# MARTE:
# A New Standard for Modeling and Analysis of Real-Time and Embedded Systems


Dr. Sébastien Gérard
(CEA LIST, France, sebastien.gerard[at]cea.fr)

Dr. Julio Medina
(CEA LIST, France, julio.medina[at]cea.fr)

Pr. Dorina Petriu
(Carleton University, Canada, petriu[at]sce.carleton.ca)

(with the contributions of: Huascar Espinoza, Frédéric Thomas and Safouan Taha)


# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

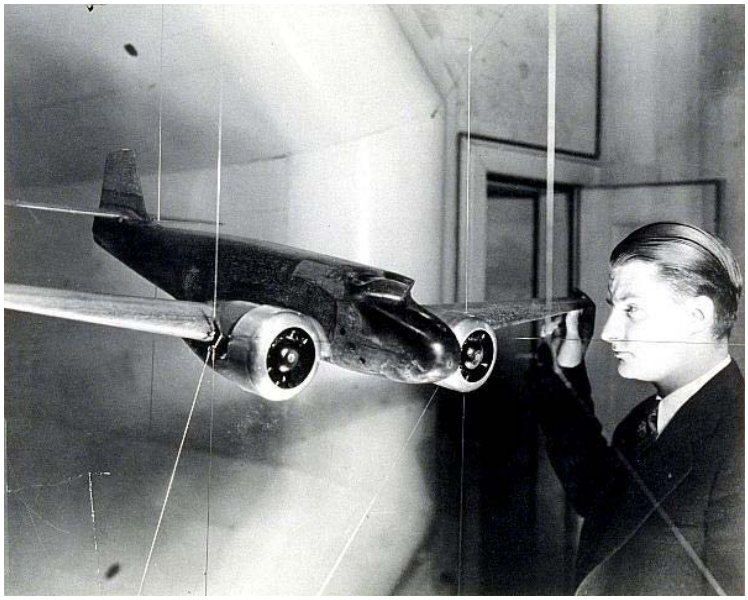
*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE

# Models in Traditional Engineering

- Probably as old as engineering



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

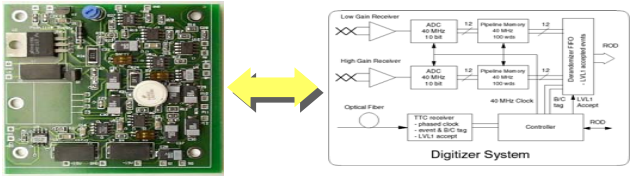


# What is a Model in MDD

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 viewpoint.
  - The viewpoint defines concern, scope and detail level of the model.




**Modeled system** **Functional model**

# 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 fidelity
- To minimize development risks
  - Through analysis and experimentation performed earlier in the design cycle
  - Enable to investigate and compare alternative solutions
- To improve communication ...
  ... to foster information sharing and reuse!
  - A model is often best suited than a long speech !
- To focus on specific domain expertise while developing software system
  - Domain Specific Language



# 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 modeled 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…

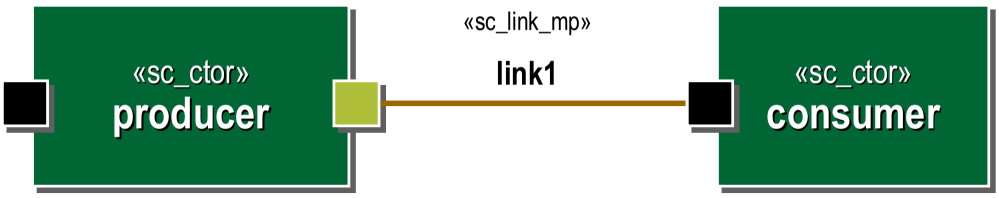
```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
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);}};
```

**Can you spot the architecture?**

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



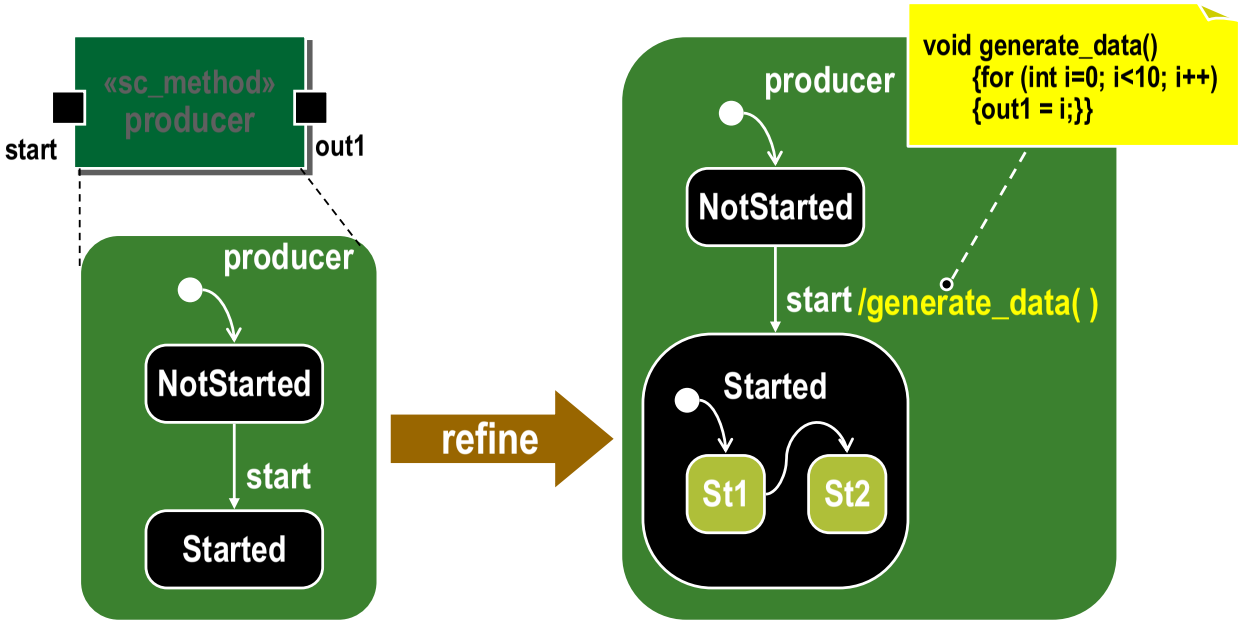
# …and its UML Model




**Can you spot the architecture?**

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

# Model Evolution: Refinement



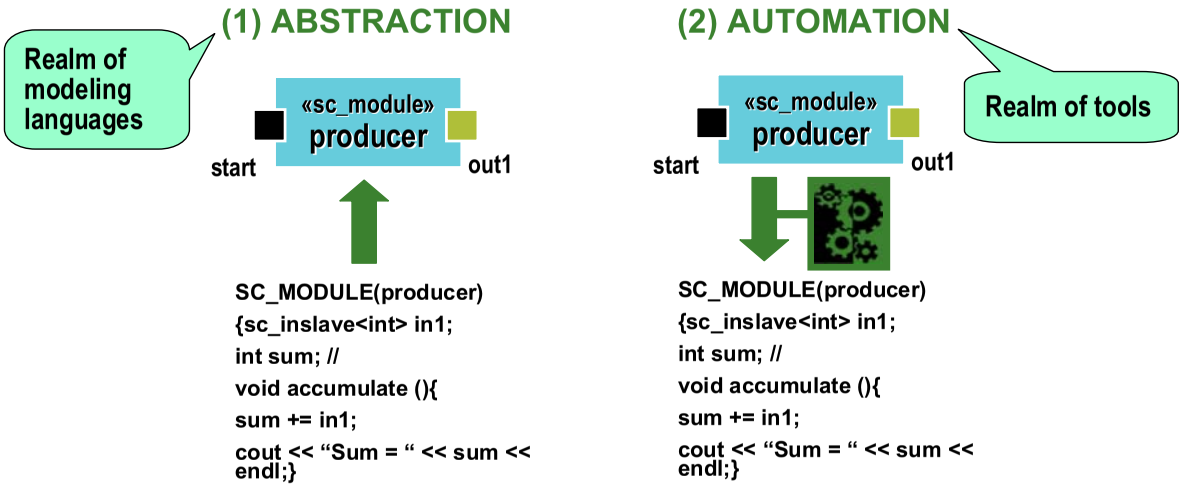

- Models can be refined continuously until the specification is complete

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



# Model-Driven Style of Development (MDD)

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




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

# The Remarkable Thing About Software

*Software has the rare property that it allows us to directly evolve models into fully-fledged implementations without changing the engineering medium, tools, or methods!*

⇒ This ensures perfect accuracy of software models, since the model and the system that it models are the same thing.

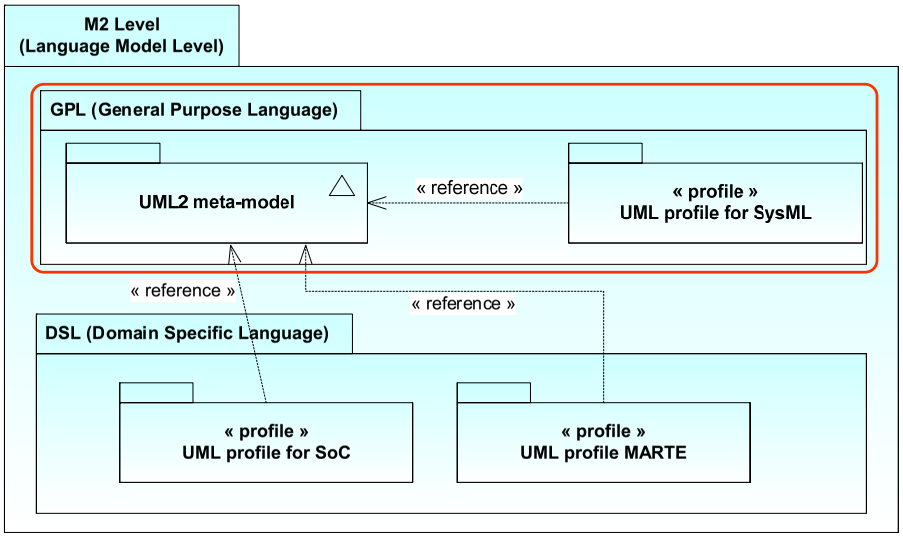
*The software model may evolve into the system it was modeling in a seamless process.*

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

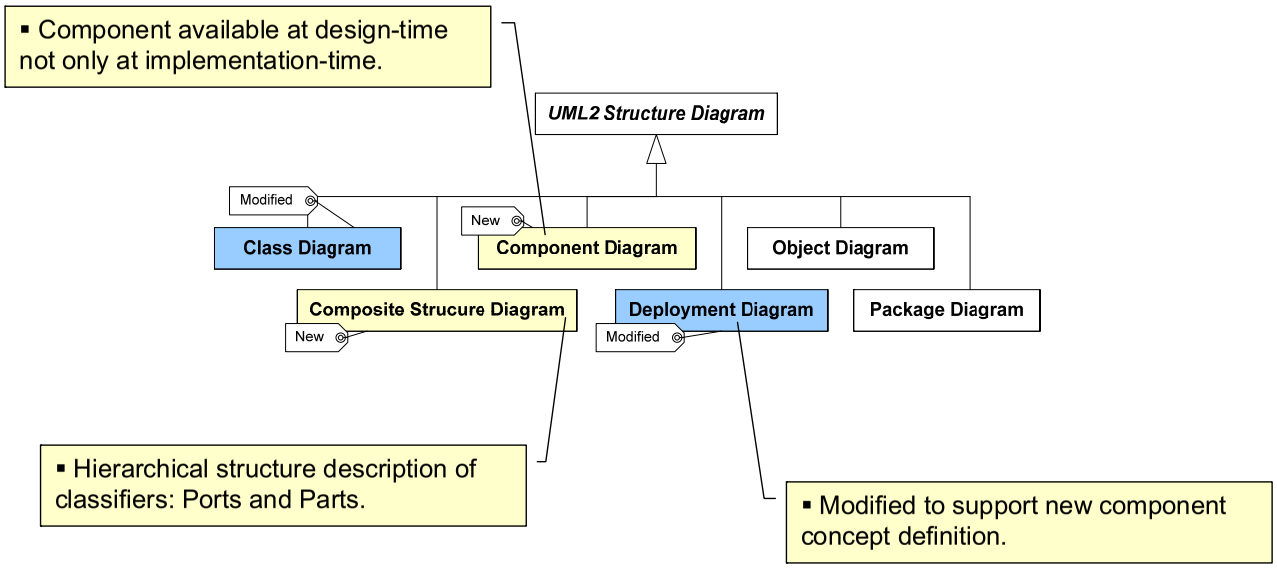


# Two Kinds of OMG Modeling Languages

- General Purpose Language (GPL)
  - E.g., UML2, SysML and CCM
- Domain Specific Language
  - E.g. LwCCM, UML profile for SoC and MARTE

