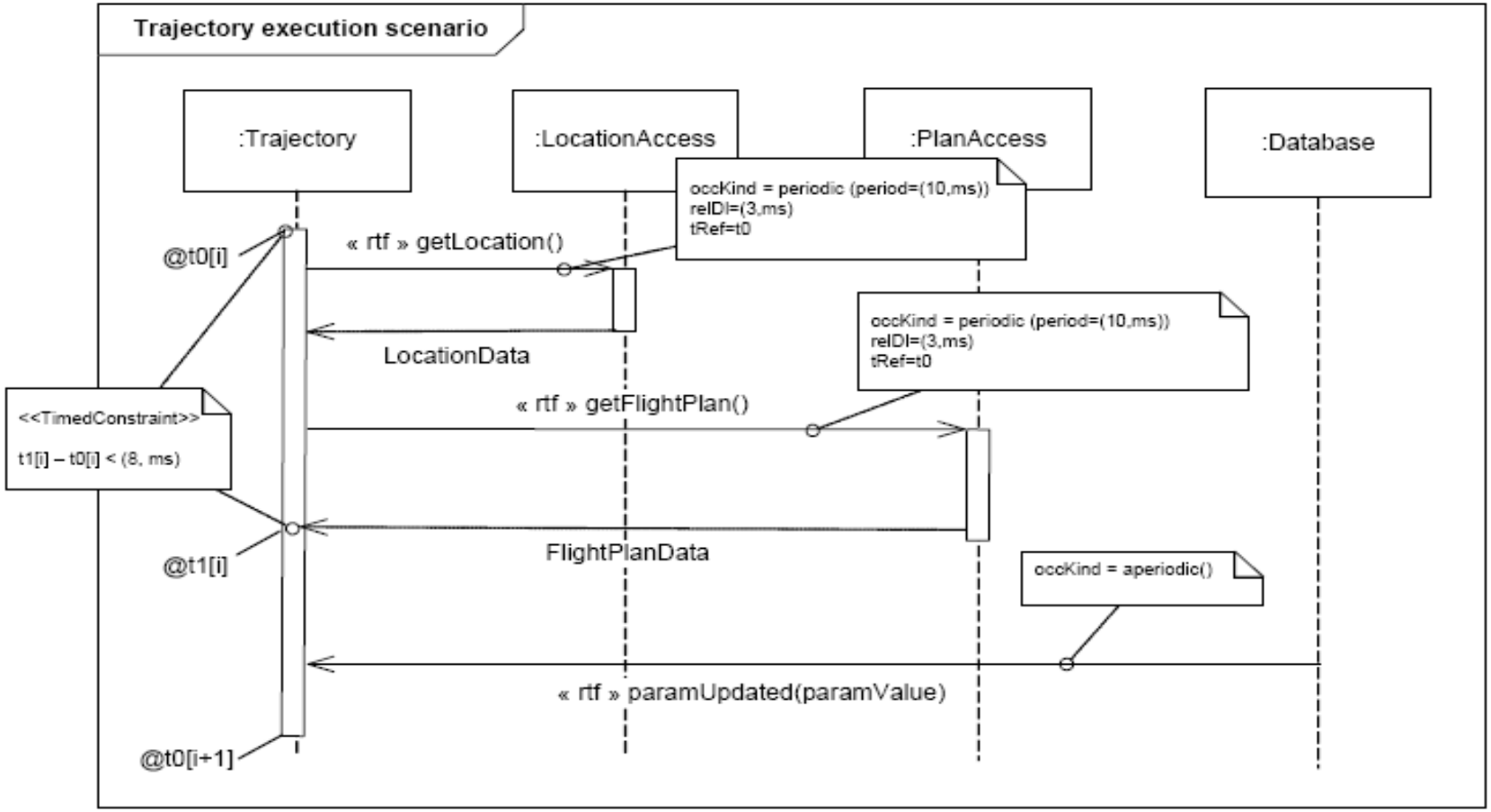
Protected passive unit exchange between components with a sequential access policy

Without a «rtUnit» stereotype, the component is considered as passive
It needs to be allocated on a computing resource (e.g., using the «allocate» stereotype)

Modeling real-time features of components (cont'd)



Modeling real-time features of components (cont'd)



Extensibility

- **All models of computation in the RTE domain not explicitly addressed by MARTE**
- **MARTE foundations (NFP, Time, GRM) allow third-parties to specify other model of computations that rely on the same semantic basis**
  - Allows one to use MARTE features along with this user-defined MoCC

Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- Part 3
  - The timing model
- Part 4
  - A component model for RT/E
- **Part 5**
  - **Platform modeling**
- Part 6
  - Repetitive structure modeling
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

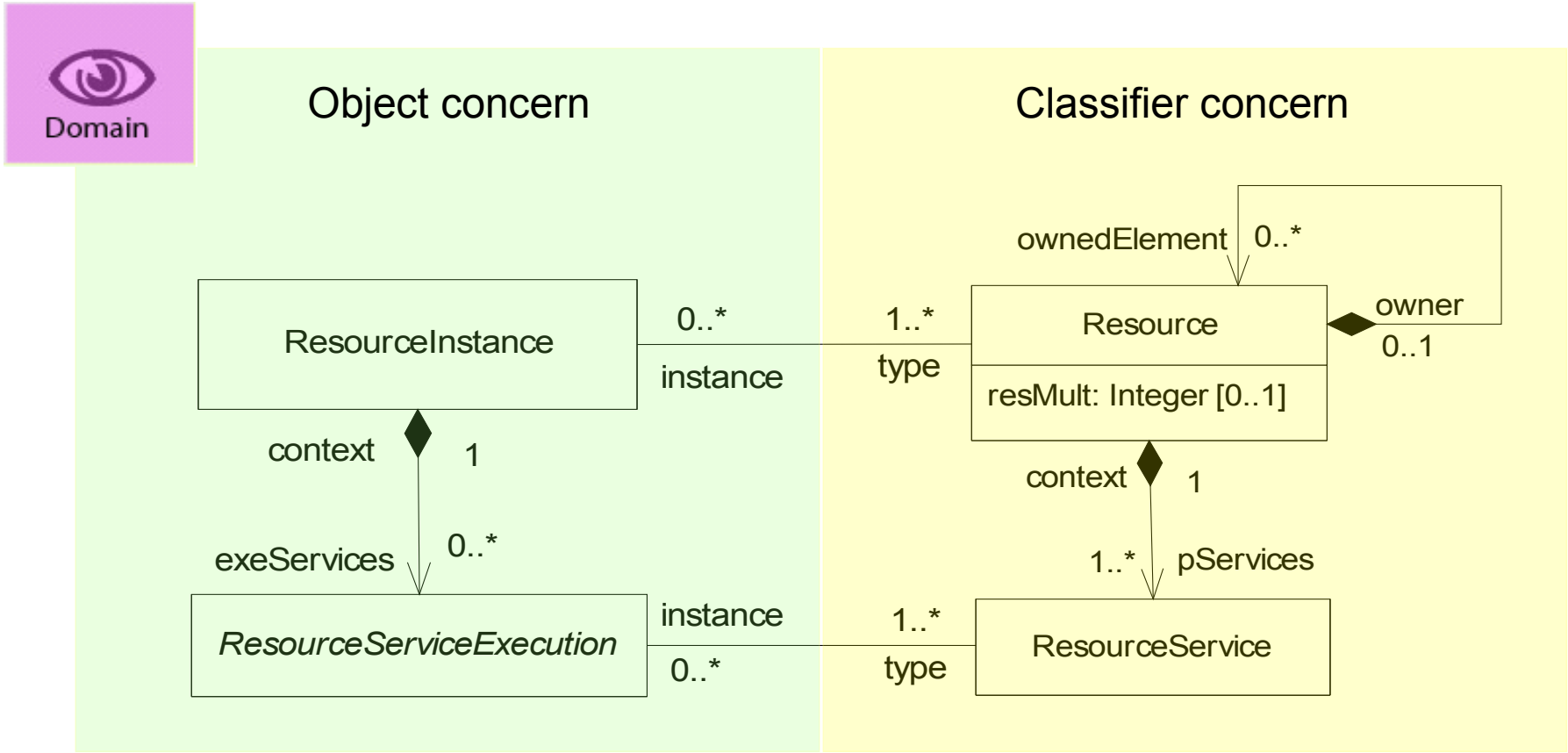
Outlines of the GRM package

- **Provides basic concepts for modeling a general (high-level) platform for processing RTE applications**

- **Includes the features for modeling processing platforms at different level of details.**
  - The level of granularity needed depends on the concern motivating the description of the platform
    - E.g., the type of the platform, the type of the application, or the type of analysis to be carried out on the model

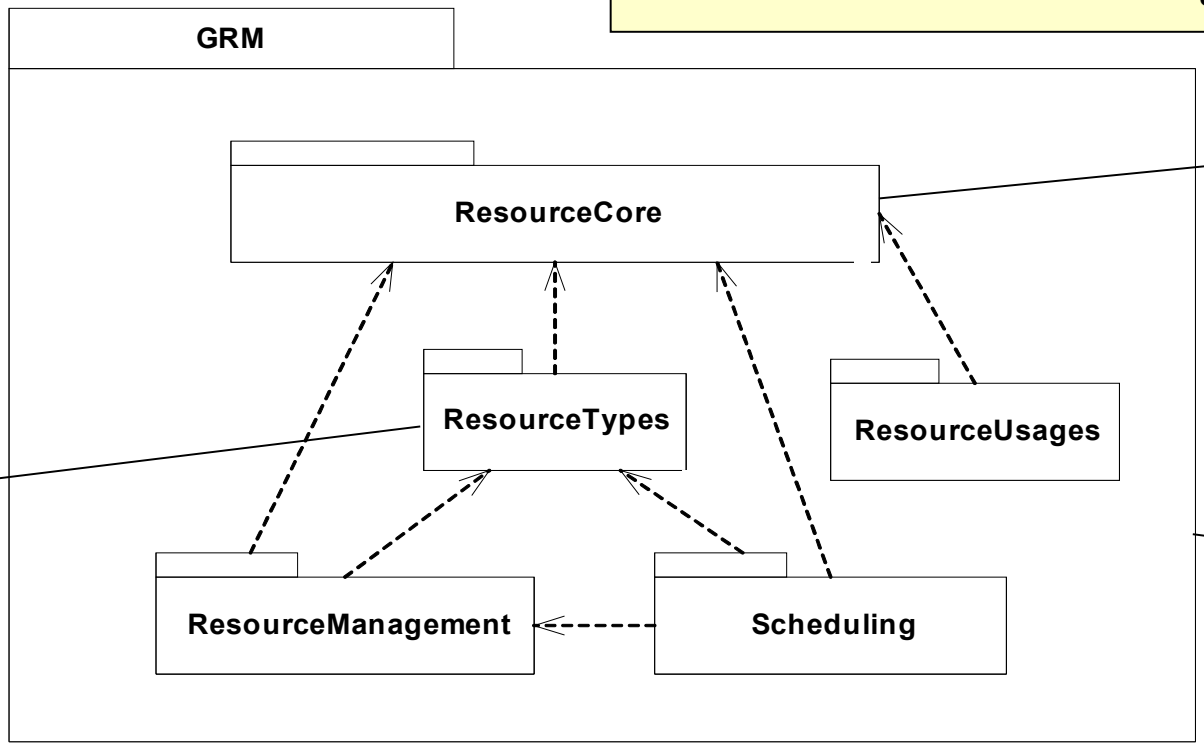
- **Build in a bottom-up process to abstract finer-level platforms**
  - Processing platform for design concern
    - See HRM and SRM
  - Processing platform for analysis concern
    - See GQAM-related ptf and further refinements for performance and schedulability analysis

Essence of the GRM Package



Generic Resource Modeling

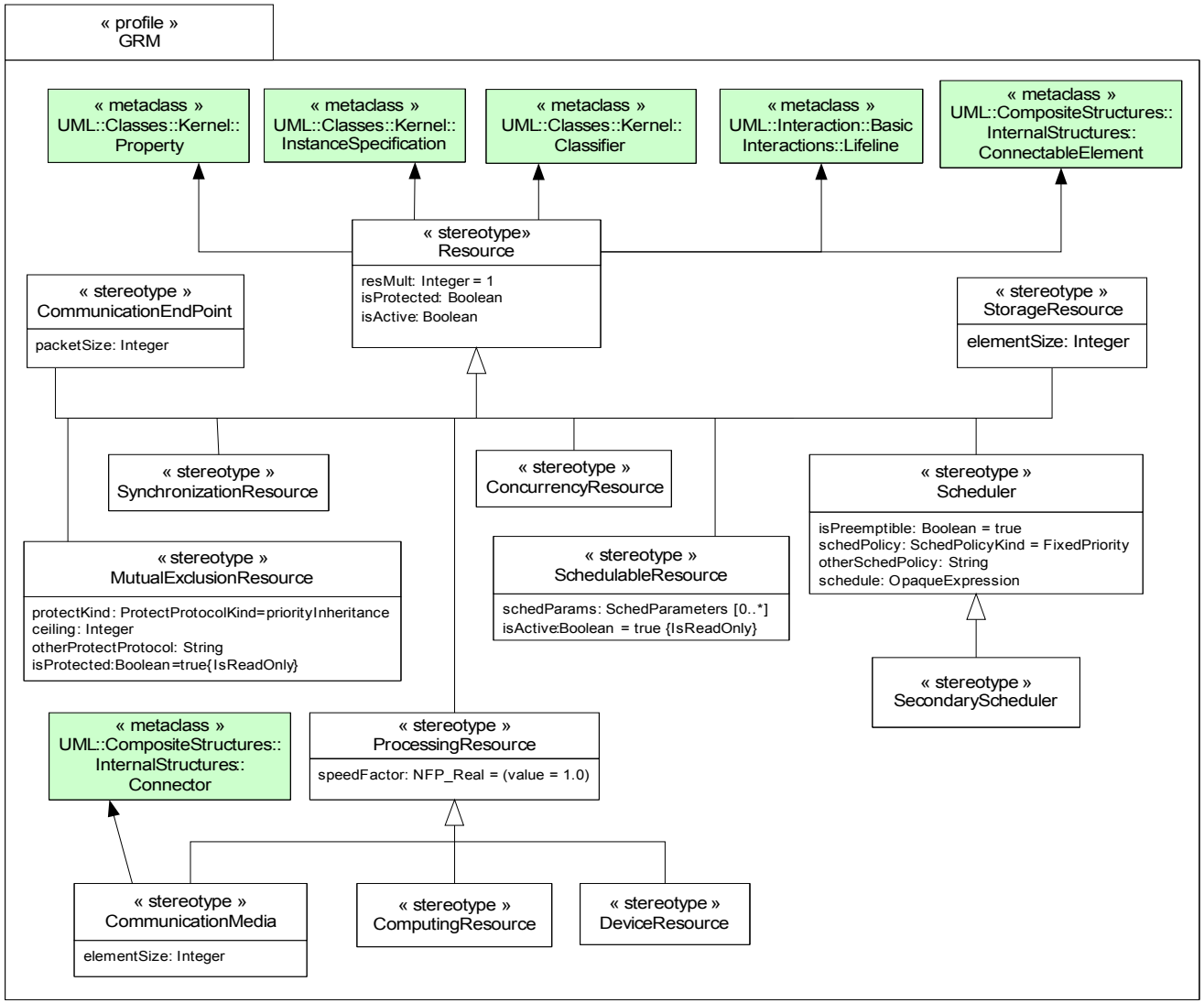
Resource offers Services and may have NFPs for its definition and usage



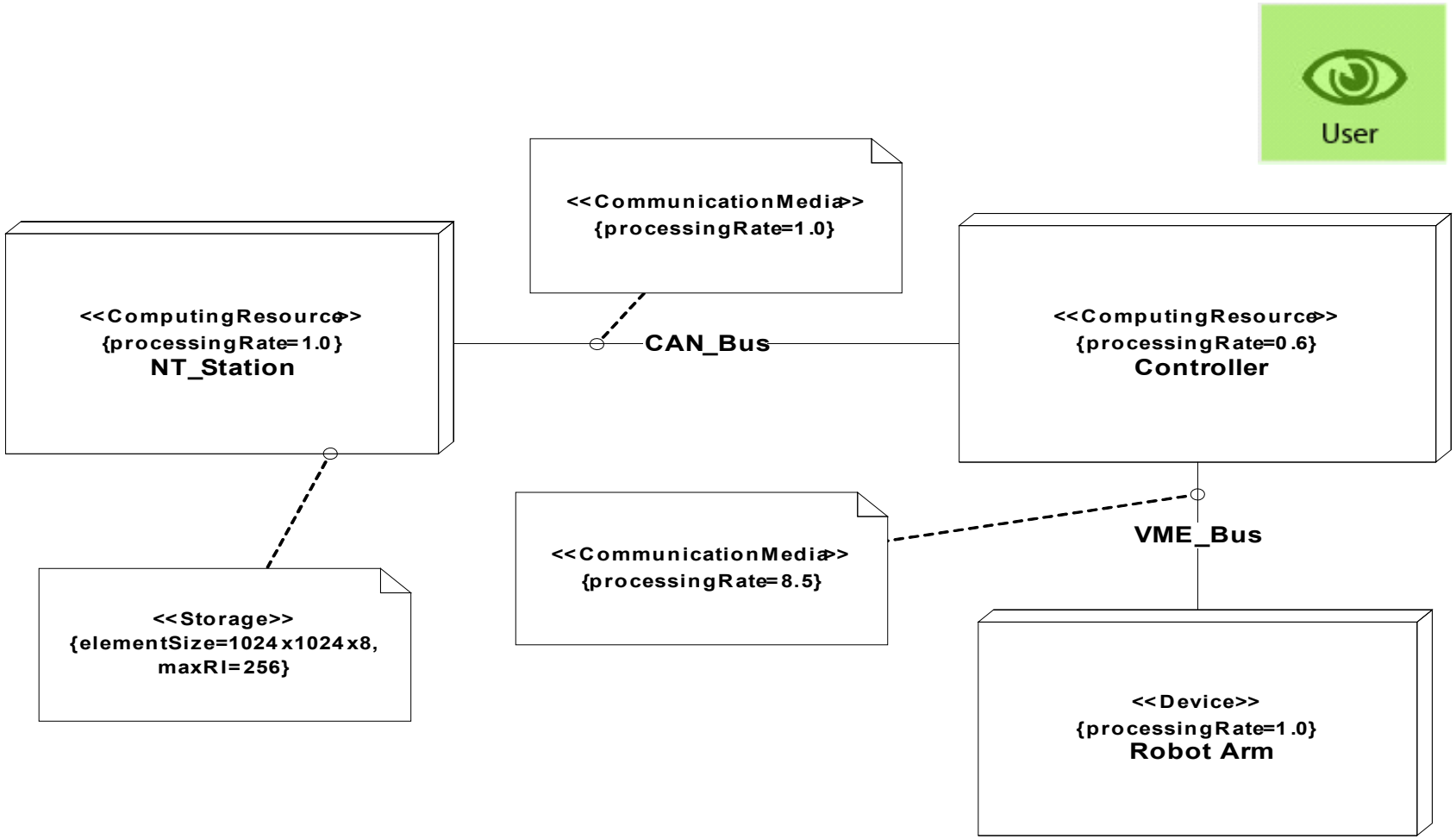
A rich categorization is provided: Storage, Synchronization, Concurrency, Communication, Timing, Computing, and Device Resources may be defined.

Shared resources, scheduling strategies and specific usages of resources (like memory consumption, computing time and energy) may be annotated.

Example of UML extensions for Generic Resources



Generic resource modeling example



Allocation & Refinement

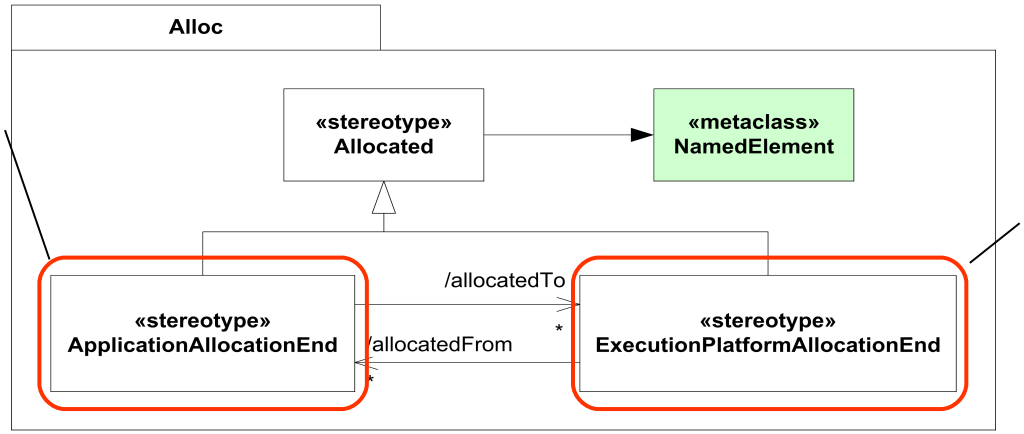
- **Basic ideas**
  - Allocate an application element to an processing platform element
  - Refine a general element into one or several more specific elements

- **Inspired by the SysML allocation**
  - Can only allocate application to execution platform
  - Can attach NFP constraints to the allocation

A two step process for allocation modeling

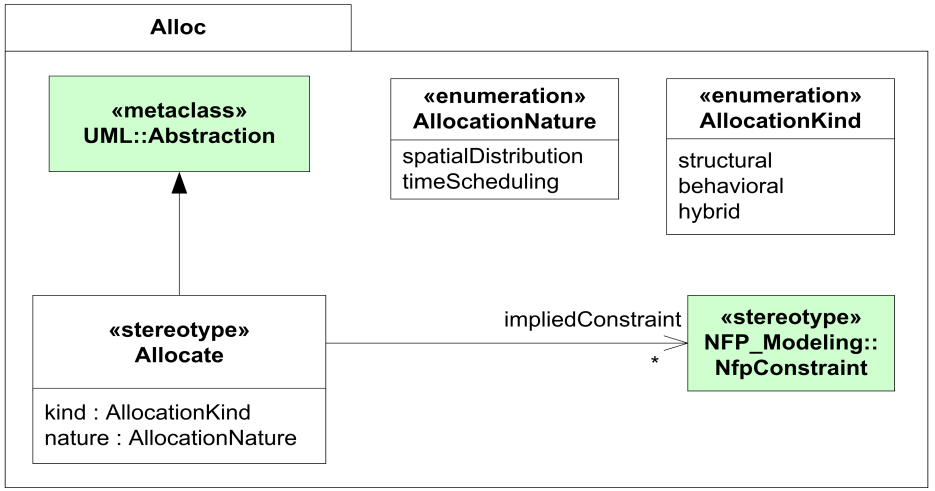
- **Identify possible sources and targets of allocations**

What can be allocated, the logical view:
➔ structure or behavior

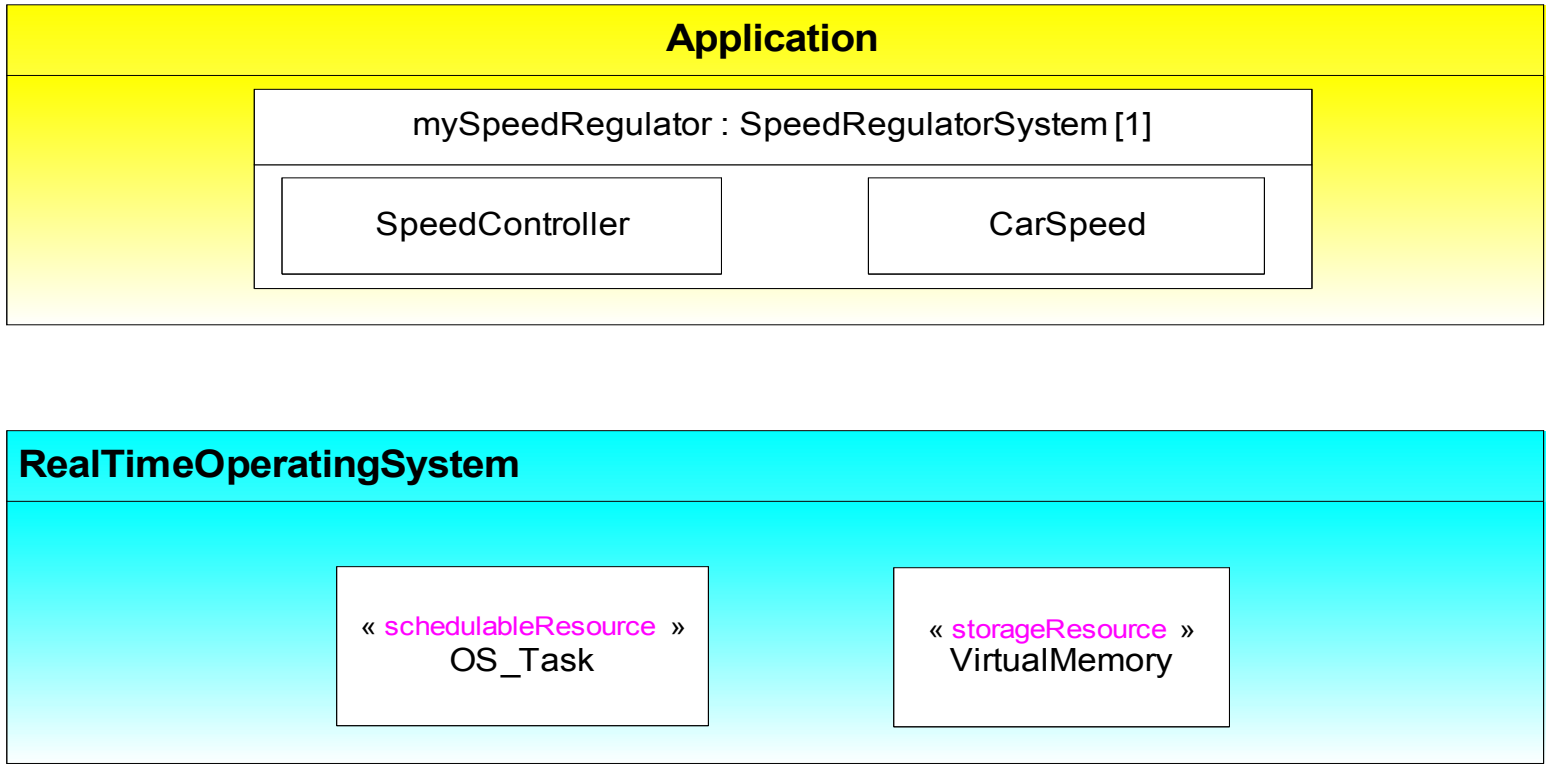


What can serve as a target of an allocation, the physical view:
➔ a resource or a service.

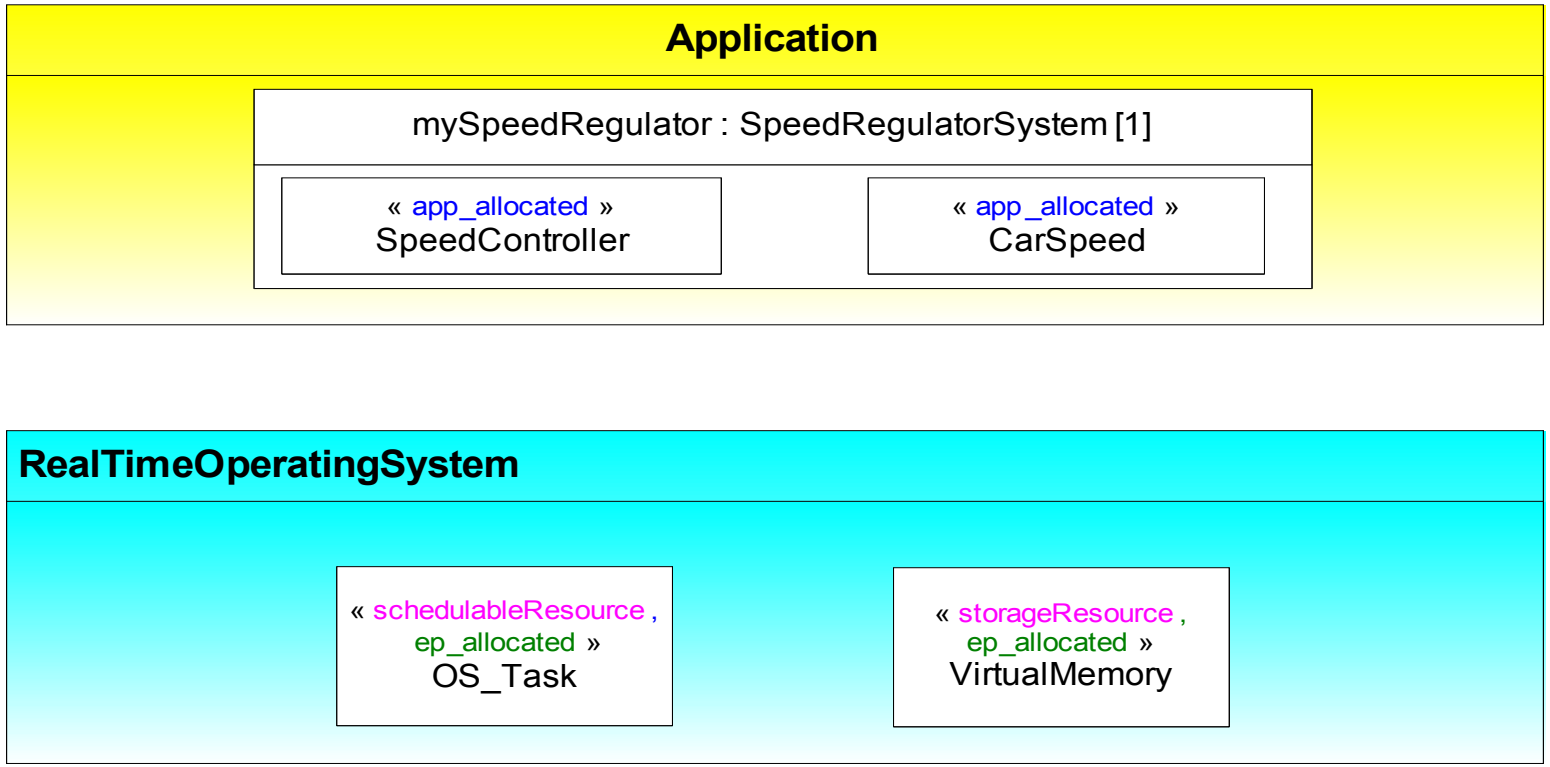
- **Define allocation relationships and its features**



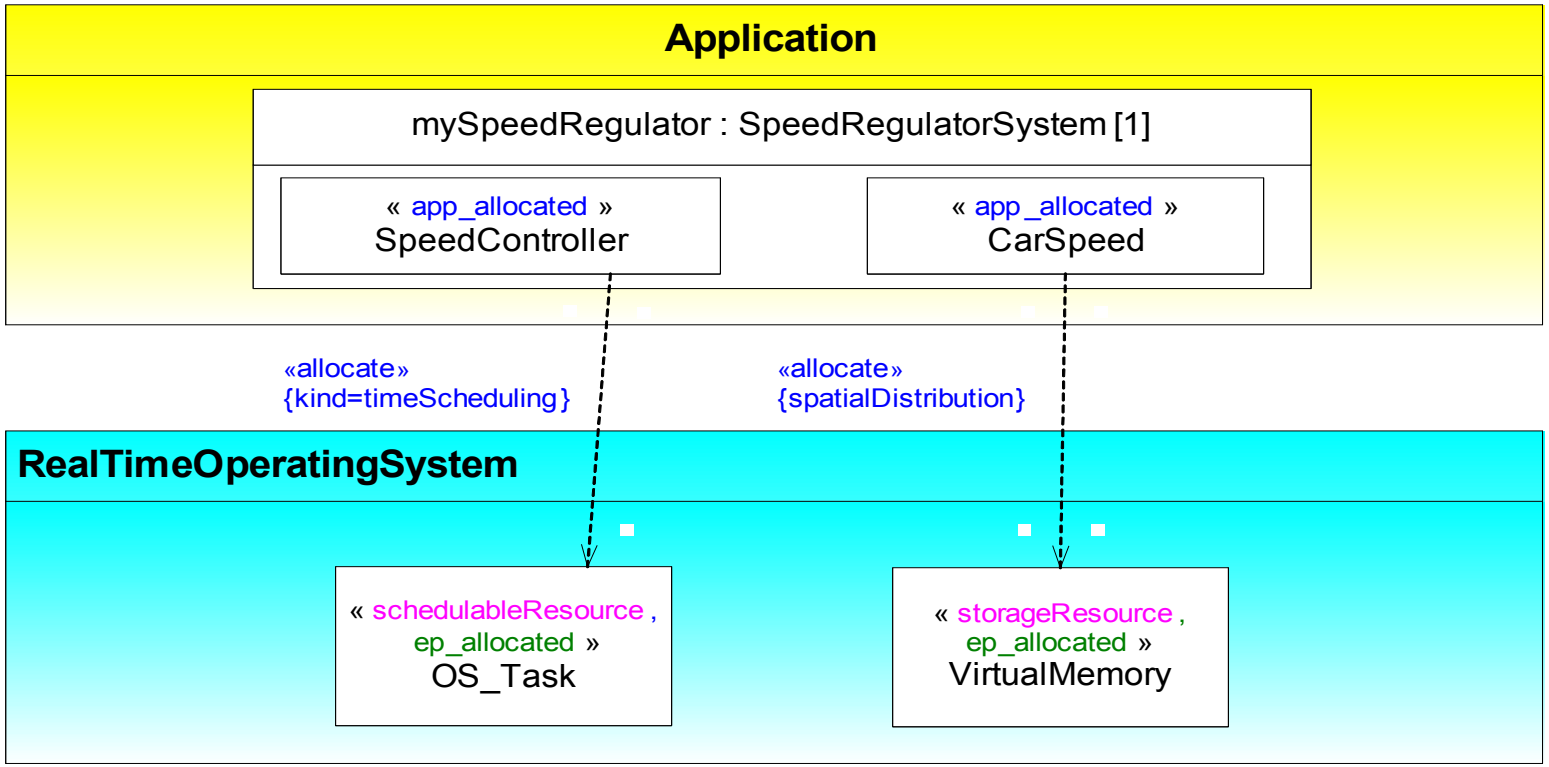
Allocation example (1)



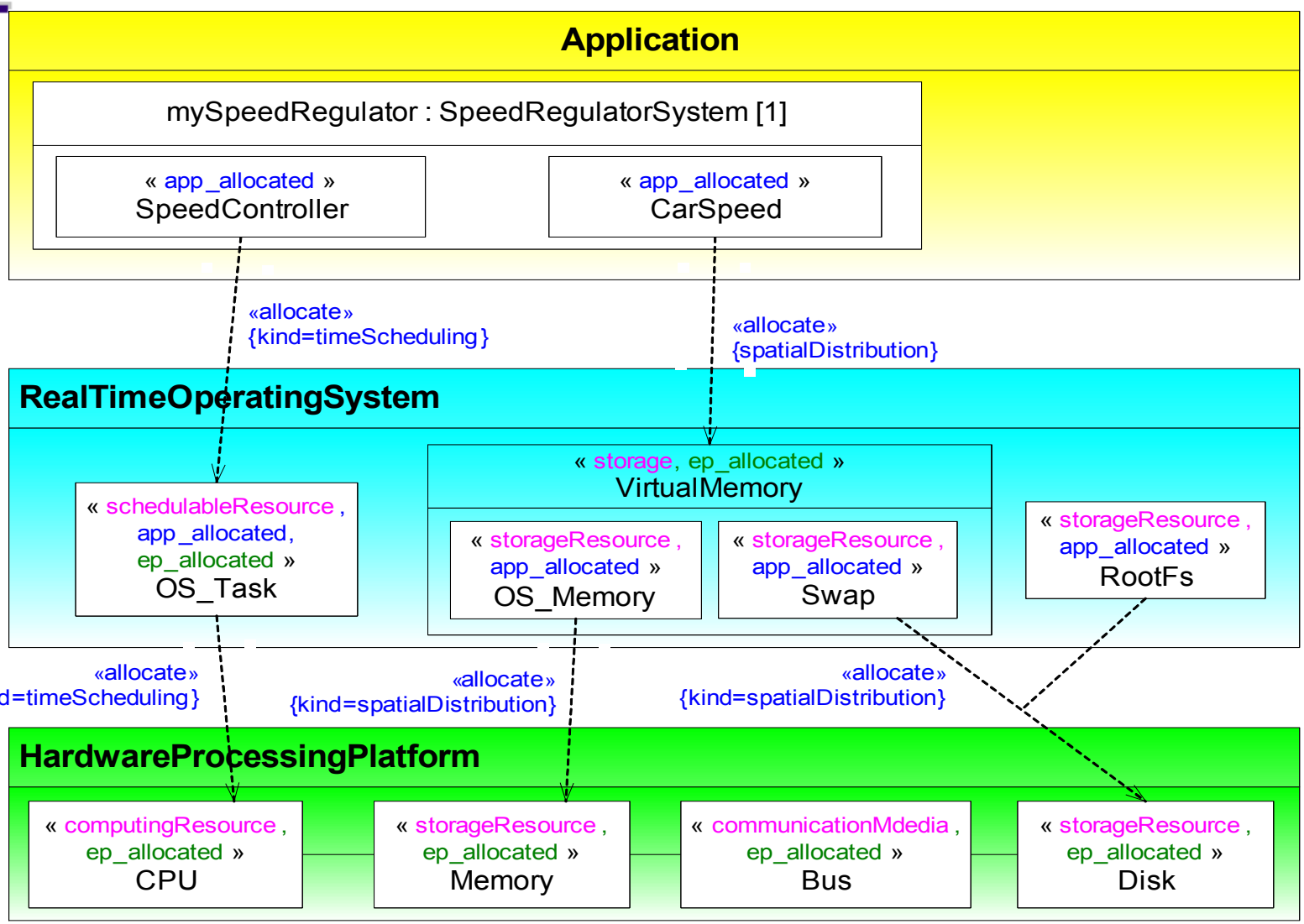
Allocation example (2)



Allocation example (3)



Allocation example (4)

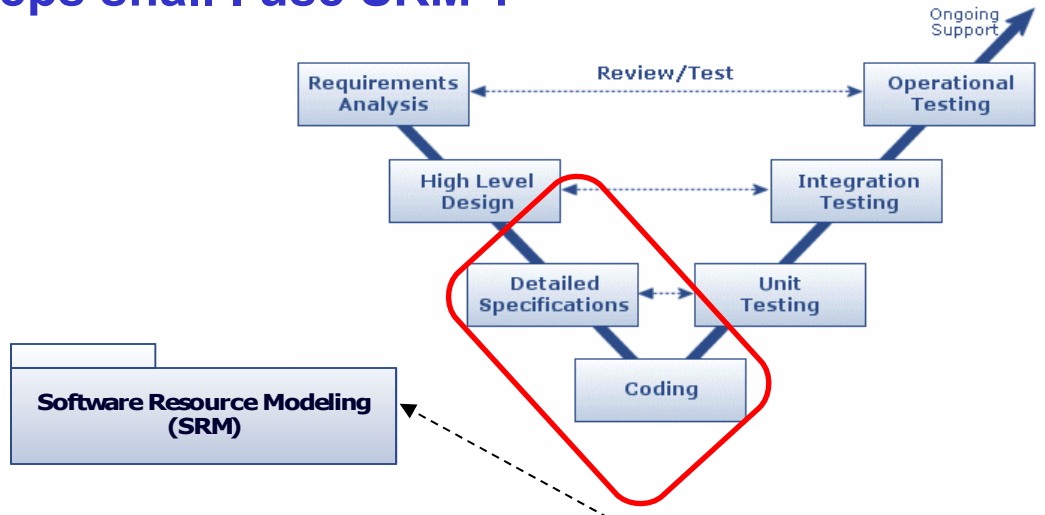


What is the Software Resource Modeling Profile (SRM) ?