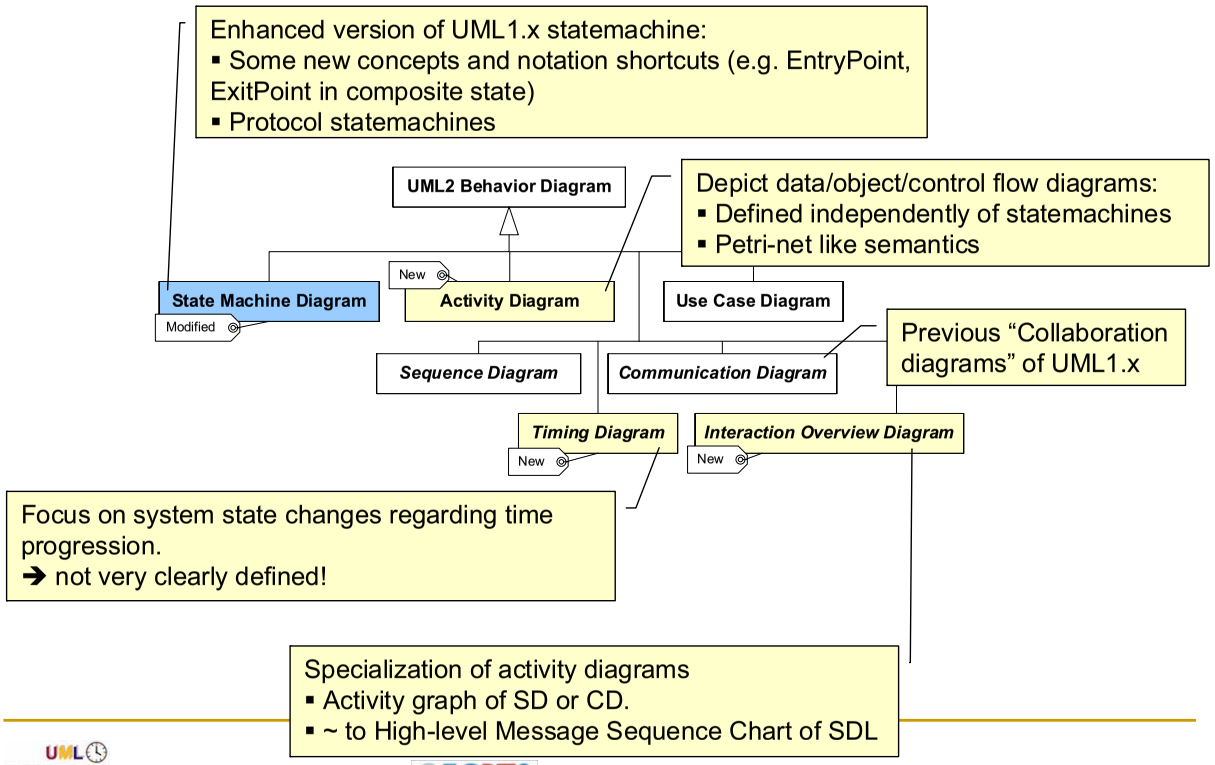


# From UML 1.x to UML 2.0: Static Point of View






# From UML 1.x to UML 2.0: Dynamic Point of View

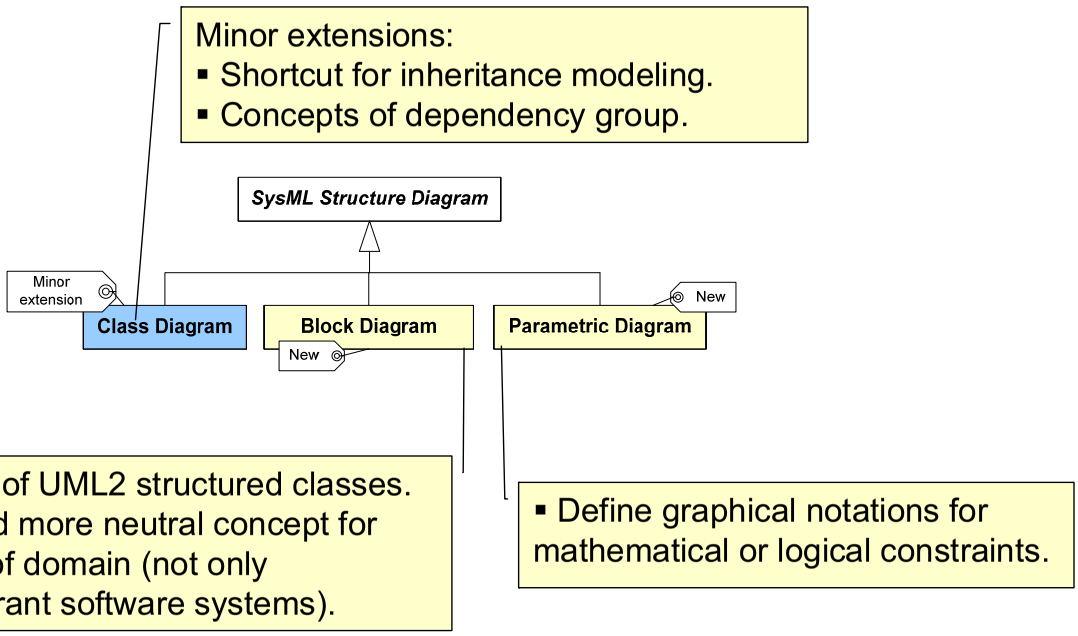


# SysML: A UML Extension for System Engineering

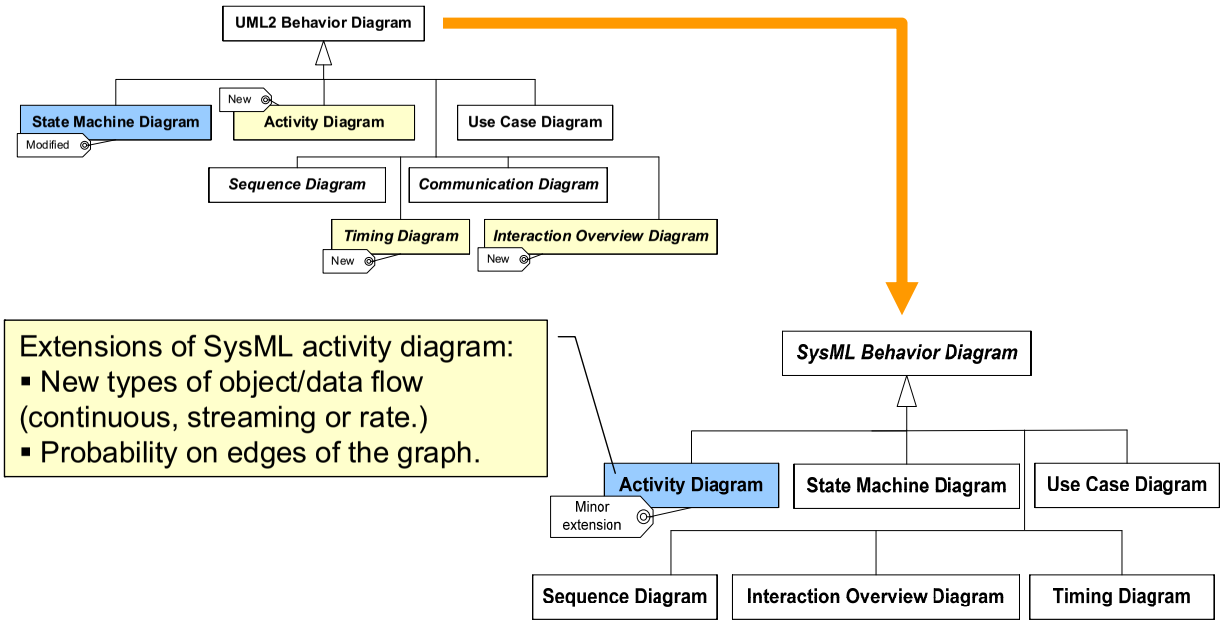
- Systems Modeling Language (www.omgsysml.org)
  - General-purpose systems modeling language
    - Specification, analysis, design, verification and validation of a broad range of complex systems
  - Expected to be customized to model domain specific applications
    - E.g., Space, Automotive, Aerospace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



# From UML 2.0 to SysML: Static Point of View



# From UML 2.0 to SysML: Dynamic Point of View




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



# Domain Specific Modeling Languages

- Use of domain concepts as language constructs
  - E.g. concepts of “memory”, “task”, “bus”, instead of general ones “components”, “nodes”, “classes”…
- Criteria for success of a DSML
  - Technical validity and absence of ambiguities (MoCCs mapping)
  - Interoperability with other technologies (verification, code generation)
  - Domain appropriateness: syntax close to embedded systems ontology
  - Support: tools, documentation, capacity for evolution

But mainly… balance between *expressive power* and *simplicity!*

- Main approaches to defining a DSML
  - Define a new language from scratch
  - Extend an existing language (add new language constructs)
  - Refine an existing language (specialize language constructs)

# Which Technics for Making an OMG DSL?

- Meta-models (using MOF)
  - For heavyweight extension mechanisms
  - Ensures full manipulation of MMs
    - Add, remove meta-classes and relationships inbetween
- Profiles
  - For lightweight extension mechanisms
  - Adaptation of existing meta-models without modifications!
    - E.g., Only possible to add constraints
  - May extend any standard MM of the OMG
    - E.g., UML profiles and CWM profiles
- Profiles vs. meta-models
  - It may depend on:
    - Scope of the extensions
    - Tooling constraints
    - Engineer level of experiment/education
    - …

Extracted from UML superstructure doc.:
"There is no simple answer for when you should create a new metamodel and when you instead should create a new profile."



# 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
- How to face these limitations?
  - Profile semantics: on-going work: 'Executable UML Foundation'
  - Clear mapping of profile extensions to domain ontology
  - Create rich profiles (expressiveness) and complement with specific methodologies (precise semantics and pragmatics)

# Main Goals for Making UML Profiles

- Define a domain specific terminology
- Define a domain specific notation
- Define concrete syntax to specific elements
  - E.g., for action
- Define semantics rule for:
  - Variation points
    - E.g., On BroadcastSignalAction, scope of target objects in Signal.
  - Ambiguous definition
  - New purpouse
- Add usage constraints on the UML MM
  - e.g. for method point of view
    - E.g., to force behavior specification via protocol state-machine
- Tag a model with additional information
  - e.g. for code generation purpose
    - E.g., for C++, «virtual», «inline»…

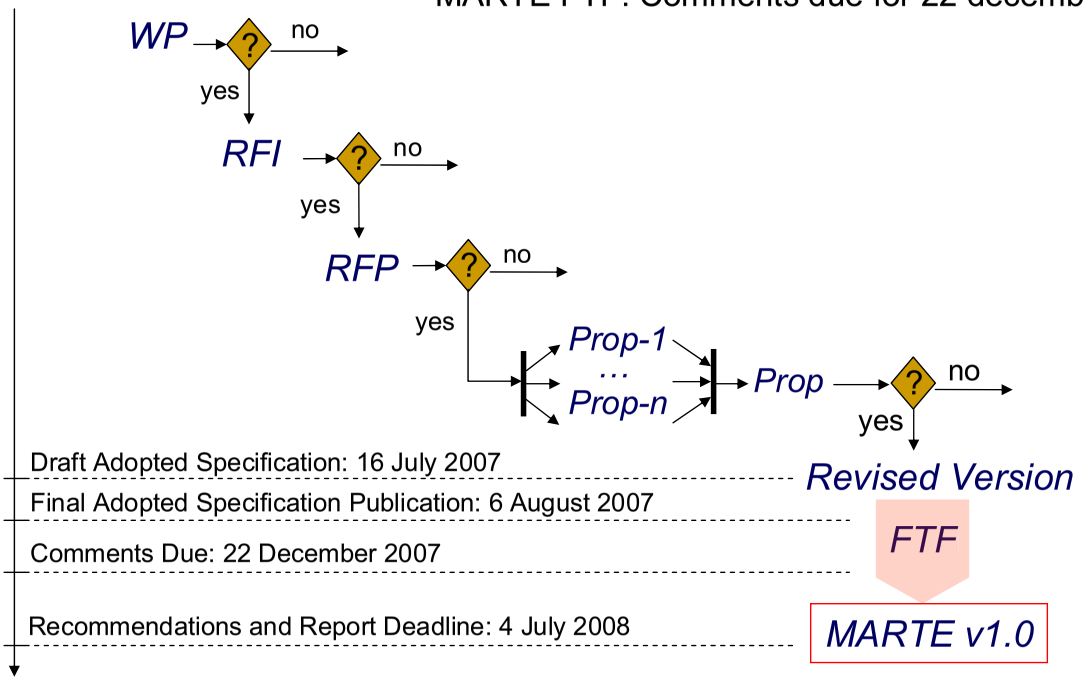


# Outlines of UML2 Extension Mechanisms

- Profiles
  - Define limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain.
  - Consists of stereotypes extended the metamodel classes.
- 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 reffered to as tagged values

# OMG Standardization Process

MARTE FTF: Comments due for 22 december 2007 !






# UML Profiles for RTES

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
- Despite its extensive use, several improvements were required:
  - Modeling HW and SW platforms, Logical Time, MoCCs, CBSE…
  - Alignment to UML2, QoS&FT, MDA,…
  - SPT's constructs were considered too abstract and hard to apply
  - …

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

# General Requirements of the MARTE RFP

- Proposals shall support Modeling and Analysis of Real-Time and Embedded (in short MARTE) systems including its software and hardware aspects.
  - The Proposals will define a metamodel and its underlying UML profile.
  - It shall be possible to use independently software and hardware parts of the profile.
  - It shall comply with existing standards
  - It shall update the SPT profile 1.1

Ref. of MARTE RFP (realtime/05-02-06):
http://www.omg.org/cgi-bin/doc?realtime/05-02-06



# The ProMARTE Team

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

* Submitter to OMG UML Profile for MARTE RFP

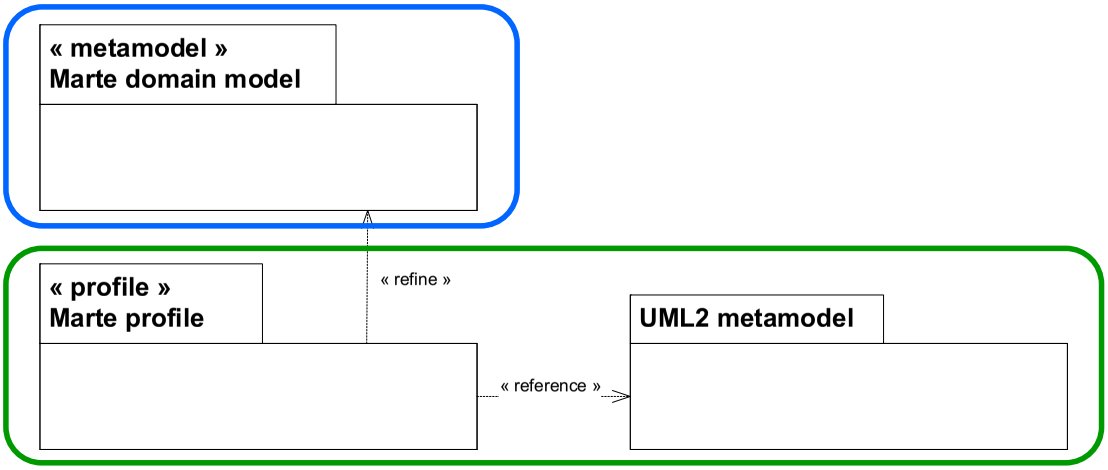
# Relationships with other OMG Standards

- Relationships with generic OMG standards
  - Profile the UML2 superstructure meta-model
  - Replace UML profile for SPT
  - Use OCL2
- 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
      - 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
      - Specialization of SysML allocation concepts and reuse of flow-related concepts
      - Ongoing discussion to include VSL in next SysML version
      - Overlap of team members

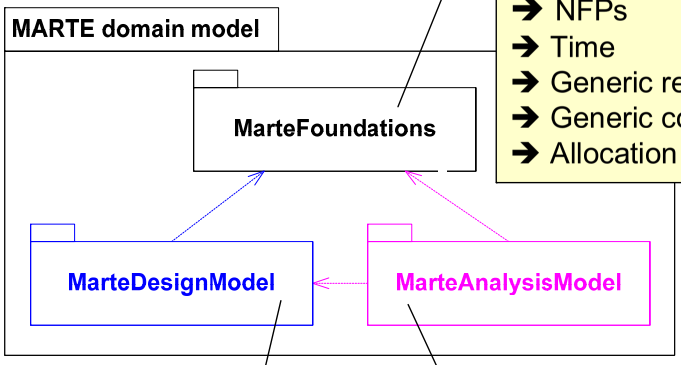


# Design Pattern Adopted for the MARTE Profile

- Stage 1 → Description of MARTE domain models
  - Purpose: Formal description of the concepts required for MARTE
  - Techniques: Meta-modeling
- Stage 2 → Mapping of MARTE domain models towards UML2
  - Purpose: MARTE domain models design as a UML2 extensions
  - Techniques: UML2 profile



# MARTE Overview




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

Specialization of MARTE fondations 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



# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT&E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

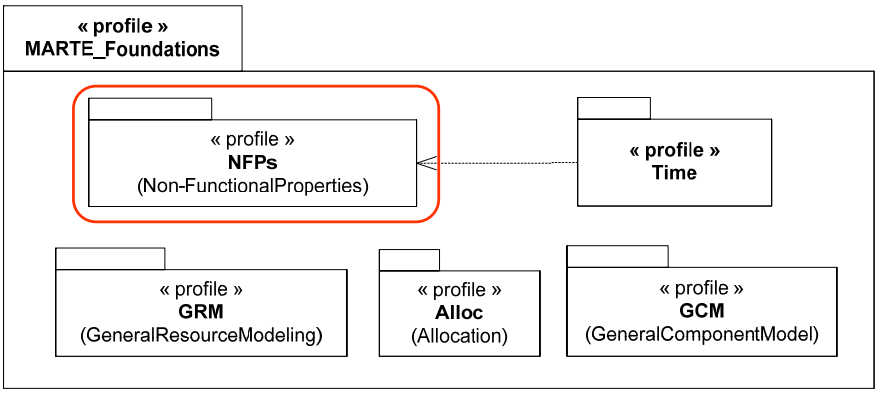
*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE

# Outlines of the MARTE Foundations

- MARTE foundations
  - Define a set of basic concepts for MDD of RTES
  - Intended to be used as basic layer at OMG for future RTE related stds
- Consists of five sub-profiles






# Non-Functional Properties (NFPs)

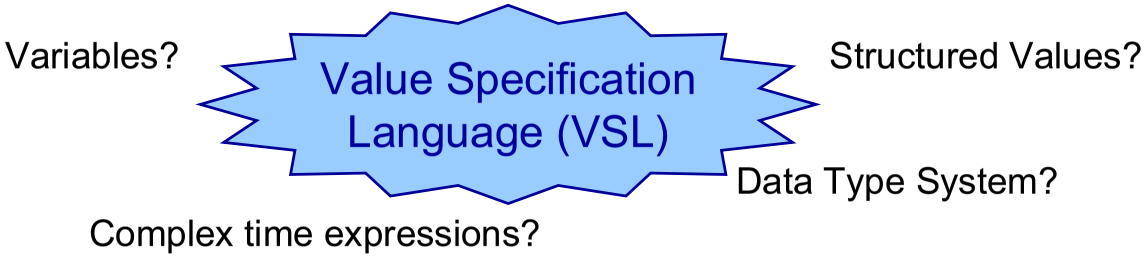
Non-functional properties describe the "fitness" of systems behavior (E.g., performance, memory usage and power consumption)

- Nature of NFPs
  - Quantitative: magnitude + unit (E.g., energy, data size and 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!!

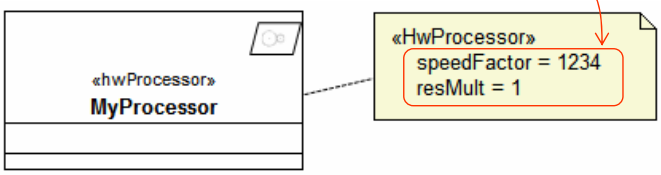

But UML expression syntax is also not sufficient!!





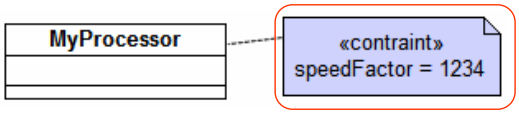

# The MARTE's NFP sub-profile

- Relies on the ability to put additional information on models
- Three possible model-based annotation mechanisms in UML
  - Value of stereotype properties




  - Slots value of classifier instance


  - Constraints



# Annotating NFPs in Tagged Values

**1) Declare NFP types**

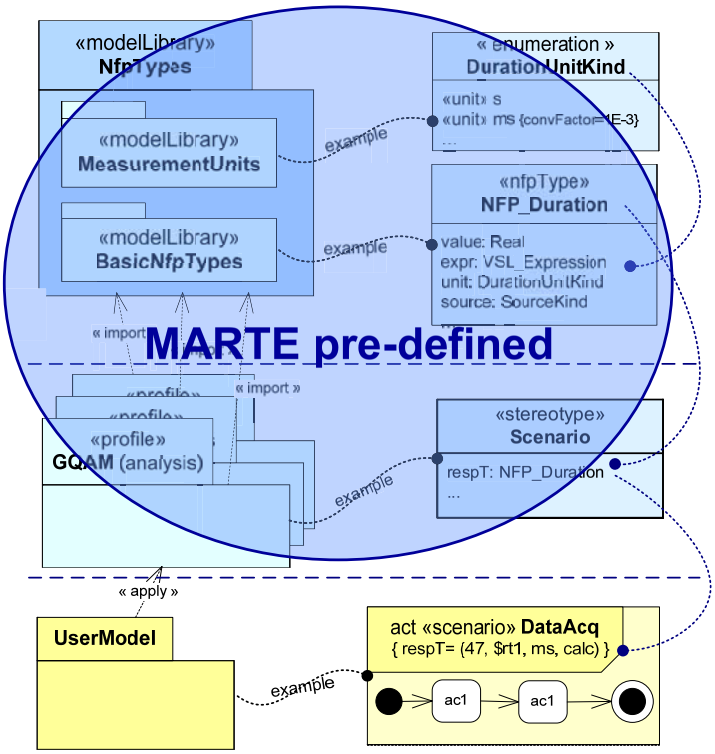
- Define measurement units and conversion parameters
- Define NFP types with qualifiers

**2) Define NFP-like extensions**

- Define stereotypes and their attributes using NFP types

**3) Specify NFP values**

- Apply stereotypes and specify their attribute values using VSL






# Annotating NFPs in Slots

**1) Declare NFP types**

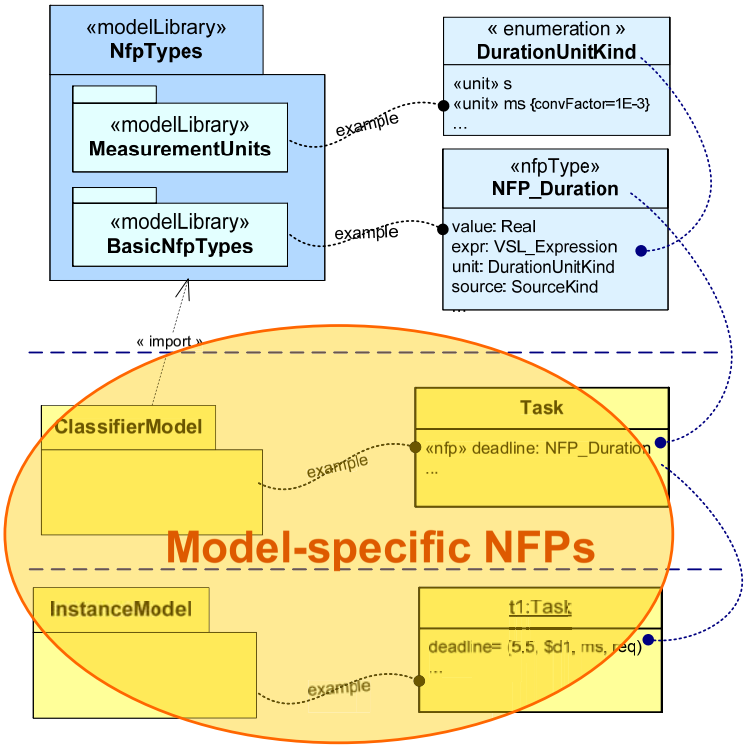
- Define measurement units and conversion parameters
- Define NFP types with qualifiers

**2) Declare NFPs**

- Define classifiers and their attributes using NFP types
- Such attributes are marked as «nfp»

**3) Specify NFP values**

- Instantiate classifiers and specify their slot values using VSL



# Annotating NFPs in Constraints

**1) Declare NFP types**

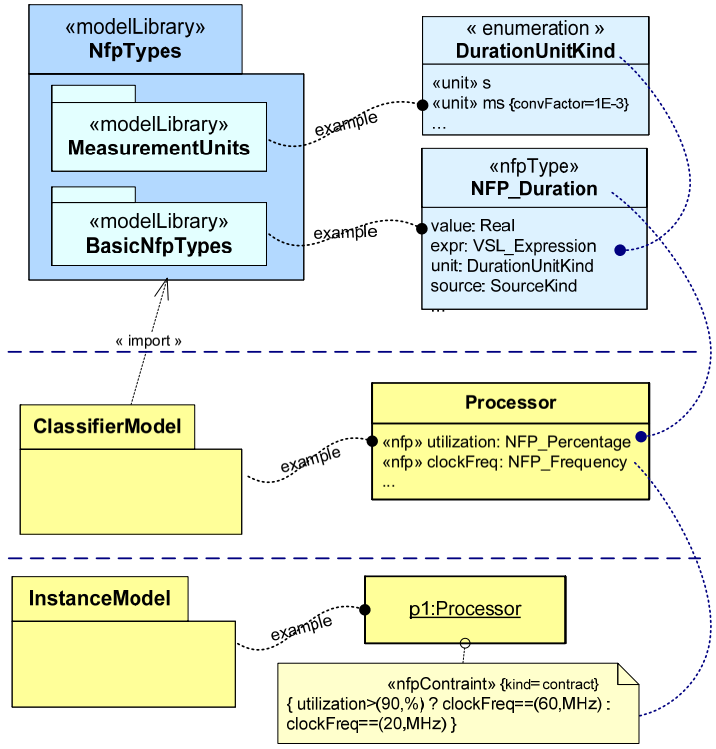
- Define measurement units and conversion parameters
- Define NFP types with qualifiers

**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






# The MARTE's NFP Modeling Framework

Three main language extensions to UML syntax:

1. Grammar for extended expressions
2. Stereotypes for extended data types
2. Complex time expressions

Value Specification Language (VSL)

# Basic Textual Expressions in VSL

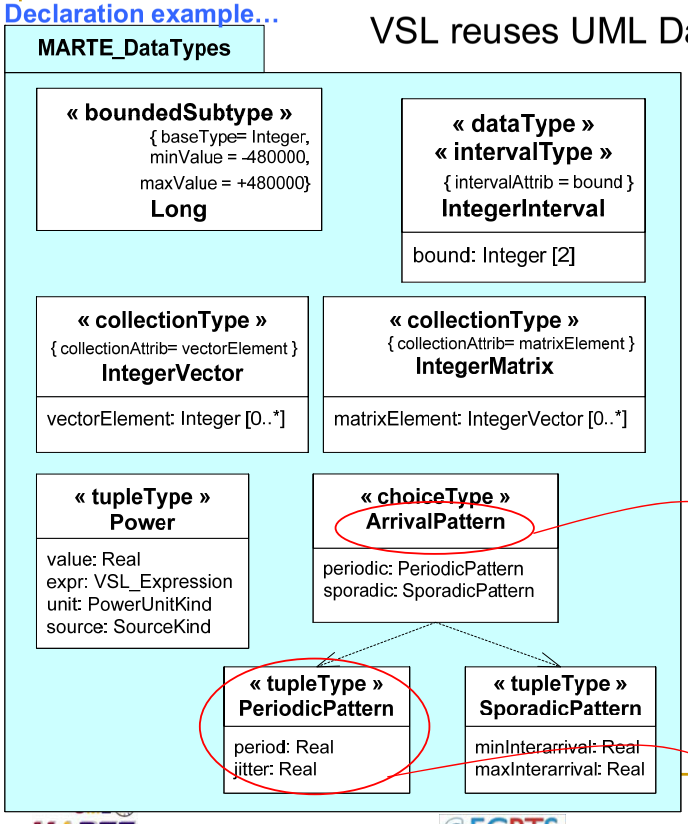
- ✓ Extended Primitive Values
- ✓ Extended Composite Values
- ✓ Extended Expressions