- **A UML profile for modeling APIs of RT/E sw execution supports**
  - Real Time Operating Systems (RTOS)
  - Dedicated Language Libraries (e.g. ADA)
- **BUT it is NOT a new API standard dedicated to the RT/E domain!**
  - SRM is the result of a very deep state of the art and of the practices including but not limited to:
    - POSIX, ARINC 653, SCEPTRE, Linux RT, …

➔ SRM = a unified mean to describe such existing or proprietary APIs

- **In which steps shall I use SRM ?**



Why shall I use SRM for modeling RTOS APIs?

- **RTOS API modeling with UML is already possible**
  - But, generics UML is lacking RTE native artifacts!
    - **No modeling artifacts to describe specific concepts**
      - E.g. tasks, semaphores and mailboxes
  - Consequently, models rely only on naming conventions
    - ➔ **Not possible to define generic tools using these models**
      - E.g. code generator or model transformations for analysis.
- **Hence, SRM profile allows:**
  - To model precise multitasking designs
  - To be able to describe generic generative tools
  - To describe SW exemodels in an unified and standard way
    - SRM profile is a sub-profile of the MARTE standard

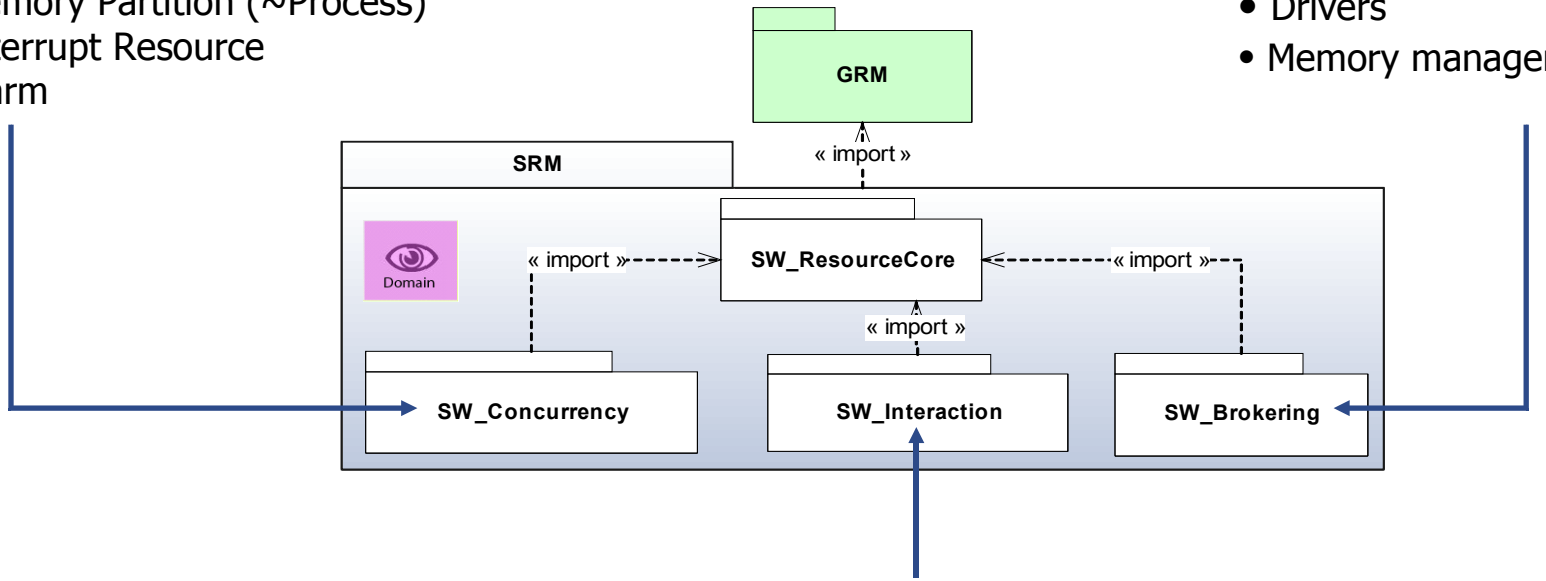
What is supported by the SRM profile ?

Concurrent execution contexts:

- Schedulable Resource (~Task)
- Memory Partition (~Process)
- Interrupt Resource
- Alarm

Hardware and software resources brokering:

- Drivers
- Memory management

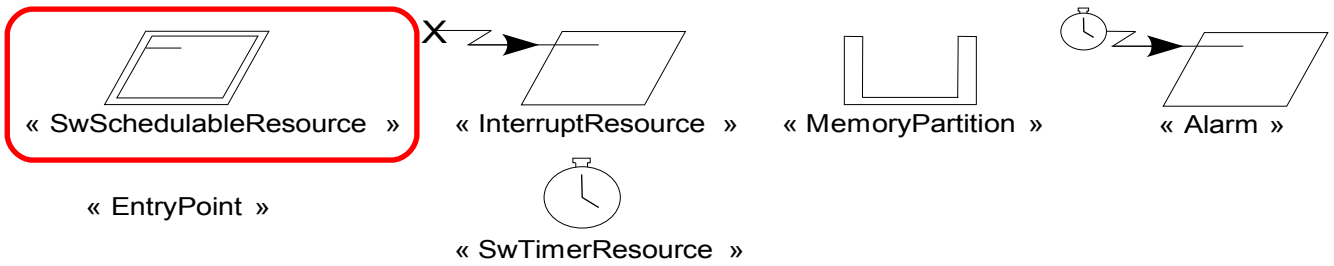


Interactions between concurrent contexts:

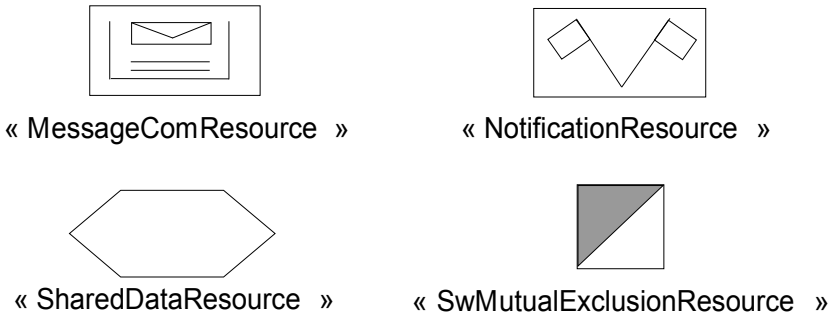
- Communication
  - ✓ Shared data
  - ✓ Message (~Message queue)
- Synchronization
  - ✓ Mutual Exclusion (~Semaphore)
  - ✓ Notification Resource (~Event mechanism)

Snapshot of the UML extensions provided by SRM

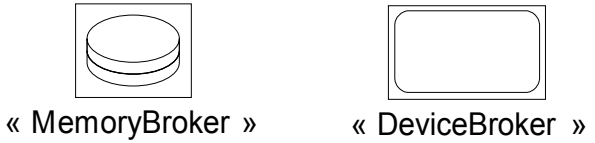
SRM::SW_Concurrency



SRM::SW_Interaction



SRM::SW_Brokering



The OSEK/VDX case study

- **OSEK/VDX standard (http://www.osek-vdx.org)**
  - Automotive industry standard for an open-ended architecture for distributed control units in vehicles
  - OSEK/VDX architecture consists of three layers:
    - OSEK-COM layer: Communication
      - **Data exchange support within and between electronics control units (ECUs)**
    - OSEK-NM layer : Network Management
      - **Configuration determination and monitoring**
    - OSEK-OS layer: Operating System
      - **API specification of RTOS for automotive ECU**

Overview of the OSEK/VDX-OS layer

- Main characteristics
  - A single processor operating system
  - A static RTOS where all kernel objects are created at compile time
- Main artifacts
  - Support for concurrent computing
    - **Task**
      - A task provides the framework for the execution of functions
    - **Interrupt**
      - Mechanism for processing asynchronous events
    - **Alarm & Counter**
      - Mechanisms for processing recurring events
  - Support for synchronizations of concurrent computing
    - **Event**
      - Mechanism for concurrent processing synchronization
    - **Resource**
      - Mechanism for mutual concurrent access exclusion

Focus on the OSEK/VDX Task definition

- **Semantic**
  - An OSEK-VDX task provides the framework for computing application functions. A scheduler organizes the sequence of task executions.

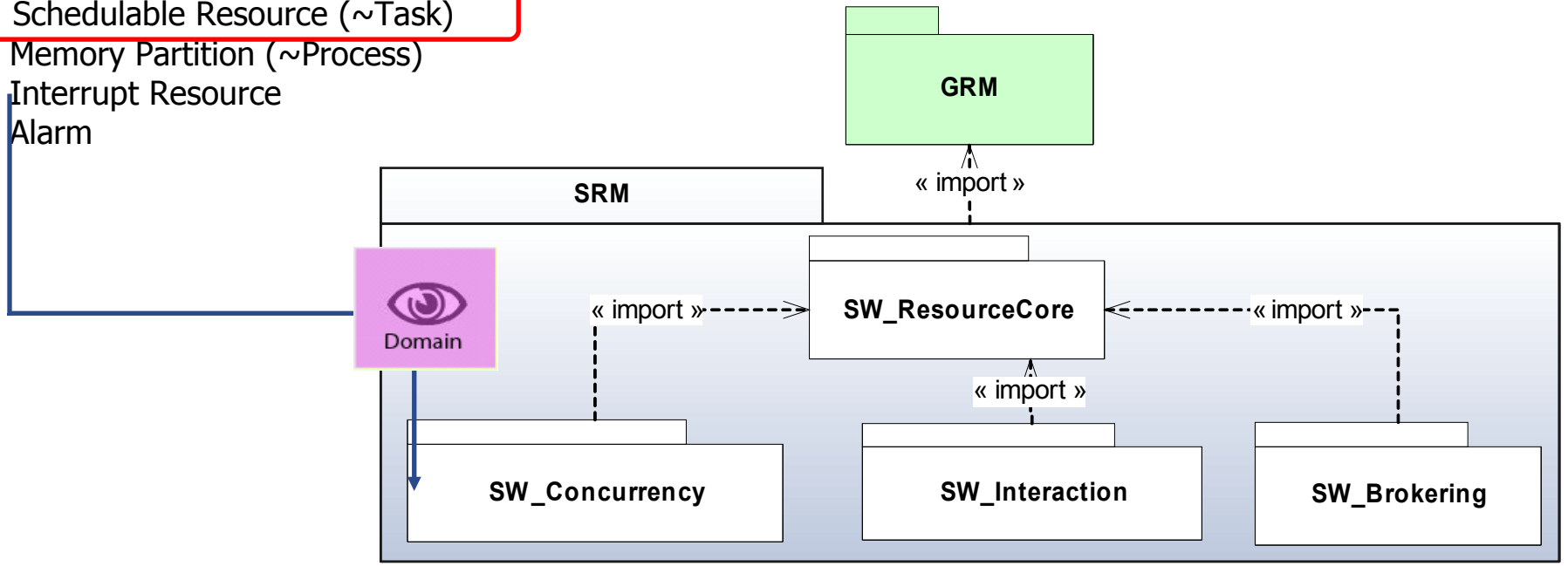
- **Example of properties**
  - **Priority: UINT32**
    - Priority execution of the task
  - **StackSize: UINT32**
    - Stack size associated with the execution of the task

- **Example of provided services**
  - **ActivateTask (TaskID: TaskType)**
    - Switch the task, identified by the **TaskID** parameter, from suspended to ready state
  - **ChainTask (TaskID: TaskType)**
    - Terminate of the calling task and activate the task identified by the **TaskID** parameter

Which SRM concepts for OSEK Task?

Concurrent execution contexts:

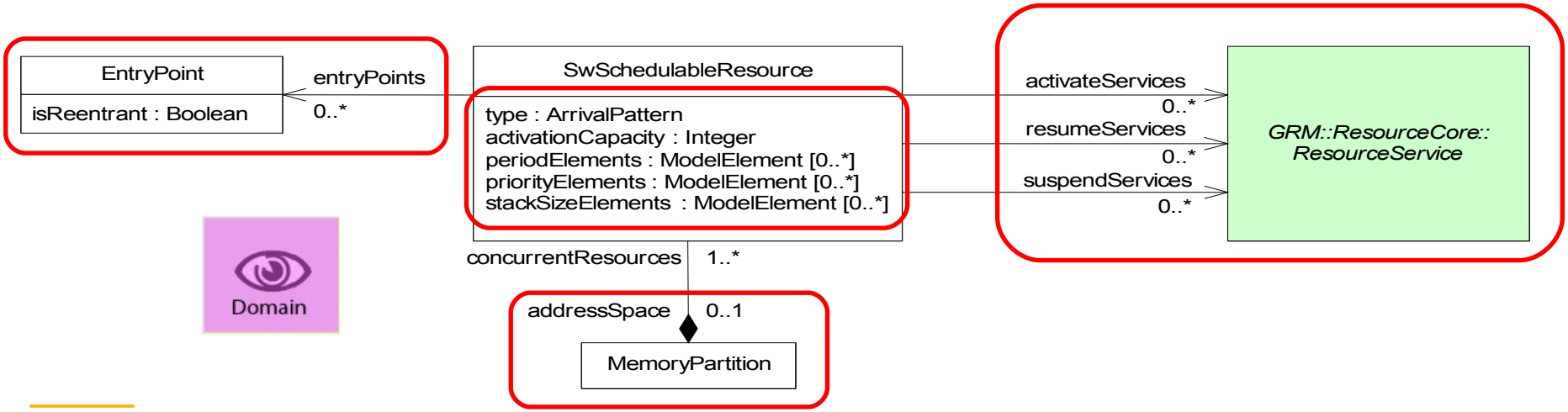
- Schedulable Resource (~Task)
- Memory Partition (~Process)
- Interrupt Resource
- Alarm



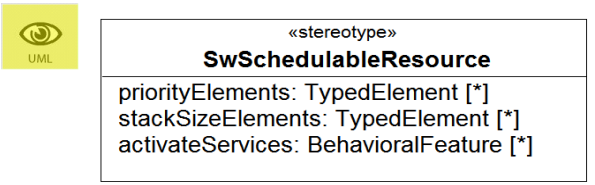
Details of «SwSchedulableResource»

- **Semantic (from MARTE::SRM::Concurrency package)**
  - Resource which executes, periodically or not, concurrently to other concurrent resources

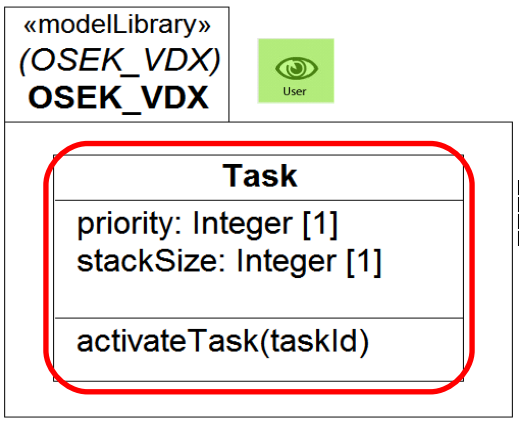
  ==> SRM artifacts for modeling OSEK-VDX Task!
- **Main features**
  - Owns an entry point referencing the application code to execute
  - May be restricted to execute in a given address space (i.e. a memory partition)
  - Owns properties: e.g., Priority, Deadline, Period and StackSize
  - Provides services: e.g., activate, resume and suspend
- **Extract from the SRM::SwConcurrency meta model**



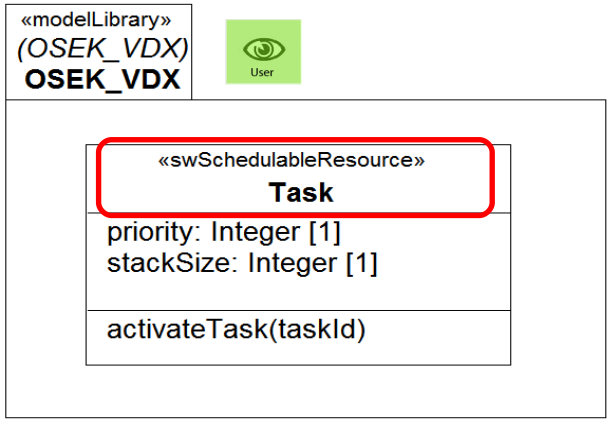
Model of an OSEK Task with «SwSchedulableResource»



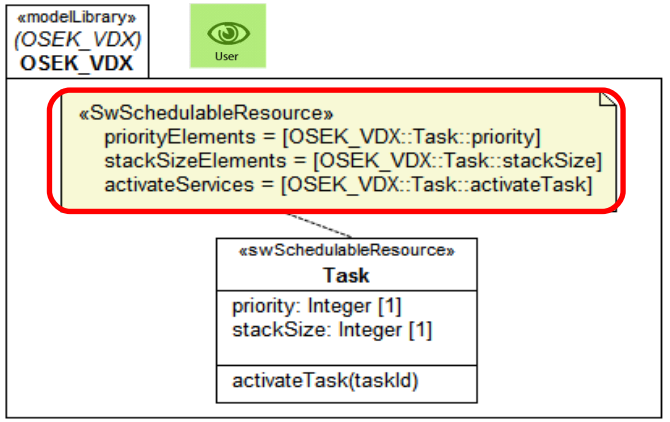
- **Define a UML model for OSEK_VDX::Task**
  a. Add model library applying the SRM profile
  b. Add a class and defines its features (properties and operations)
- **Applying the «SwSchedulableResource» stereotype**
- **Fulfill the tagged values of the applied stereotype**



(Step 1)



(Step 2)

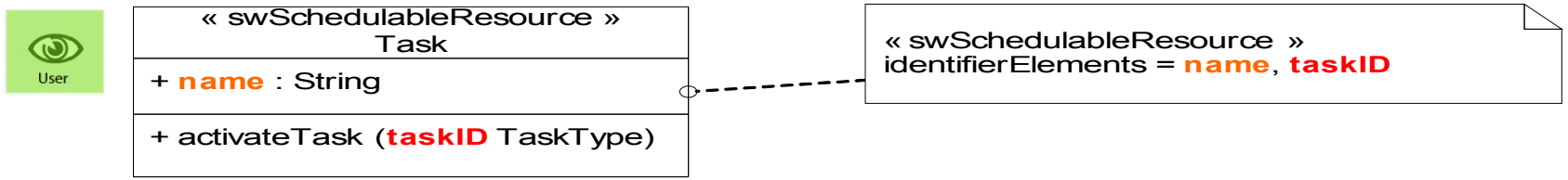


(Step 3)

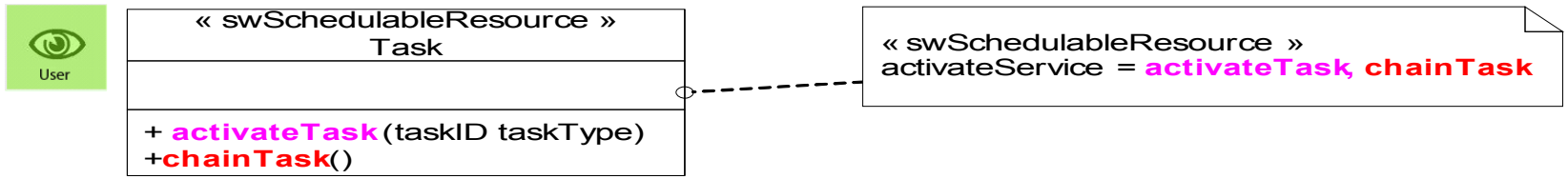
Models have been realized with the Papyrus Eclipse-based open-source tool for UML2:
http://www.papyrusuml.org

SRM modeling facilities

- **How to model multiple candidates for the same semantics ?**
  - Answer : All stereotype tags have multiple multiplicities. Thus, it is possible to reference multiple candidates for the same tag.
  - Examples
    - Both *name* attributes and *taskId* parameter are task identifier

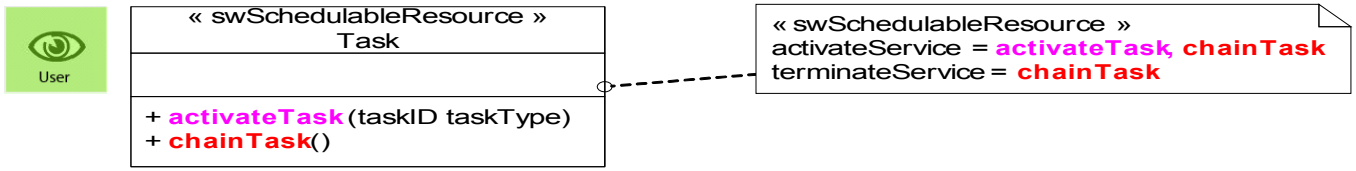


    - Both *activateTask* and *chainTask* operations are task activating services

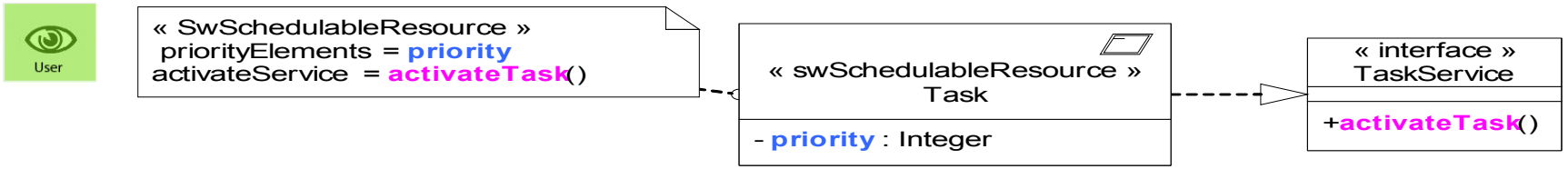


SRM modeling facilities (seq.)

- **How to model a feature which have multiple semantics ?**
  - Answer : Feature can be referenced by several different tags
    - Example
      - **The *chainTask* service is both a terminate service and an activate service**



- **Is it possible to reference a feature even if the feature owner is not the stereotyped element ?**
  - Answer : Yes, there is no constraints on the feature owner

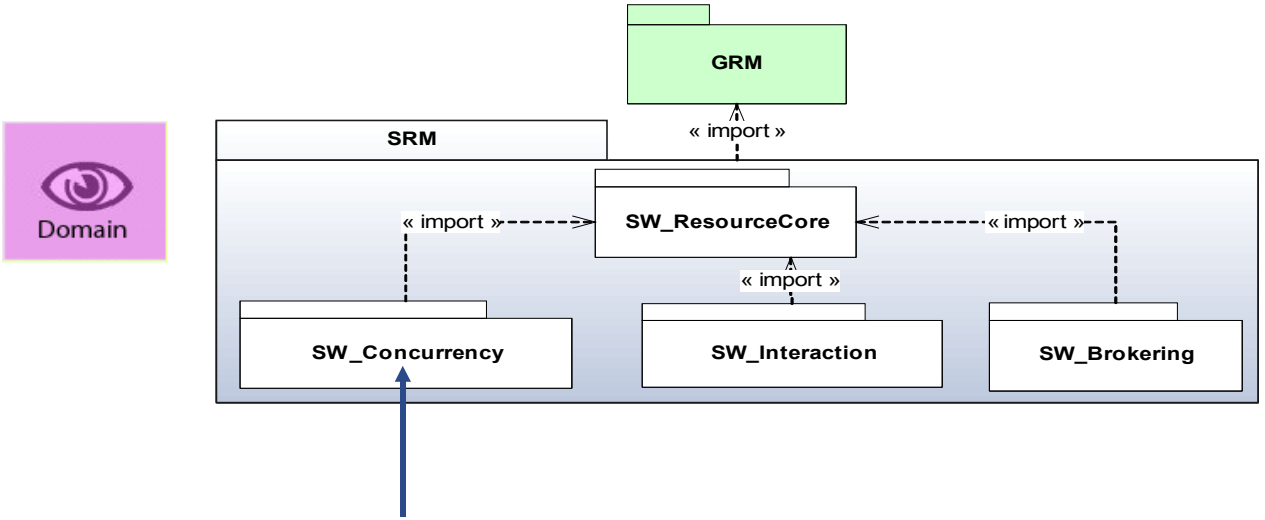


- **SRM allows multiple usages**
  - User can use constraints, such as OCL rules, to limit those possibilities

Focus on the OSEK/VDX Event definition

- **Semantics:**
  - The event mechanism is a means of synchronization that initiates state transitions of tasks to and from the *waiting* state.
  - Example of owned properties
    - **Mask : EventMaskType**
      - Define the mask associated with the event
  - Examples of provided services
    - **SetEvent (TaskID: TaskType, Mask: EventMaskType)**
      - The events of the task referenced by the TaskID parameter are set according to the event mask specified by the Mask parameter.
      - Calling the service SetEvent causes the task identified by the TaskID parameter to be transferred to the ready state, if it was waiting for at least one of the events specified in the Mask parameter.
    - **WaitEvent (Mask: EventMaskType)**
      - The state of the calling task is set to *waiting*, unless at least one of the events specified in the Mask parameter has already been set.

Which SRM concepts for OSEK Event?



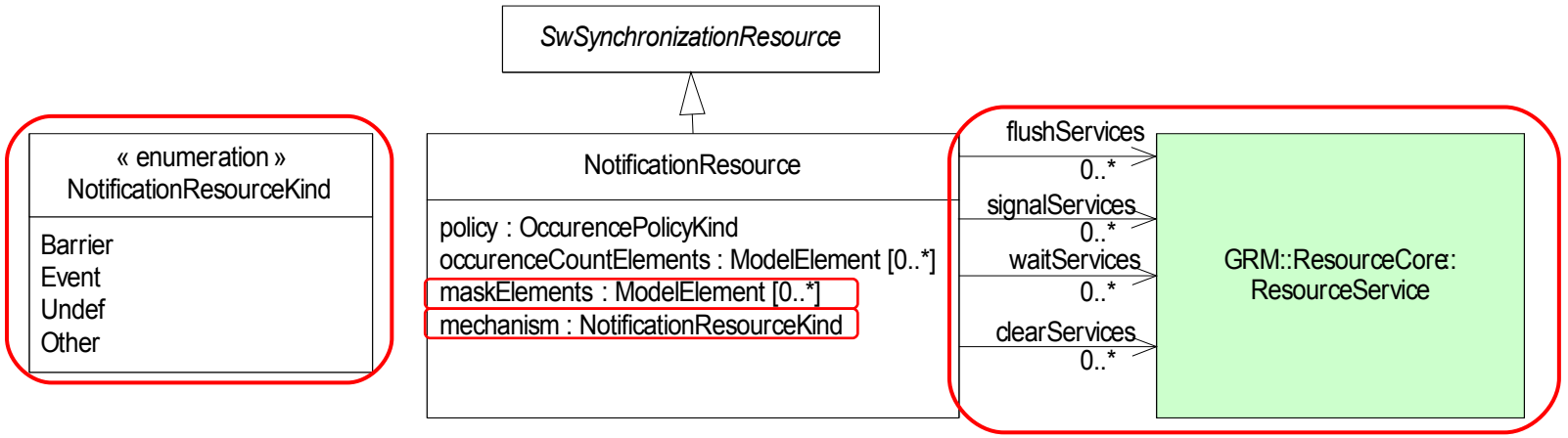
Interactions between concurrent contexts:

- Communication
  - ✓ Shared data
  - ✓ Message (~Message queue)
- Synchronization
  - ✓ Mutual Exclusion (~Semaphore)
  - ✓ Notification Resource (Event mechanism)

Details of «NotificationResource»

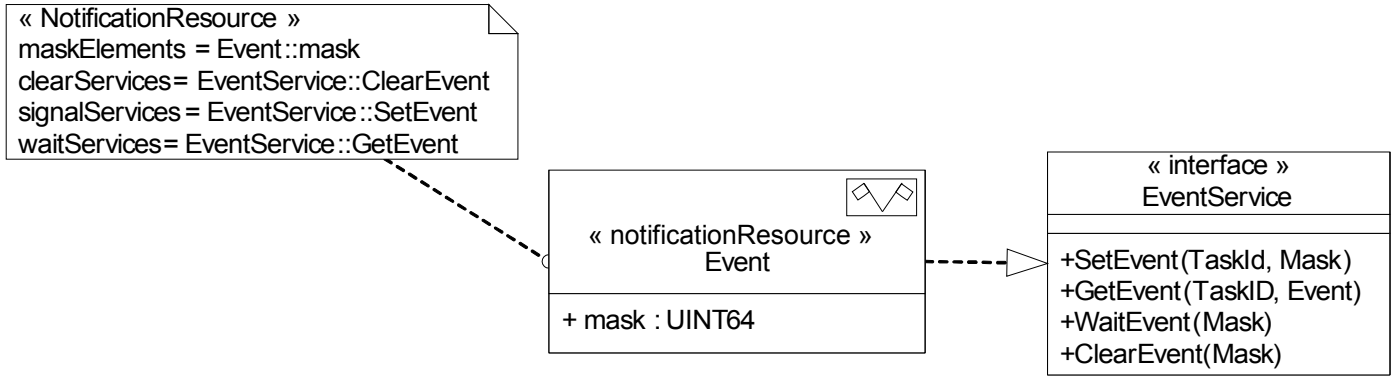
- **Semantic**
  - *NotificationResource* supports control flow by notifying the occurrences of conditions to awaiting concurrent resources

  ==> SRM artifacts for modeling OSEK-VDX Event!
- **Main features**
  - Examples of owned attribute
    - *maskElements* and *mechanism*
  - Examples of provided service
    - flushServices, signalServices,waitServices and clearServices
- **Extract from the SRM::SwInteraction meta model**

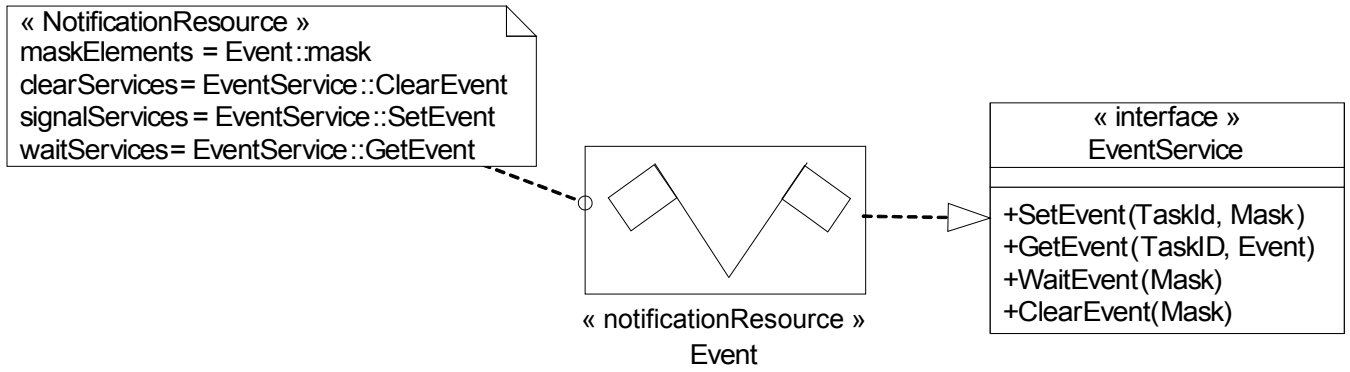


OSEK/VDX Event as a NotificationResource

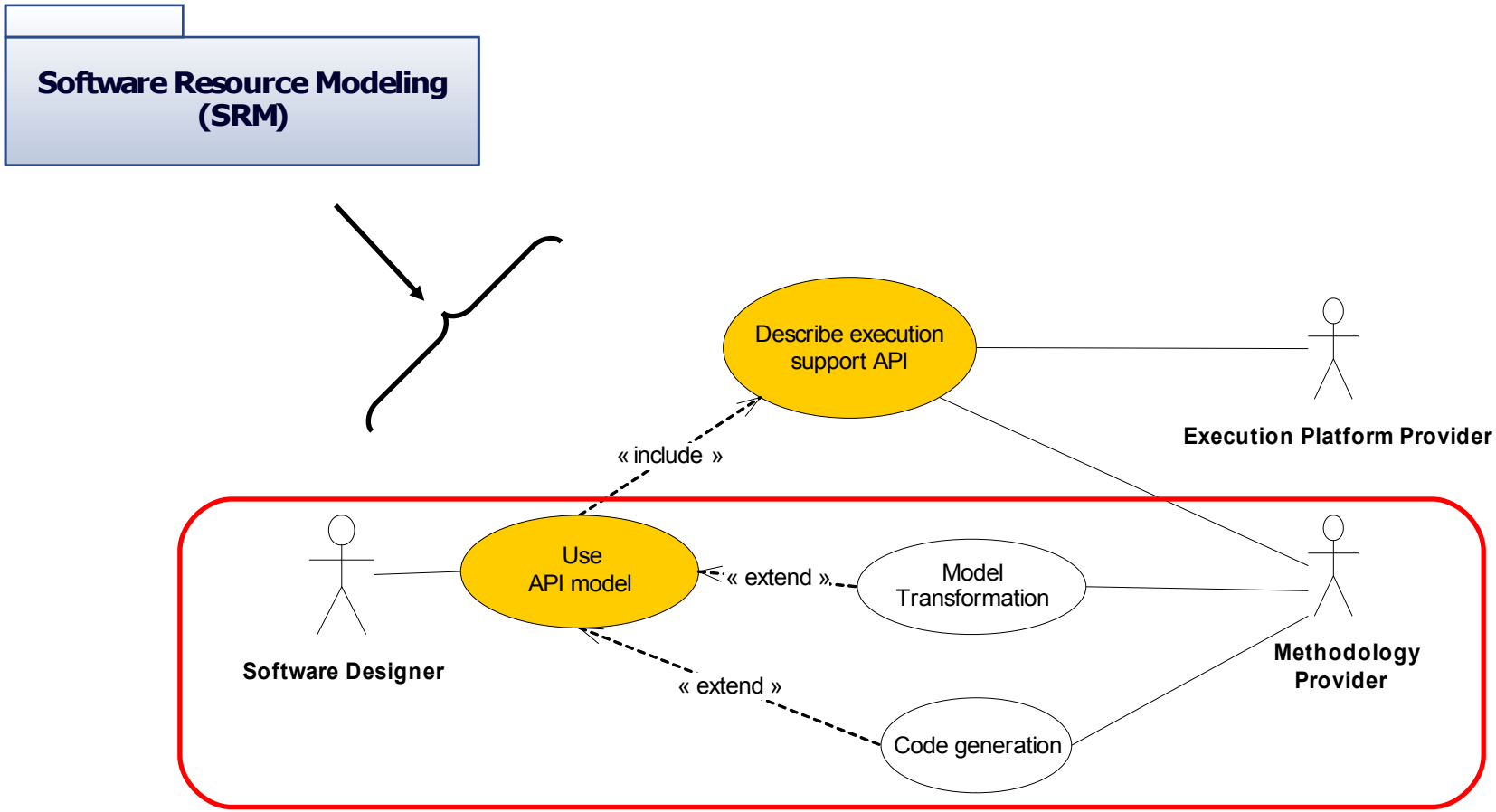
- **Stereotype icon**



- **Stereotype shape**



In which typical cases shall I use SRM ?