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



# VSL Extended Data Types

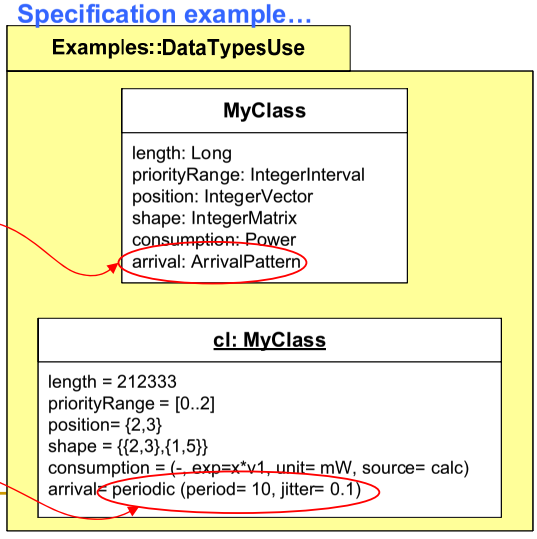
Declaration example…



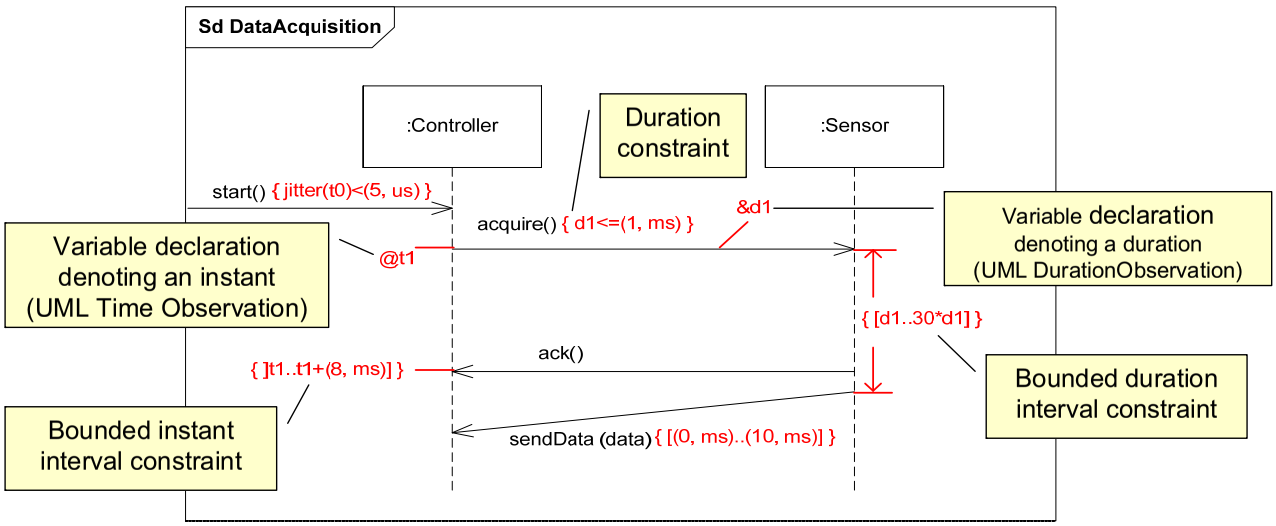

VSL reuses UML DataType constructs, but adds…

- ✓ BoundedType
- ✓ IntervalType
- ✓ CollectionType
- ✓ TupleType
- ✓ ChoiceType

Specification example…



# Examples of Time Expressions with VSL



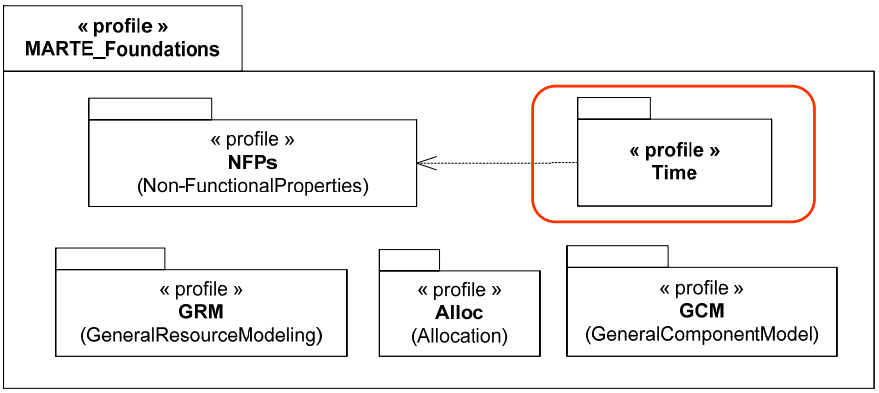



# Summary of the "MARTE's NFP"

- Synthesis of best modeling practices…
  - OCL: full constraint language, but complex 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 (occurrence index, jitters,...)
  - Formally defined by abstract and concrete syntaxes (grammar).
  - Flexibility: expression operators in model-specific libraries

# Outlines of the MARTE Foundations

- MARTE foundations
  - Define a set of basic concepts for MDD of RTES
  - Intended to be used as basic layer at OMG for future RTE related stds
- Consists of five sub-profiles

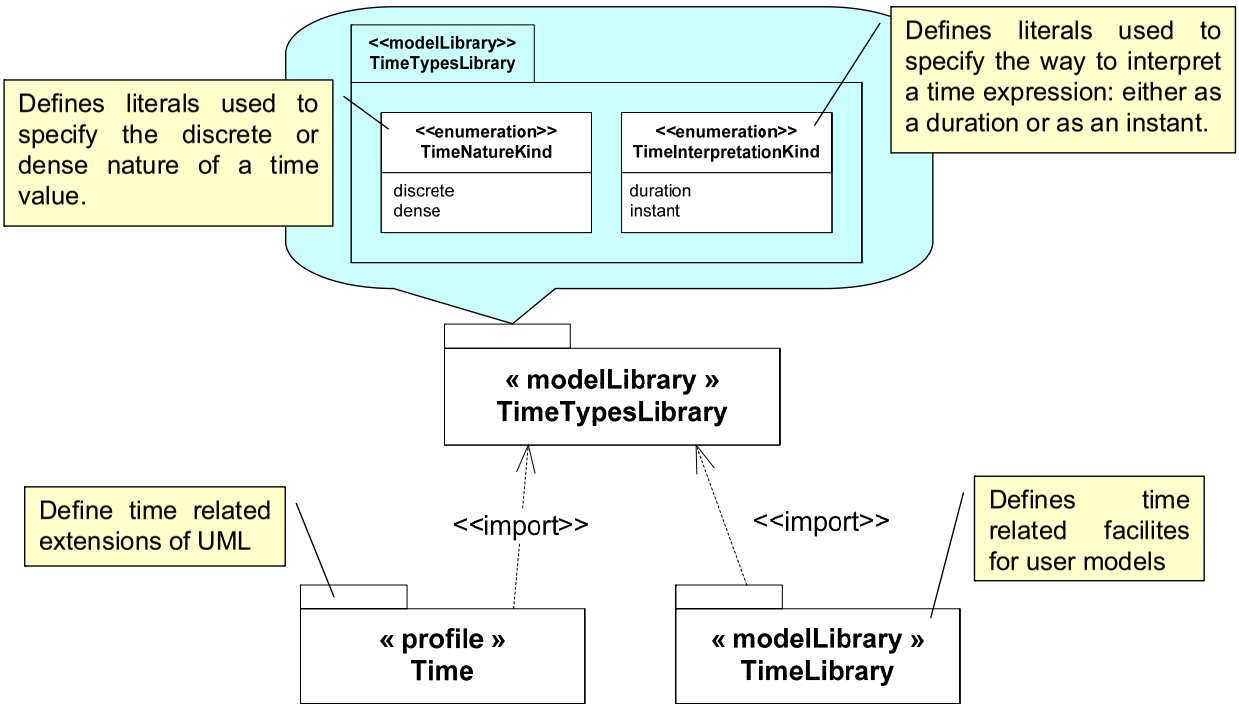





# Scope of the Time Profile of MARTE

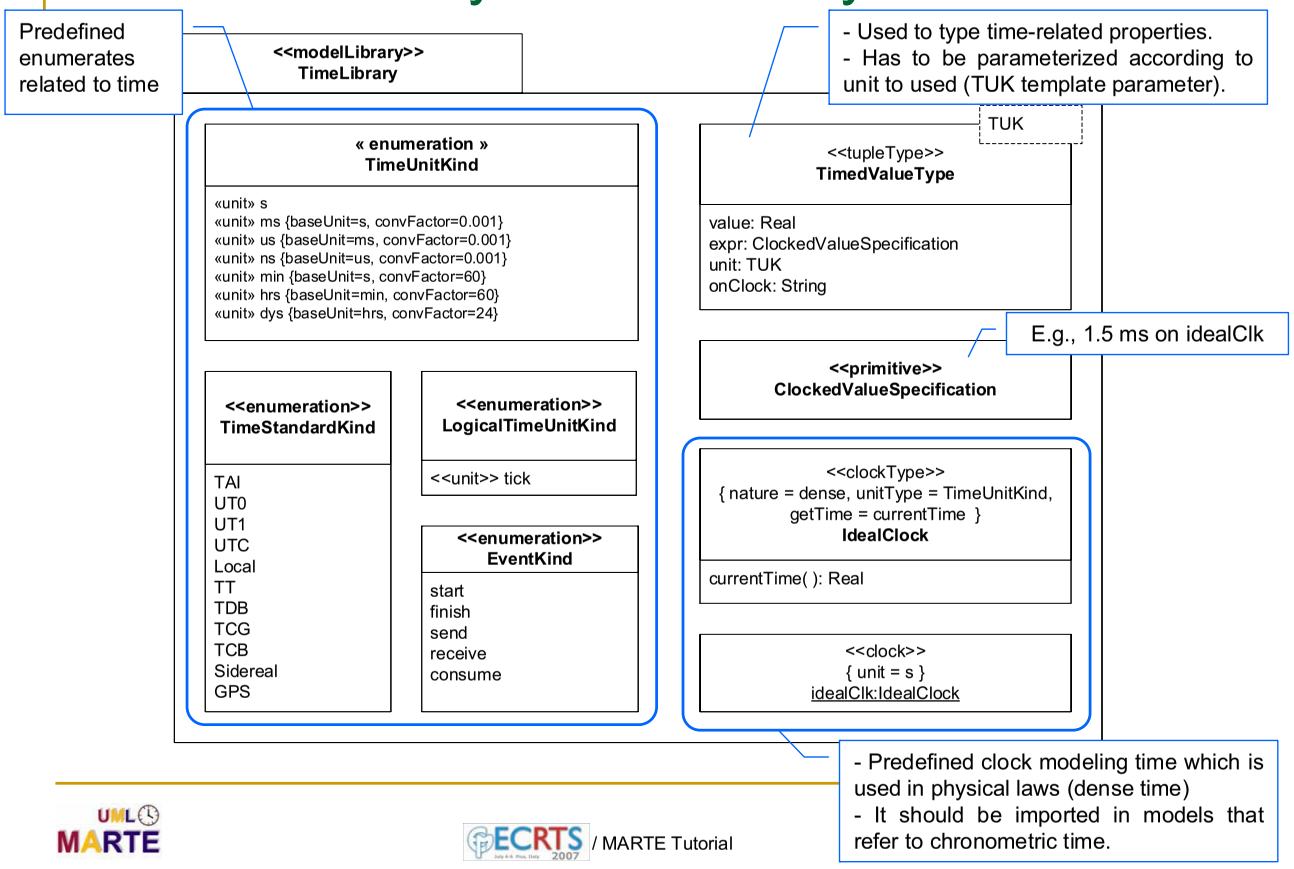
- MARTE adopts models of time that rely on partial ordering of instants
- Three basic time models
  - Chronometric time model
    - Mainly concerns with time cardinality
      - E.g., delay, duration and clock time
  - Logical time model
    - Mainly concerns events ordering.
      - E.g., ev1 is before ev2
  - Synchronous time model
    - Specialization of the logical time model
    - Introduce notion of simultaneity
      - E.g., ev1 and ev2 occurs at the same instant

# Time Packages of MARTE



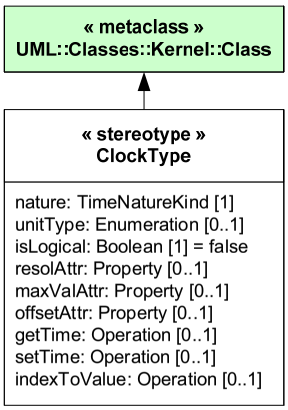


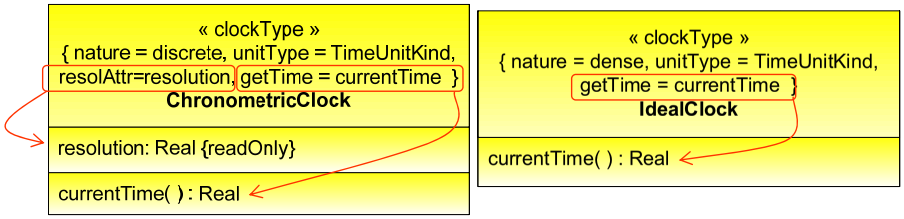

# The TimeLibrary Model Library

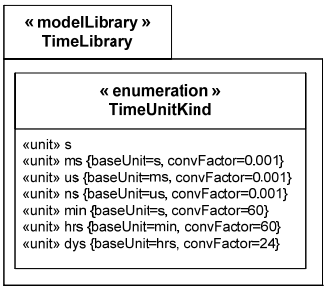


# « ClockType »

- Used to declare clock types of user applications
- Example of properties defining the clock type
  - nature: TimeNatureKind [1]
    - Dense or discrete
  - isLogical: Boolean [1]
    - Default value is false
  - resolAttr: Property [0..1]
    - Refers a property which slot will define the clock resolution
- Examples



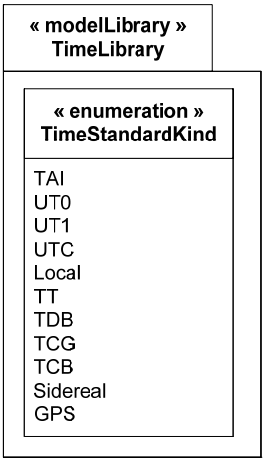



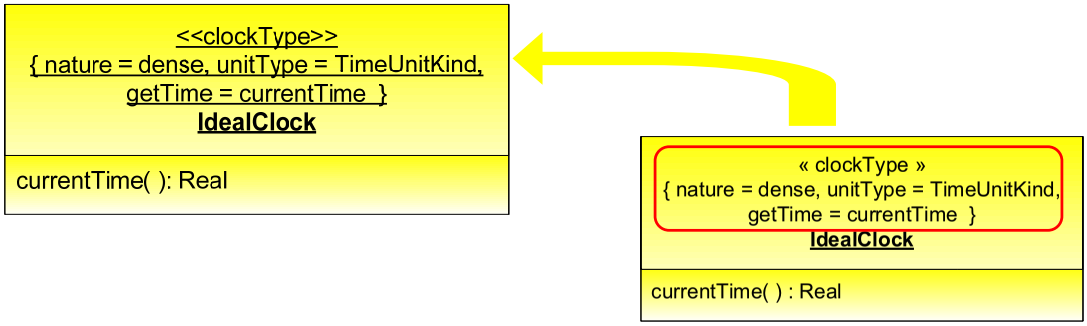






# « Clocks »

- Clocks are instance specification of ClockTypes
- Clocks properties are:
  - standard: TimeStandardKind [0..1]
  - unit: Unit [0..1]
  - type: ClockType [1]
- Example



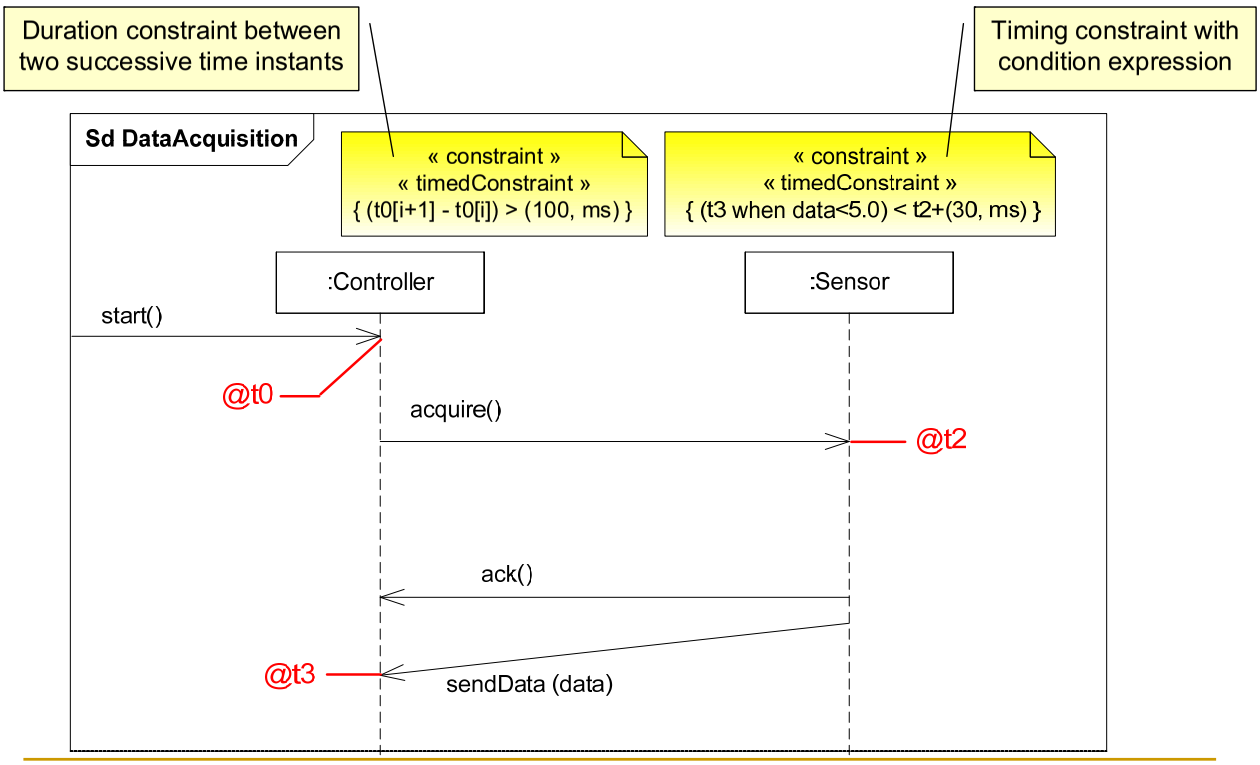



# Timed & Clock Constraints and TimedDomain

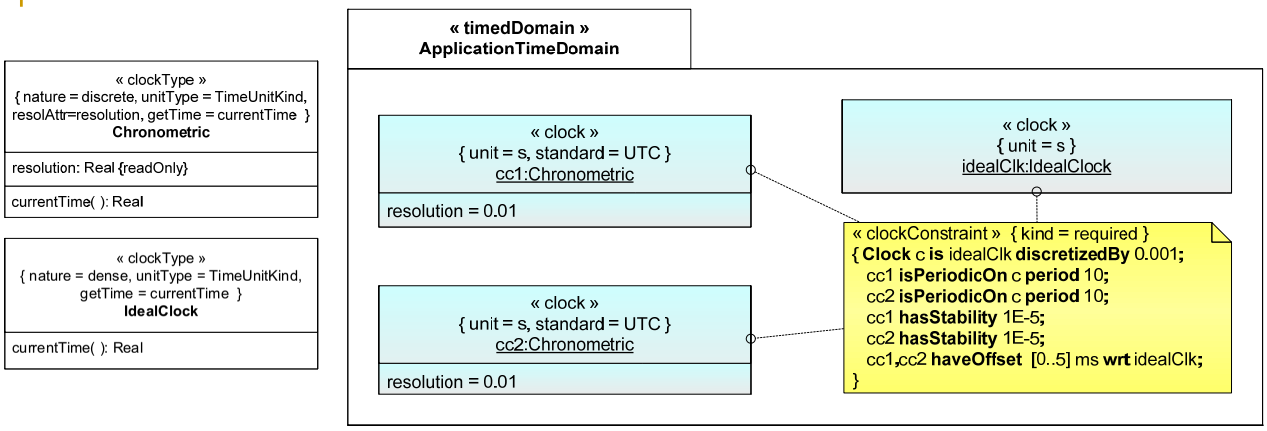
- « TimedConstraint »
  - Imposes constraints on either instant values or duration values associated with model elements bound to clocks.
  - Can be conveniently expressed in VSL
- « ClockConstraint »
  - Imposes dependency between clocks
  - Refers to a set of clocks and possibly to other model elements
  - Described via an opaque expression using a dedicated language
    - E.g. CCSL (Clock Constraint Specification Language) defined in Annex C
- « TimedDomain »
  - Namespace for:
    - Clock and ClockTypes user definitions
    - But also for ClockConstraints (see example later)
  - Model elements of the *TimedDomain* may refer to *Clocks* to express that their behavior depends on time