Use examples of one RTOS modeled with SRM

- **Example 1: Model-based design of multitask applications**
  - Illustrated on a robot controller application
- **Example 2: OS configuration file generation**
  - Generation of the OSEK OIL configuration files
- **Example 3: Assistance to port applications**
  - From OSEK to ARINC multitask design

Case study: A simple robot controller software

- **Goal**
  - A motion controller system for an exploration autonomous mobile robot.
- **Robot features**
  - Pioneer Robot (P3AT)
    - Four driving wheels
    - A camera
    - Eight sonar sensors, etc.
- **Design features of the robot controller**
  - OSEK/VDX execution support
    - Simulation on Trampoline (http://trampoline.rts-software.org/)
  - Two periodic tasks
    - **Data acquisition task**
      - Get position data from sonar sensors every 1 ms
    - **trajectory computing task**
      - Set new speed every 4 ms

Robot Simulator
http://playerstage.sourceforge.net/gazebo/gazebo.html

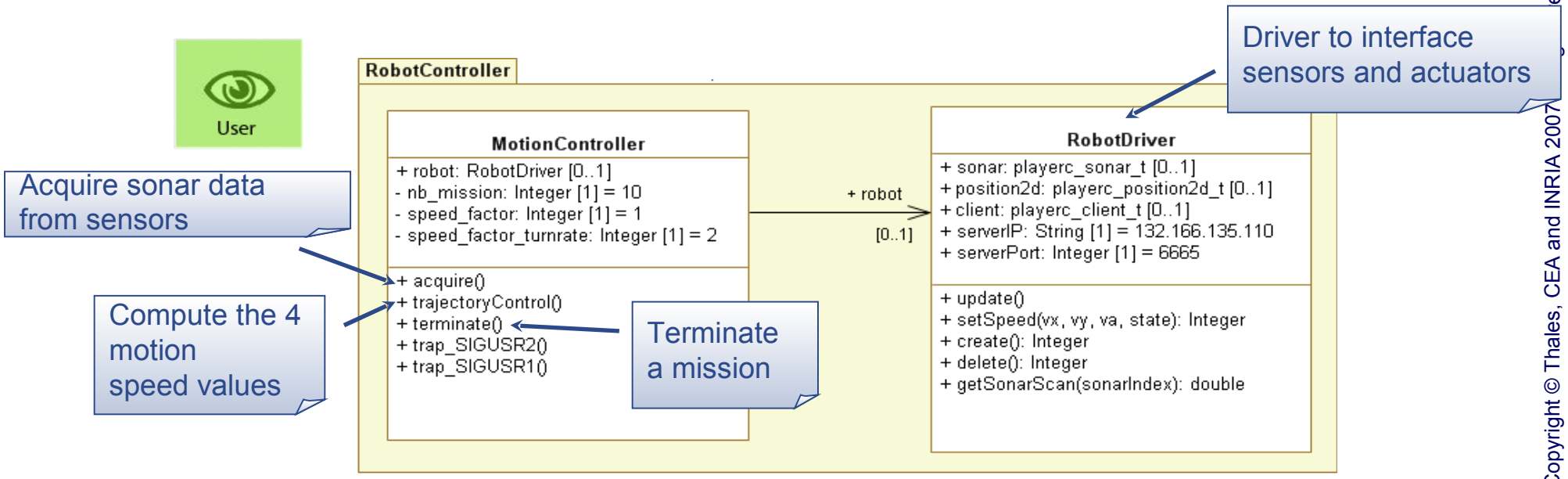
Purpose and context of the example 1

- **Provide a multitask design of the robot controller**
  - Target of the design is an OSEK/VDX-based platform

- **Design process**
  - A platform provider supplies the OSEK/VDX model library
    - Model library is described with the SRM Profile (as previously shown)
  - A user designs a multitask model of the application
    - Step 1: Describe the application model (also called functional model)
    - Step 2: Propose a multitask design using the OSEK model library artifact

Application design

- **Application model at the functional level**
  - One robot controller entity
    - Aims at controlling the robot motions
    - Main functions
      - **Acquire the sonar data**
      - **Compute the new speed of each 4 motions and send new orders**
  - A robot driver entity
    - Aims at interfacing robot sensors and actuators with the control application

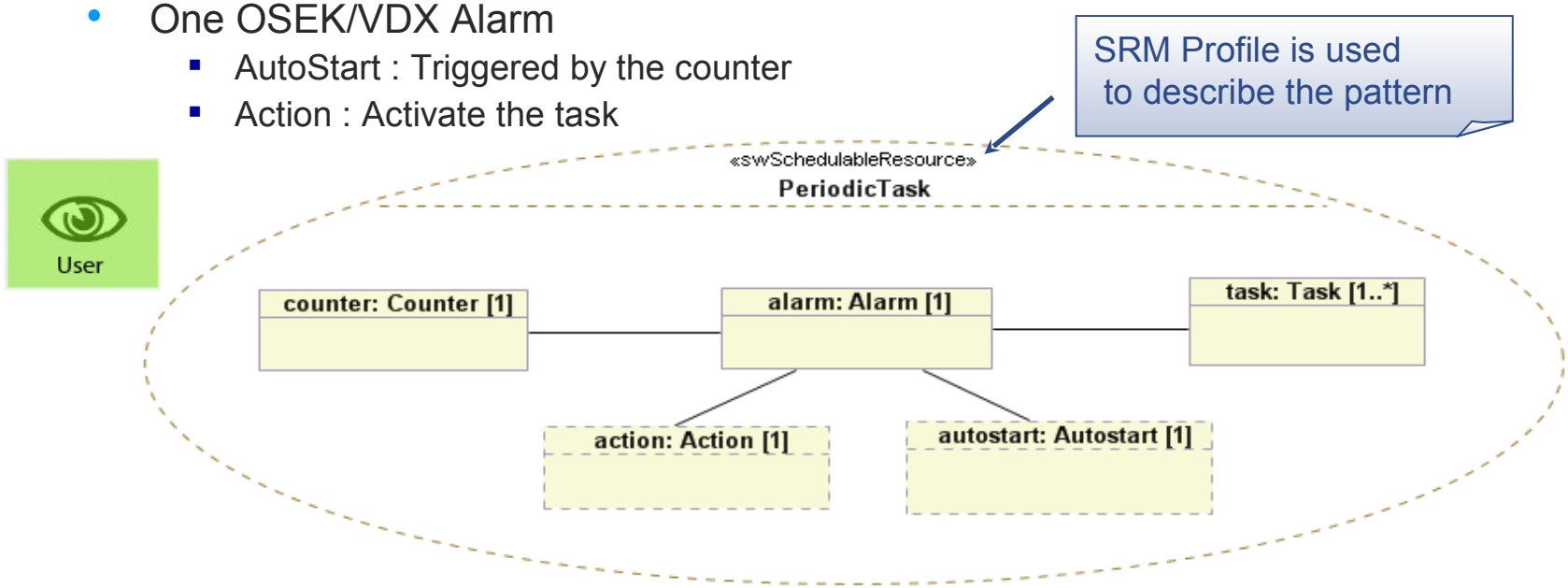


Principles of the applied multitask design

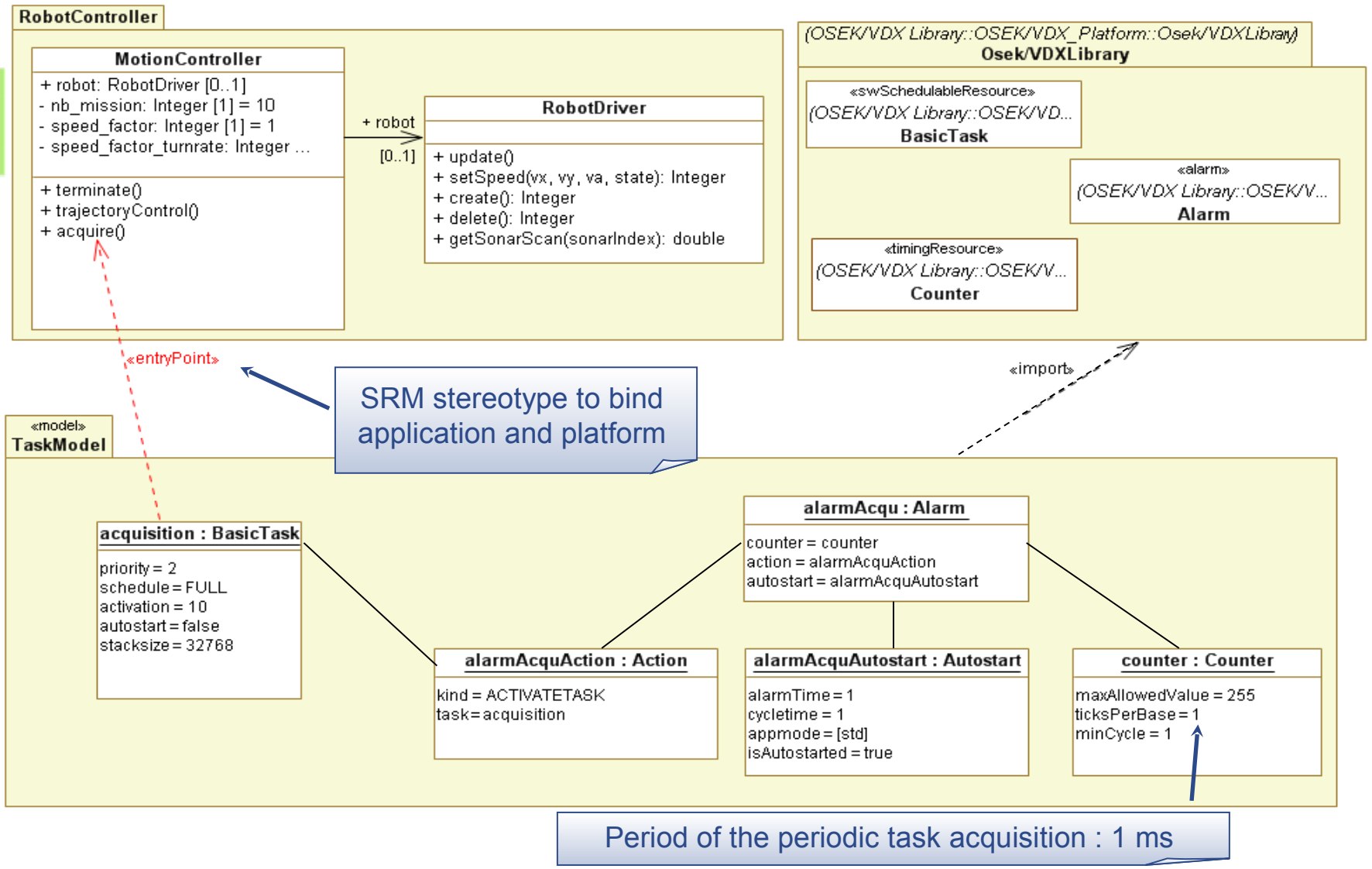
- **Two periodic tasks**
  - For data acquisition
    - Get position data from sonar sensors
    - Entry point
      - **Operation MotionController::acquire()**
    - Periodic
      - **Period = 1 ms**
  - For trajectory control
    - Compute and assign new speed order
    - Entry point
      - **Operation MotionController::trajectoryControl()**
    - Periodic
      - **Period = 4 ms**

Periodic task in OSEK/VDX

- **A design pattern for implementing periodic task on OSEK/VDX-based platforms**
  - One OSEK/VDX Counter
    - Counter period = period of the required periodic task
  - One OSEK/VDX Task
    - Entry point : periodic task Entry Point
  - One OSEK/VDX Alarm
    - AutoStart : Triggered by the counter
    - Action : Activate the task



Basic Robot Controller task models



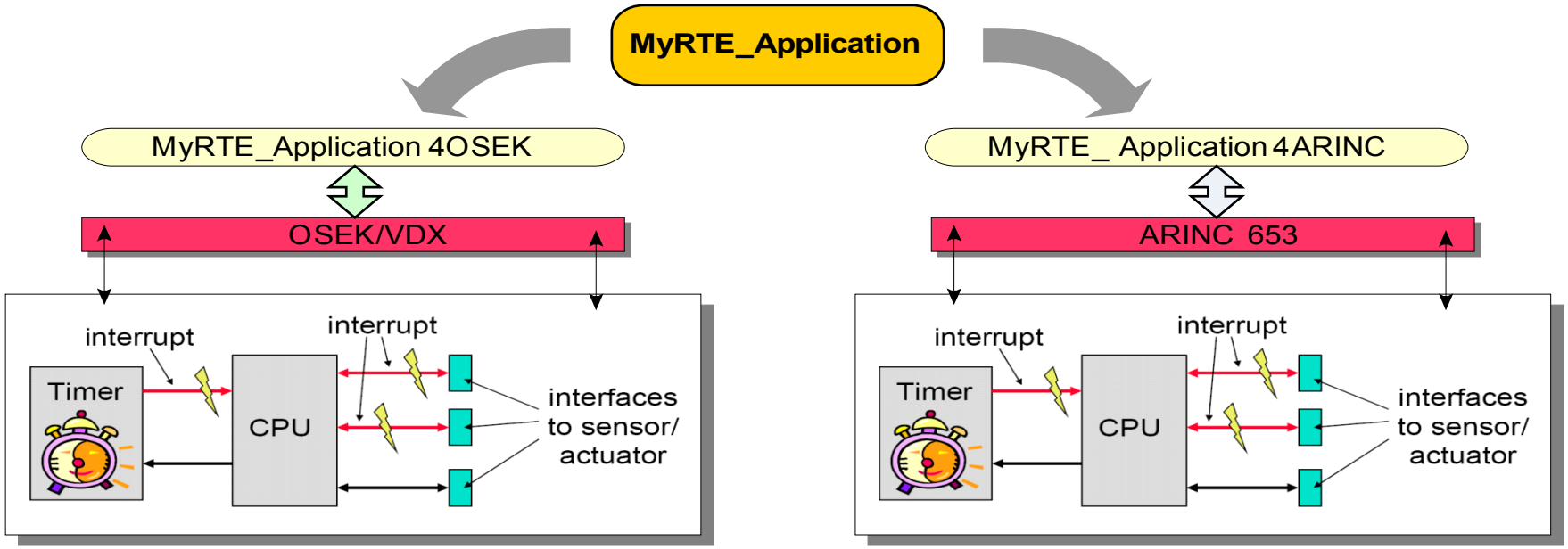
Example 2: OSEK Configuration File generation

- **Purpose**
  - Generation of the OSEK OIL configuration files from the multi-task design of the robot controller
- **OIL: OSEK Implementation Language**
  - http:://osek-vdx.org
  - The goal of OIL is to provide a mechanism to configure an OSEK application for a particular CPU
  - Principle
    - For each CPU, there must be an OIL description
    - All OSEK system objects are described using OIL objects
    - OIL descriptions may be :
      - **hand-written**
      - **or generated by a system configuration tool**

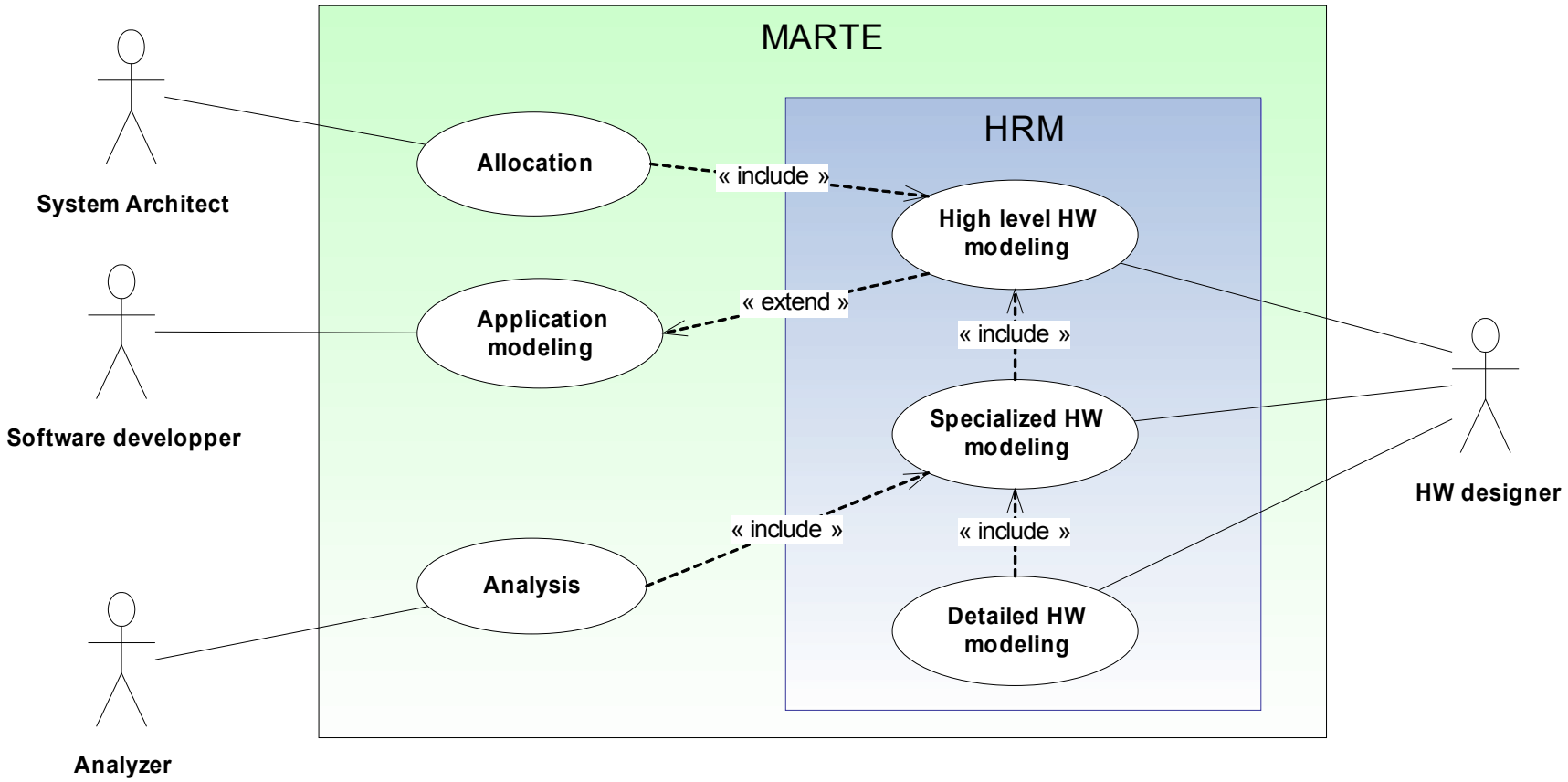
```
OIL_VERSION = "2.5" : "RobotController" ;

IMPLEMENTATION OSEK {
};

CPU cpu {
  APPMODE std {
  };

  COUNTER counter {
                MAXALLOWEDVALUE =  255 ;
                TICKSPERBASE =  1 ;
                MINCYCLE =  1 ;
  };
  ALARM alarmAcqu {
                COUNTER =  counter ;
                ACTION =  ACTIVATETASK {
                              TASK =  acquisition ;
                } ;
                AUTOSTART =  TRUE {
                              ALARMTIME =  1 ;
                              CYCLETIME =  1 ;
                              APPMODE =  std ;
                } ;
 };

  TASK acquisition {
                PRIORITY =  2 ;
                SCHEDULE =  FULL ;
                ACTIVATION =  10 ;
                AUTOSTART =  FALSE ;
                STACKSIZE =  32768 ;
  };
...
```

Example 3: Assist user to port multitask designs

- **Purpose**
  - Assist user to port the multitask design to an ARINC-653 RTOS
    - ARINC 653 standard provides avionics application software with the set of basic services to access the operating system and other system-specific resources.



HRM use cases



3 use cases = 3 levels of details

HRM use cases -- High level hardware modeling

- **How?**
  - High level of **abstraction**
  - **Architectural** view of the HW platform
  - With key properties:
    - E.g., instruction set and memory size.
  - A formal view of usual **block diagrams**
- **For**
  - High level description of existing and targeted HW platform
  - First steps of design of new HW architecture
- **By**
  - System architects
  - Software developers

HRM use cases -- Specialized hardware modeling

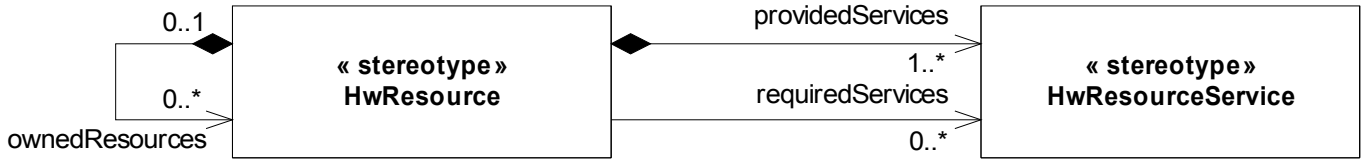
- **How?**
  - Specialized HW **description** model
  - Nature of details depends on the **point of view**
    - Ex1 : autonomy analysis requires power consumption modeling
    - Ex2 : WCET analysis need details on processor speed, communication bandwidth and memory organization…
- **For analysis purpose**
- **By analyzers**

HRM use cases -- Detailed hardware modeling

- **How?**
  - HRM is a detailed HW architecture design language
  - Level of details depends on the description **accuracy**
    - Ex1: Functional simulator of a processor only requires its instruction set family
    - Ex2: Performance simulation need a fine description of processors micro-architecture.
- **For**
  - Model-based datasheets description
  - Simulation
    - generation of configurations for simulation tools
- **By**
  - HW designers

HRM structure

- **Hierarchical taxonomy of hardware concepts**
  - Successive **inheritance** layers
  - **From** generic concepts (GRM-like)
    - *HwComputingResource, HwMemory, HwCommunicationResource…*
  - **To** specific and detailed resources
    - *HwProcessor, HwBranchPredictor, HwCache, HwMMU, HwBus, HwBridge, HwDMA…*
  - All HRM concepts are *HwResource(s)*



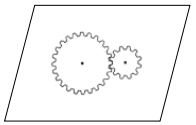
- Two **modeling views to separate concerns**

Logical / Physical

HRM structure -- Logical modeling

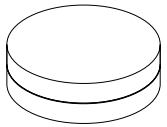
- **Provides a functional description**
- **Based on a functional classification of hardware resources:**

. HwComputing .

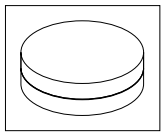


« HwProcessor », « HwPLD », « HwASIC »

. HwStorage .

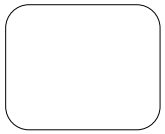


« HwCache », « HwRAM », « HwROM », « HwDrive »

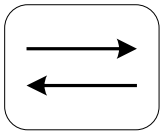


« HwMMU », « HwDMA »

. HwDevice .

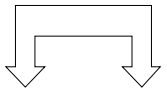


« HwDevice », « HwSupport »

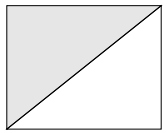


« HwI/O »

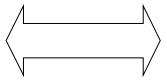
. HwCommunication .



« HwBridge »

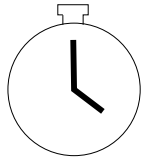


« HwArbiter »



« HwMedia », « HwBus »

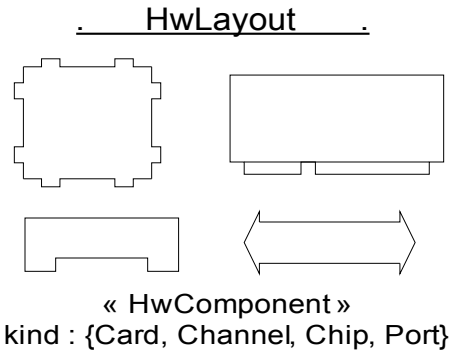
. HwTiming .

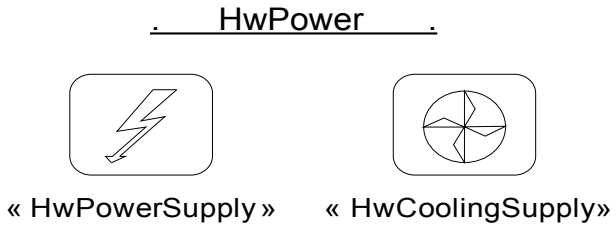


« HwClock », « HwTimer »

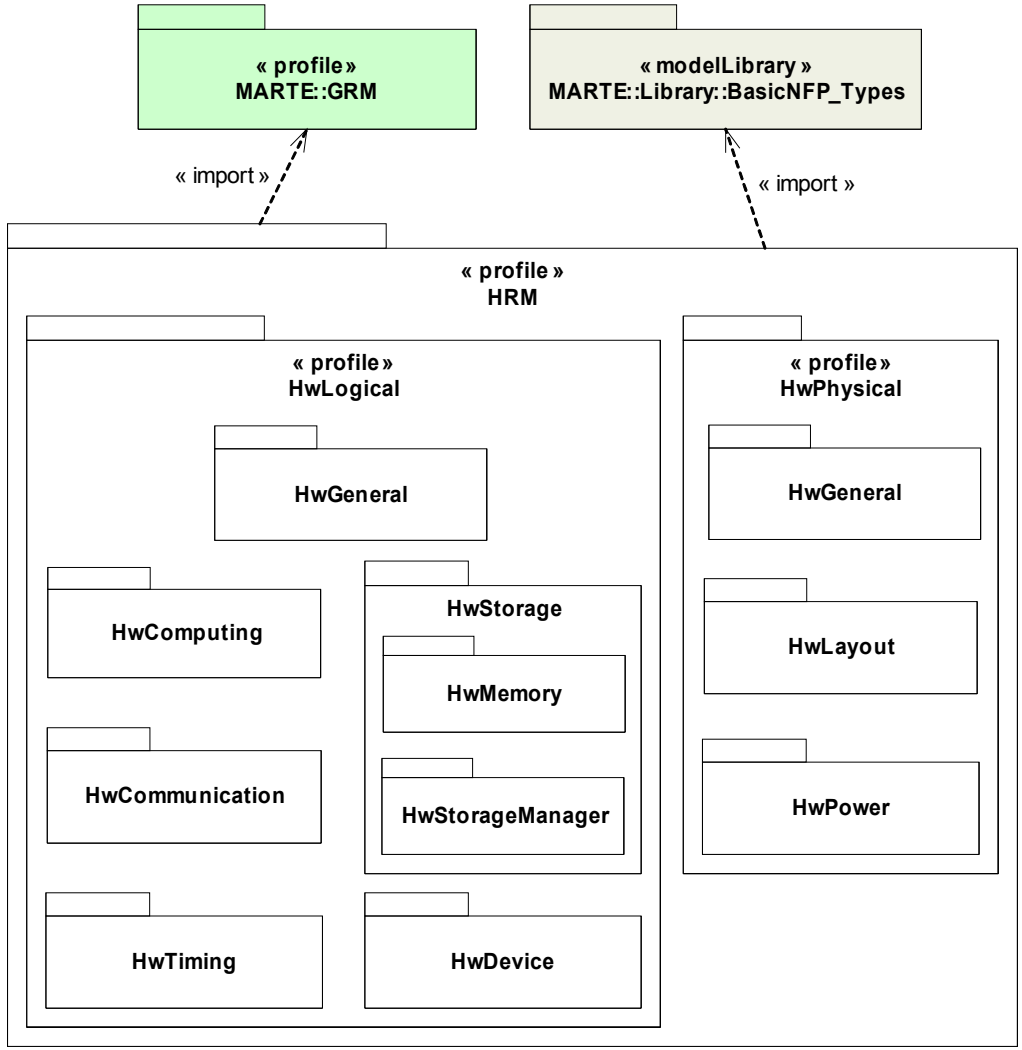
HRM structure -- Physical modeling

- **Provides a physical properties description**
- **Based on both following packages**
  - HwLayout
    - Forms: Chip, Card, Channel…
    - Dimensions, area and arrangement mechanism within rectilinear grids
    - Environmental conditions: e.g. temperature, vibration, humidity…
  - HwPower
    - Power consumption and heat dissipation

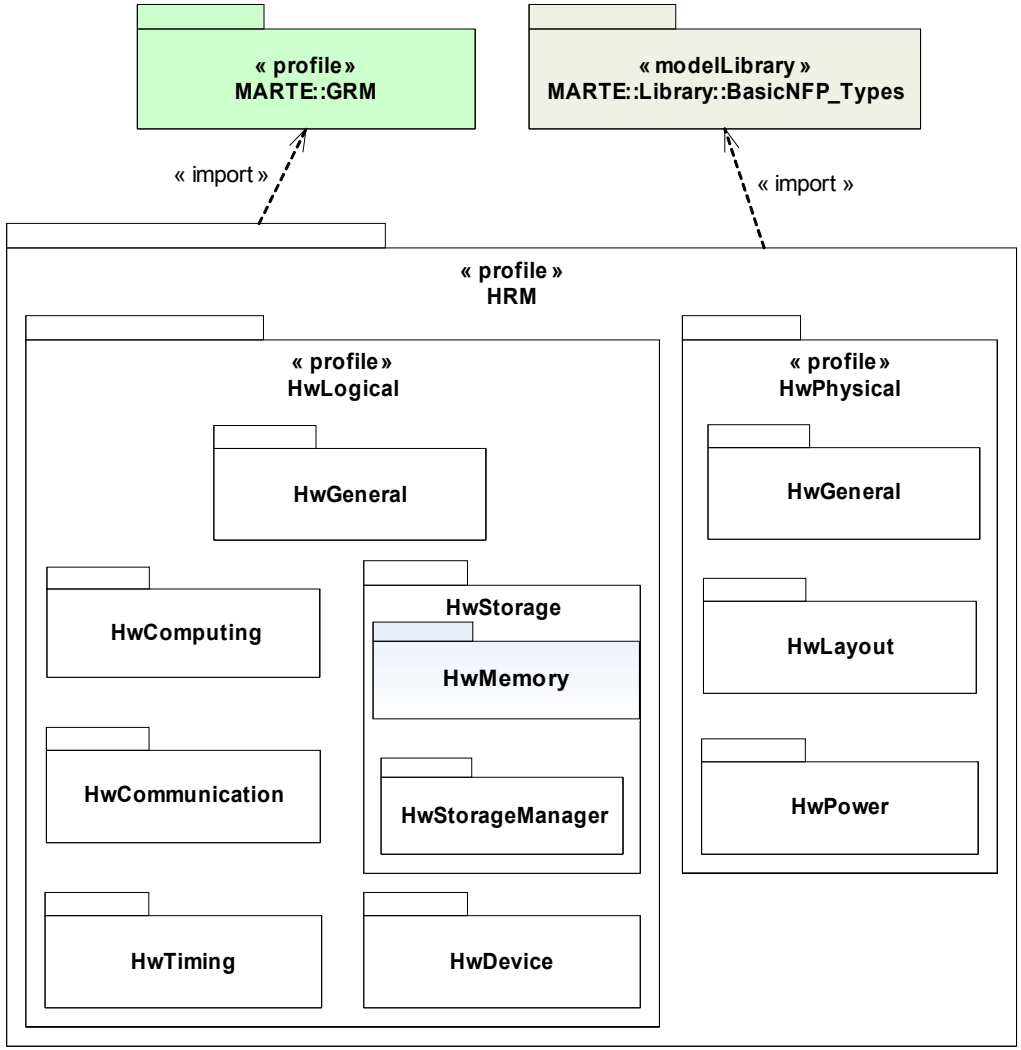




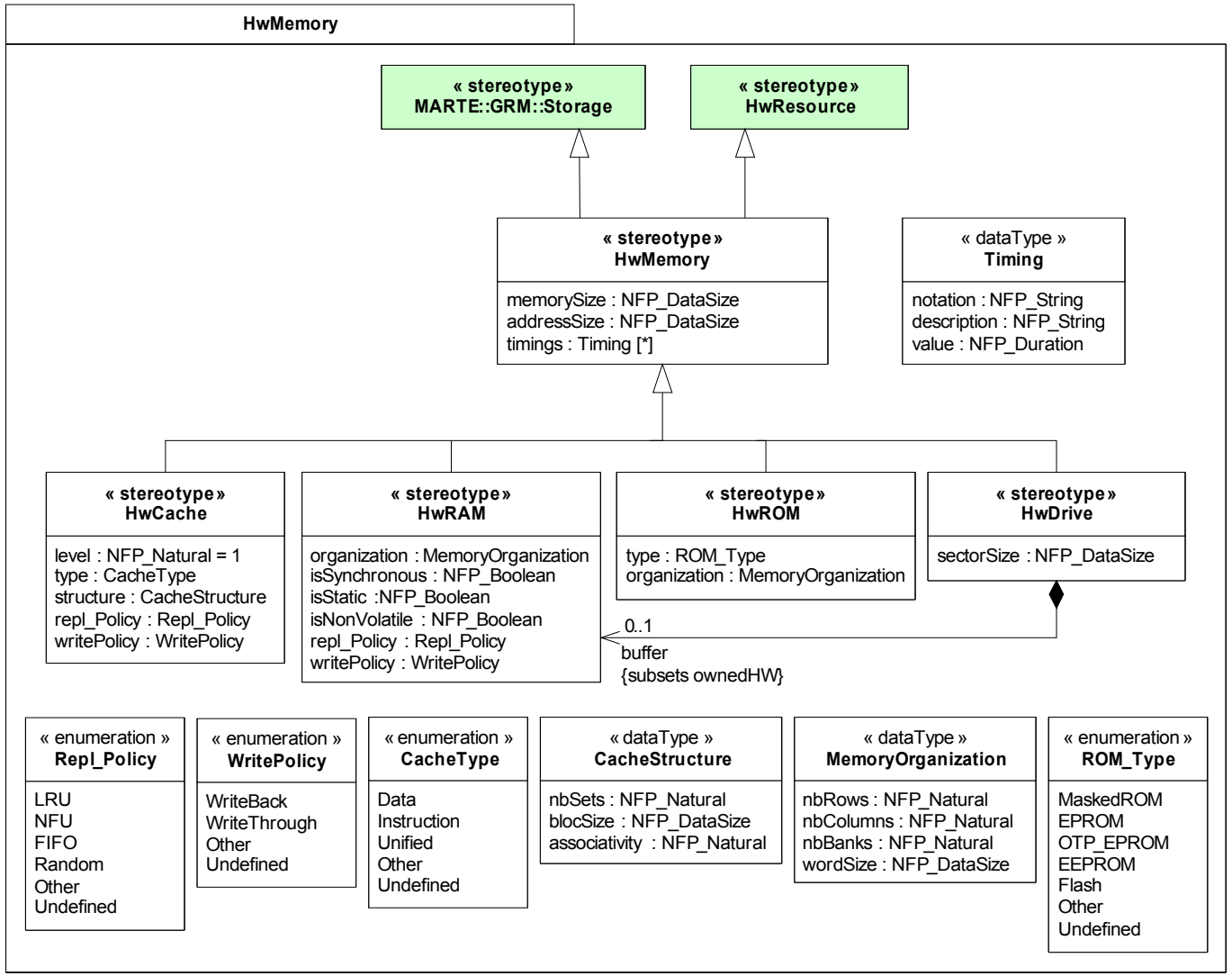
HRM profile overview



HRM profile -- HwMemory



HRM profile -- HwMemory



HRM profile -- HwMemory -- HwCache

« stereotype » HwCache
level : NFP_Natural = 1 type : CacheType structure : CacheStructure repl_Policy : Repl_Policy writePolicy : WritePolicy

- HwCache is a processing memory where frequently used data can be stored for rapid access
- Detailed description of the HwCache is necessary for performance analysis and simulation

« enumeration » Repl_Policy
LRU NFU FIFO Random Other Undefined

« enumeration » WritePolicy
WriteBack WriteThrough Other Undefined

« enumeration » CacheType
Data Instruction Unified Other Undefined

« dataType » CacheStructure
nbSets : NFP_Natural blocSize : NFP_DataSize associativity : NFP_Natural

HRM profile -- HwMemory -- HwCache

« stereotype » HwCache
level : NFP_Natural = 1 type : CacheType structure : CacheStructure repl_Policy : Repl_Policy writePolicy : WritePolicy

- Specifies the cache level.
  - Default value is 1

« enumeration » Repl_Policy
LRU NFU FIFO Random Other Undefined

« enumeration » WritePolicy
WriteBack WriteThrough Other Undefined

« enumeration » CacheType
Data Instruction Unified Other Undefined

« dataType » CacheStructure
nbSets : NFP_Natural blocSize : NFP_DataSize associativity : NFP_Natural

HRM profile -- HwMemory -- HwCache

« stereotype » HwCache
level : NFP_Natural = 1 type : CacheType structure : CacheStructure repl_Policy : Repl_Policy writePolicy : WritePolicy

- Specifies the HwCache structure
- HwCache is organized under sets of blocks.
- Associativity is the number of blocks within each set.
  - If associativity = 1, cache is direct mapped
  - If nbSets = 1, cache is fully associative.
- OCL rule
  - *memorySize = nbSets x blocSize x associativity*

« enumeration » Repl_Policy
LRU NFU FIFO Random Other Undefined

« enumeration » WritePolicy
WriteBack WriteThrough Other Undefined

« enumeration » CacheType
Data Instruction Unified Other Undefined

« dataType » CacheStructure
nbSets : NFP_Natural blocSize : NFP_DataSize associativity : NFP_Natural

HRM profile -- HwMemory -- HwCache

« stereotype » HwCache
level : NFP_Natural = 1 type : CacheType structure : CacheStructure repl_Policy : Repl_Policy writePolicy : WritePolicy

- Specifies the cache write policy
  - WriteBack: Cache write is not immediately reflected to the backing memory.
  - WriteThrough: Writes are immediately mirrored.

« enumeration » Repl_Policy
LRU NFU FIFO Random Other Undefined

« enumeration » WritePolicy
WriteBack WriteThrough Other Undefined

« enumeration » CacheType
Data Instruction Unified Other Undefined

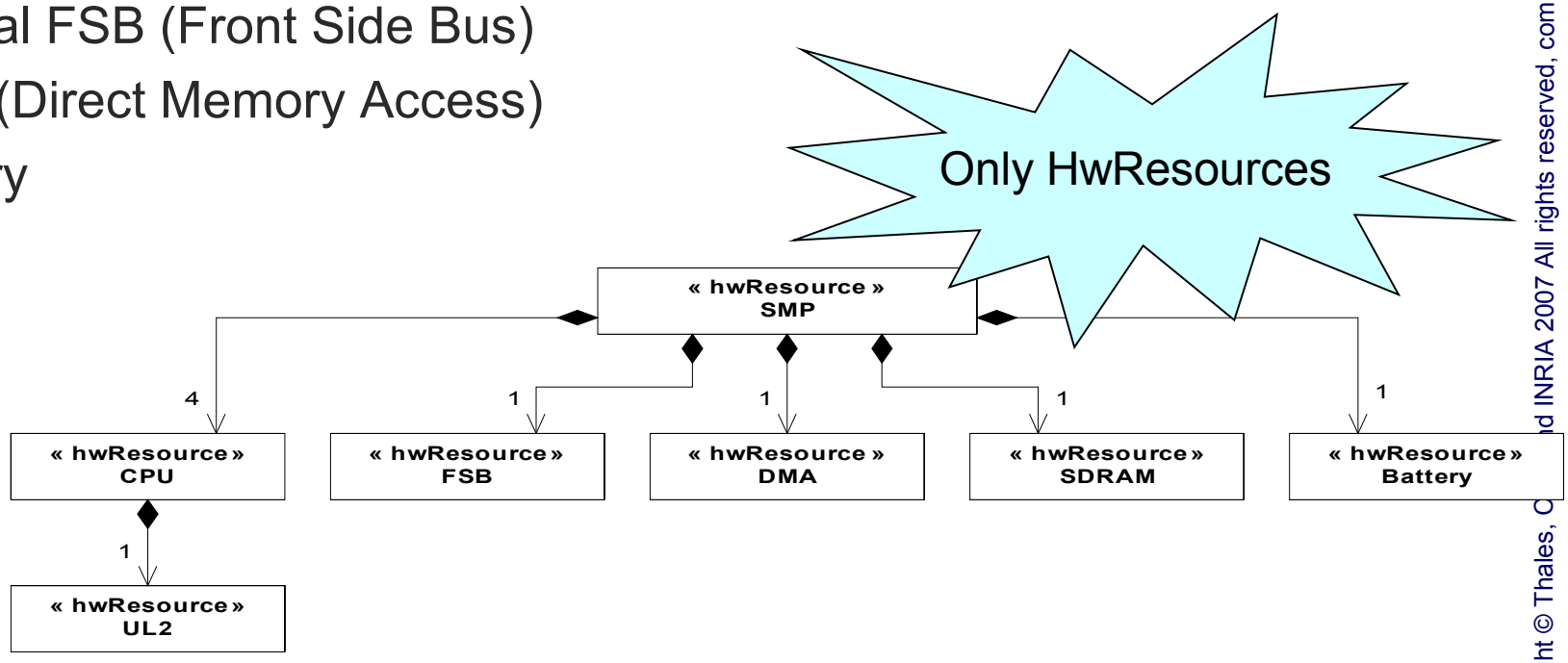
« dataType » CacheStructure
nbSets : NFP_Natural blocSize : NFP_DataSize associativity : NFP_Natural

HRM usage

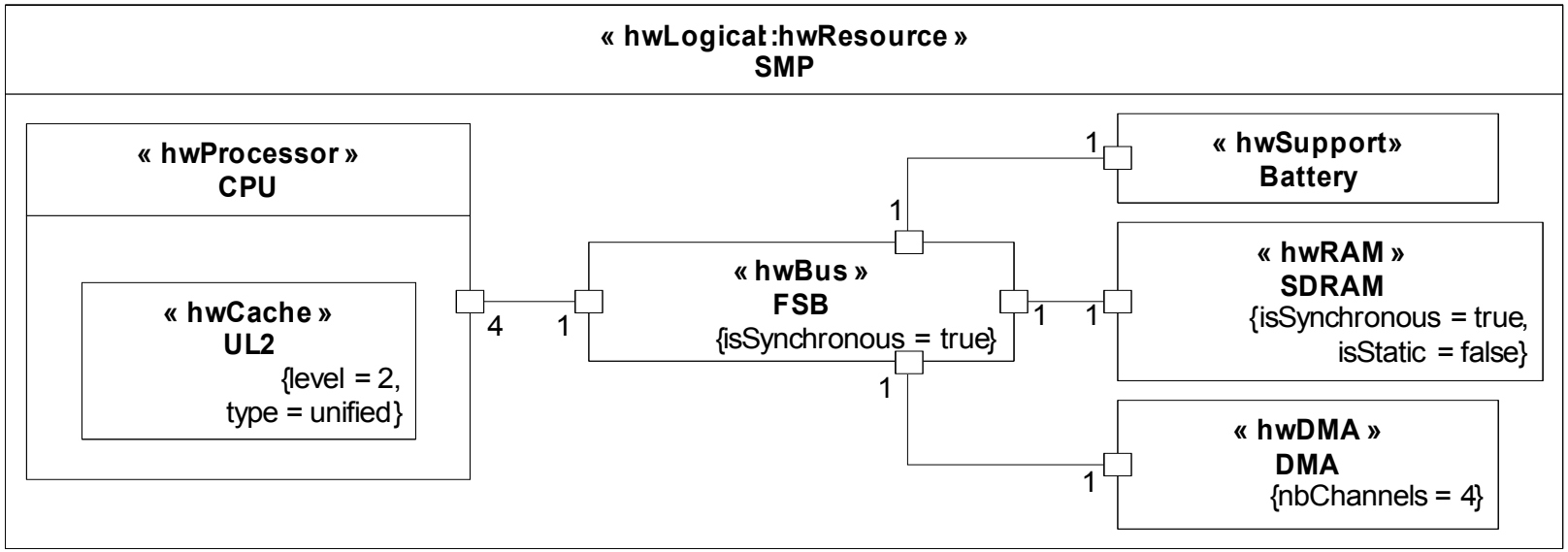
- **HRM stereotypes extends the main structural UML metaclasses**
  - Classifier, Class
  - InstanceSpecification, Property
  - Association (HwMedia, HwBus…), Port (HwEndPoint)
- **HRM can be used with all Structural UML diagrams:**
  - Class diagram
  - Component diagram
  - Composite Structure Diagram (well adapted for HW)
- **HRM profile application**
  - Definitions of the stereotype properties are optional
    - Specified **if** needed
    - Specified **when** needed (**Refinement**)
      - **At class level for technology definition (e.g. type of *HwCache*)**
      - **At instance level for component definition (e.g. size of *HwCache)***