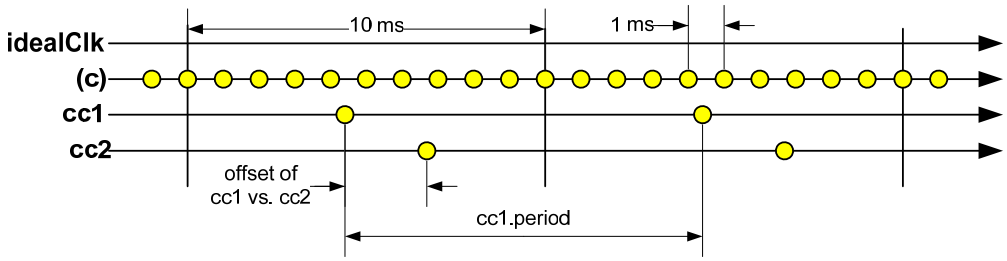


# Example of Timed Constraints



# Example of TimedDomain with a ClockConstraint

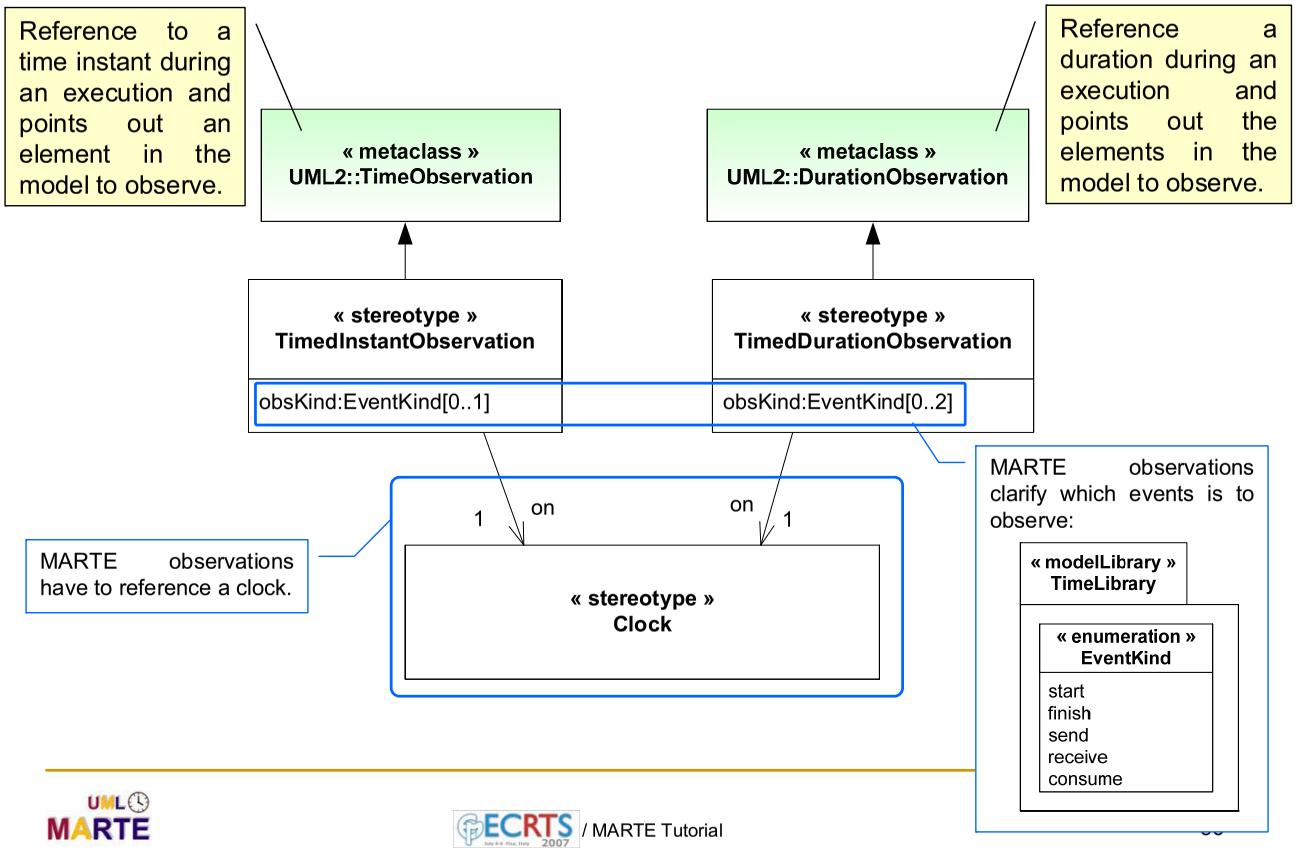


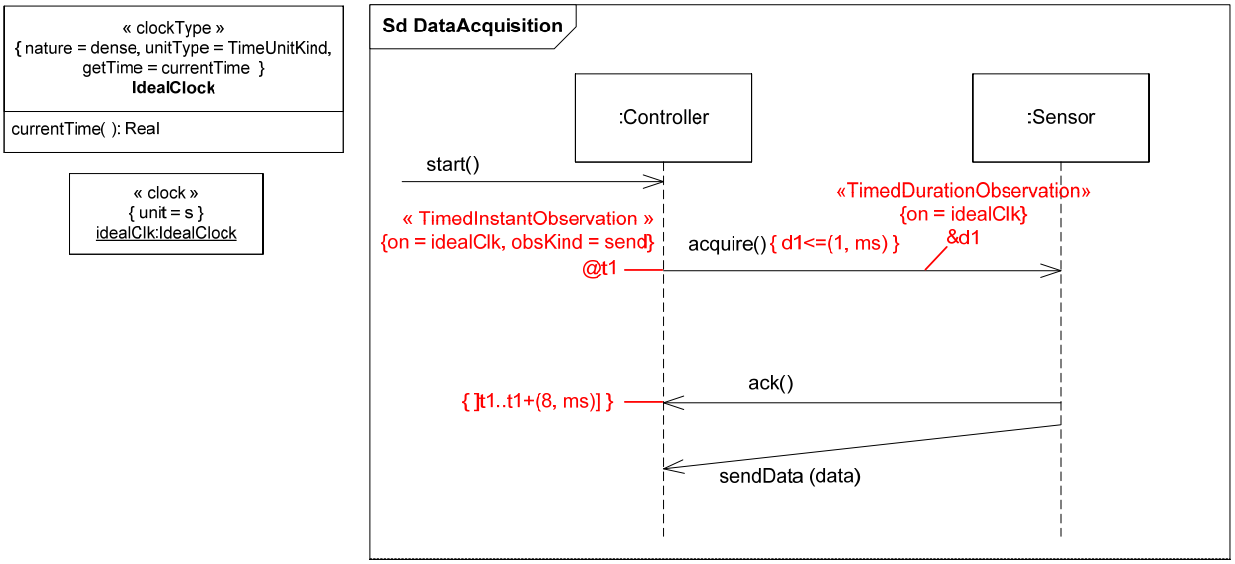

Instants chronograms of clocks cc1 and cc2



# Timed Instant and Duration Observations

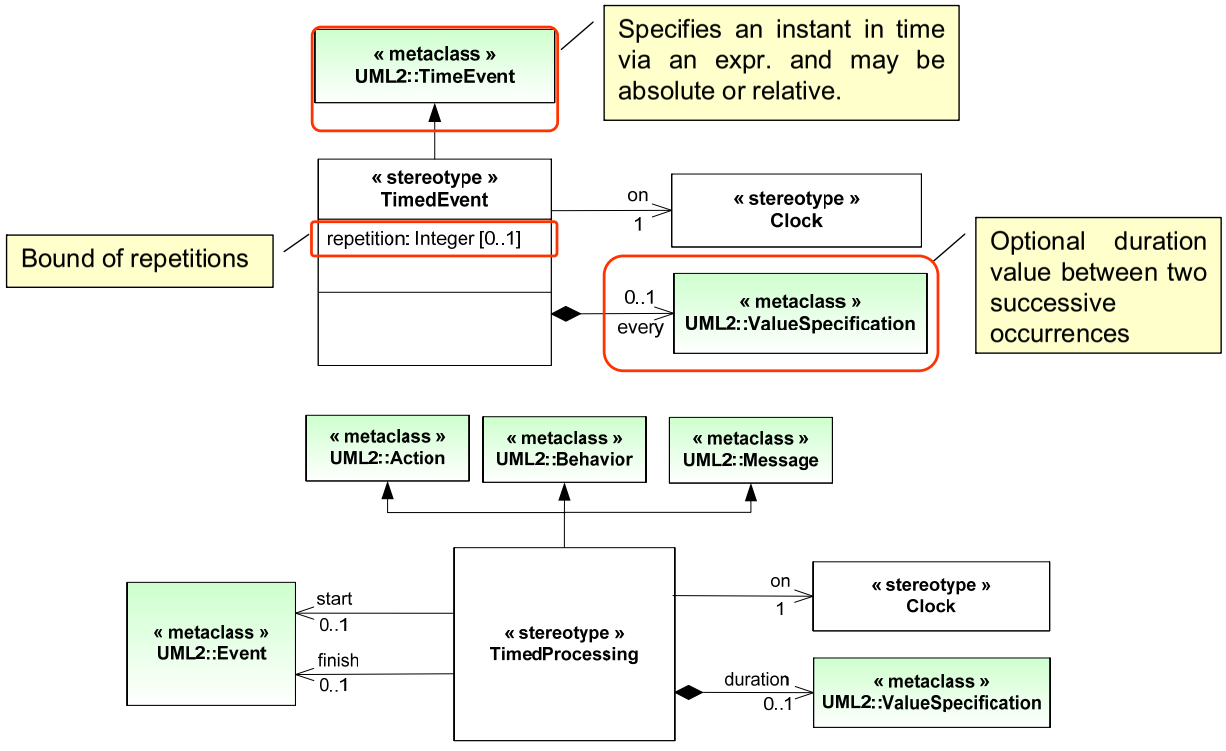


# Examples of «TimedInstantObservation» and «TimedDurationObservation»

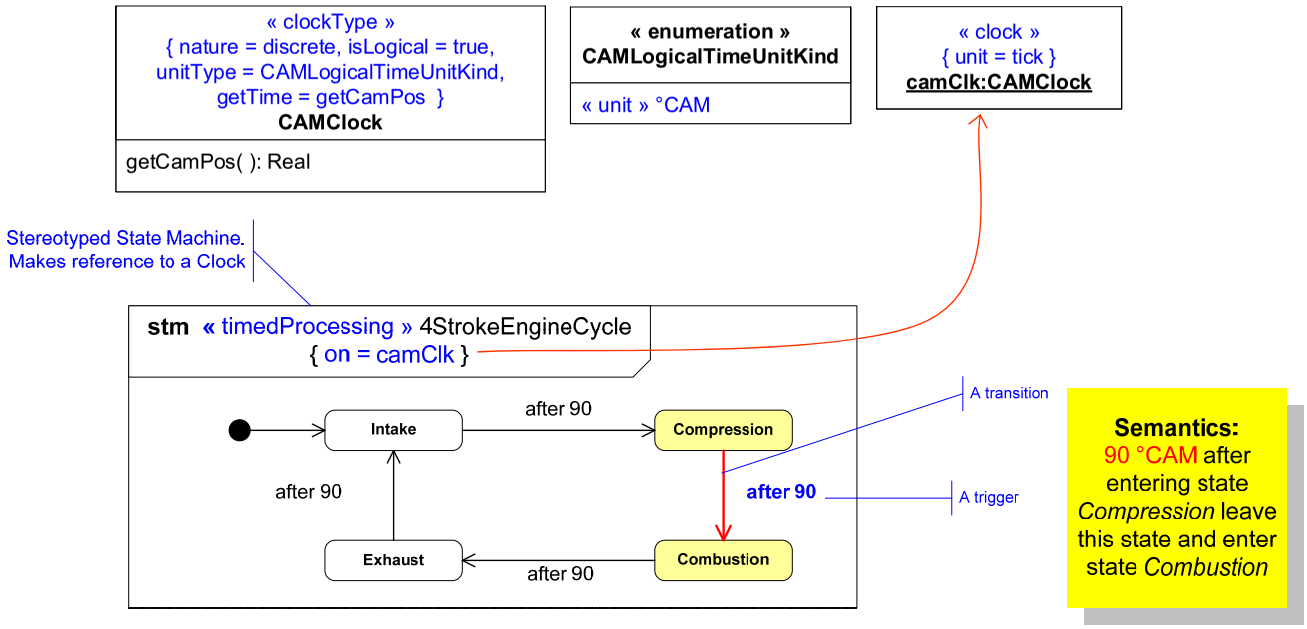





# Timed Events and Processings



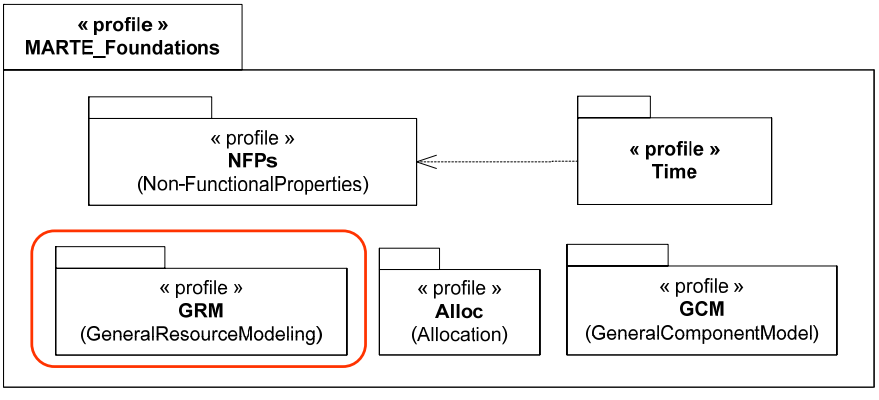
# Example of «TimedProcessing» & «TimedEvent» with Logical Time






# Outlines of the MARTE Foundations

- MARTE foundations
  - Define a set of basic concepts for MDD of RTES
  - Intended to be used as basic layer at OMG for future RTE related stds
- Consists of five sub-profiles

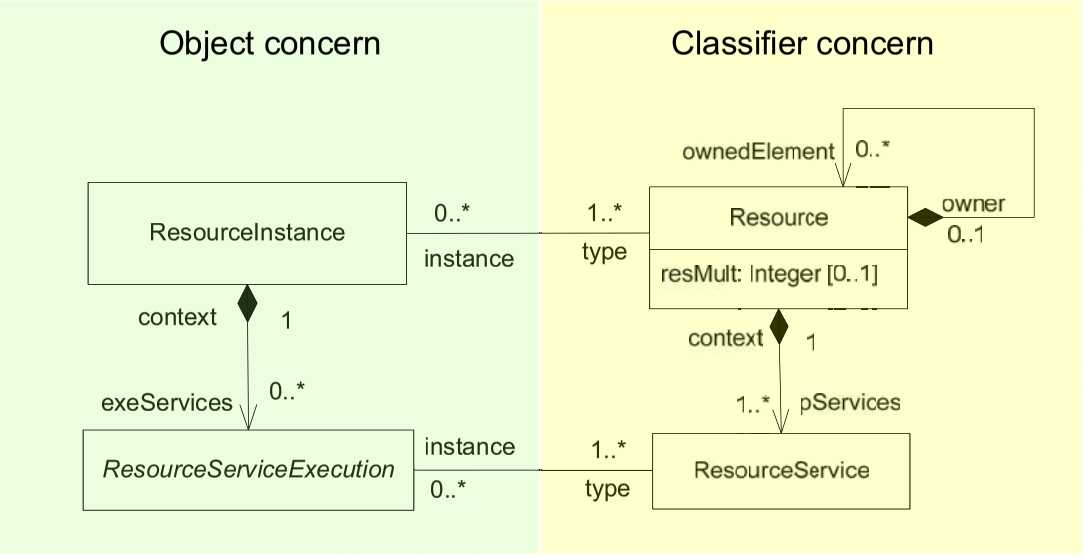


# Outlines of the GRM package

- 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

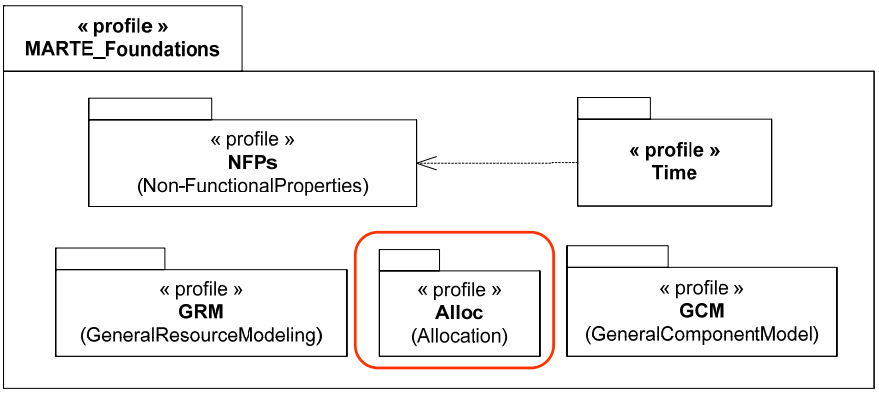


# Essence of the GRM Package



# Outlines of the MARTE Foundations

- MARTE foundations
  - Define a set of basic concepts for MDD of RTES
  - Intended to be used as basic layer at OMG for future RTE related stds
- Consists of five sub-profiles





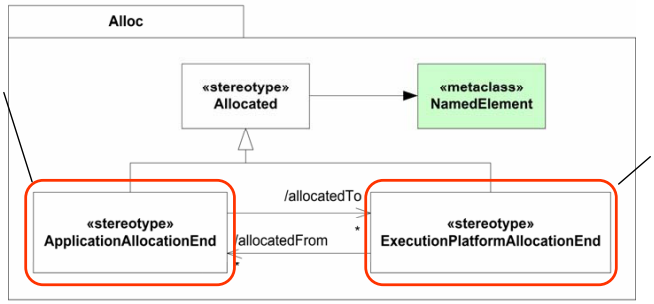

# Allocation & Refinement

- 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

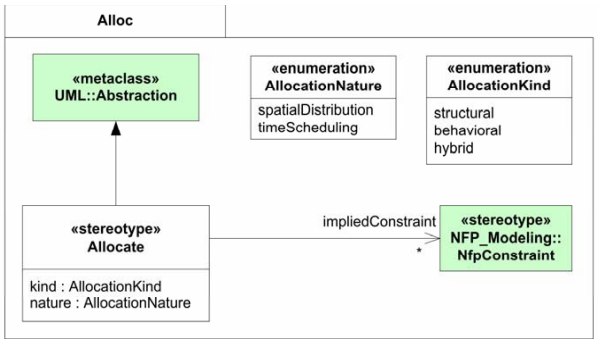
- Identify possible sources and targets of allocations

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



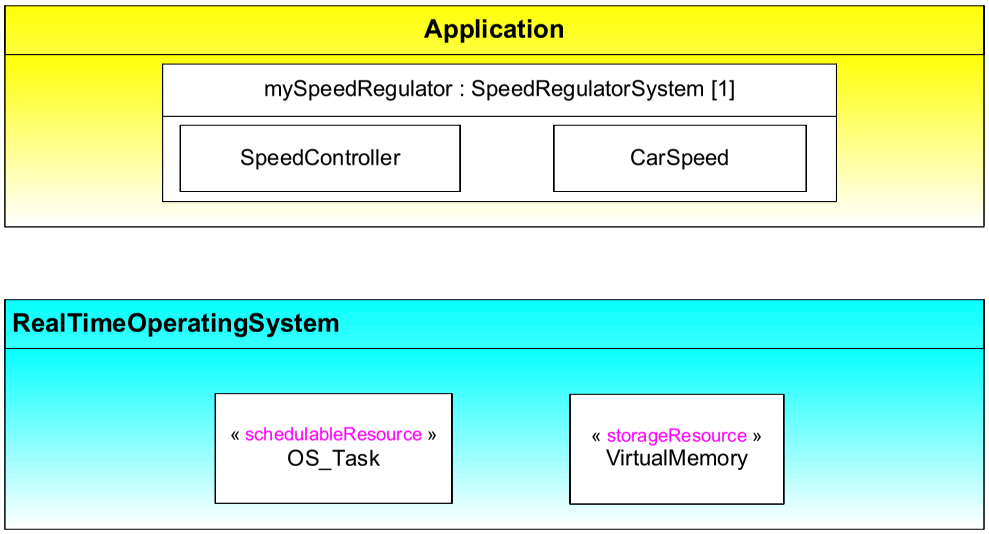

What can serve as a target of an allocation, the physical view:
➔ a resource or a service.