Very early Hw Architecture Description

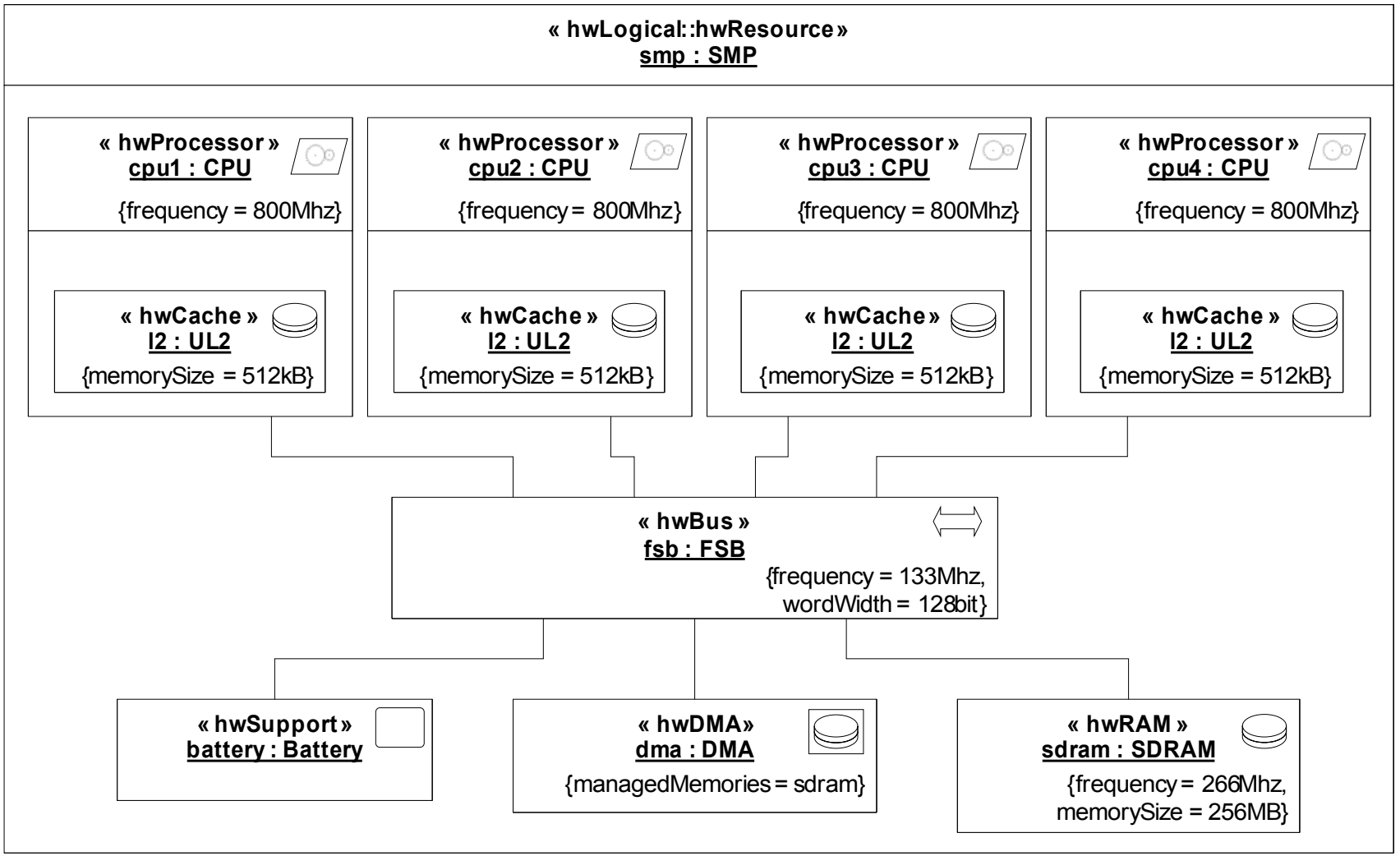
- SMP (Symmetric MultiProcessing) hardware platform
  - 4 identical processors
    - Unified Level 2 cache for each
  - Shared main memory (SDRAM)
  - Central FSB (Front Side Bus)
  - DMA (Direct Memory Access)
  - Battery



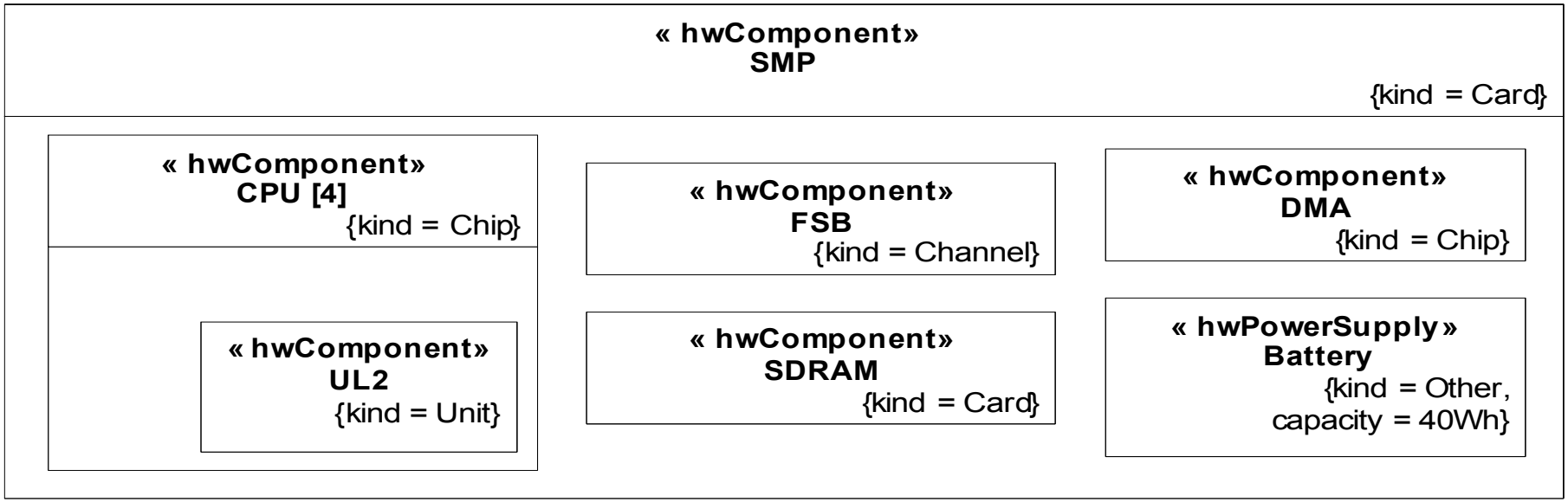
HRM usage example: Logical view 1



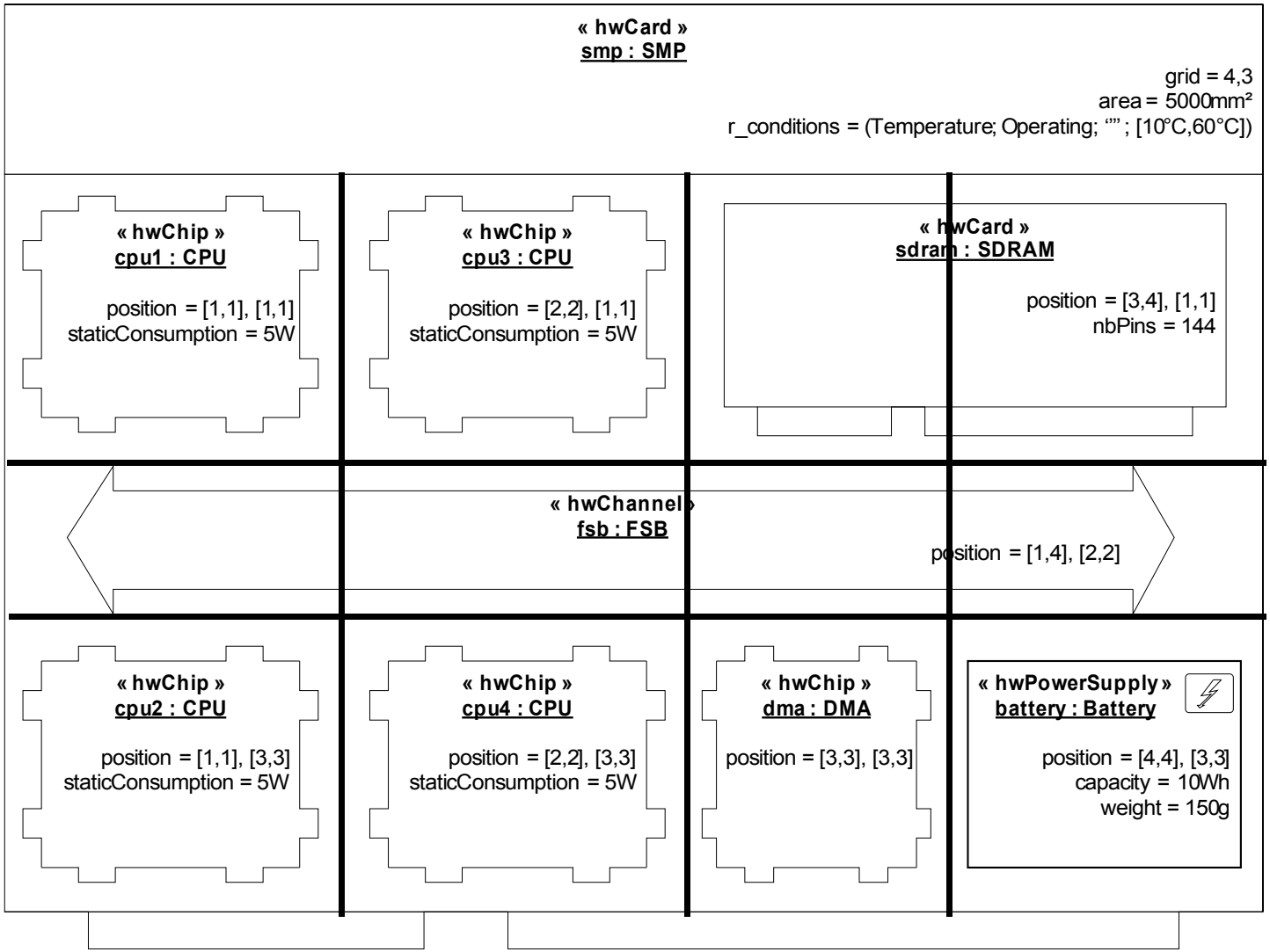
HRM usage example: Logical view 2



HRM usage example: Physical view 1



HRM usage example: Physical view 2



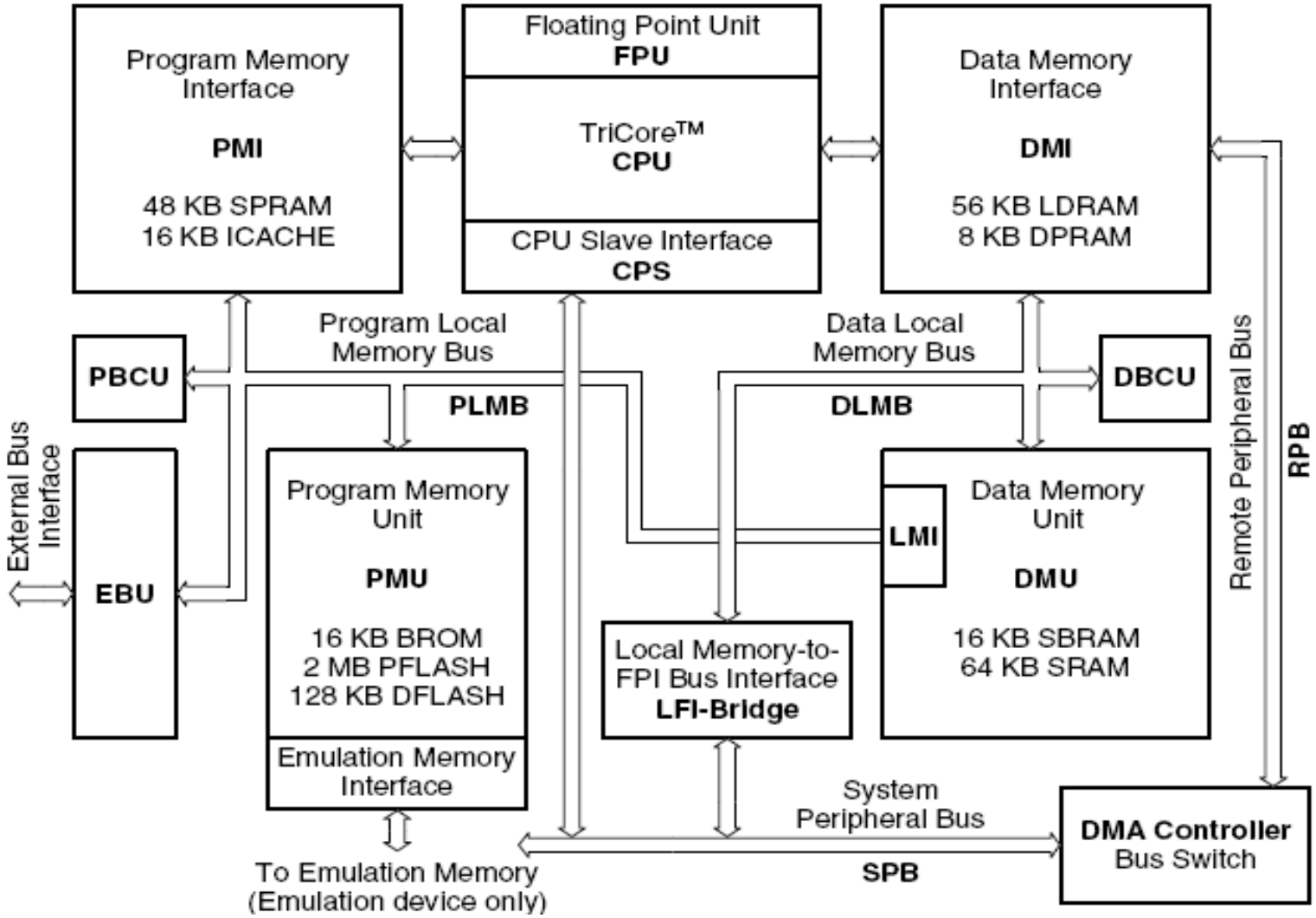
HRM case study -- TC1796 (µController)

- **Advanced 32-bit TriCore™-based Next Generation Microcontroller for Real-Time Embedded systems**
  - Automotive control systems
  - Industrial robotic control
- **Features**
  - Super-scalar TriCore CPU
    - Superior real-time performance
      - **Efficient interrupt handling**
    - 4 stage pipeline
    - DSP capabilities
    - 150 MHz operational frequency

HRM case study -- TC1796

- **Complex memory architecture**
  - Embedded Program Memory (>2MByte): PMI (ICACHE, SPRAM), PMU (BROM, PFLASH, DFLASH)
  - Data Memory : DMI(LDRAM, DPRAM), DMU(SRAM, SBRAM)…
  - Extendable memory using an external bus
- **High performance triple bus structure**
  - Two Local memory busses (64-bit) to program and data memories
  - 32-bit system peripheral bus to on-chip peripherals
  - 32-bit remote peripheral bus to external peripherals
  - Independent bus control units
- **16-channel DMA controller…**

Block diagram of the TC1796 CPU-Subsystem



HRM case study -- TC1796 CPU-Subsystem

See models examples

on www.papyrusuml.org

HRM application -- HW emulation

- **UML models have now a precise standard XML representation (using the XMI definition).**
- **Then, all model manipulations and transformations can be easily done using widely known XML technologies.**
  - Eclipse plugins (EMF, UML2…), Acceleo…
- **The steps are:**
  - **Describe** the HW models in UML using HRM
  - **Parse** and Capture all the required HW properties
  - **Verify** coherency and completion
  - **Generate** the configuration file for the target emulation tool
  - **Simulate** the application software on the emulated HW

Examples of Possible Hw Emulators

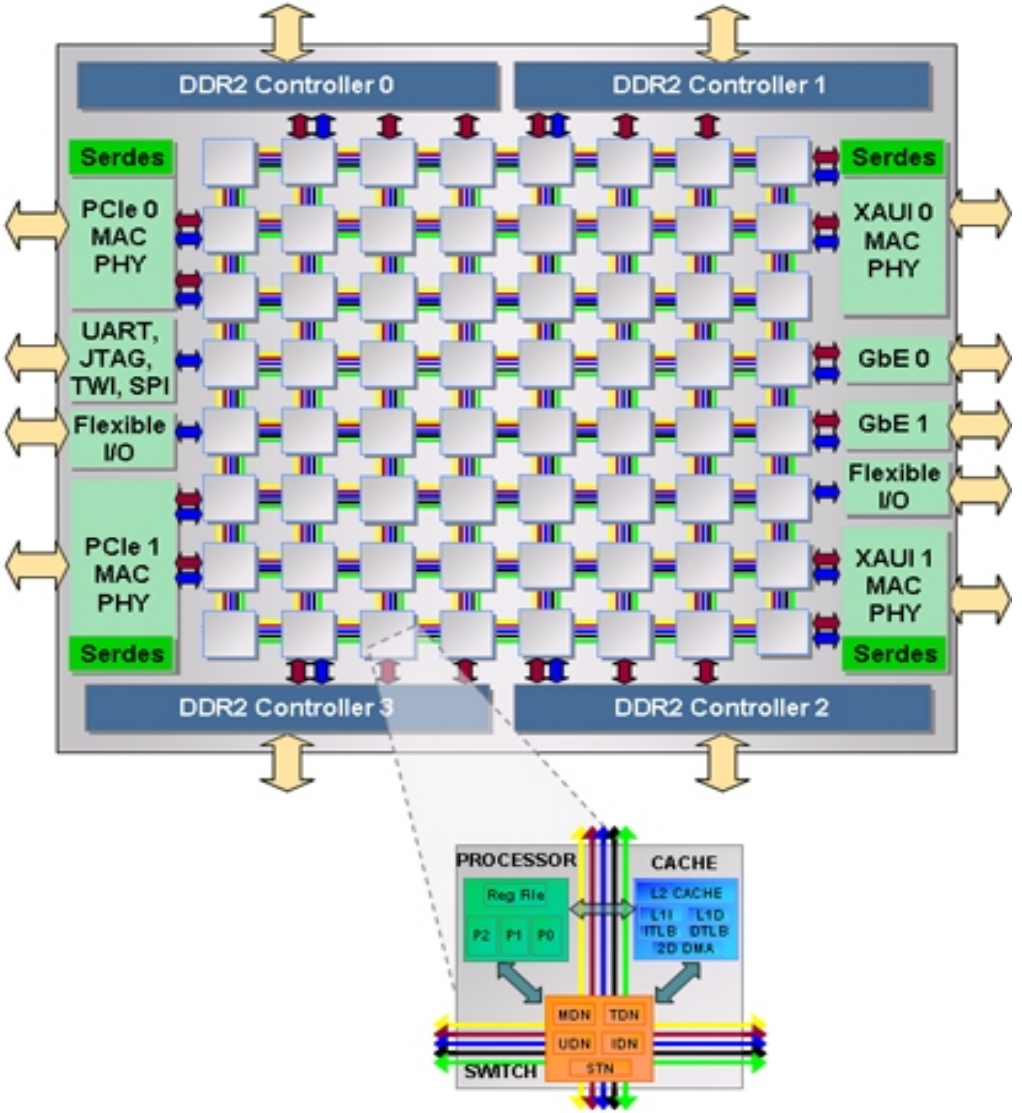
- Simics (Virtutech, www.virtutech.com/)
  - Support for most HW components
  - Functional and Performance simulation
  - Enable to run heavy software applications (e.g., linux)
  - Free for academics
- **Skyeye (www.skyeye.org/)**
  - Support for ARM-like processors, most of memories and peripherals
  - Functional simulation
  - Enable to run only light sw applications (E.g., µLinux and ARMLinux)
  - GPL
- **SimpleScalar (www.simplescalar.com/)**
  - Academic tool easy to extend
  - Performance simulation
  - Run C code

Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- Part 3
  - The timing model
- Part 4
  - A component model for RT/E
- Part 5
  - Platform modeling
- **Part 6**
  - **Repetitive structure modeling**
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

Embedded System Hardware is now *Repetitive*

- **Multicore**
  - Today 4 to 8 cores
  - Tomorrow: 16 to 64 cores
- **Processor meshes**
  - Ex: Tilera Tile64
- **SIMD units**
  - Data parallelism



The Future of Embedded Applications is *Parallel*

- **Multimedia**
  - Video coding/decoding
  - HDTV
- **Detection systems**
  - Radar
  - Sonar
- **Telecom**
  - Software radio
  - Wireless communications

Computation models

- Multidimensional signal processing
- Stream processing
- Data parallelism

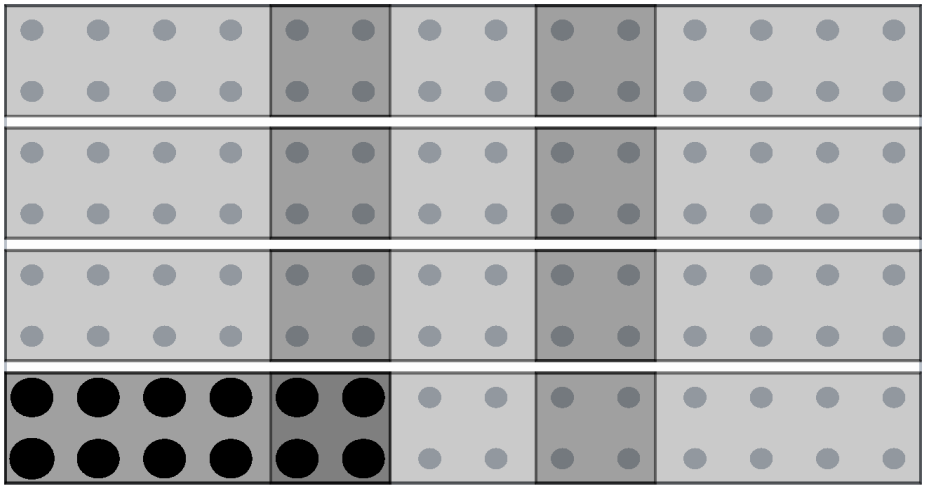
Repetitive Structure Modeling

- **Motivation**
  - ***Multidimensional regular parallelism***
    - Nested loops
    - Multiprocessor Systems
  - ***Compact representation***
    - Application
    - Hardware platform
    - Association
- **Form**
  - New notation / stereotypes

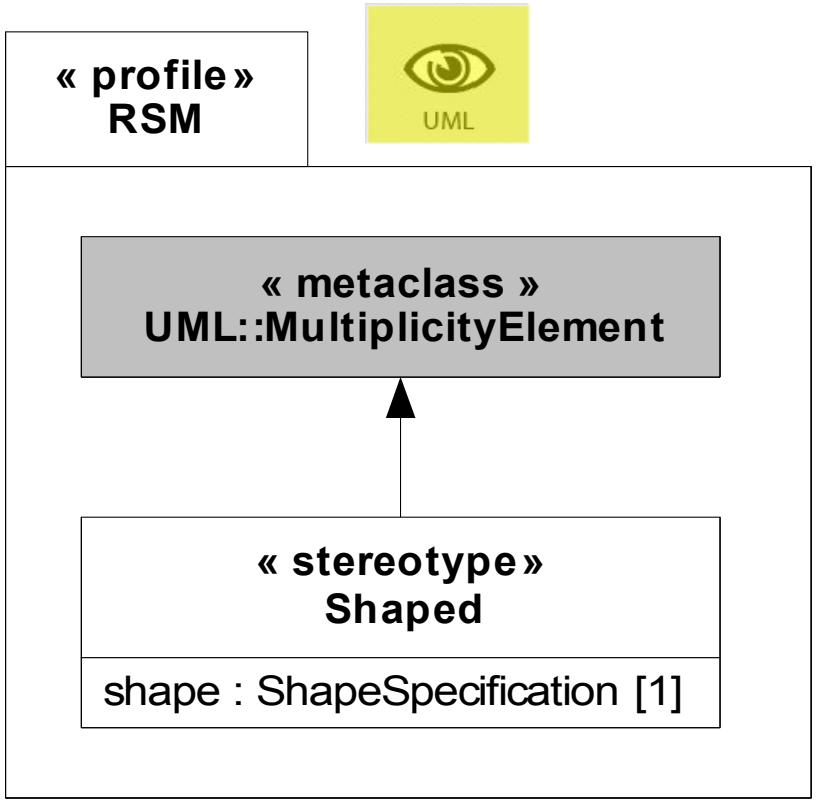
Concepts of Repetitive Structure Modeling

- Concepts
  - **Shape** (extension of *multiplicity*)
    - To model multidimensional arrays
  - **Link topology** (extension of *connector* and *allocate*)
    - To model the topology of the links between multidimensional arrays
    - Pattern-based regular topologies

- **Basic idea: *regular tiling* of multidimensional arrays by multidimensional sub-arrays**
  - Regular spacing of points inside a tile
  - Regular spacing of tiles
  - Inherits from the Array-OL language



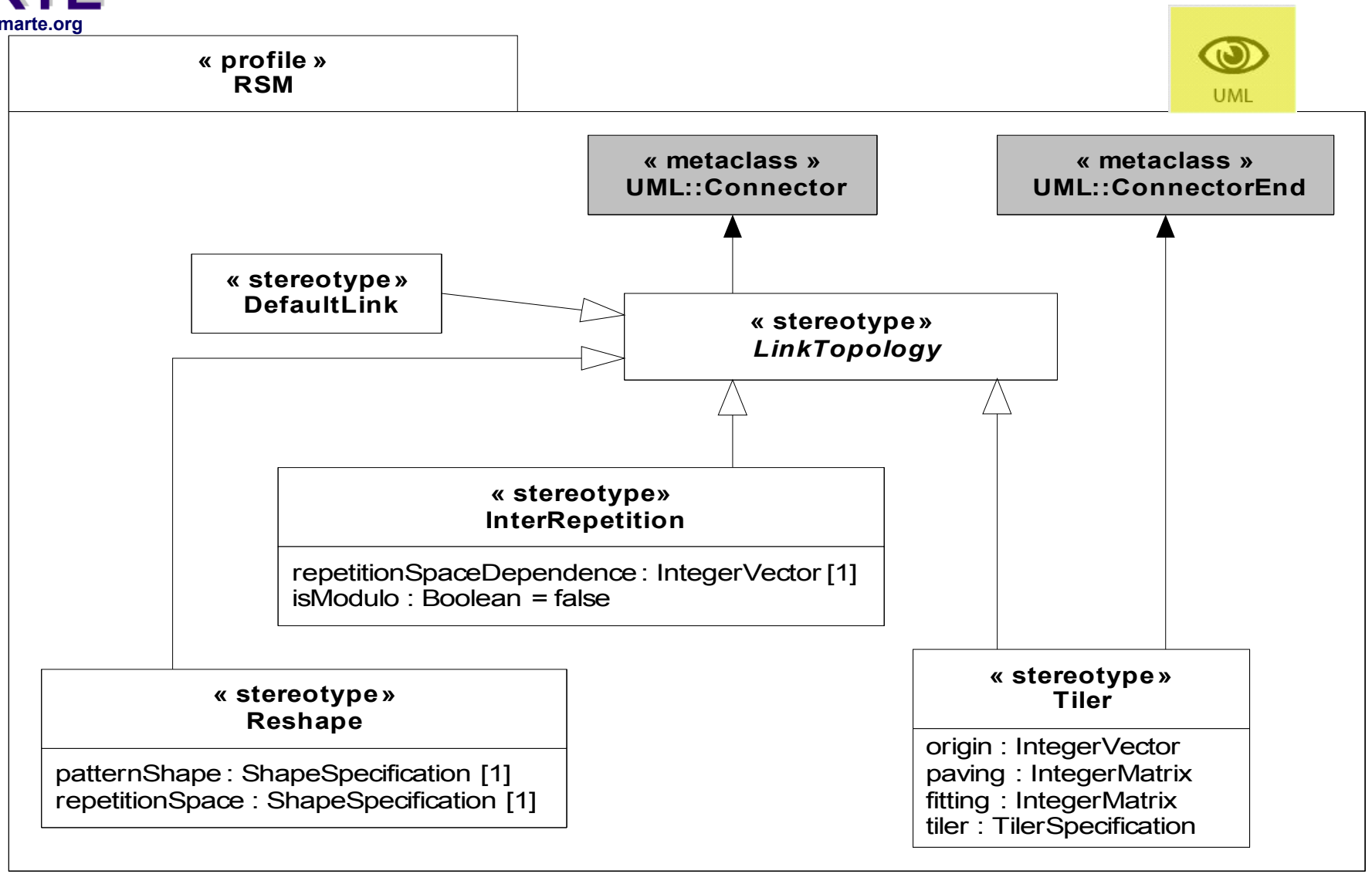
Shape Modeling



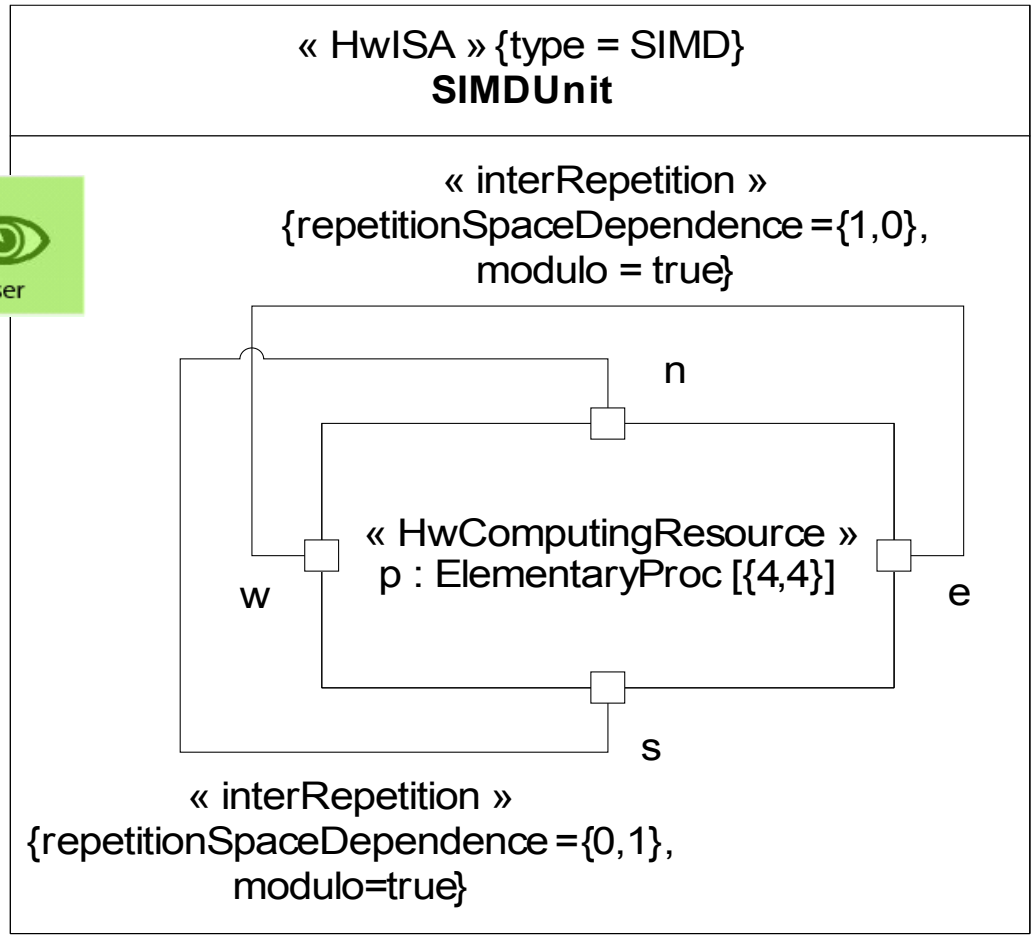
- **New notation**
  - Refinement of the multiplicity notation
  - Vector of UnlimitedNaturals
- **Examples**
  - 16 → {4,4}
  - * → {512,128,*}

« dataType » « collectionType » {collectionAttrib = size} **ShapeSpecification**
size : UnlimitedNatural [0..*]

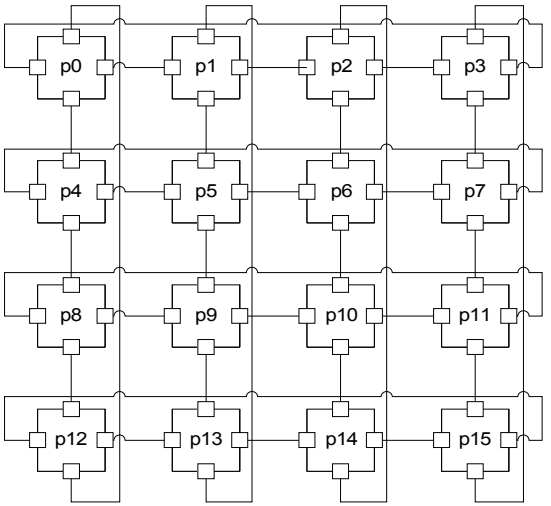
Link Topology Modeling



Hardware Platform Example



- **SIMD unit**
  - 16 processors
- **Topology**
  - Toroidal 4×4 grid
  - Bidirectional connections
    - North-South
    - East-West



Tiling an Array

« stereotype » Tiler
origin : IntegerVector paving : IntegerMatrix fitting : IntegerMatrix tiler : TilerSpecification

« dataType » « tupleType » TilerSpecification
origin : IntegerVector paving : IntegerMatrix [1] fitting : IntegerMatrix

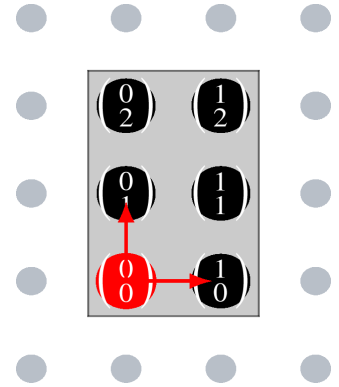
- **Needed shapes**
  - Array shape
  - Pattern shape
  - Repetition space shape
- **Tiler**
  - *Fitting*: regular spacing of the points of the tiles
    - Index ***i***
    - Scanning the pattern
  - *Paving*: regular spacing of the tiles
    - Index ***r***
    - Scanning the repetition space

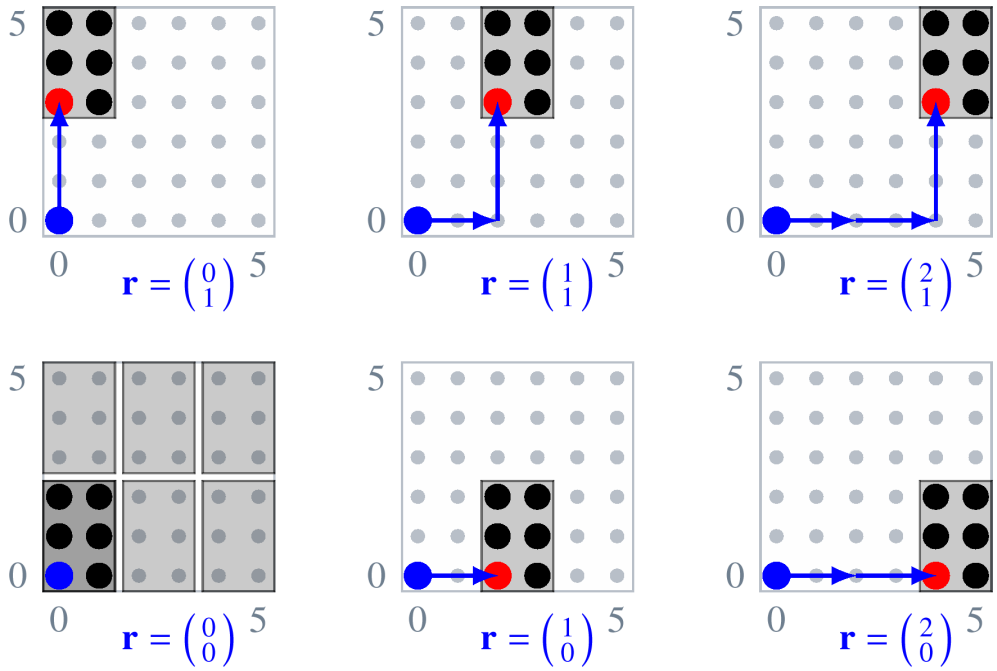
origin+(paving fitting).$\begin{pmatrix} r \\ i \end{pmatrix}$*mod* array.shape

Graphical Interpretation of a Tiler (1/2)

$$F = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathbf{o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad P = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$$

$$\mathbf{s}_{pattern} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \quad \mathbf{s}_{array} = \begin{pmatrix} 6 \\ 6 \end{pmatrix} \quad \mathbf{s}_{repetition} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$



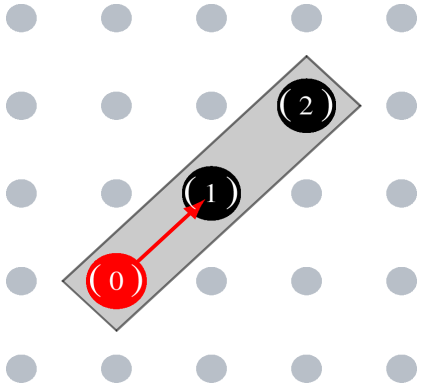


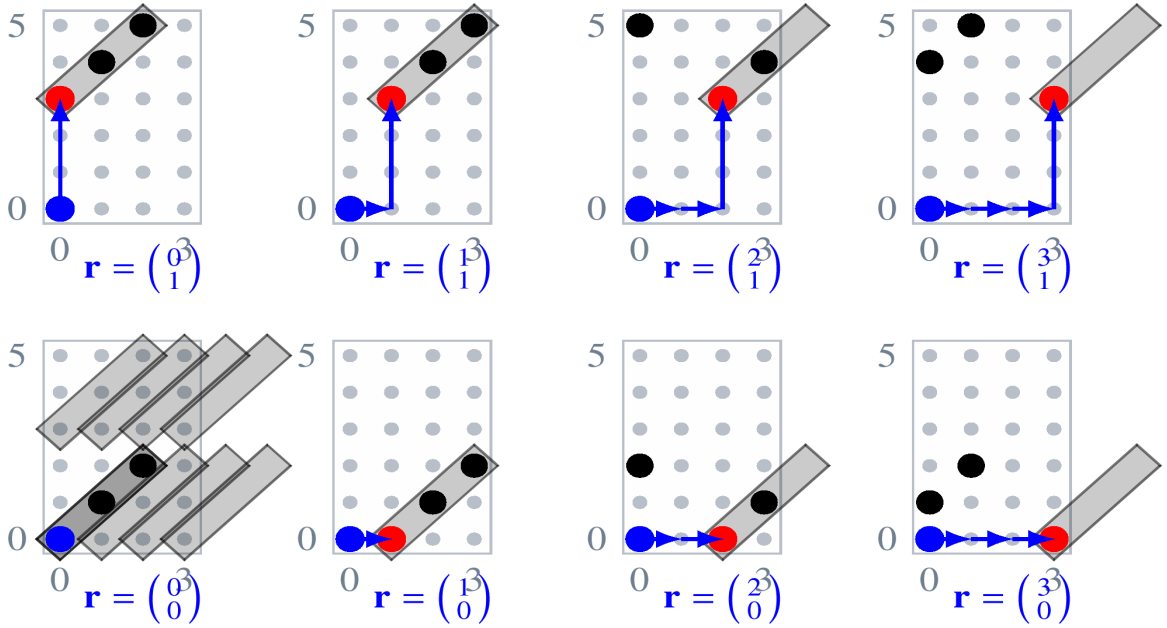
- **Fitting**
  - Column vectors
    - Basis of the tile
  - Pattern shape
    - Bounds of the fitting
- **Paving**
  - Column vectors
    - Basis of the placement of the tiles
  - Repetition space
    - Bounds of the paving
  - Origin
    - Coordinates of the reference point of the reference tile

Graphical Interpretation of a Tiler (2/2)

$$F = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \mathbf{o} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$$

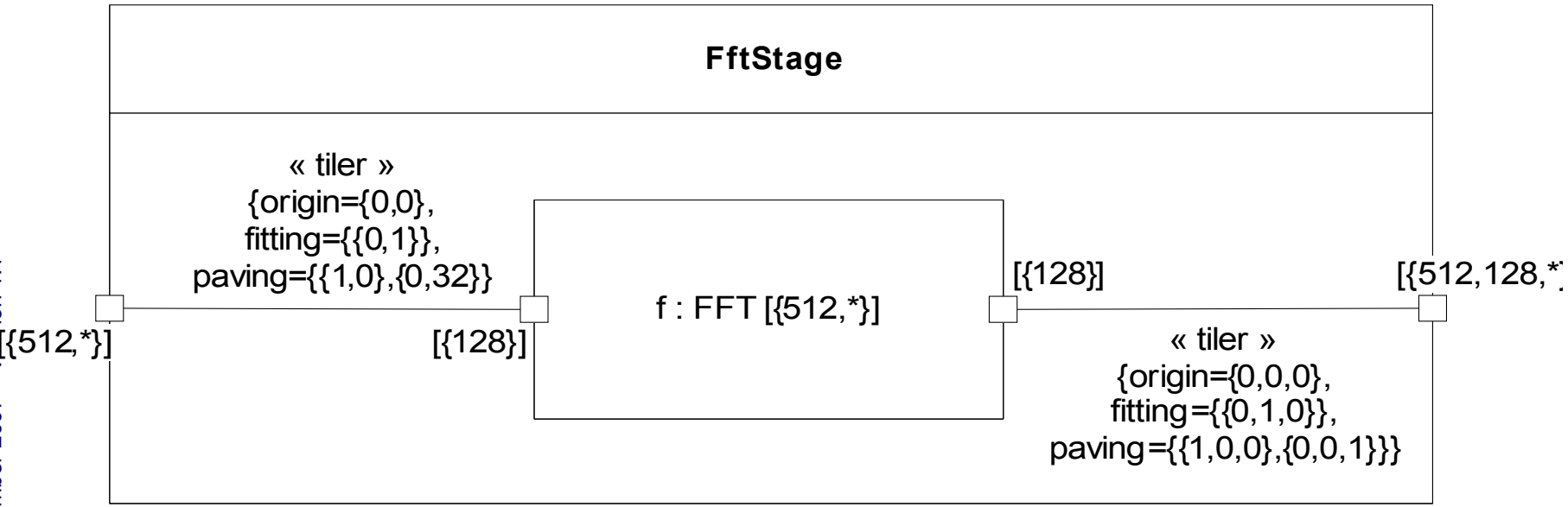
$$\mathbf{s}_{pattern} = \begin{pmatrix} 3 \end{pmatrix} \quad \mathbf{s}_{array} = \begin{pmatrix} 4 \\ 6 \end{pmatrix} \quad \mathbf{s}_{repetition} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$





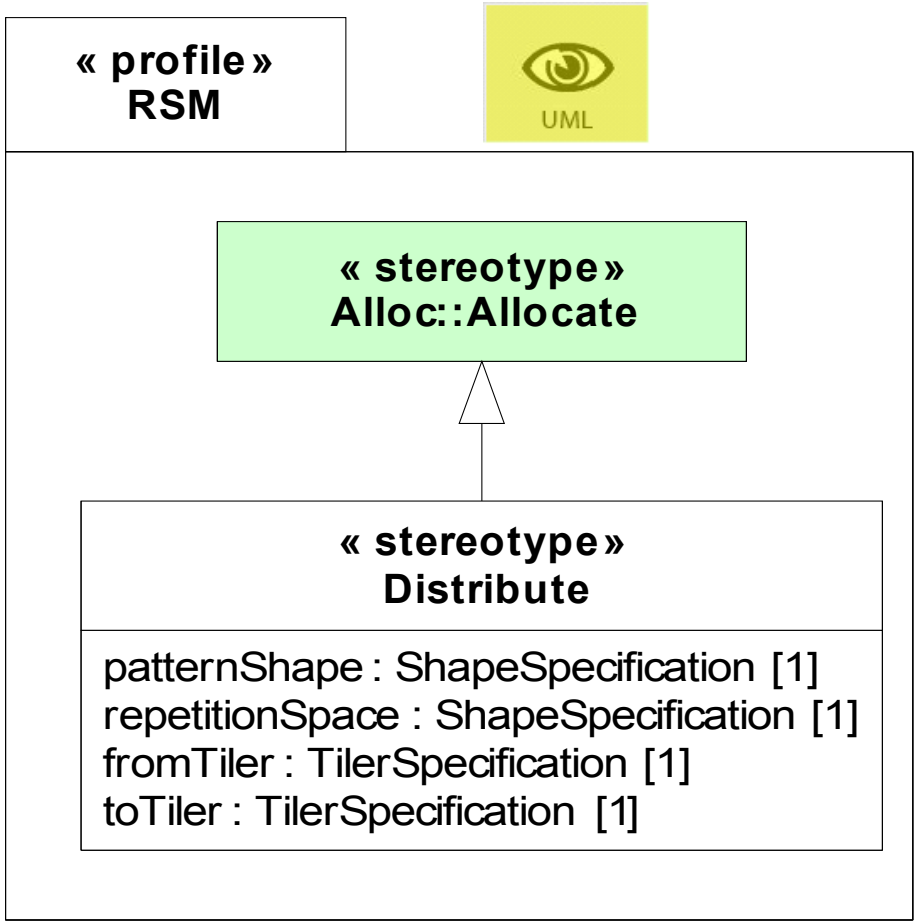
- **Fitting**
  - Column vectors
    - Basis of the tile
  - Pattern shape
    - Bounds of the fitting
- **Paving**
  - Column vectors
    - Basis of the placement of the tiles
  - Repetition space
    - Bounds of the paving
  - Origin
    - Coordinates of the reference point of the reference tile

Application Example



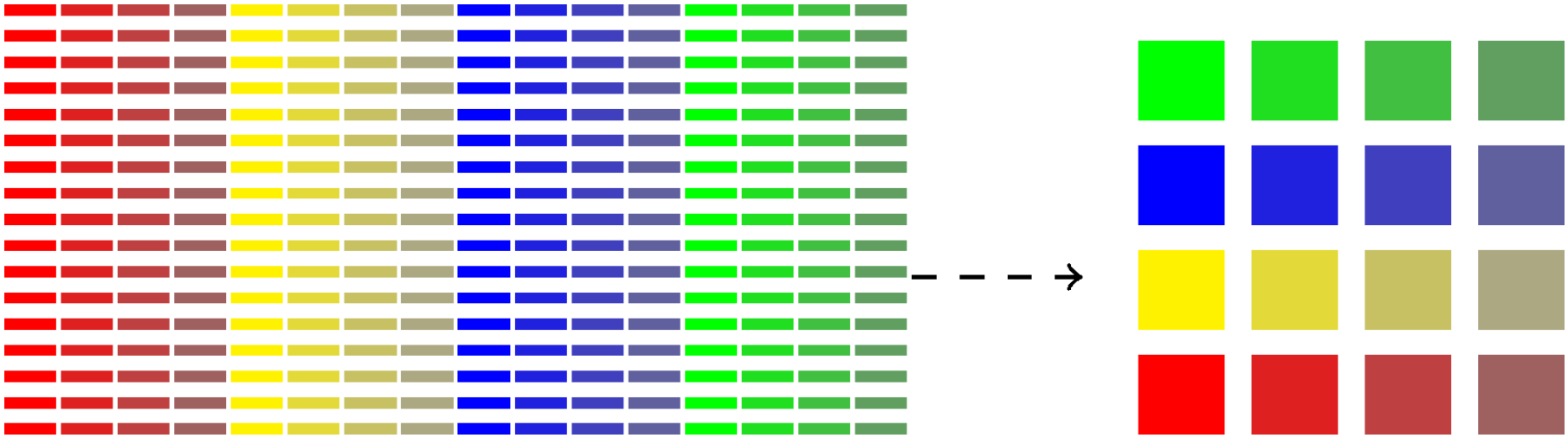
- **Samples from 512 hydrophones around a submarine**
  - Shape of the input data = 512×∞
- **Repetition of FFTs**
  - For each hydrophone
  - Sliding window of 128 samples every 32 time steps

Distribution



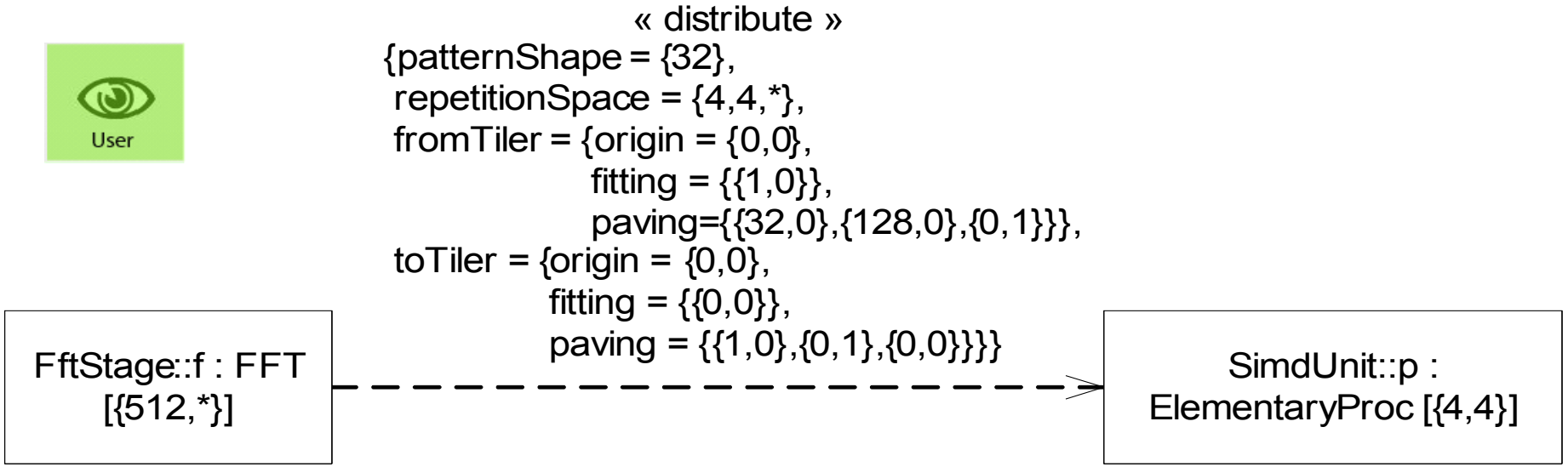
- **Refinement of Allocation**
- **Similar to the reshape stereotype of the connectors**
- **Principle**
  - Tiling both ends
    - Two tilers
  - With the same tiles
    - One pattern shape
    - One repetition space
- **Power of expression**
  - At least all HPF data distributions

Distribution Example



- **Distribution of the FFT computations to the SIMD unit**
  - No spatial distribution of the infinite dimension (time steps)
  - Bloc distribution of the 512 FFTs for each time step
    - Size of the bloc = 32
    - On the 16 elementary processors

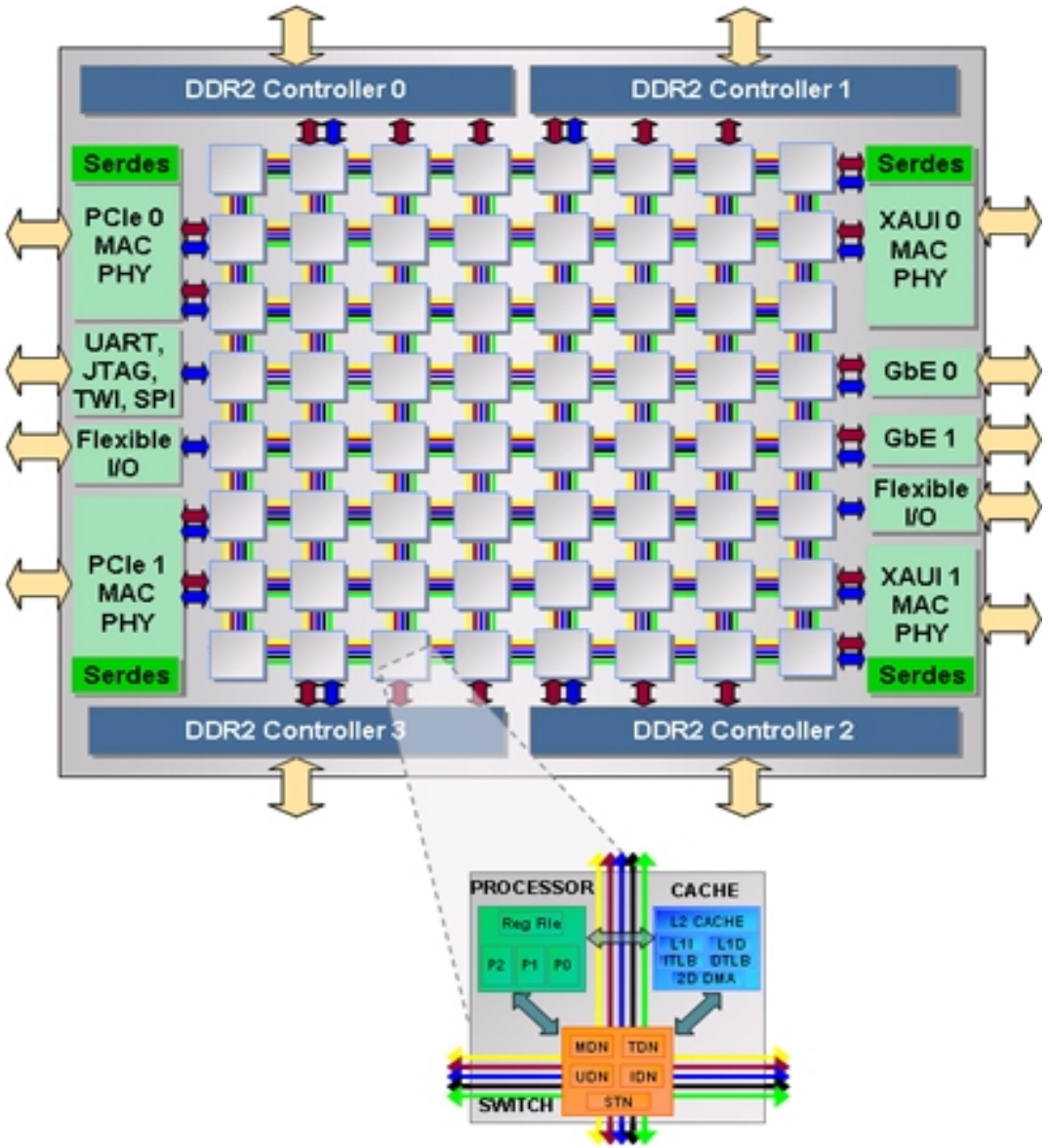
Distribution Example



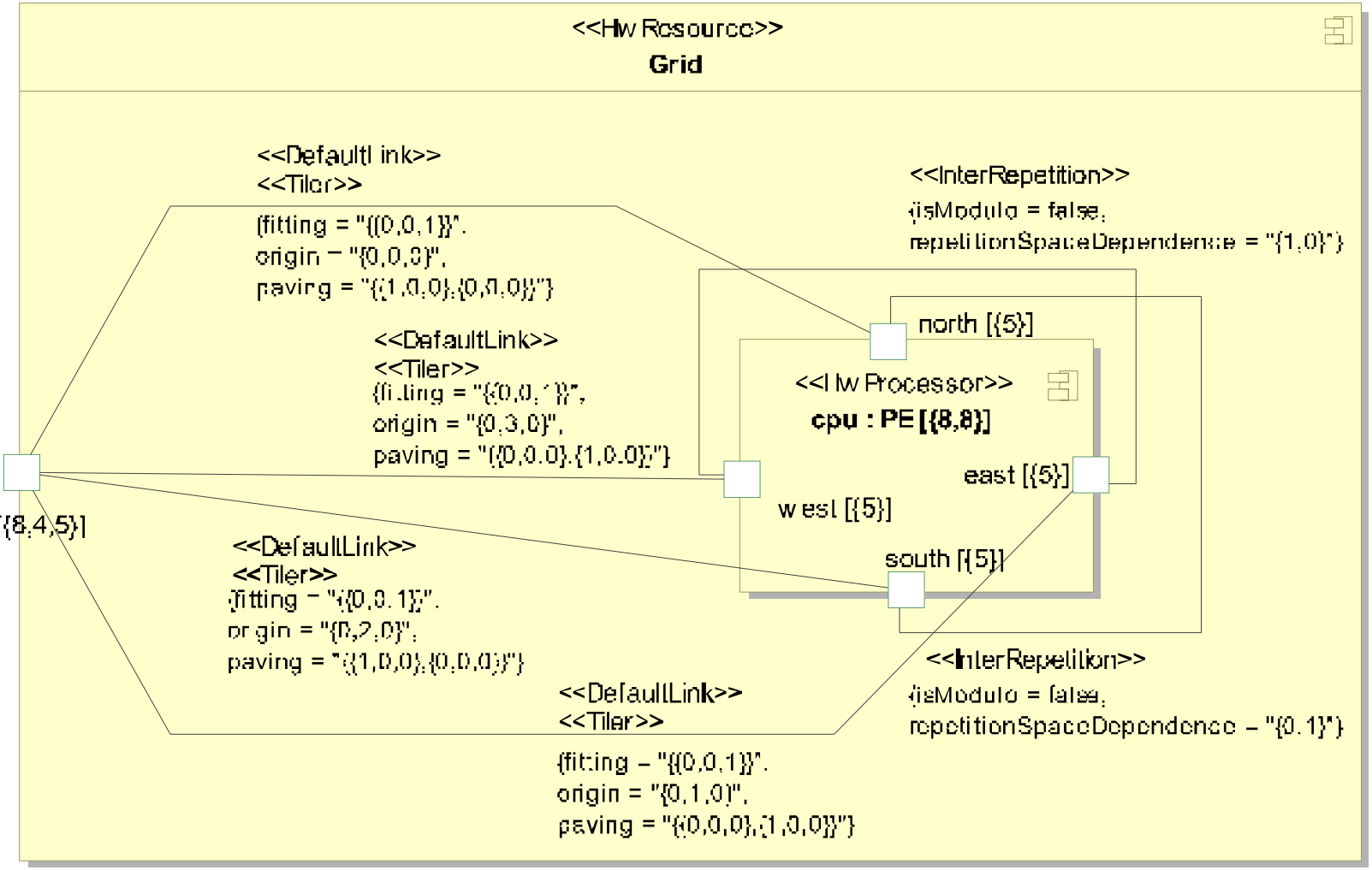
- **Distribution of the FFT computations to the SIMD unit**
  - No spatial distribution of the infinite dimension (time steps)
  - Bloc distribution of the 512 FFTs for each time step
    - Size of the bloc = 32
    - On the 16 elementary processors

Complex Hardware Example:Tile64

- **Challenge**
  - Model the architecture
  - In the most compact way
- **Proposal**
  - 8x8-repetition of the processing element
  - 4-repetition of the DDR2 controller
  - Factorization of the ports

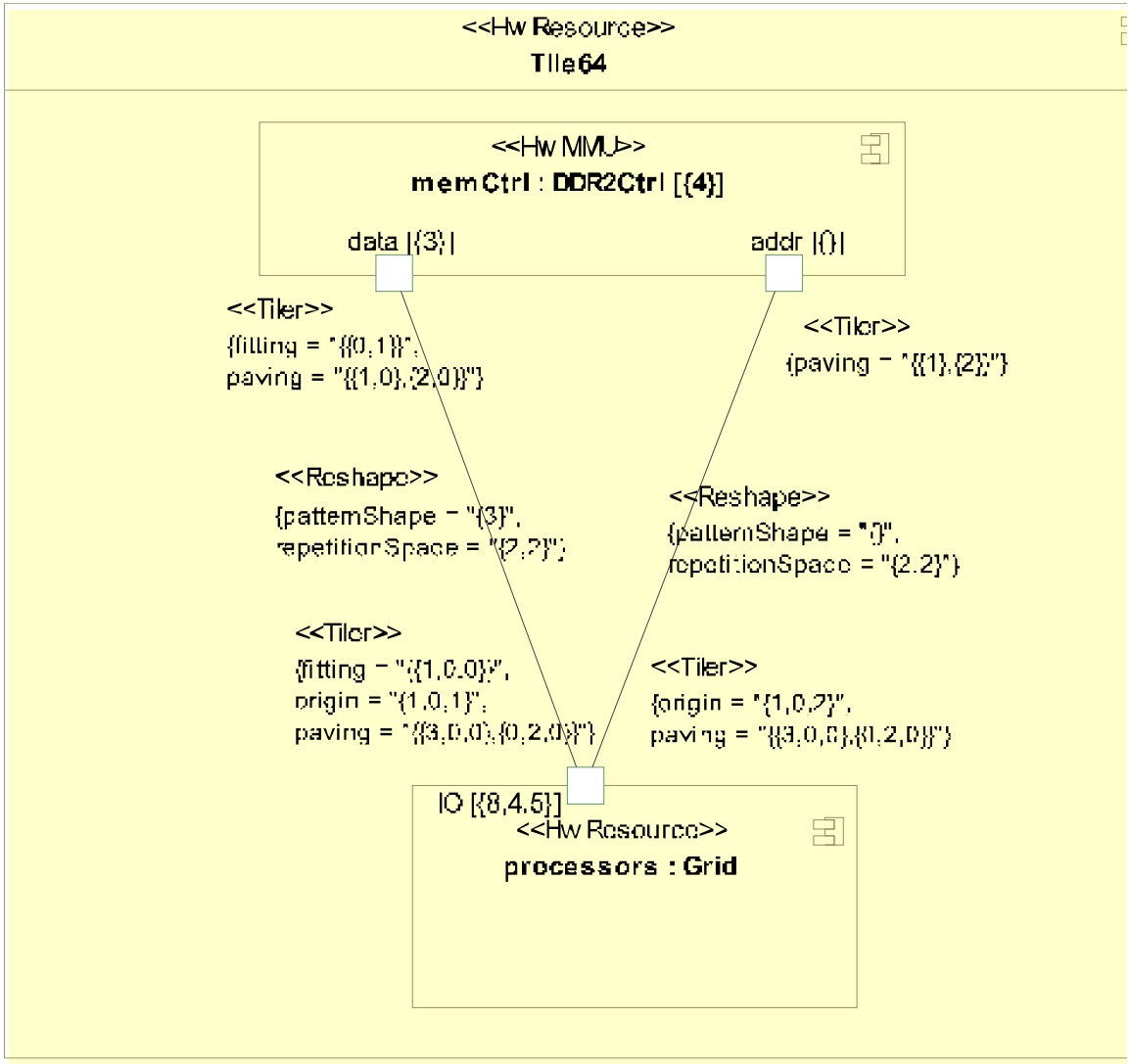


Processing Element Repetition



User

DDR2 Controler Connection to the Grid



User

Conclusion on RSM

- **General mechanism to handle**
  - Multidimensional structures (arrays)
  - Tiling by sub-structures (non orthogonal or sparse tiles possible)
  - Links between such structures (cyclic or non cyclic connection patterns possible)
- **Necessary to handle massive regular parallelism**
  - Compactness of the model
  - Efficiency, maintainability, readability
- **Relations with the rest of MARTE**
  - Uses VSL
  - Benefits from the component model (flow ports)
  - Applies to both application and hardware components
  - Extends allocation
- **Limitations**
  - Handles only arrays (no fancier shapes)
  - Would benefit from a custom (visual) tiler editor
    - Under development

Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- Part 3
  - The timing model
- Part 4
  - A component model for RT/E
- Part 5
  - Platform modeling
- Part 6
  - Repetitive structure modeling
- Part 7
  - **Model-based analysis for RT/E**
- Part 8
  - MARTE and AADL
- Part 9
  - Conclusions

Goals in Non-Functional (or Quantitative) Analysis

It offers a mathematically-sound way to calculate NFPs of interest based on other available NFPs and the system behavior

- **Different Goals for Evaluate & Verify System Architectures**
  - Point evaluation of the output NFPs for a given operating point defined by input NFPs
  - Search over the parameter space for feasible or optimal solutions
  - Sensitivity analysis of some output results to some input parameters
  - Scalability analysis: how the system performs when the problem size or the system size grow.

MARTE Features for Quantitative Analysis

- **Improvements w.r.t. SPT**
  - Extend implementation and scheduling models
    - e.g. distributed systems, hierarchical scheduling
  - Extend the set of analysis techniques supported
    - e.g. offset-based techniques
  - Extend timing annotations expressiveness
    - Overheads (e.g. messages passing)
    - Response times (e.g. BCET & ACET)
    - Timing requirements (e.g. miss ratios and max. jitters)
- **New features w.r.t. SPT**
  - Support for sensitivity analysis
  - Improve modeling reuse and component-based design.
  - Support of the “Y-chart” approach: application vs. platform models

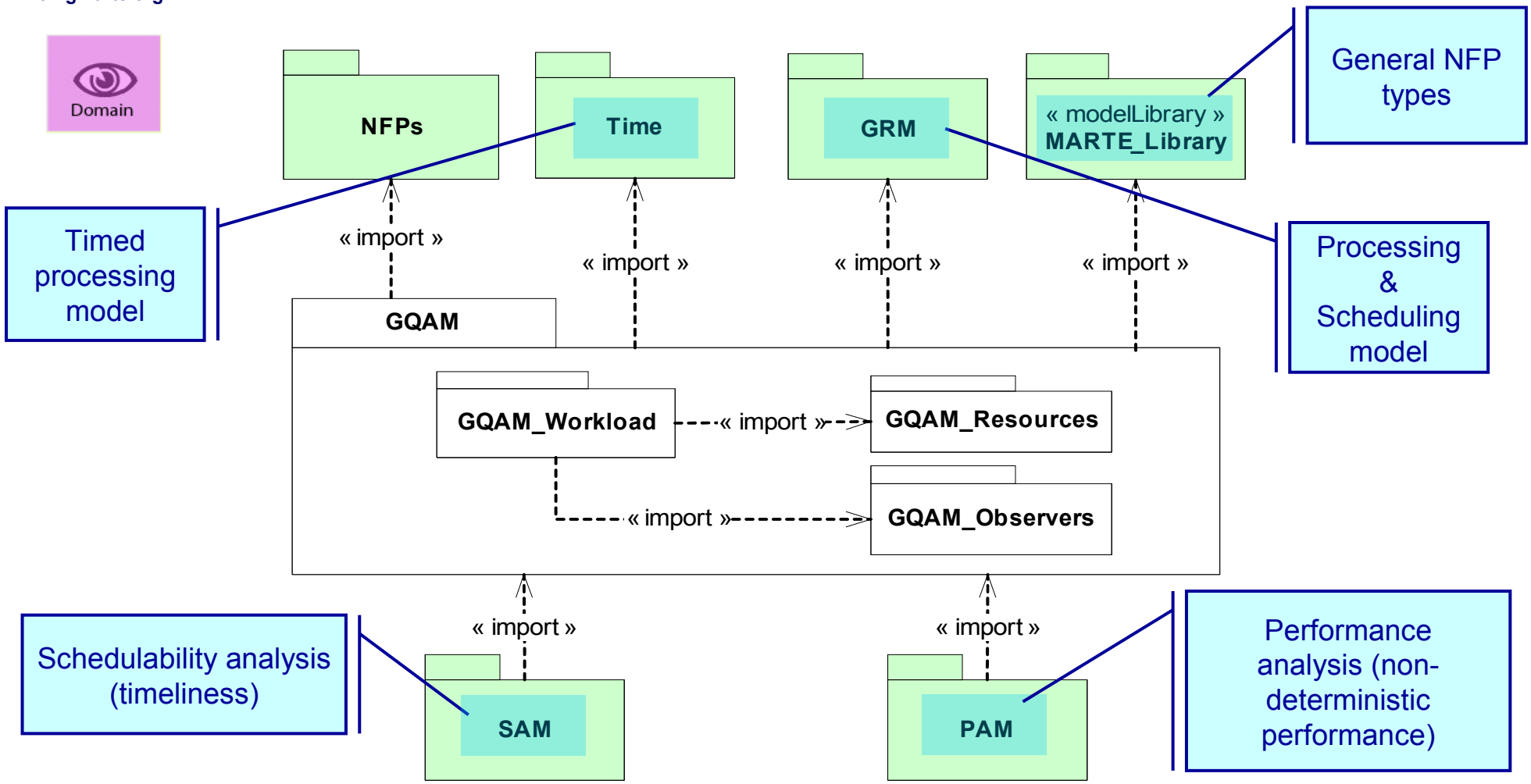
UML-Based Analysis Foundations

- **GQAM Profile factorizes common constructs and NFPs**
  - Stereotypes define "analysis" abstractions
    - workload events, scenarios,…
    - schedulable entities, shared resources, processing nodes, schedulers…
  - Stereotype attributes define pre-defined NFPs
    - e.g. event arrival patterns, end-to-end deadlines, wcet-bcet-acet,…
- **The analysis sub-profiles define model well-formedness rules**
  - It includes "constraints" to construct "analyzable" models, w.r.t…
  - "Analysis Model Viewpoints" (e.g., schedulability analysis viewpoint)
  - Specialized constraints must be refined by technique-specific approaches

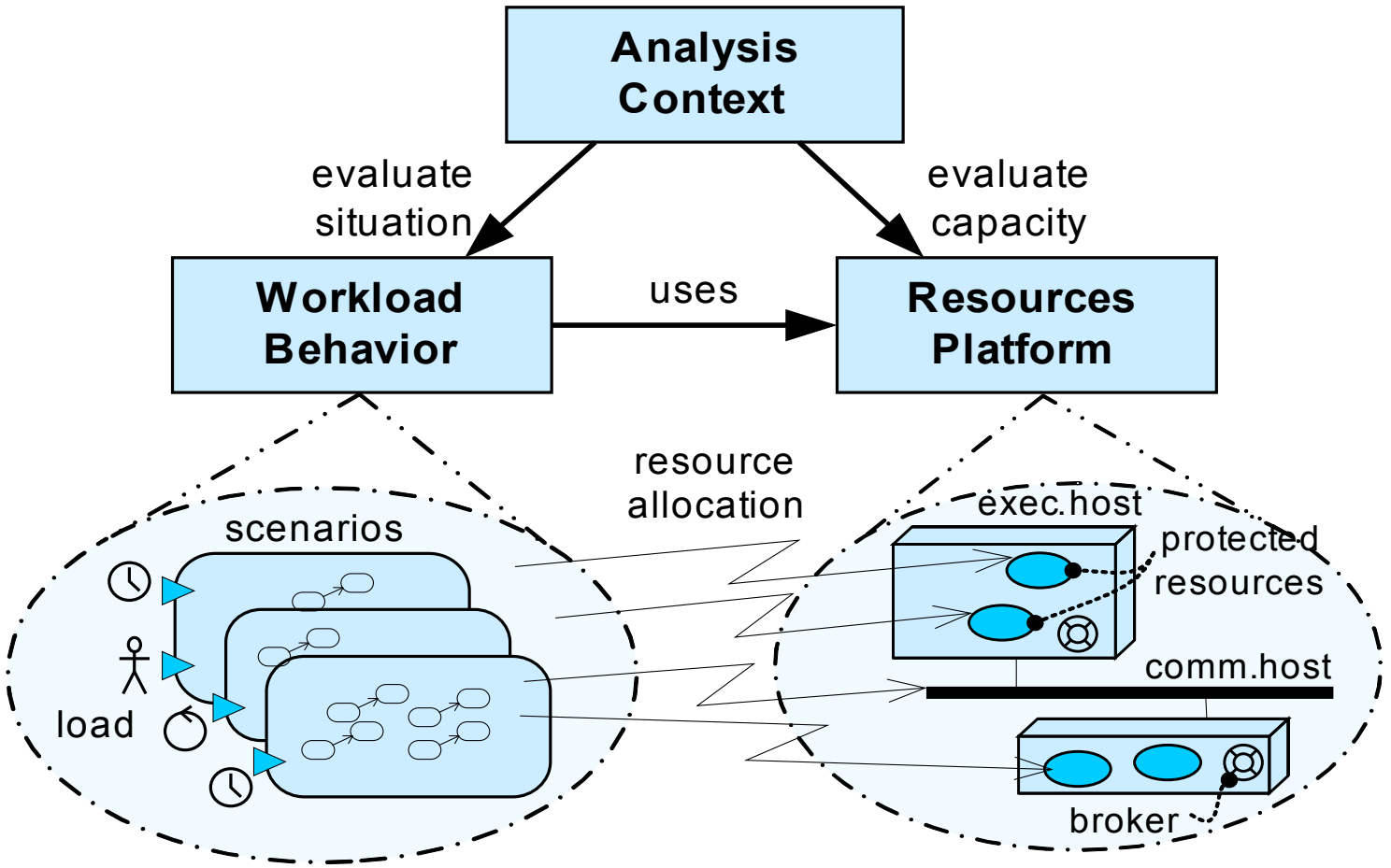
The MARTE analysis sub-profiles provide standard constructs to map UML models on well-established analysis techniques

➔MARTE "Foundations" and "GQAM" allow for extending to further techniques

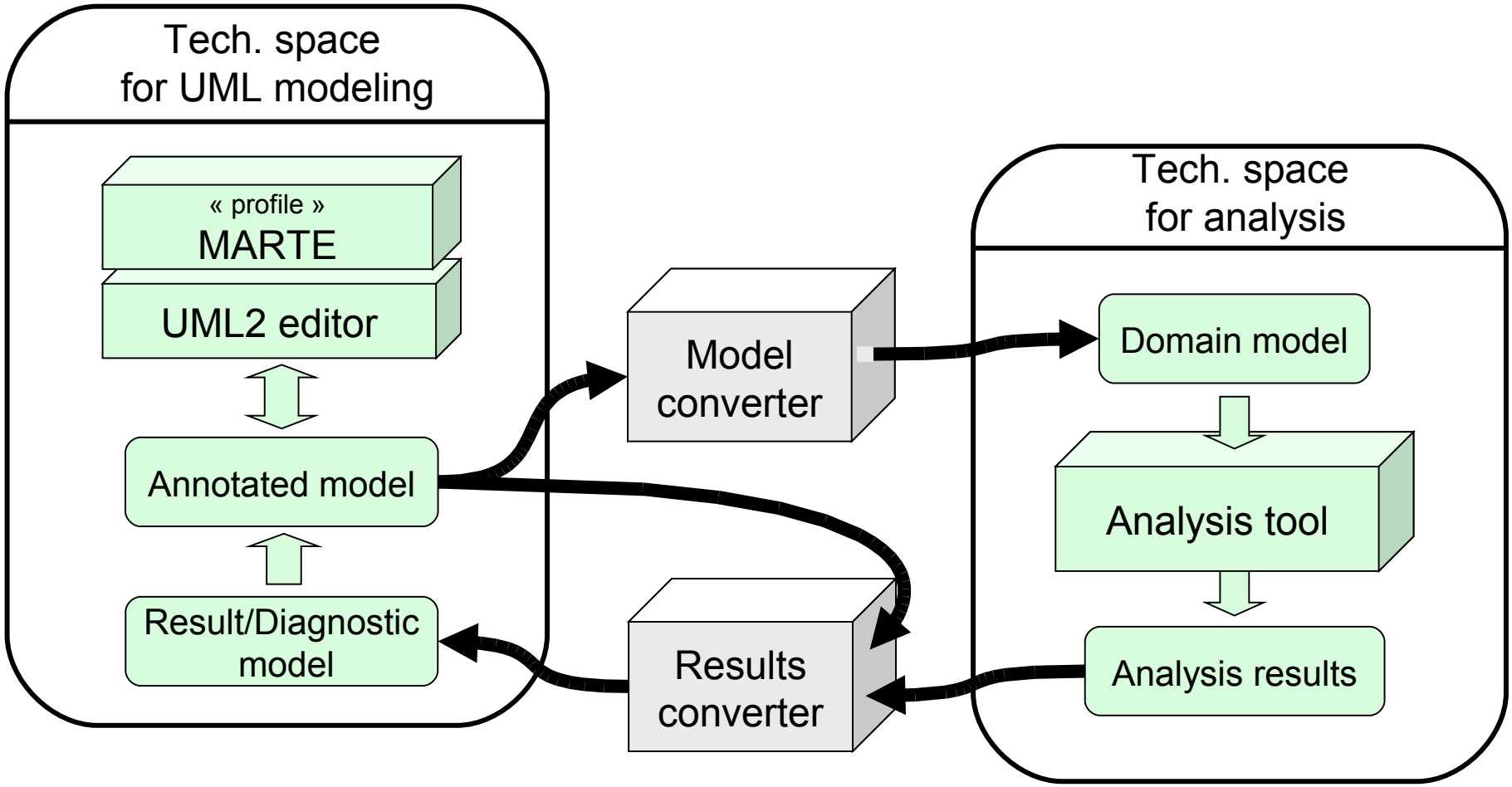
GQAM: Dependencies and Architecture



GQAM: Analysis Modeling Structure



Processing Schema for Analysis

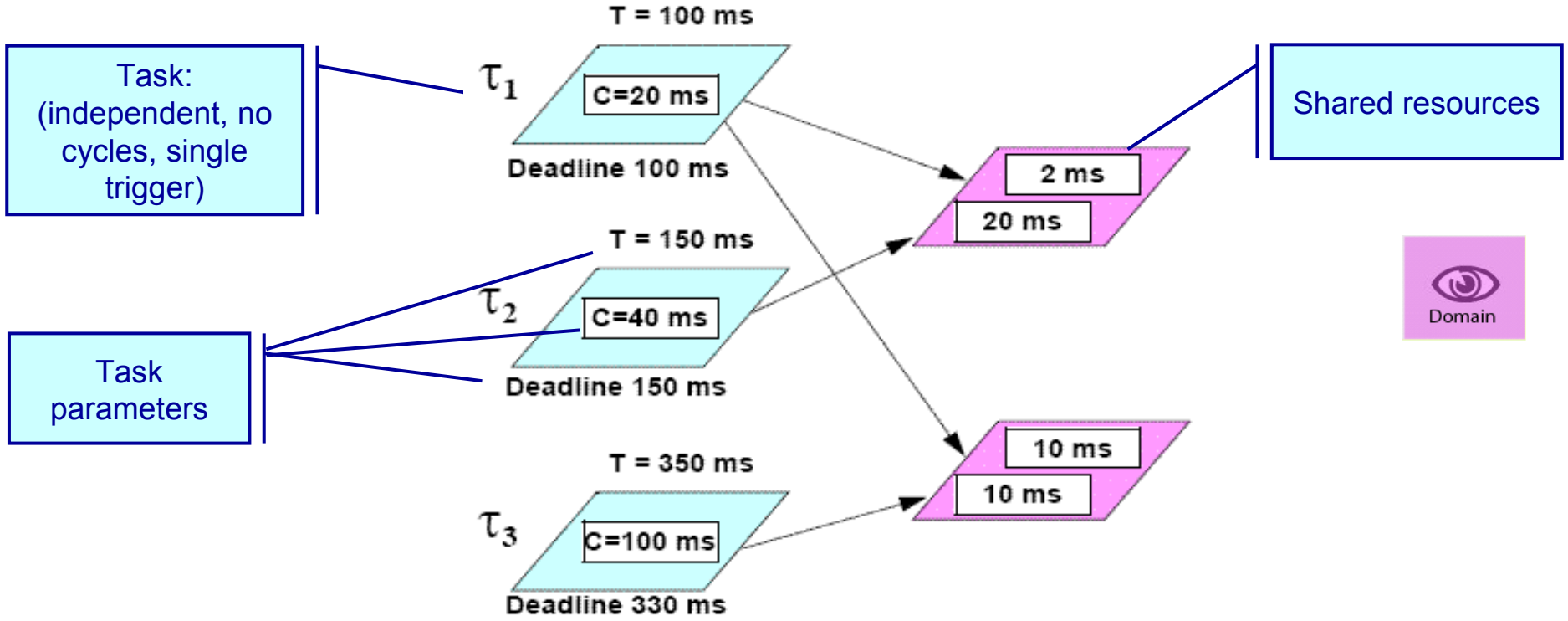


Schedulability Analysis

Provides the ability to evaluate time constraints and guarantee worst-case behavior of a system or particular piece of software

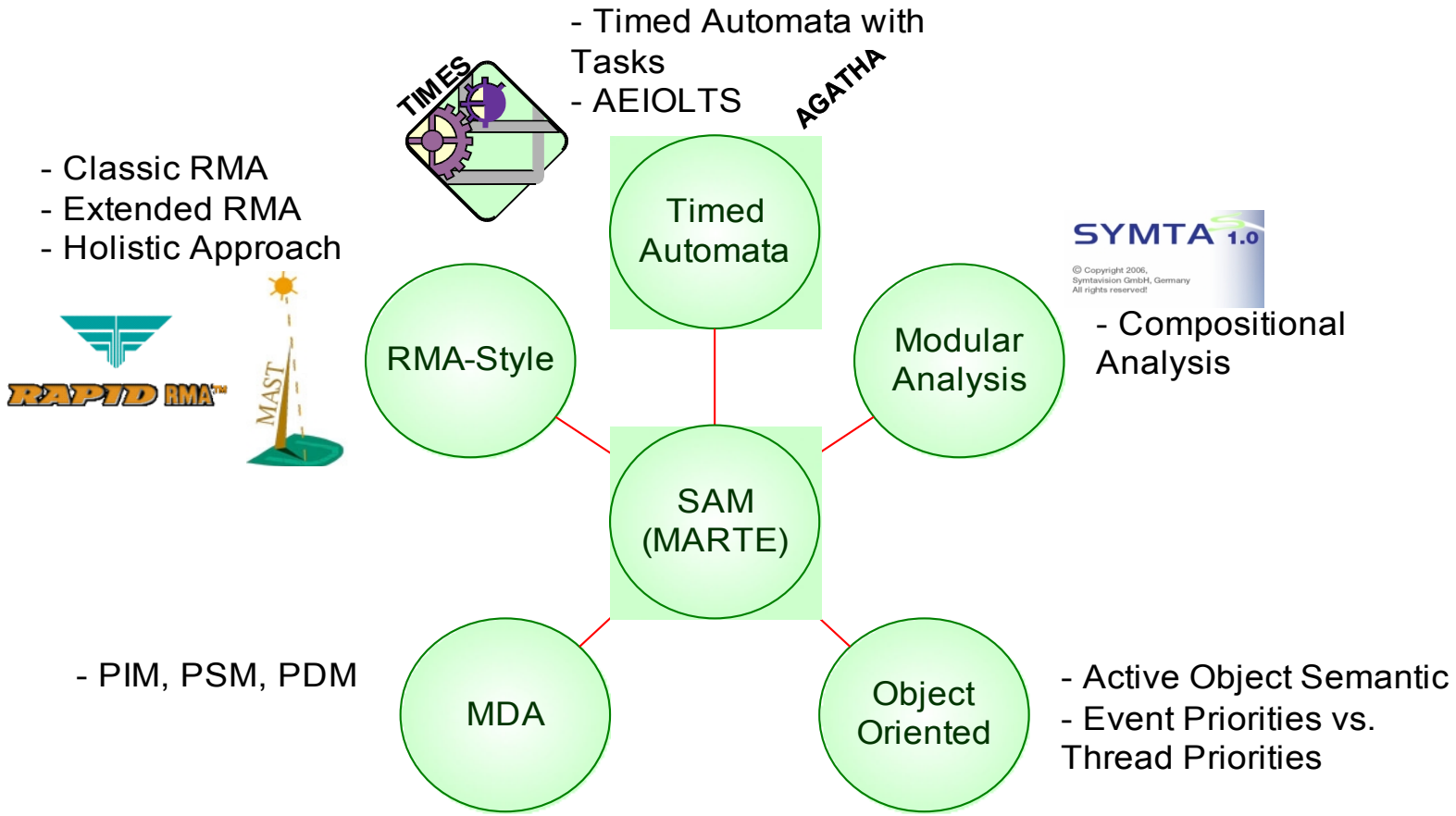
- **Schedulability analysis offers:**
  - Offline guarantees. E.g., worst-case latencies and worst-case resource usage.
  - At different development stages.
    - Early analysis: to detect potentially unfeasible real-time architectures.
    - Later analysis: to discover temporal-related faults, or to evaluate the impact of migrations (e.g., scheduling strategies).
- **Provide answer to questions such as for example...**
  - Will we miss any deadline if we switch a processor from a normal operation mode to a lower-consumption mode?
  - If yes, how can we modify task workloads for allowing our system to still work?

A Simple Example (Classical Scheduling Theory)



- **Three main analysis approaches for verify timeliness:**
  - Critical instant calculation
  - Utilization bound test
  - Response time calculation

SAM: Integration Different Approaches

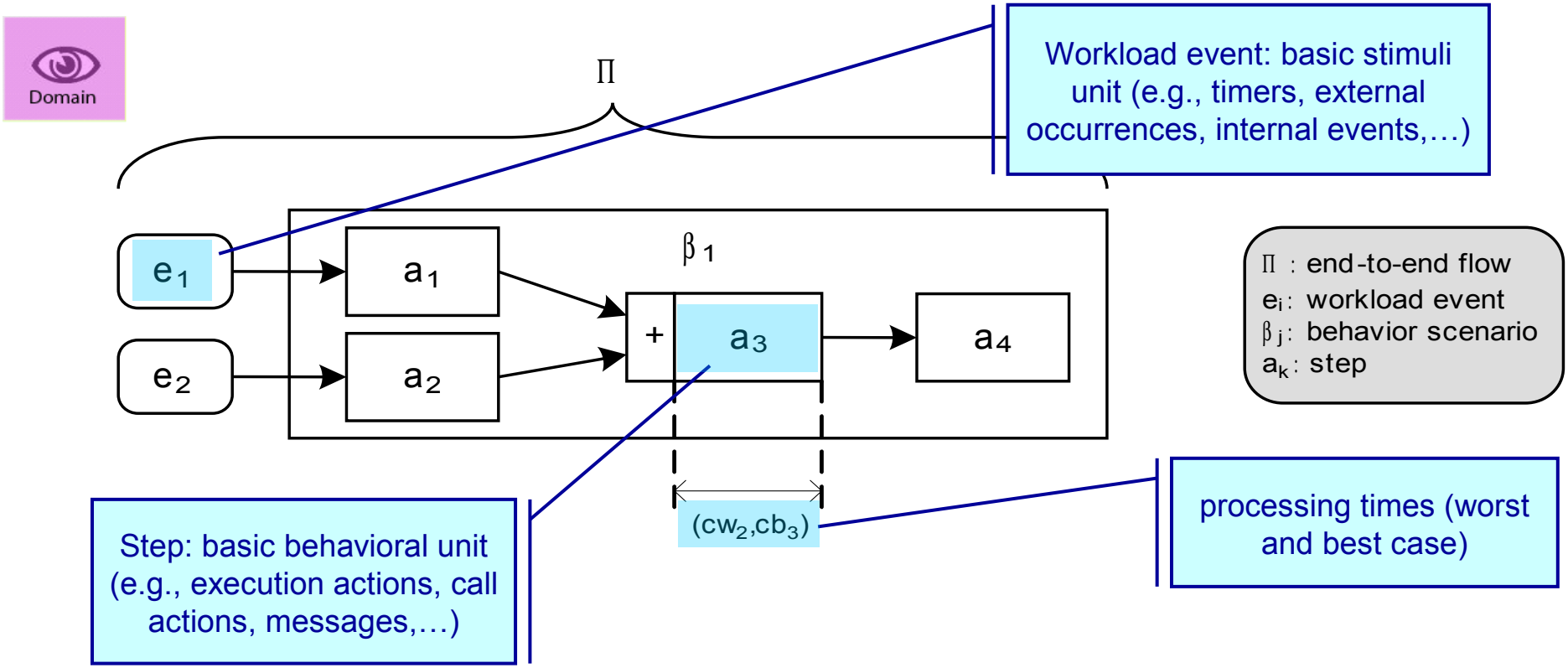


Other Sched. Analysis tools: Livedevices' Real-Time Architect, CoMET from VaST, Vector's CANAlyzer…

SAM: The Notion of End-To-End Flow

An "End-To-End Flow" is the basic workload unit to be evaluate by by schedulability analysis tools.

→ An end-to-end flow refers to the entire causal set of steps triggered by one or more external workload events.

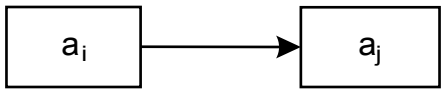


SAM: Precedence Relations

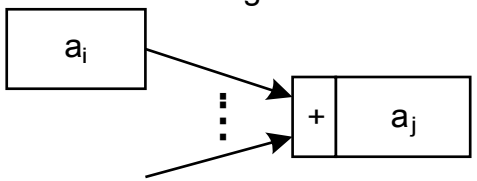
Execution and communication steps may be causally related by one of the following precedence relations:

Domain

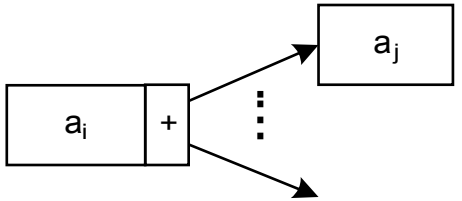
Sequencial



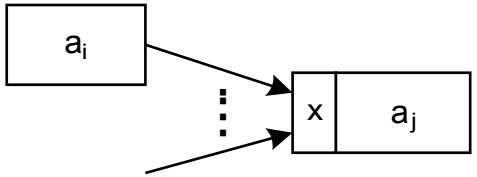
Merge OR:



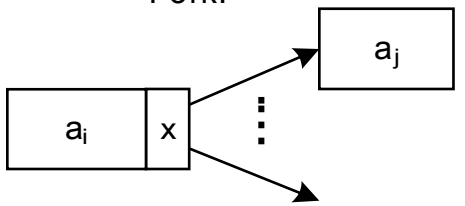
Decision OR:



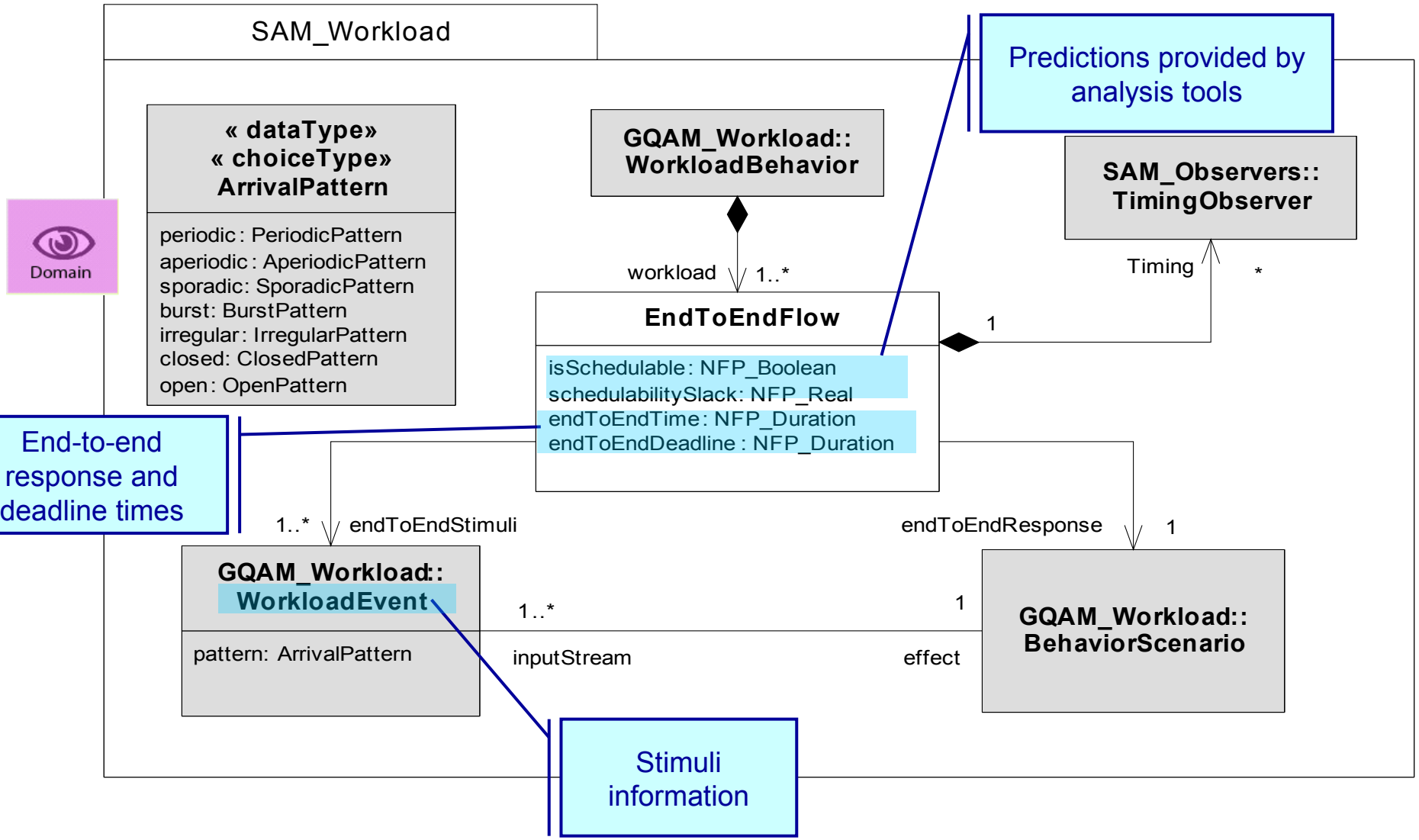
Join:



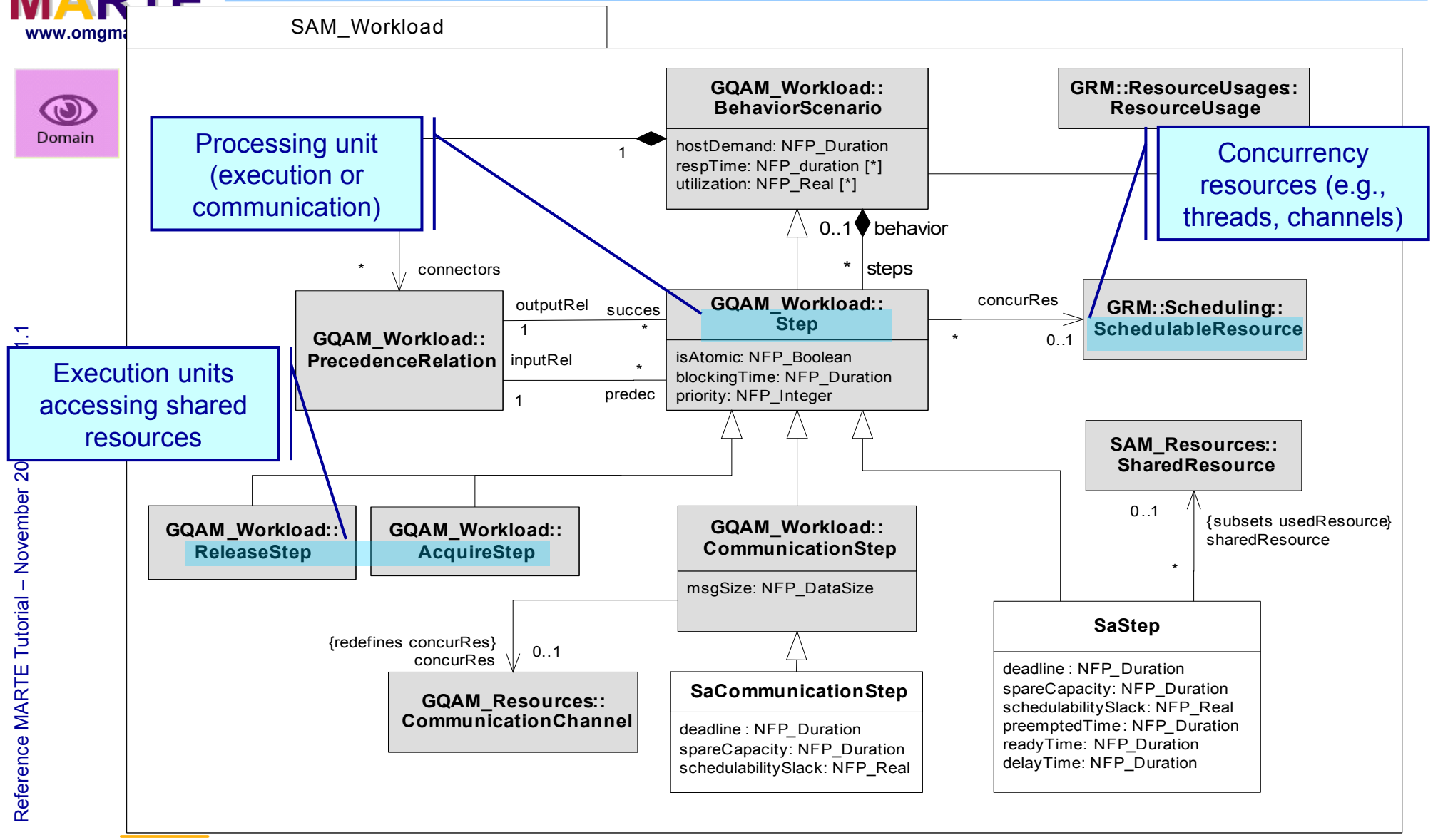
Fork:



SAM: Workload Domain Metamodel (end-to-end)



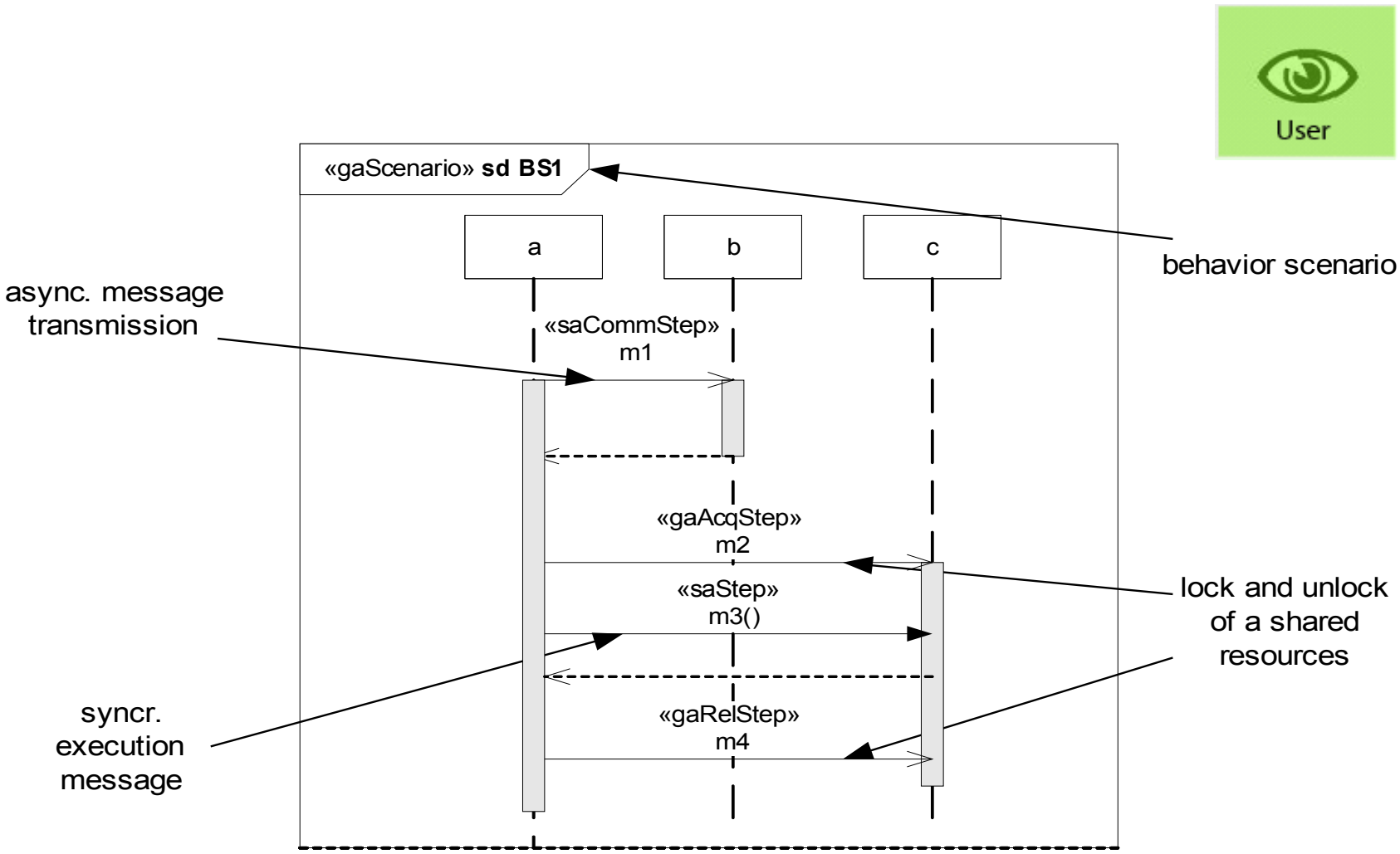
SAM: Workload Domain Metamodel (detailed behav.)



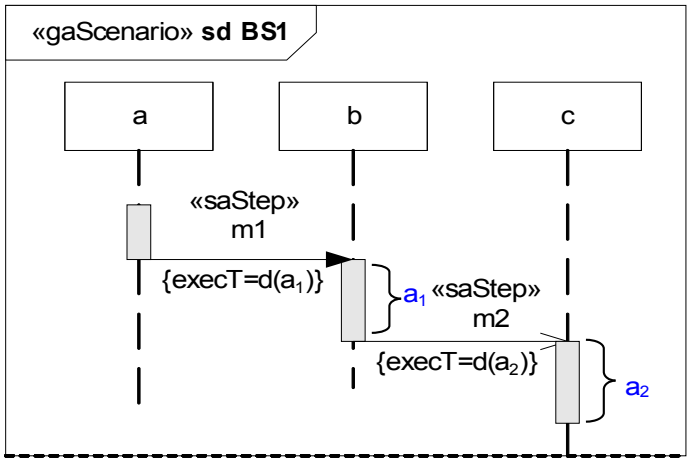
SAM: Example of Stereotype Extensions Usage

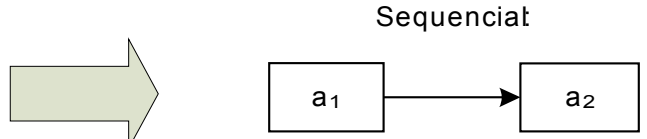
SAM Domain Model	SAM Stereotype	UML Metaclasses	Context
WorkloadBehavior	GaWorkloadBehavior	UML::Interactions::Fragments:: **CombinedFragments**	Modeled in a high-level interaction
EndToEndFlow	SaEnd2EndFlow	UML::Interactions::Fragments:: **InteractionOperand**	Modeled in a high-level interaction
WorkloadEvent	GaWorkloadEvent	UML::Interactions::BasicInteractions:: **Message**	Modeled in a high-level interaction
BehaviorScenario	GaScenario	UML::Interactions::BasicInteractions:: **Interaction**	Modeled as a low-level interaction nested within a higher-level interaction
Step **CommunicationStep** **ReleaseStep** **AcquireStep**	SaStep SaCommStep GaRelStep GaAcqStep	UML::Interactions::BasicInteractions:: **Message**	Messages in low-level interactions

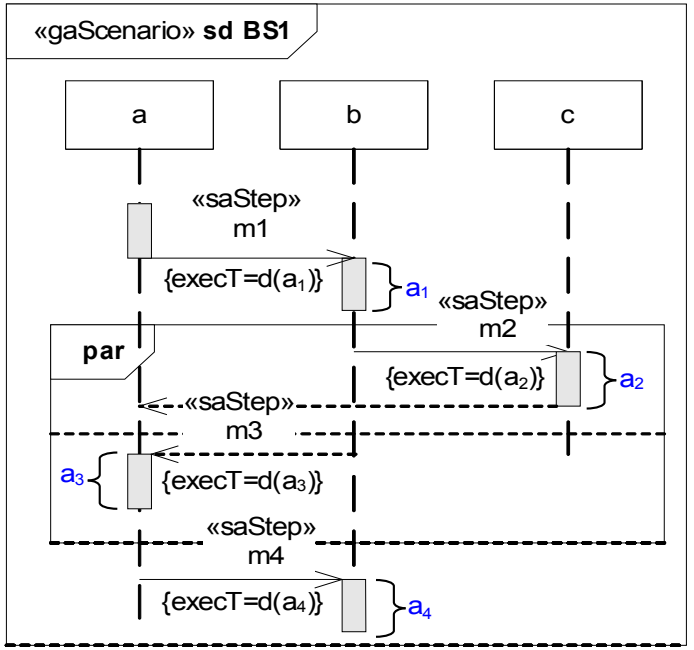
SAM: Examples of Behavior Annotations

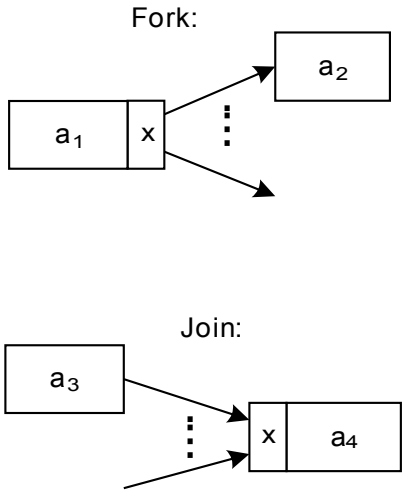


SAM: Example of Precedence Relations Annotation

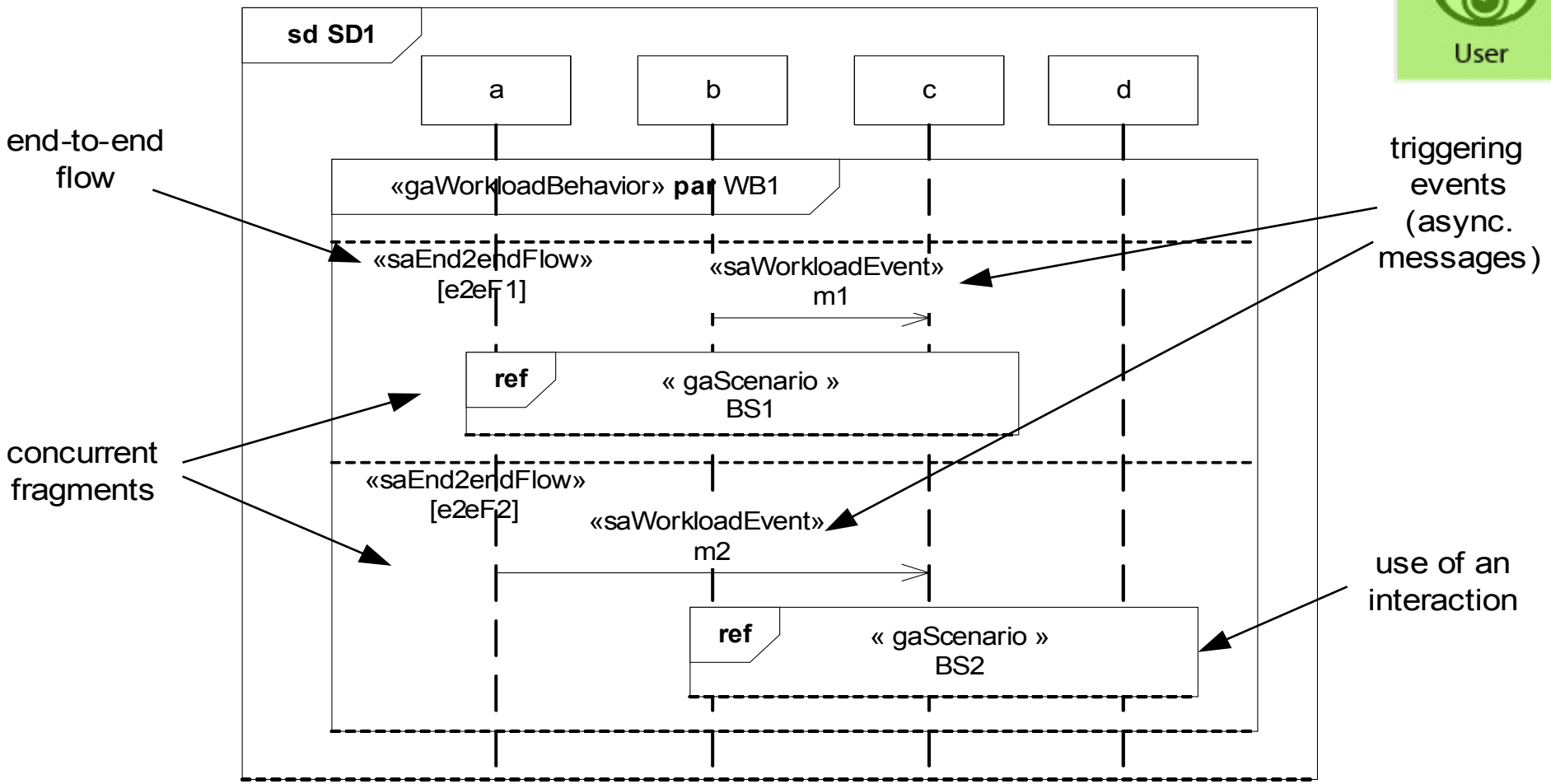






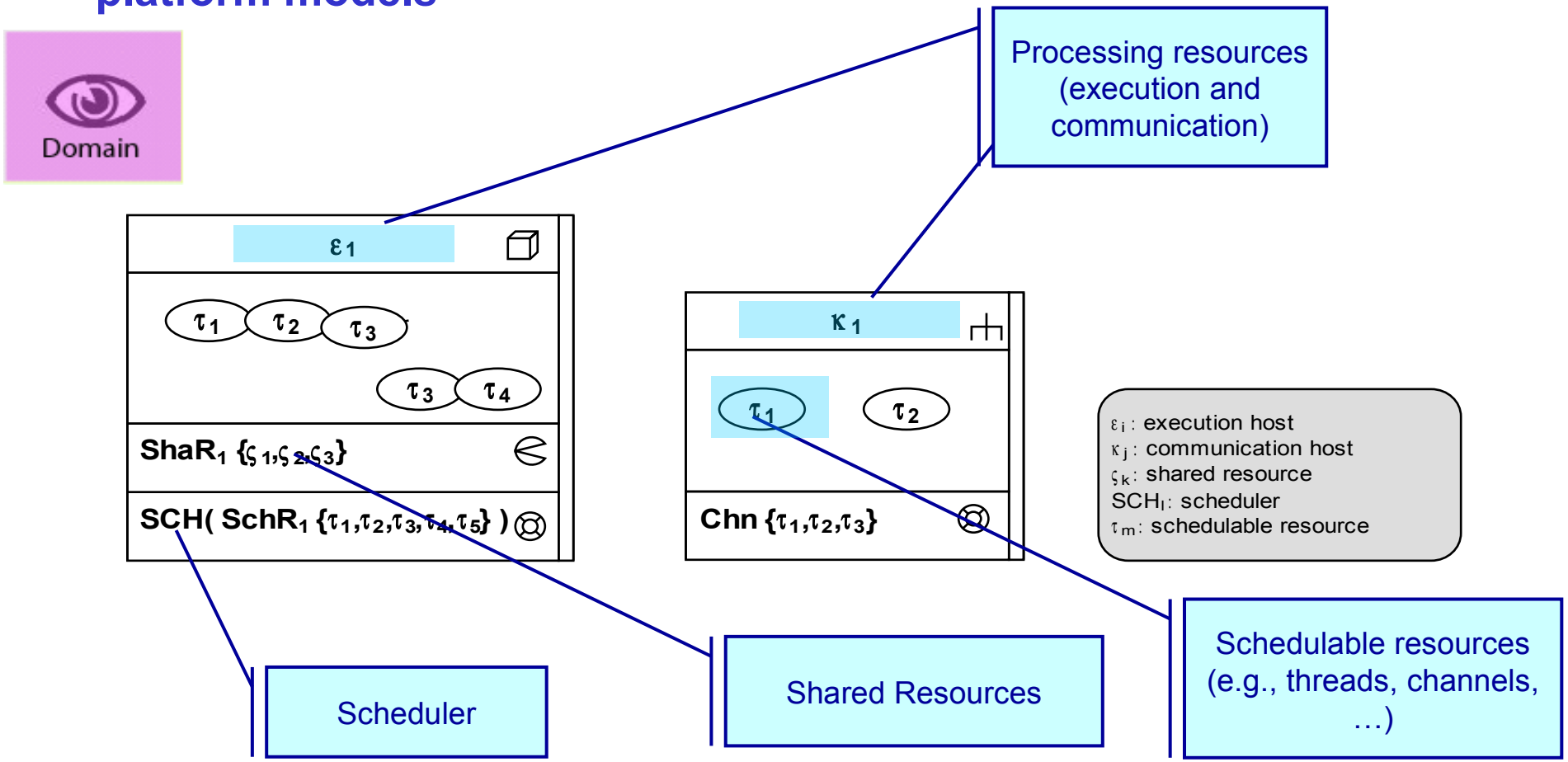


SAM: Example of Workload Annotations

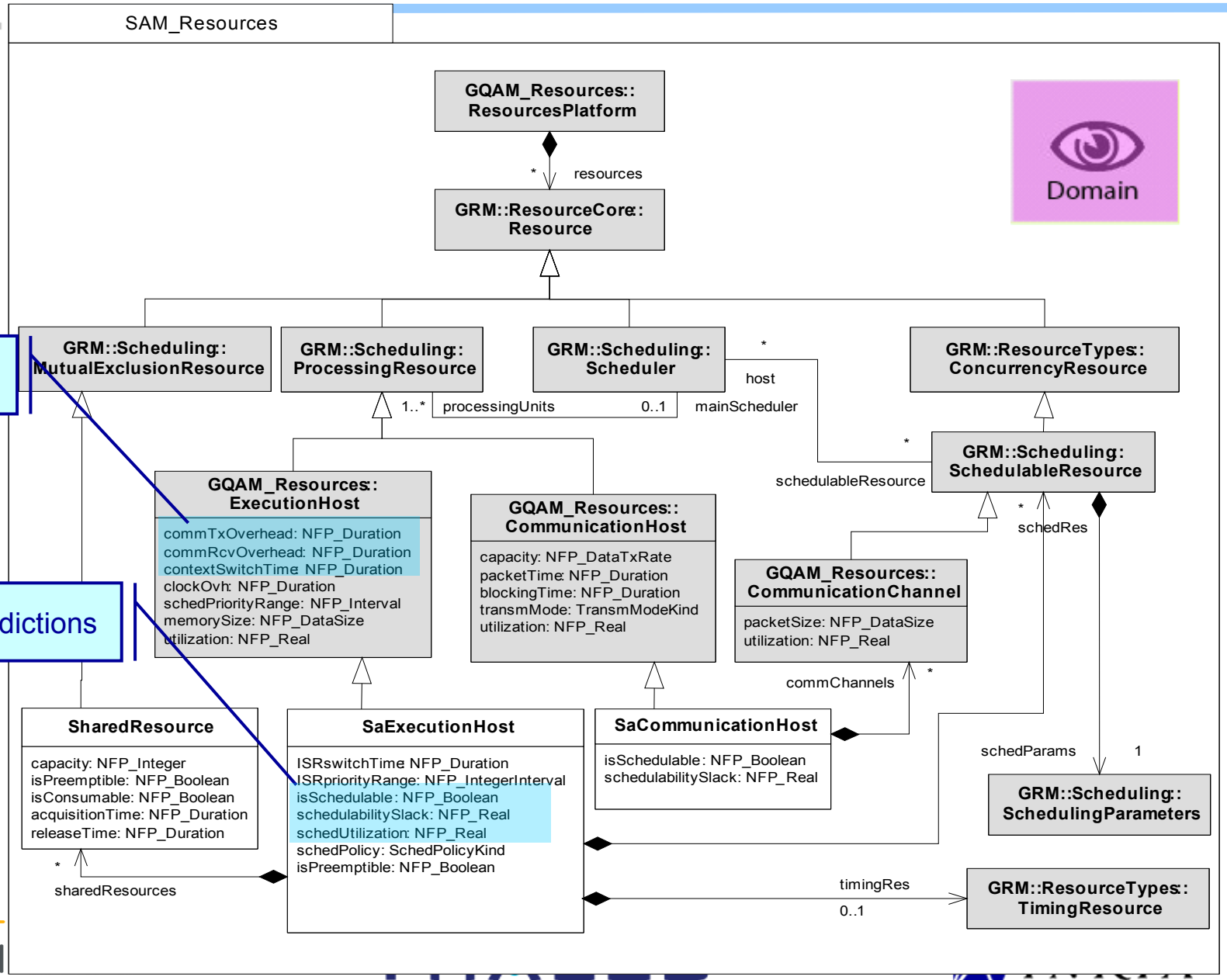


SAM: Resources Concepts

- **Provide additional (analysis-specific) annotations to annotate resources platform models**



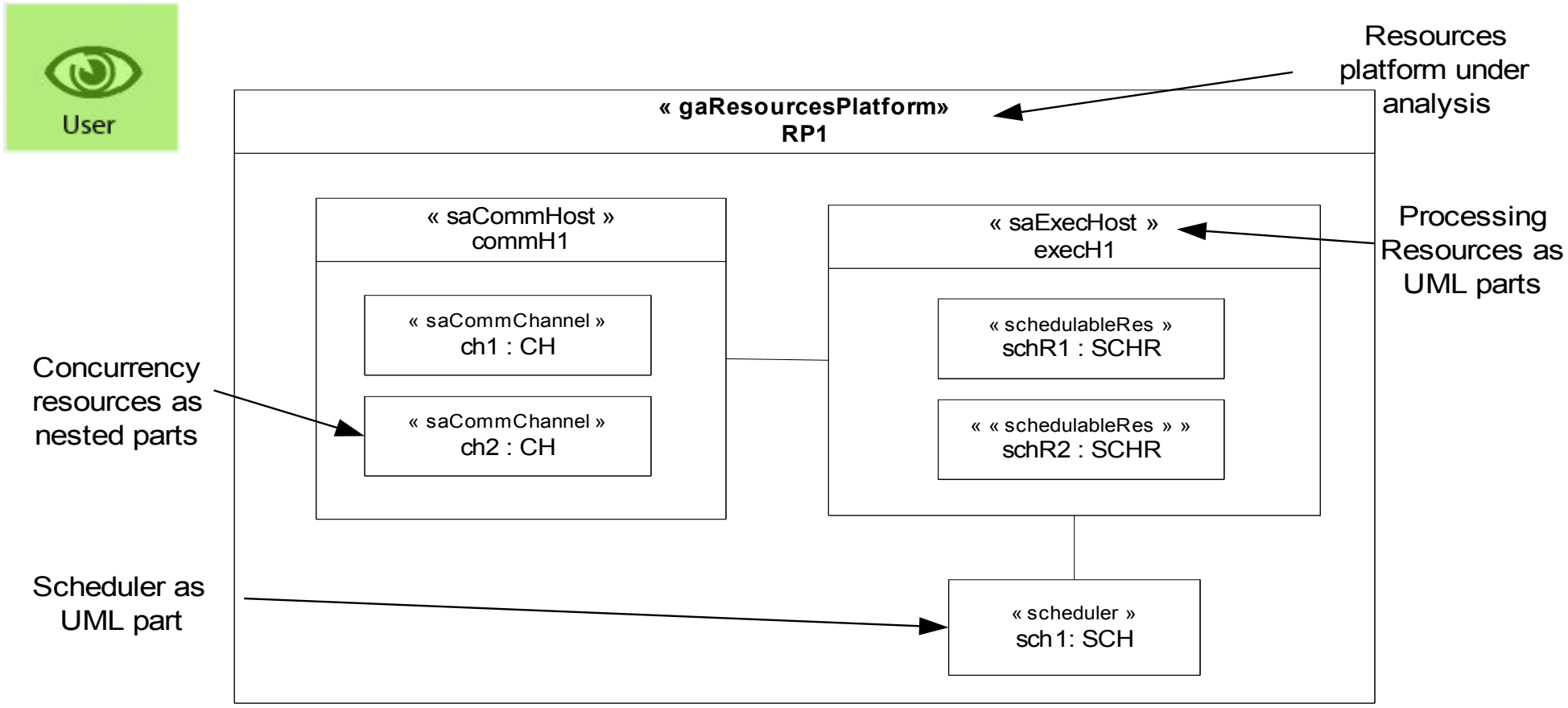
SAM: Resources Domain Metamodel



SAM: Examples of the Stereotypes Usage

SAM Domain Model	SAM Stereotype	UML Metaclasses	Context
ResourcesPlatform	GaResourcesPlatform	UML::StructuredClasses:: **SctructuredClass**	Main container of resources
SaExecutionHost **SaCommunicationHost** **GRM::Scheduler**	SaExecHost SaCommHost Scheduler	UML:: StructuredClasses:: **Property**	Parts of the resources platform
GRM::SchedulableResource **SaCommChannel**	SchedulableRes SaCommChannel	UML:: StructuredClasses:: **Property**	Parts of processing resources

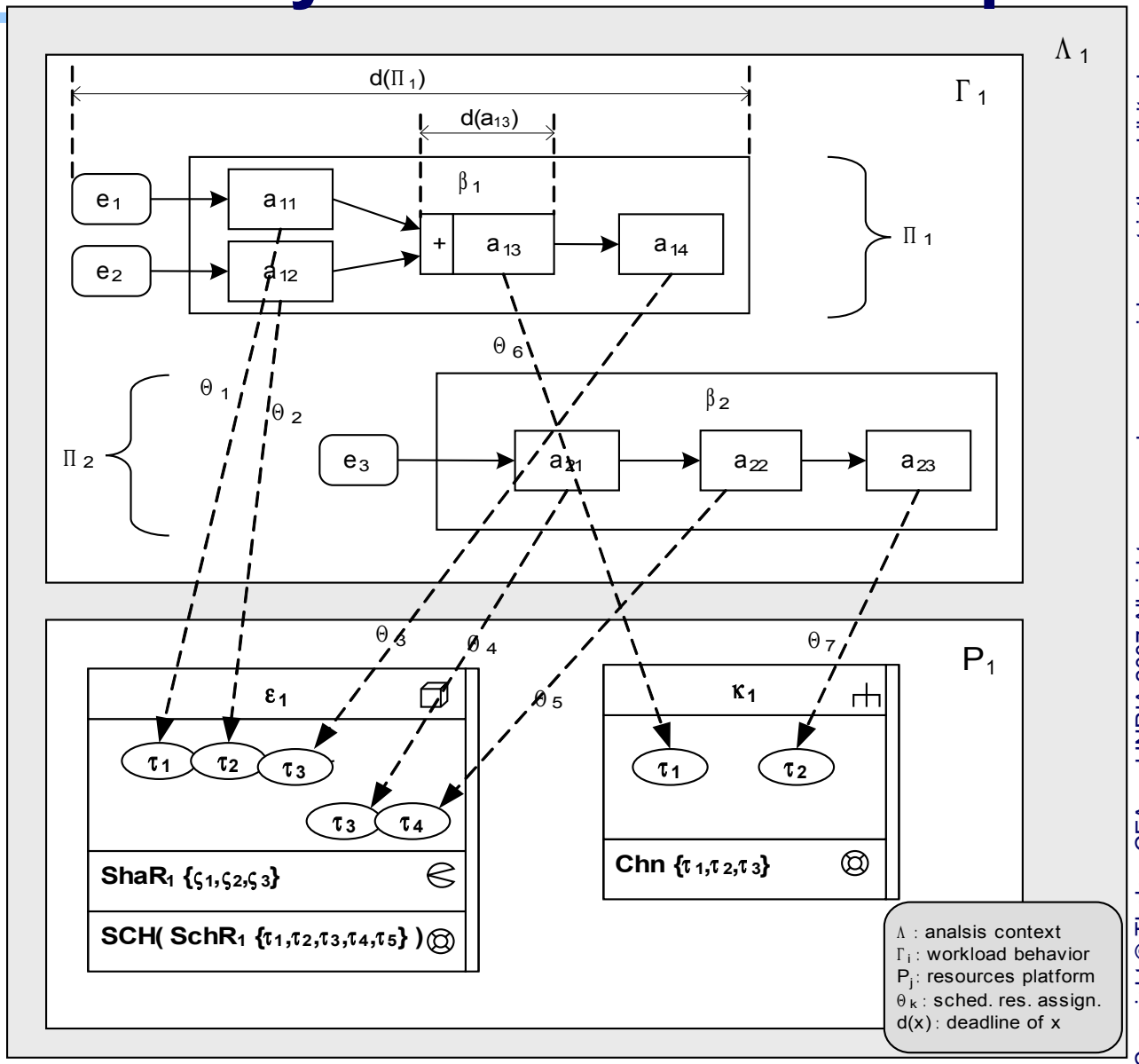
SAM: Example of Resources Stereotype Usage



SAM: Analysis Context concepts

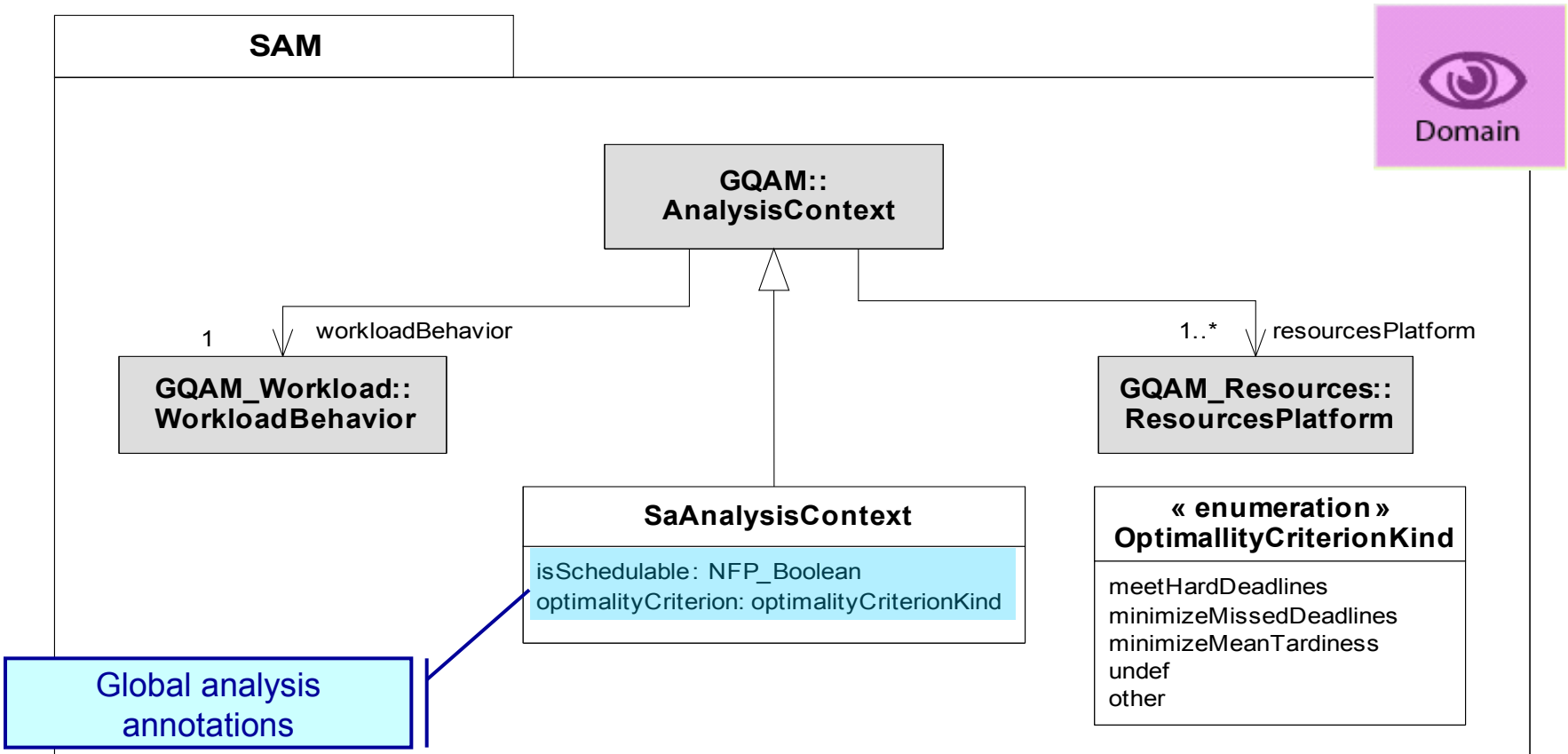
- An analysis context is the root concept used to collect relevant quantitative information for performing a specific analysis scenario.

- An analysis context integrates workload behavior models and resources platform models.

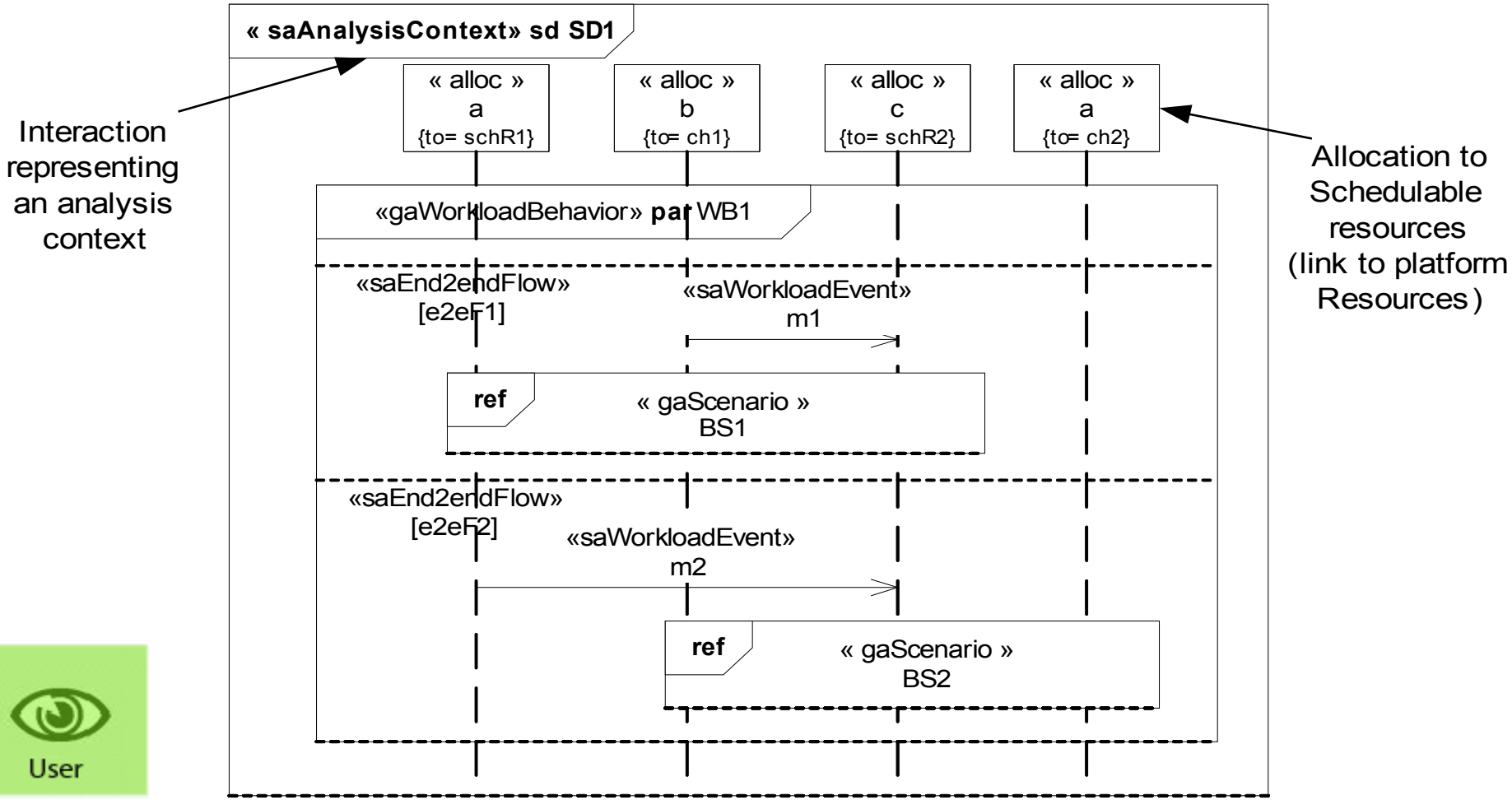


Domain

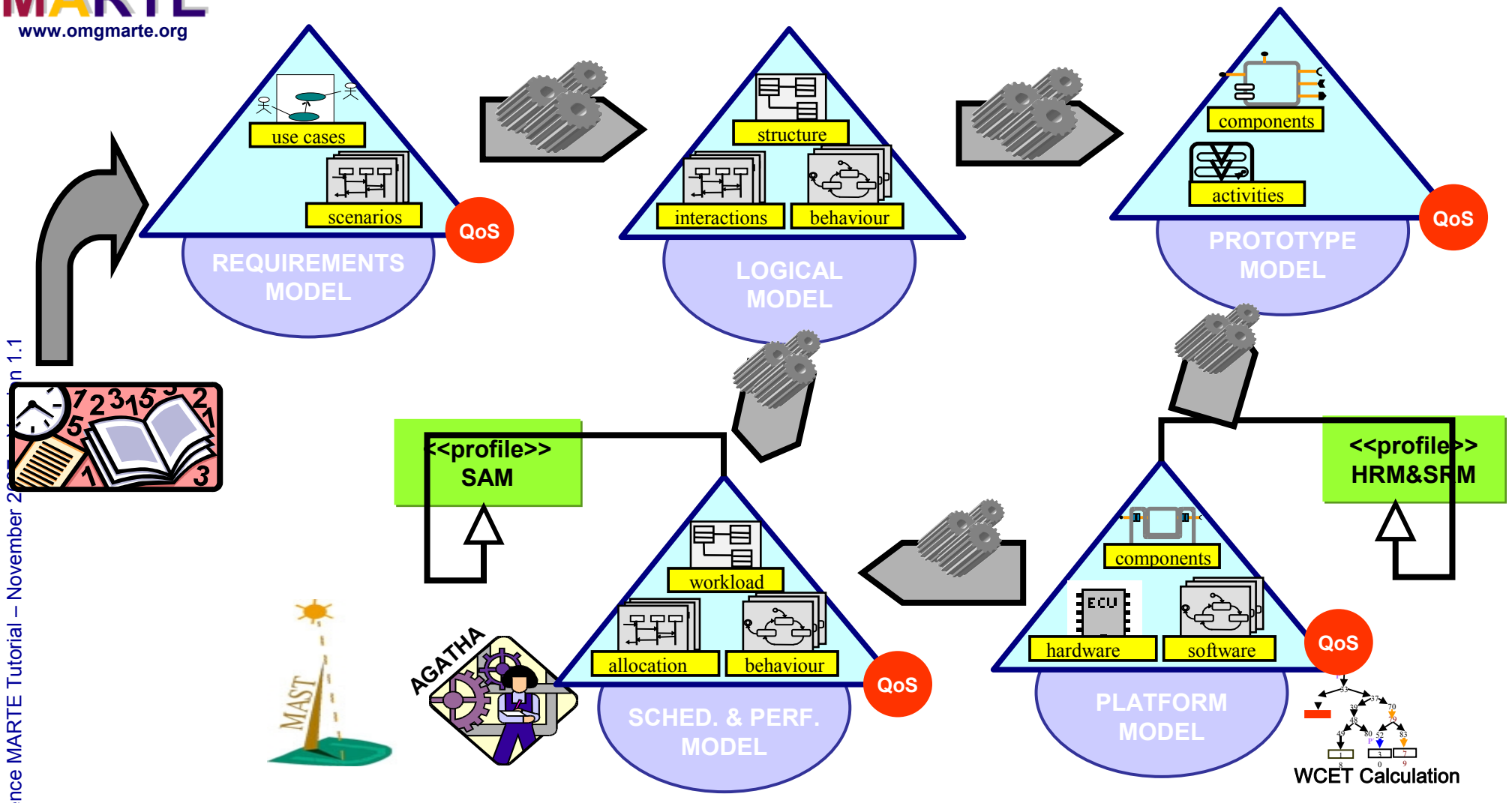
SAM: Analysis Context Domain Metamodel



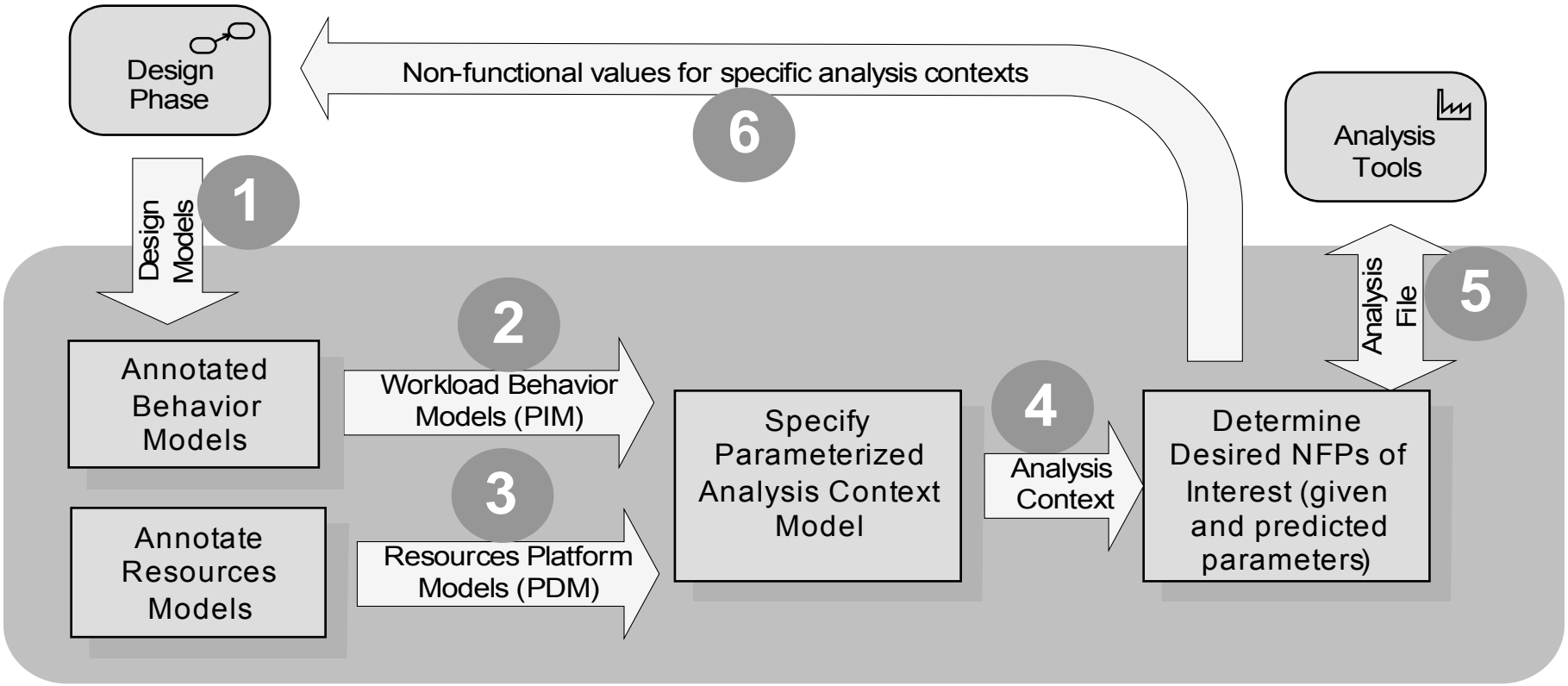
SAM: Example of Analysis Context Stereotype Applic.



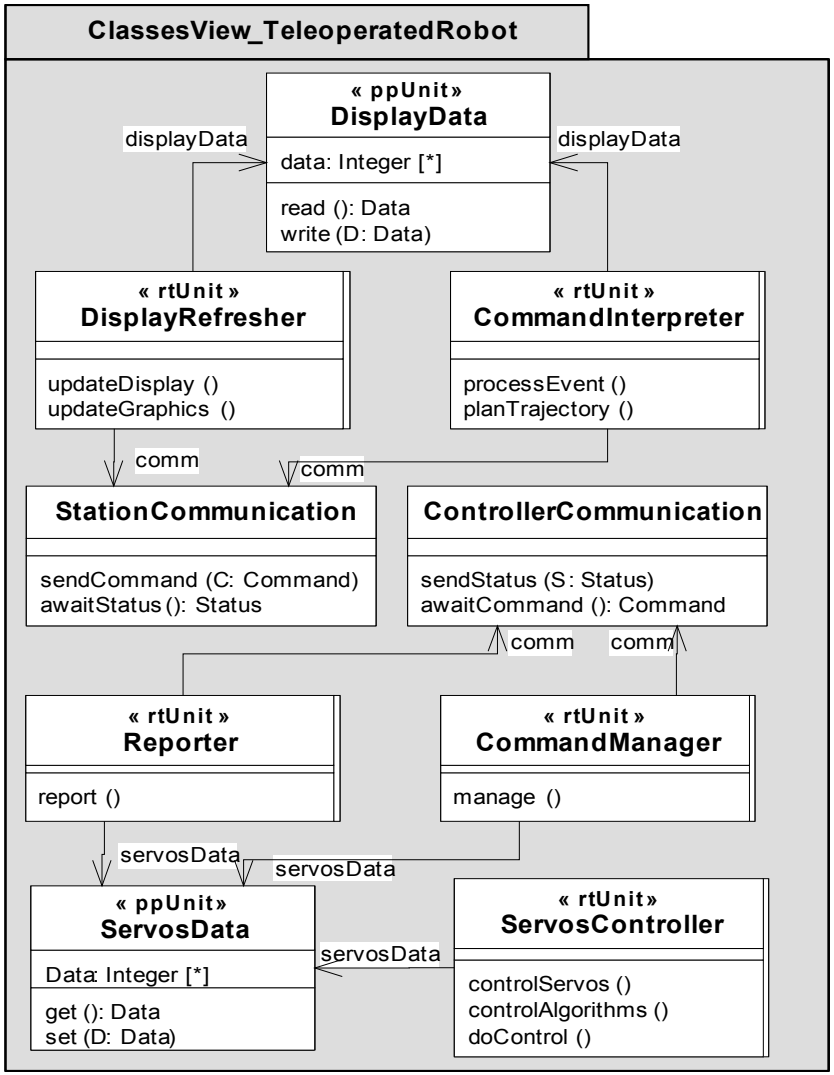
Example of Global Development Process

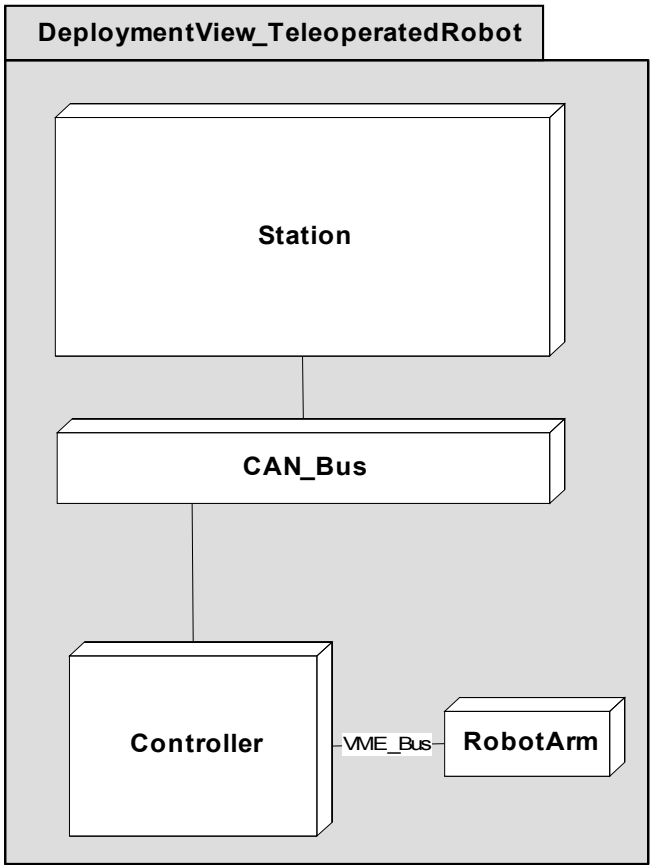


General Procedure to Use the SAM Profile

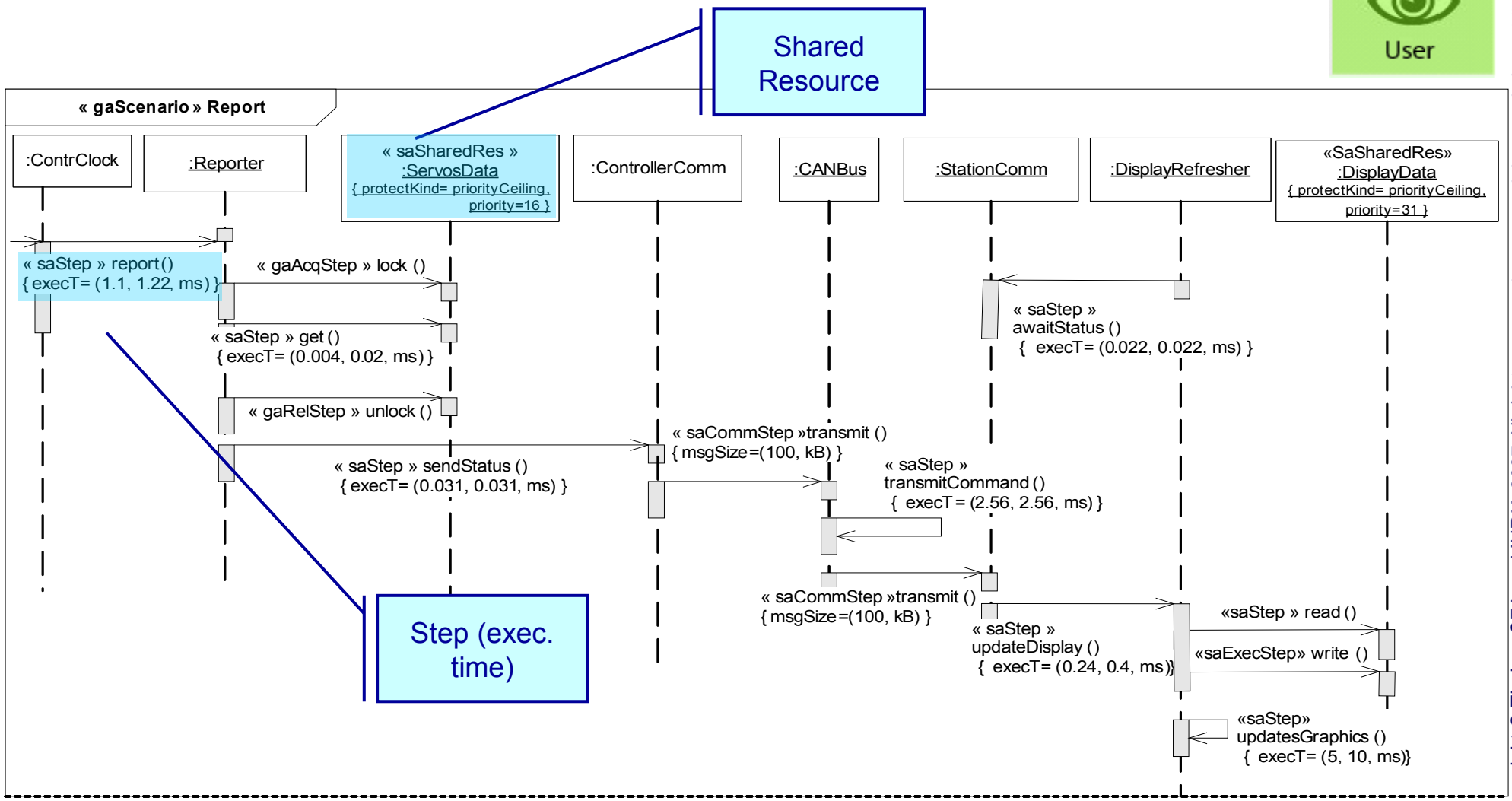


Example: A Teleoperated Robot

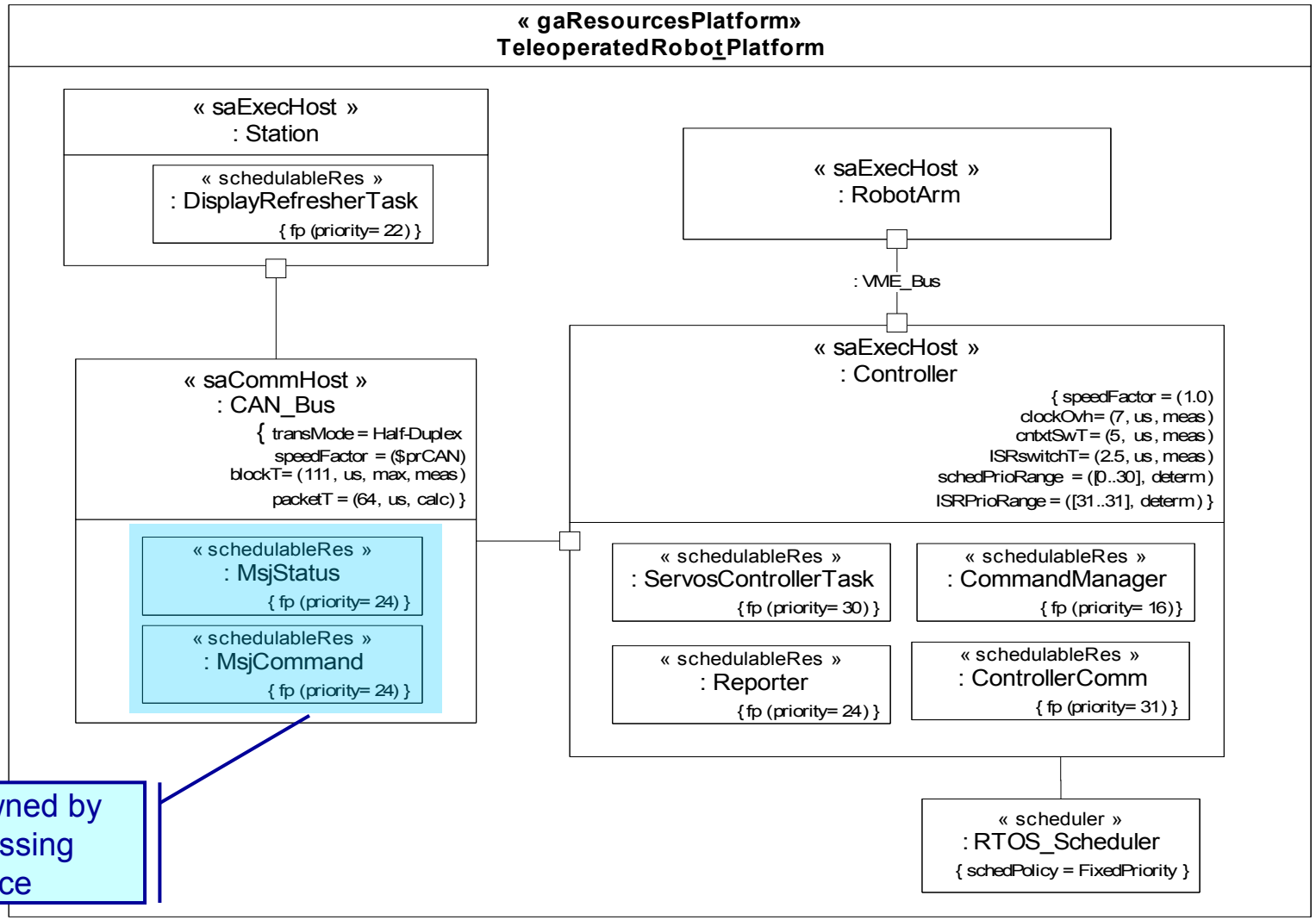




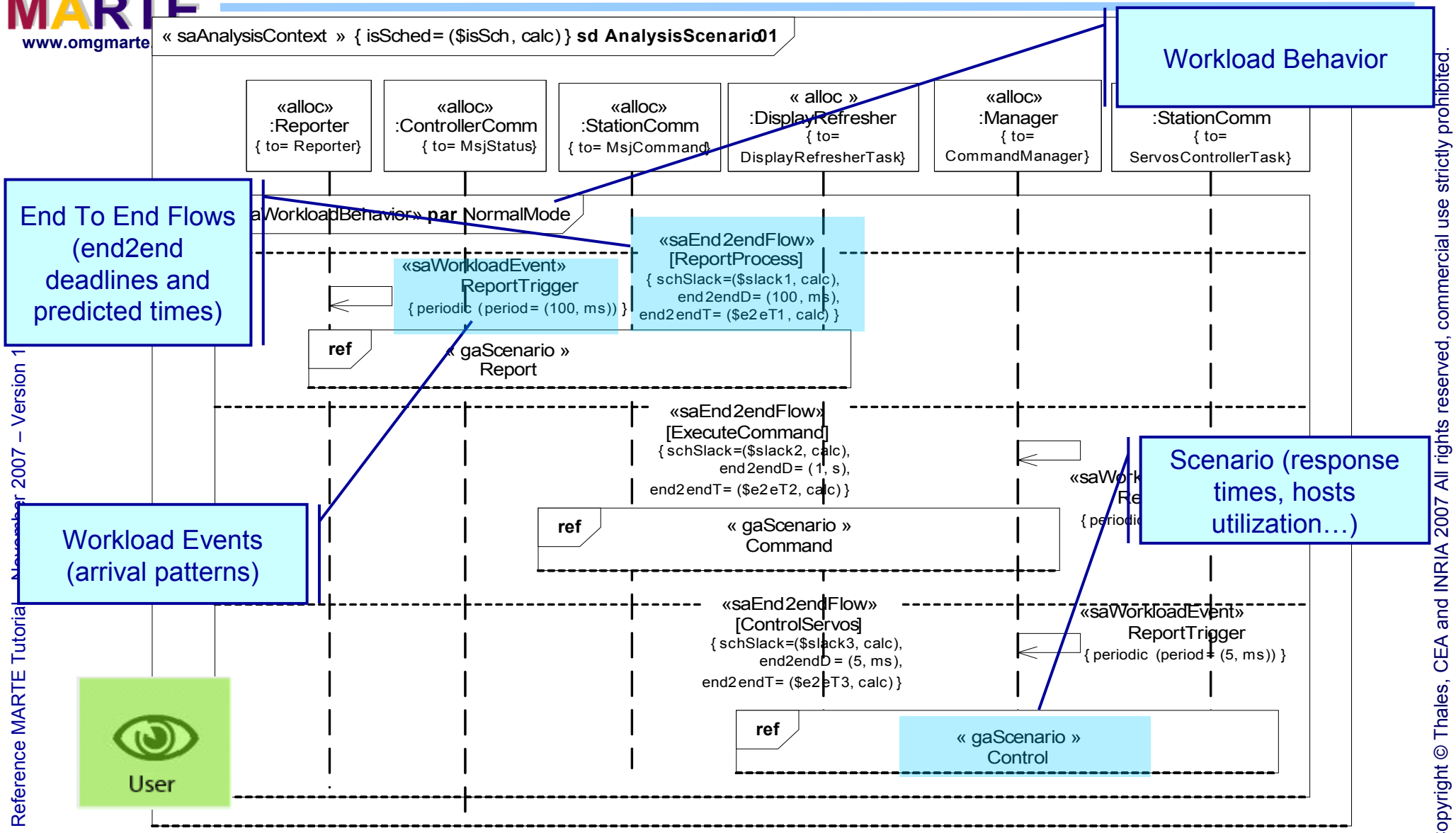
Example of Annotated Scenario with SAM



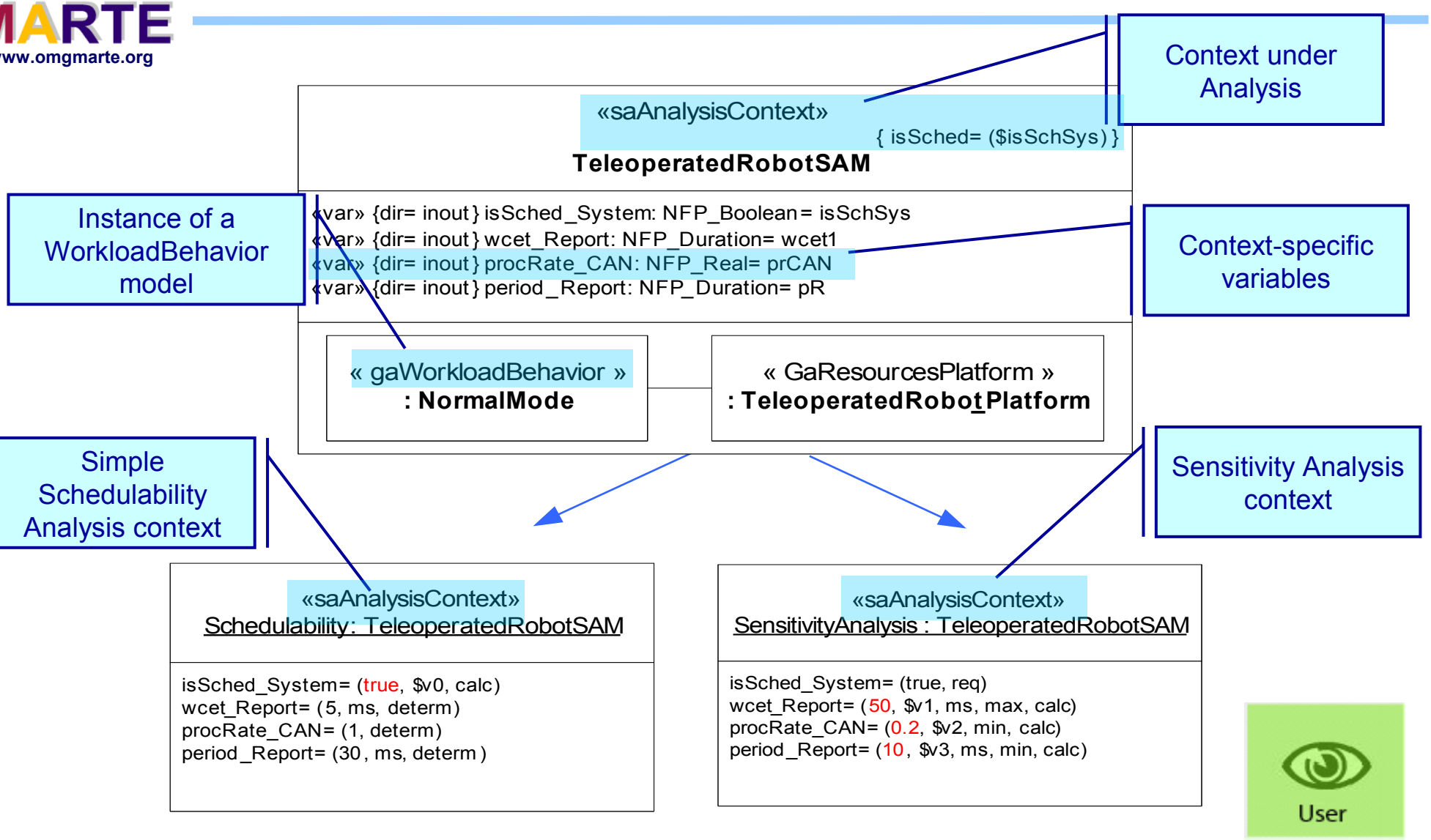
Example of Annotated Resources Model with SAM



Example of Analysis Context Model



Example of Parametric Analysis Context



MARTE Tooling

- **Current Implementations supporting MARTE**
  - Full MARTE Profile & Libraries for Eclipse UML2
  - VSL edition assistant and type checker as a Eclipse plug-in for the UML Papyrus tool and RSA 7.0

- **On-going work:**
  - Eclipse plug-ins to transform UML models annotated with the SAM profile to input files of MAST, SymTA/S, Cheddar and RapidRMA tools

MARTE Open Source Implementation in

UML Papyrus: **www.papyrusuml.org**
IBM RSA: **www.omgmarte.org**

Conclusions on MARTE's Analysis

- **Industrial Use of V&V can benefits from MDE**
  - Analysis task must be cohesively integrated with Design tasks
  - Application of individual analysis techniques should be regarded as an essential part of an integrated V&V methodology

- **Methodological support is still under way:**
  - Complex analysis scenarios for Interface-Based Design, Multiobjective Design Space Exploration…
  - Means to manage NFP measurement models
  - Methods to map/transform MoCCs into analysis models

Agenda

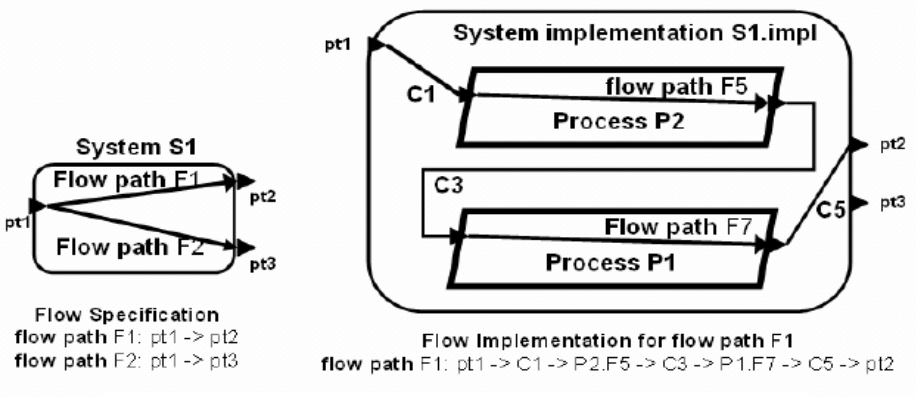
- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- Part 3
  - The timing model
- Part 4
  - A component model for RT/E
- Part 5
  - Platform modeling
- Part 6
  - Repetitive structure modeling
- Part 7
  - Model-based analysis for RT/E
- Part 8
  - **MARTE and AADL**
- Part 9
  - Conclusions

AADL Overview

AADL Architecture Analysis Description Language

- Architecture Description Language dedicated to RTES
- International standard at SAE (AS5506, 2004)
- Adapted for many critical computer system domain
  - Automotive, space, robotics, industrial control, medical, avionics, …
- Allow specification, analysis and automated integration of real-time performance critical
  - Timing,
  - Safety,
  - Schedulability,
  - Fault tolerant,
  - Security,
  - Distributed computing systems,….

What does AADL look like ?



AADL

A graphical representation

```
  compute12: data port compute1.outed ->> compute2.ined;
  compute23: data port compute2.outed -> compute3.ined;
  actuateconn: data port compute3.outed ->> actuate.ined;
  bus access db -> sense.devbus;
  bus access db -> actuate.devbus;
flows
  etelatency: end to end flow sense.flow1 -> senseconn -> com-
pute1.flow1
            -> compute12 -> compute2.flow1 -> compute23 -> com-
pute3.flow1
            -> actuateconn -> actuate.flow1 { latency => 153 ms;};
end application.twosamplesteps;
```

A textual representation

Guidelines for modeling AADL application in MARTE

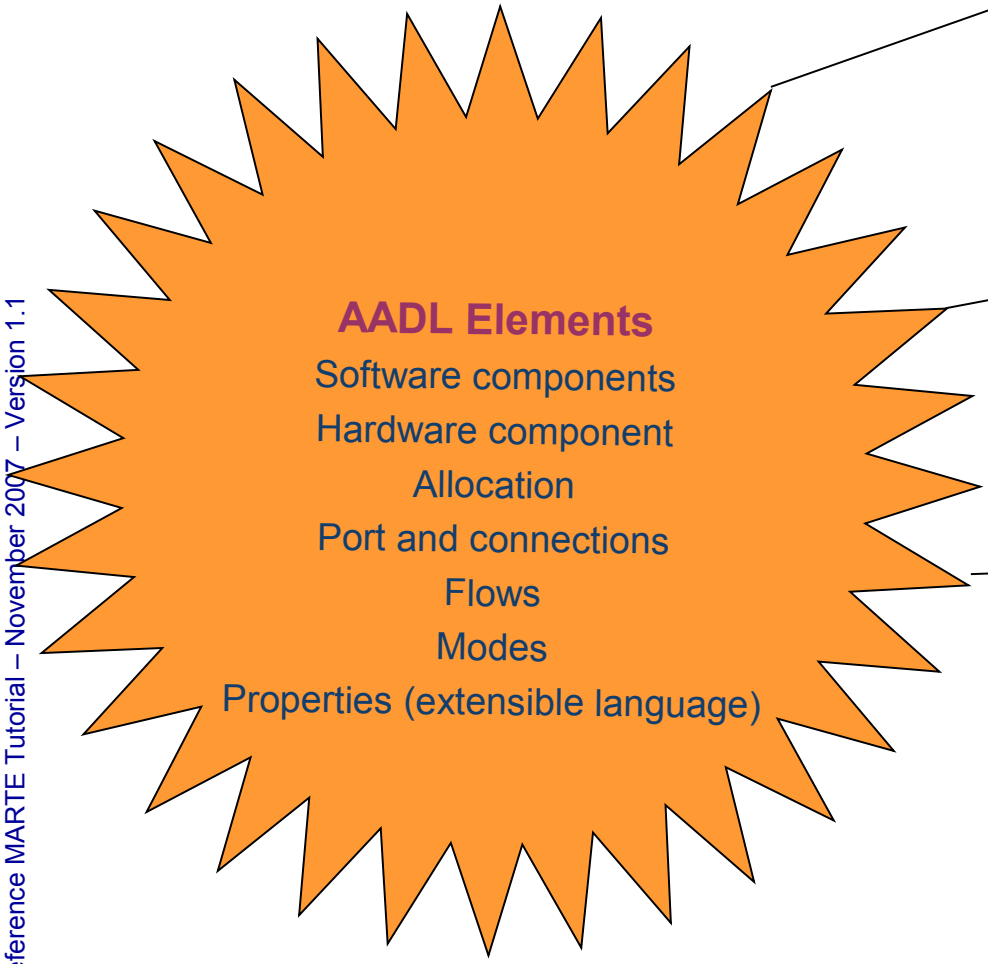
MARTE

- Generic for Real time Embedded System application modeling and analysis
- Address early and detailed design stages
- Complementary and consistent views make the model more understandable
  - Platform execution model can be explicitly modelized
  - Full integration of non-functional properties in the model (Time, performance, scheduling features…)

AADL

- Specific to synchronous data flow application modeling and analysis
- Address detailed design stages
- Based on an implicit execution platform model
  - specific thread execution automata
  - Applications and platform execution semantics have to be in line
- Lack of non-functional properties model integration

AADL useful for….



Non functional aspects through properties specification

- Temporal, safety, reliability,…

Architecture and design

- Components and interfaces, connections
- Data and control flows
- Run time architecture

Analysis and verification aspects

- End-to-end latency
- Age of signal data and jitter
- …

MARTE-AADL concepts mapping

- First AADL – MARTE alignment based on AADL constructs and features and MARTE artifacts.
- Next step: Deeper map AADL component semantics, AADL properties and implicit AADL platform semantics on MARTE concepts (MARTE concepts properties and NFP, VSL language)

AADL concepts	MARTE concepts
Software components	memoryPartition, wSchedulableRessource,.
Hardware components	hwProcessor, hwMemory,.
Binding	Allocated
AADL features	MARTE flowPorts, UML required/provided interfaces…
Subcomponents	UML Parts
Port Connections	UML delegation/assembly
Flow specification	UML object flows
Modes	UML state machines
Properties	Not yet documented

AADL component (Type and Implementation)

AADL Component Types

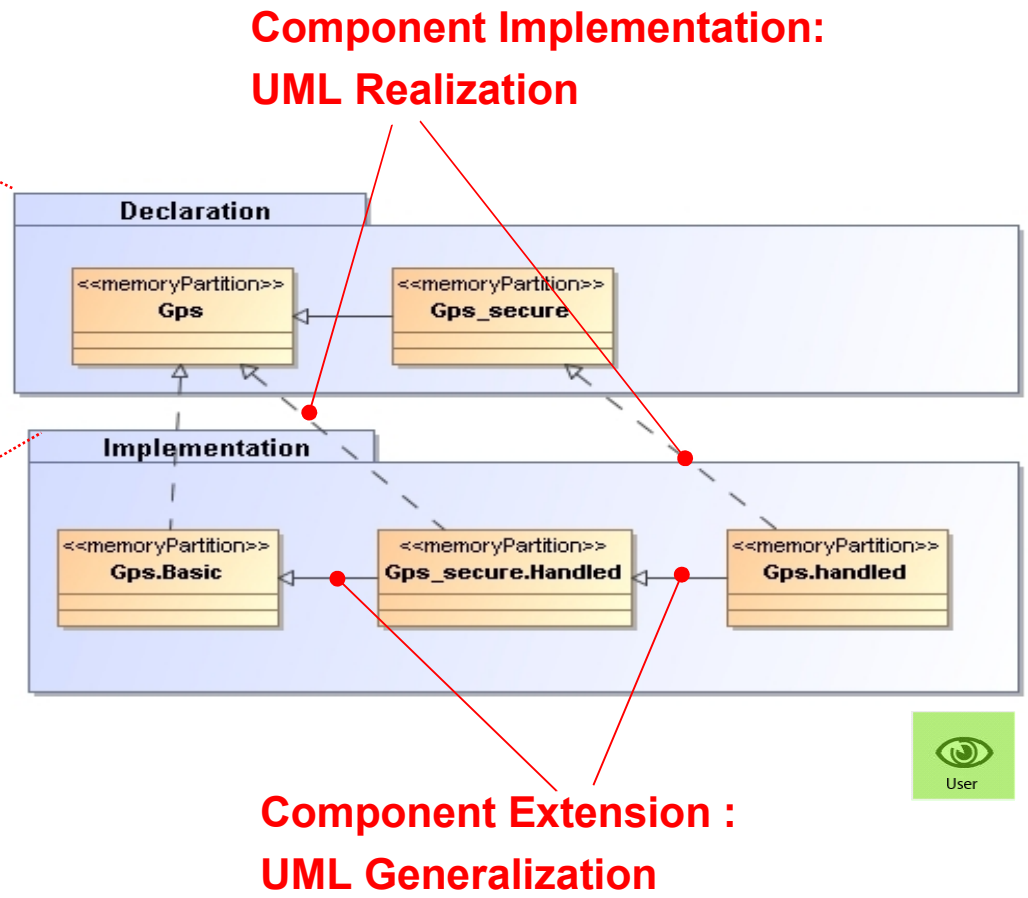
- Specifies a functional interface in terms of features (ports, port groups, flow specifications, subcomponent access..), properties
- UML package containing AADL component declaration)

AADL Component Implementations

- Describes the internal structure and behavior of that component in terms of subcomponents, connections and flows across them, and behavioral modes
- UML package containing AADL component implementations
- AADL implementation liked to AADL declaration through UML Realization

AADL Component Extension

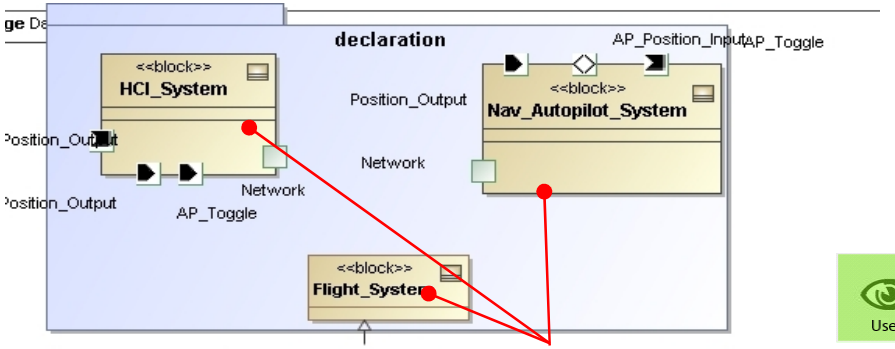
- UML Generalization



AADL System and AADL Subcomponents

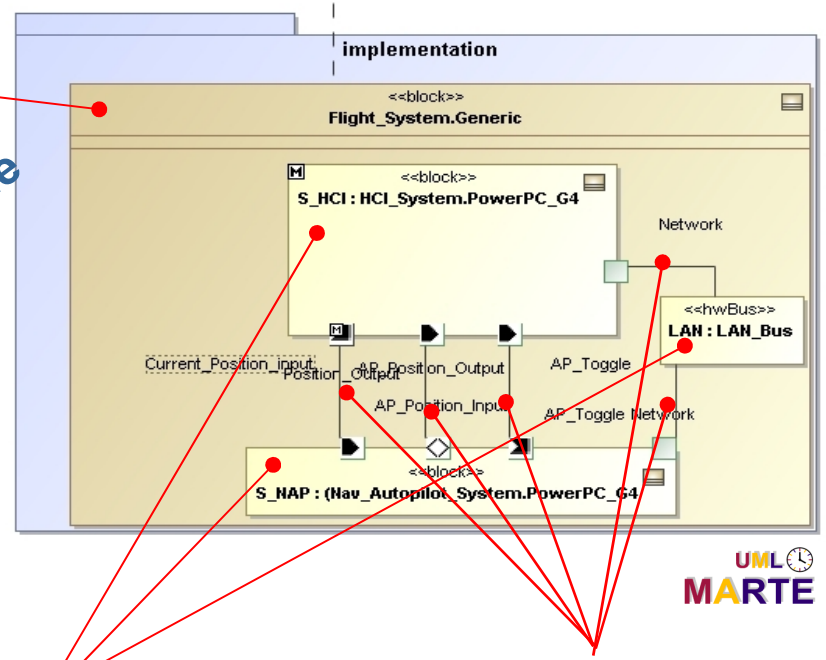
- **AADL System**
  - Represents a composite software, execution platform or system components
  - Represented by a "SysML" block

- AADL Subcomponents
  - represented by UML parts



System Types

System Implementation



Generated code

```
system Flight_System
end Flight_System;

system implementation Flight_System.Generic
subcomponents
    S_HCI : system HCI_System.PowerPC_G4;
    S_NAP : system Nav_Autopilot_System.PowerPC_G4;
    LAN : bus LAN_Bus;   ….
```

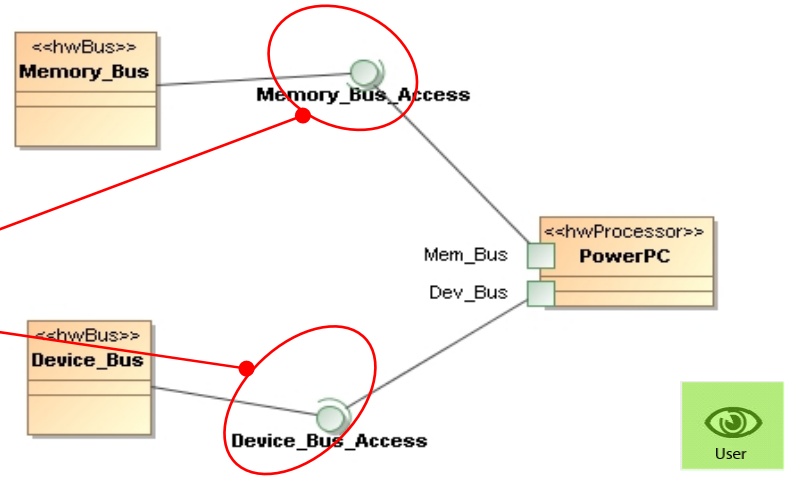
AADL

System subcomponents

Subcomponents relationships

Required data/bus access

- AADL Bus and Data components access
- **Provide data/service to other AADL components**
- **Access required from other AADL components**
  - Declaration part : Provide/required access modeled in MARTE via provided / required UML Interfaces
  - Implementation part :
    - Access design by delegation / assembly connectors
    - Resources may be specialized with real time features or associated to services (synchronization, concurrency access …)

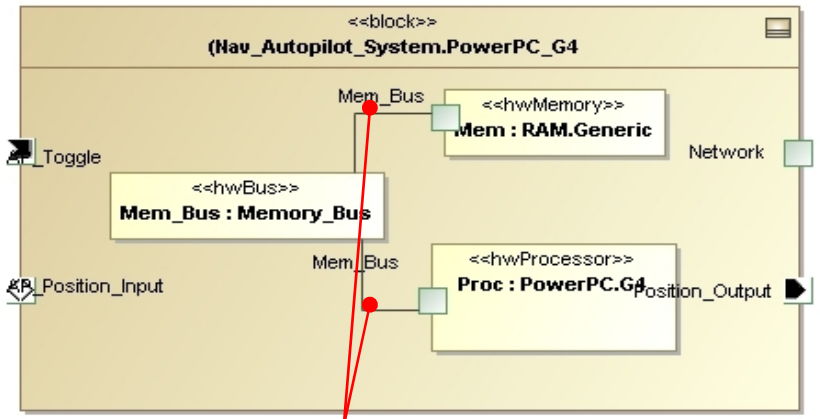


Access declaration

Generated code

```
processor powerPC
Features
    MemBus : requires bus access Memory_Bus;
    Dev_Bus : requires bus access Device_Bus
End PowerPC;

bus Memory_Bus
end Memory_Bus;
```

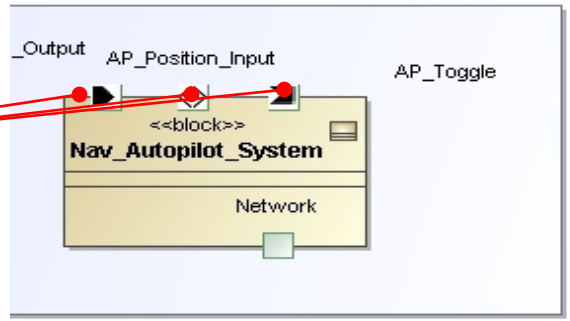
AADL



Access connections

Ports and Connections

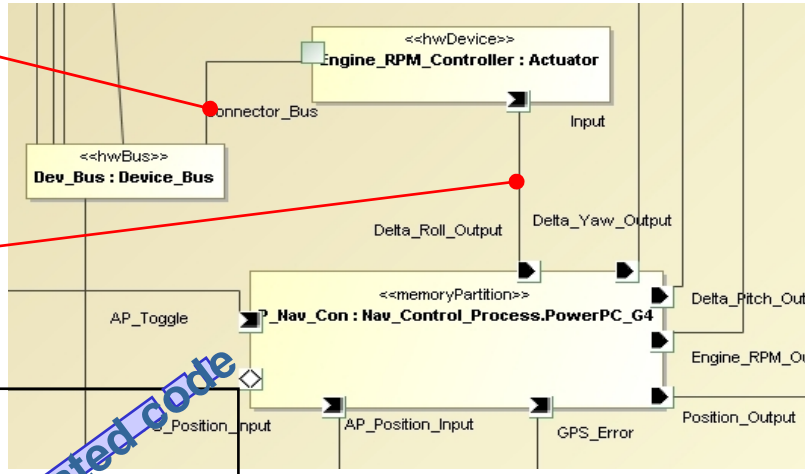
- **AADL Ports**
  - Flow ports are represented by MARTE Flow Ports




```
system Nav_Autopilot_Systemfeatures
    AP_Toggle : in event port;
    AP_Position_Input : in event data port Nav_Types::Position.GPS; …
```

AADL

- **AADL Port, Bus and Memory Connections**
  - Bus/data access data are represented by UML connectors between the port providing the access and the device



- **Flows ports connections are represented by UML connectors (delegation or assembly connectors)**

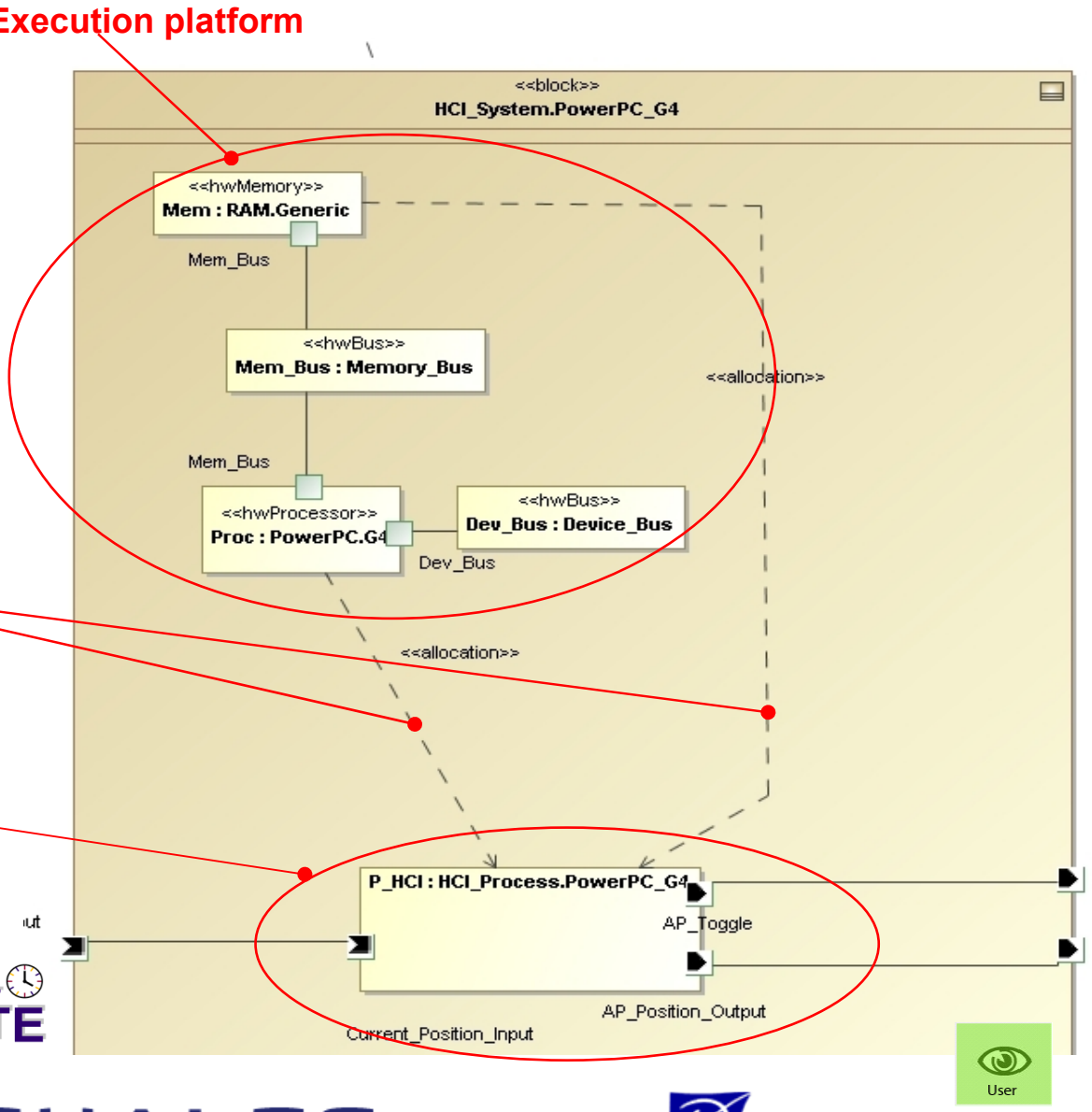
```
system implementation Nav_Autopilot_System.PowerPC_G4
….
bus access Dev_Bus -> Engine_RPM_Controller.Connector_Bus;
….
data port P_Nav_Con.Engine_RPM_Output -> Engine_RPM_Controller.Input ;
…
```

Generated code

AADL

System binding

- **AADL System bindings**
  - Are represented by MARTE <<allocation>> stereotyped UML Dependency



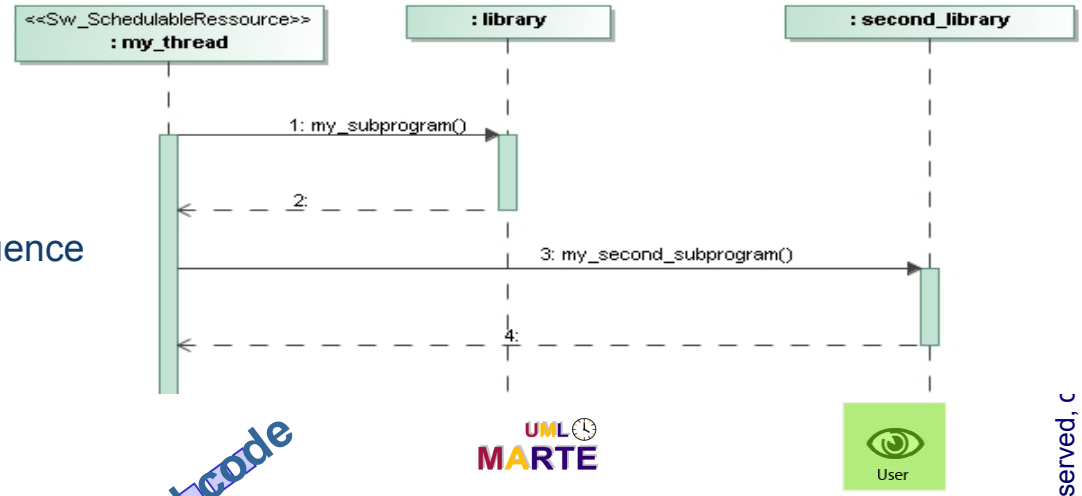
Subprogram call

Subprogram

- Are represented by Operation

Subprogram calls

- Are represented through UML Messages on sequence diagrams

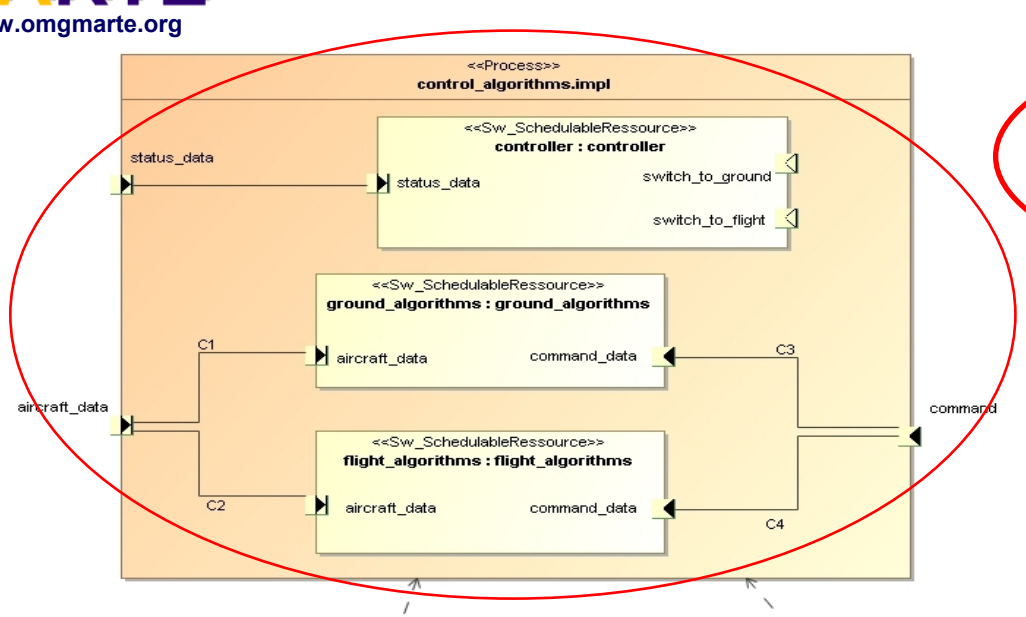


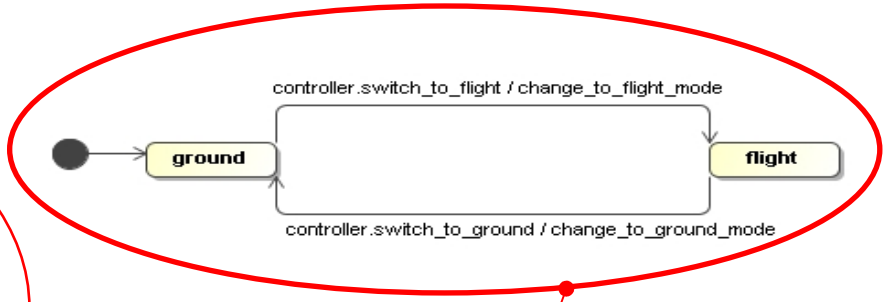
Generated code

AADL

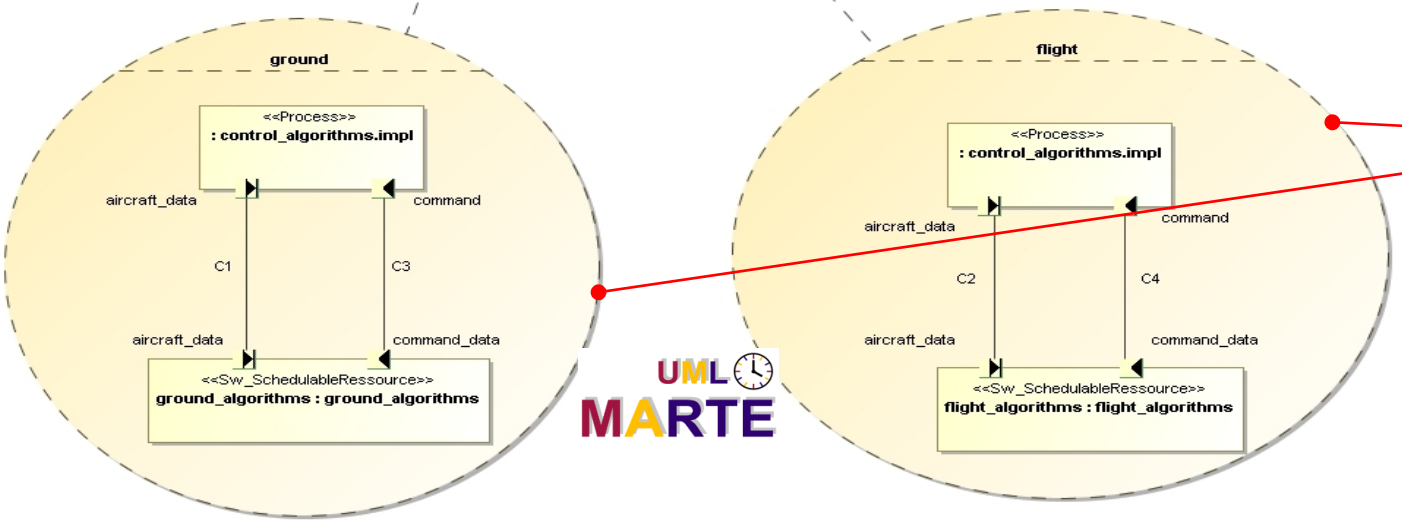
```
thread implementation my_thread.impl
calls {
        first_subpgr : subprogram my_subprogram;
        second_subpgr : subprogram my_second_subprogram;
    }
end my_thread.impl;
```

Modes with MARTE guidelines





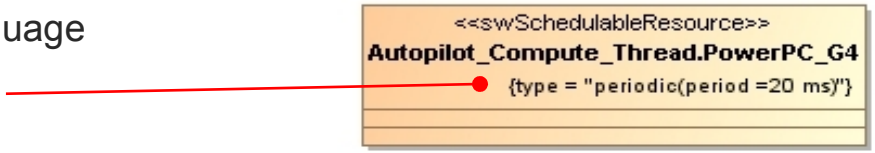
Mode transition modelized by an UML state machine



Different mode configuration through Collaboration diagrams

Properties

- **AADL Properties**
  - Are represented by <<AADL properties>> stereotyped UML comments
  - Will be aligned using NFPs, VSL language and MARTE concepts properties



UML MARTE

MARTE-AADL code generator

- MARTE to AADL code generator is already available
- Bridge between MARTE/AADL and Cheddar (Scheduling analysis) already tested

```
process implementation Nav_Control_Process.PowerPC_G4
subcomponents
                T_GPS_Reader : thread GPS_Sampling_Thread.PowerPC_G4 in modes (GPS_UP_AP_UP, GPS_UP_AP_DOWN);
                T_AP_Compute : thread Autopilot_Compute_Thread.PowerPC_G4 in modes (GPS_UP_AP_UP);          T_AP_Params : thread
Autopilot_Modify_Parameters_Thread.PowerPC_G4;
                D_AP_Destination : data Nav_Types::Position.Simple;
                D_AP_Airspeed : data Nav_Types::Integer;
                D_AP_Altitude : data Nav_Types::Integer;
connections
                data port GPS_Position_Input -> T_GPS_Reader.Position_Input in modes (GPS_UP_AP_UP, GPS_UP_AP_DOWN);
                data port T_GPS_Reader.Position_Output -> Position_Output in modes (GPS_UP_AP_UP, GPS_UP_AP_DOWN);
                data port T_GPS_Reader.Position_Output -> T_AP_Compute.Position_Input in modes (GPS_UP_AP_UP);
                T_AP_Compute.Delta_Roll_Output -> Delta_Roll_Output in modes (GPS_UP_AP_UP);
                data port T_AP_Compute.Delta_Yaw_Output -> Delta_Yaw_Output in modes (GPS_UP_AP_UP);
                data port T_AP_Compute.Delta_Pitch_Output -> Delta_Pitch_Output in modes (GPS_UP_AP_UP);
                data port T_AP_Compute.Engine_RPM_Output -> Engine_RPM_Output in modes (GPS_UP_AP_UP);
                event data port AP_Position_Input -> T_AP_Params.AP_Position_Input                modes
                GPS_UP_AP_DOWN : initial mode;
                GPS_UP_AP_UP : mode;
                -- <INITIAL_MODE> -[ <EVENT> ]-> <FINAL_MODE>
                GPS_UP_AP_DOWN -[ AP_Toggle ]-> GPS_UP_AP_UP;
                GPS_UP_AP_DOWN -[ GPS_Error ]-> GPS_DOWN;
                GPS_UP_AP_UP -[ GPS_Error ]-> GPS_DOWN;
end Nav_Control_Process.PowerPC_G4; …
…
process implementation HCI_Process.PowerPC_G4
subcomponents
                T_Screen_Display_Thread : thread Screen_Display_Thread.PowerPC_G4;
                T_Pilot_Input : thread Pilot_Input_Thread.PowerPC_G4;
connections
                event port T_Pilot_Input.AP_Toggle -> AP_Toggle;
                event data port T_Pilot_Input.AP_Position_Output -> AP_Position_Output;
                T_Pilot_Input.AP_Toggle -> T_Screen_Disp.AP_Toggle;
```

CODE GENERATED

Agenda

- **Part 1**
  - Introduction to MDD for RT/E systems & MARTE in a nutshell
- **Part 2**
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)
- **Part 3**
  - The timing model
- **Part 4**
  - A component model for RT/E
- **Part 5**
  - Platform modeling
- **Part 6**
  - Repetitive structure modeling
- **Part 7**
  - Model-based analysis for RT/E
- **Part 8**
  - MARTE and AADL
- **Part 9**
  - **Conclusions**

MARTE Frontiers and Challenges

- **MARTE define the language constructs only!**
  - Common patterns, base building blocks, standard NFP annotations
  - Generic constraints that do not force specific execution models, analysis techniques or implementation technologies
- **It does not cover methodologies aspects:**
  - Interface-Based Design, Design Space Exploration
  - Means to manage refinement of NFP measurement models
  - Concrete processes to storage, bind, and display NFP context models
  - Mapping to transform MoCCs into analysis models

MARTE is to the RTES domain as UML to the System & Software domain: a family of large and open specification formalisms!

Related links

- **The official MARTE web site: www.omgmarte.org**
  - Tutorials, events, projects related and tools
  - On open source Eclipse plug-in for UML2 graphical modeling
  - MARTE implementation available within IBM RSA 7.0
    - Included the VSL editor

- **www.papyrusuml.org**
  - On open source Eclipse plug-in for UML2 graphical modeling
  - MARTE implementation available within the V1.8 release of the tool
    - Already available on:
      - **https://speedy.supelec.fr/Papyrus/svn/Papyrus/extensions/MARTE/head/**
    - Working on:
      - **https://speedy.supelec.fr/Papyrus/svn/Papyrus/core/...**