- Define allocation relationships and its features

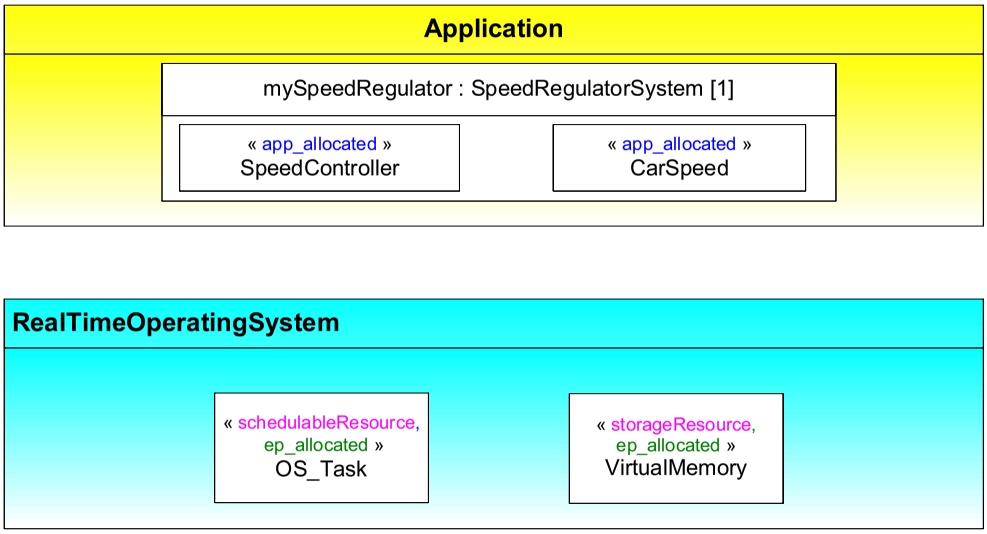





# Allocation example (1)

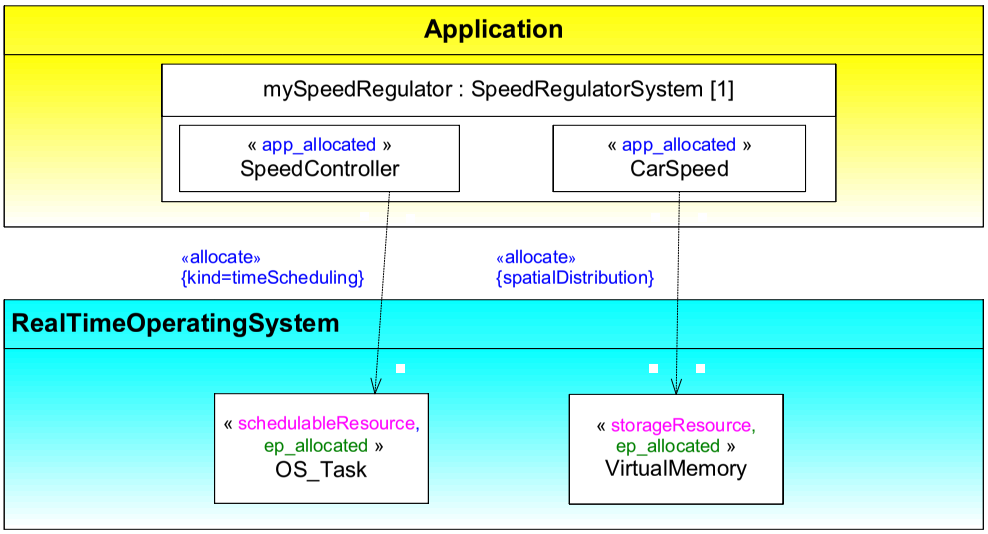


# Allocation example (2)

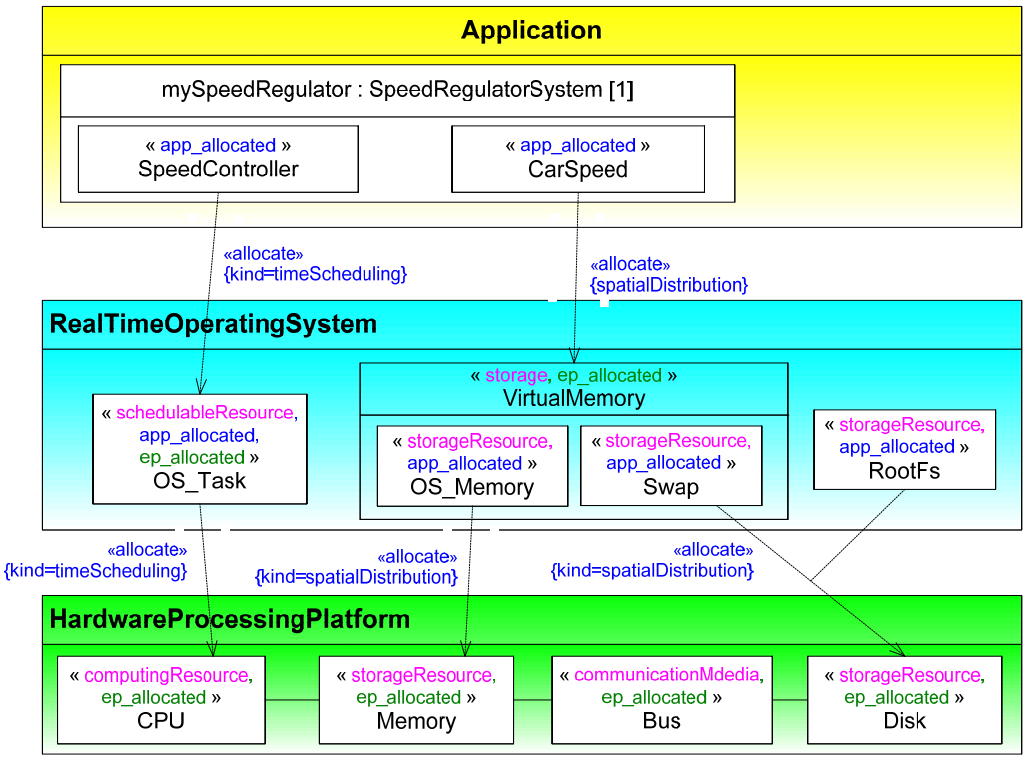





# Allocation example (3)



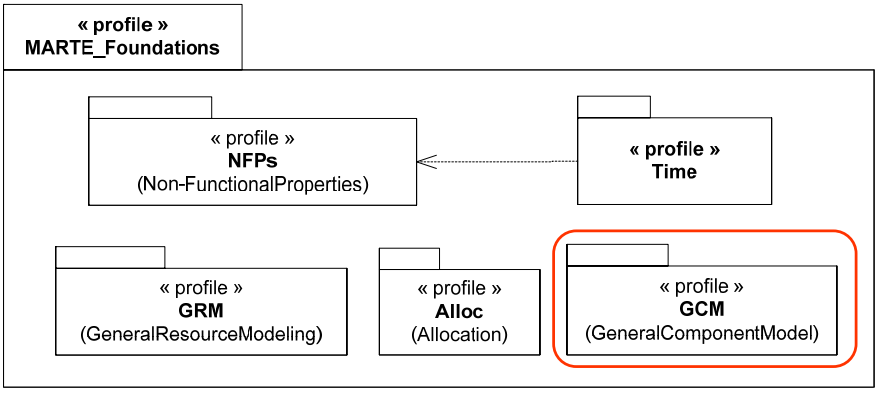
# Allocation example (4)






# Outlines of the MARTE Foundations

- MARTE foundations
  - Define a set of basic concepts for MDD of RTES
  - Intended to be used as basic layer at OMG for future RTE related stds
- Consists of five sub-profiles

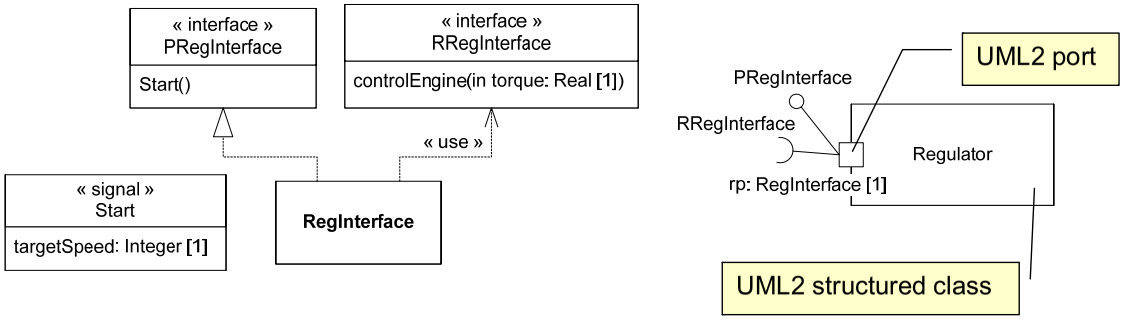


# The Marte General Component Model

- Defined within MARTE to support CBSE and to provide a common denominator among various existing component models, which in principle do not target exclusively the RTE domain
- Introduce no new component-related concepts
  - Has to cope with various component models
    - E.g., UML2, SysML, Spirit, AADL, Lightweight-CCM, EAST-ADL2, Autosar, …
- Main features
  - Mainly refined UML structured classes, on top of which a support for SysML blocks has been added
  - But also compatible with a support for Lightweight-CCM, AADL and EAST-ADL2, Spirit and Autosar
  - Shortcuts for UML2 modeling of components/composites diagrams

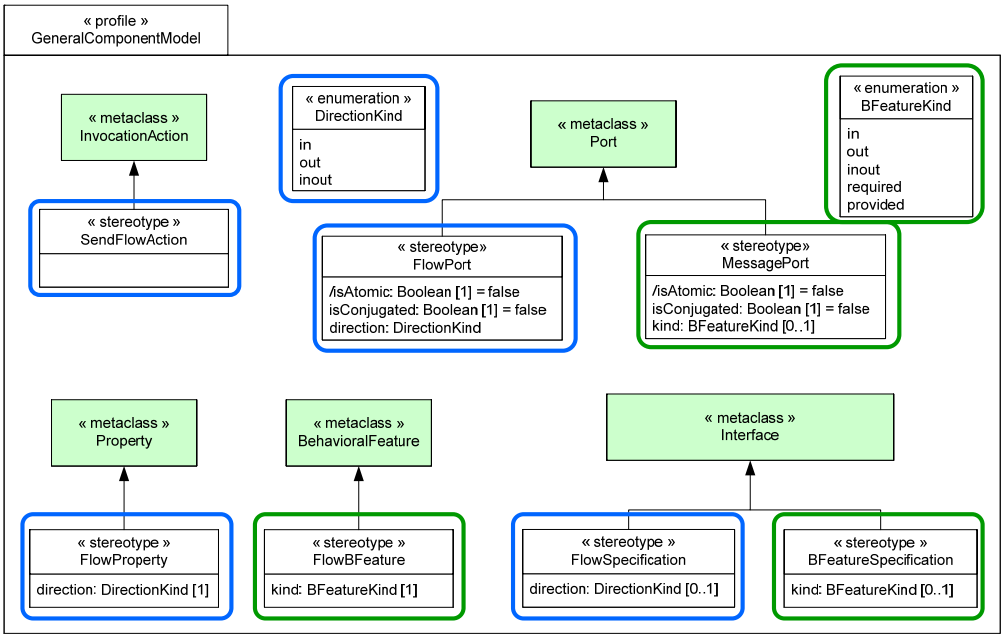


# Example of Normal UML2 Structured Class




Normal UML2 port typed by a class realizing and using interfaces: the port rp provides PRegInterface and required RRegulatorInterface

# The MARTE GCM Sub-profile



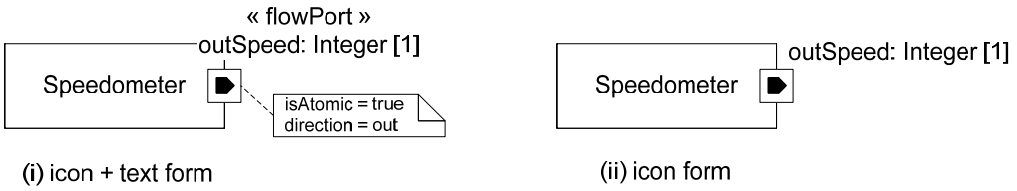

: concepts for data-based flow communication

: concepts for message-based flow communication



# Examples of FlowPort usage

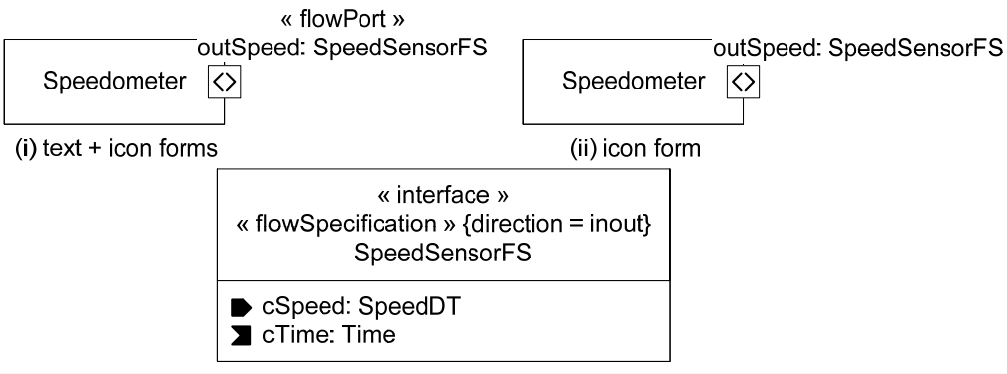
- Atomic flow port example




(i) icon + text form

(ii) icon form

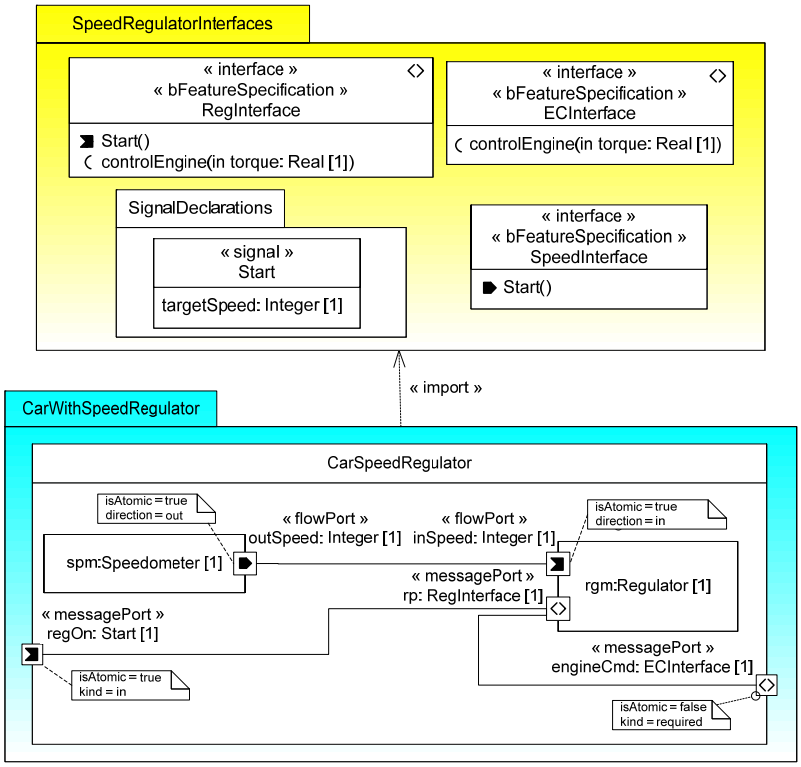
- Non-atomic flow port example




(i) text + icon forms

(ii) icon form

# Example with « MessagePort »






# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

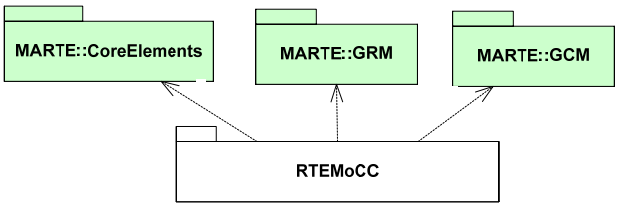
*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE

# RTE Model of Computation & Communication

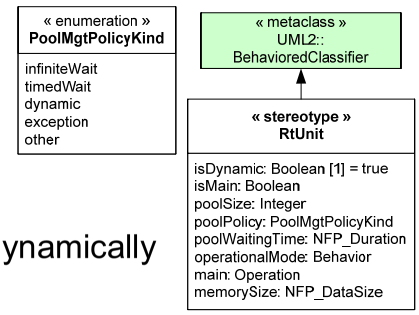
- High-level modeling concepts where RT/E concerns are embedded inside modeling artifacts (E.g., UML active/passive objects)
  - Implicit semantics
- Two families of concerns
  - Quantitative aspects
    - E.g. concurrency and behavior
  - Qualitative aspects as real-time feature
    - E.g. deadline or period
- Package dependencies



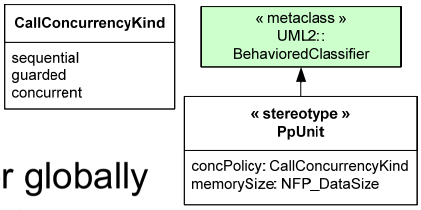



# Qualitative RT features Modeling

- « RtUnit »
  - Generalization of UML2 Active Objects
    - Autonomous processing resource, able to handle different messages at the same time
  - Owns at last one schedulable resource
    - Managed either statically (via a SR pool) or dynamically
  - May have operational mode description
  - May be main objects
    - Refer to the main operation

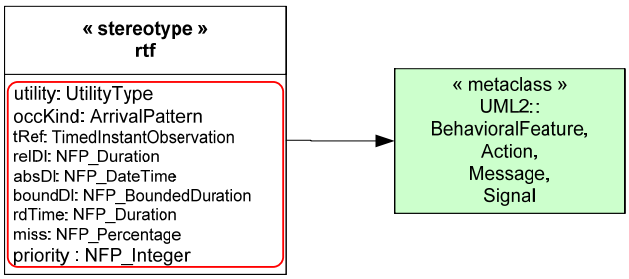



- « PpUnit »
  - Generalization of UML2 Passive Objects
  - Concurrency policy specified either locally or globally
  - Processing is either immediateRemote or deferred

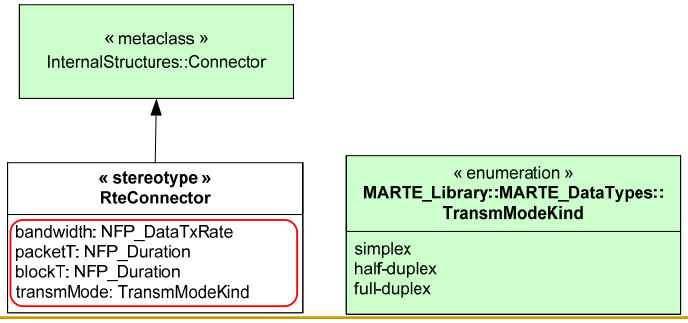


# Quantitative RT features modeling

- « RealTimeFeature »



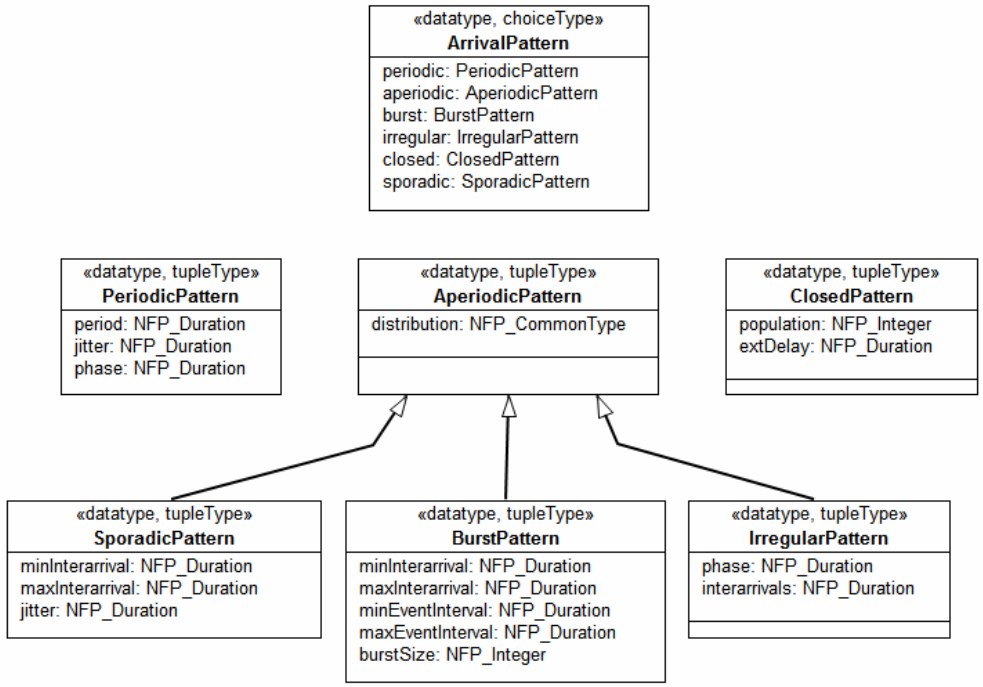

- « RteConnector »





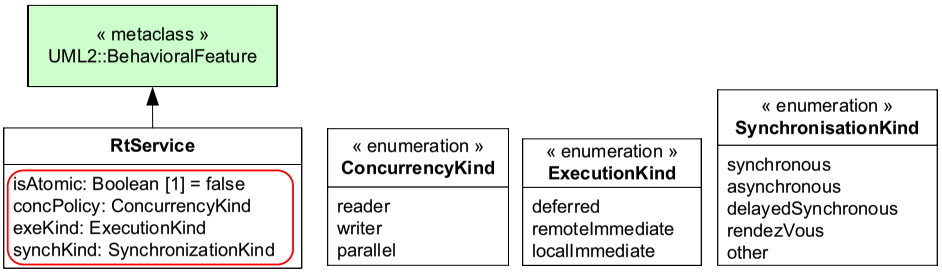

# The Arrival Pattern data type

- Defined in the MARTE_Library

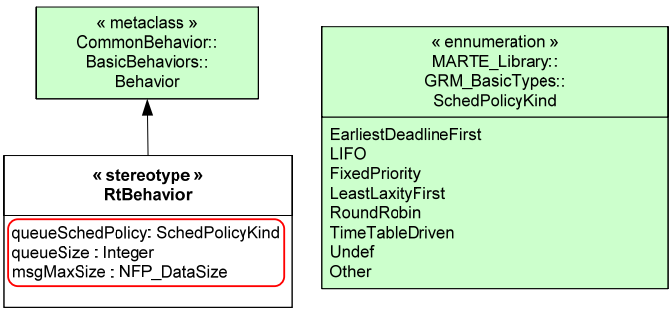


# Dynamics aspects of « RtUnit » and « PpUnit »

- Services specification

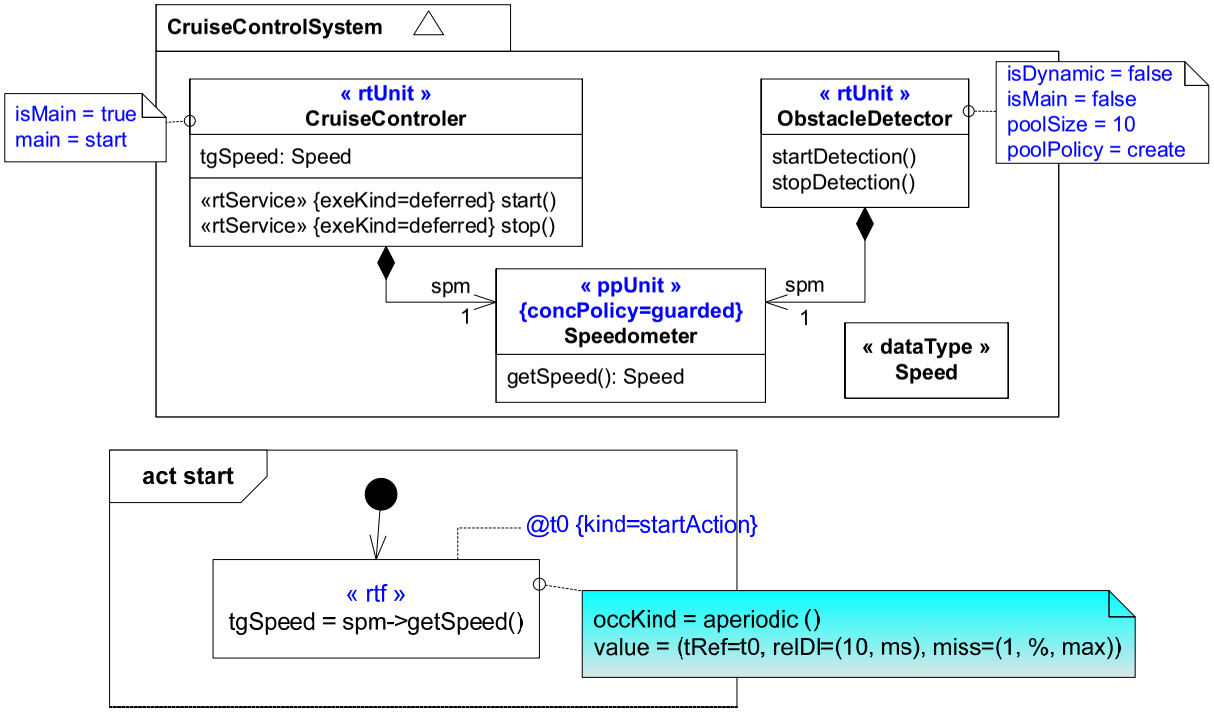



- Classifier behavior specification

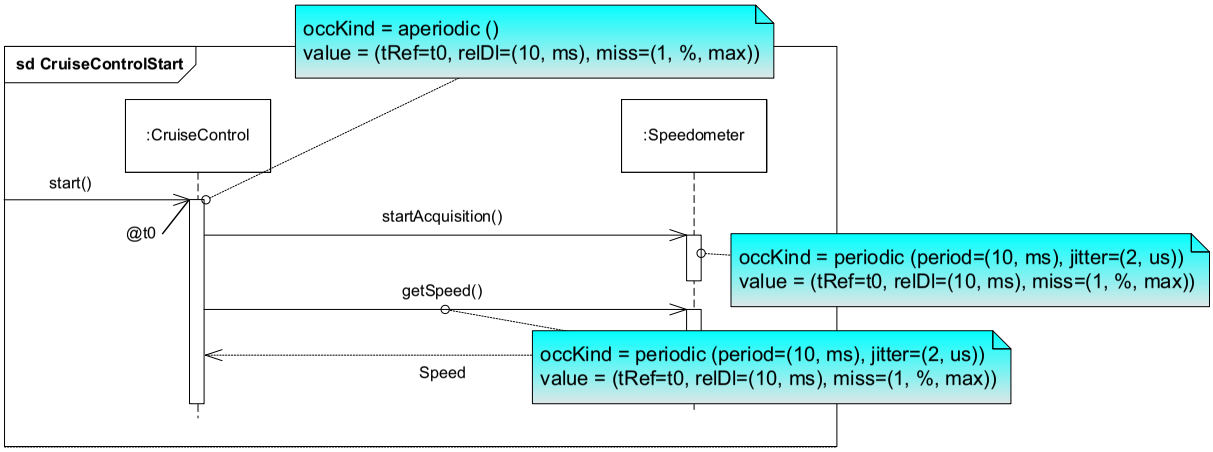





# Usage examples of the RTEMoCC extensions (1)

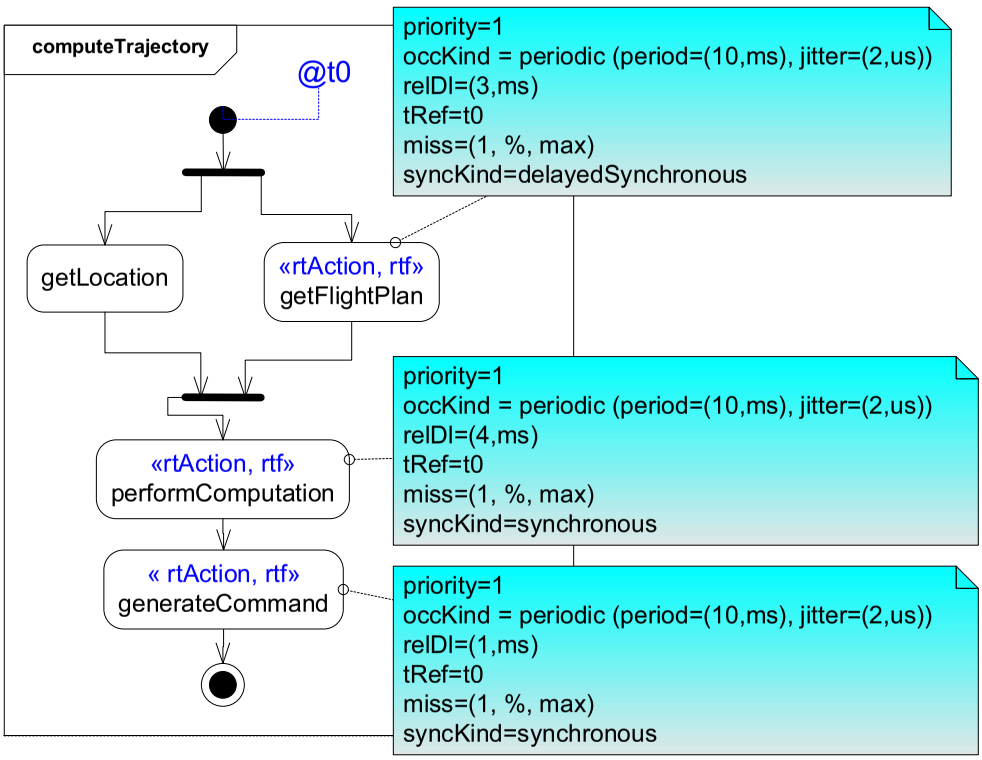


# Usage examples of the RTEMoCC extensions (2)






# Usage examples of the RTEMoCC extensions (3)



# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

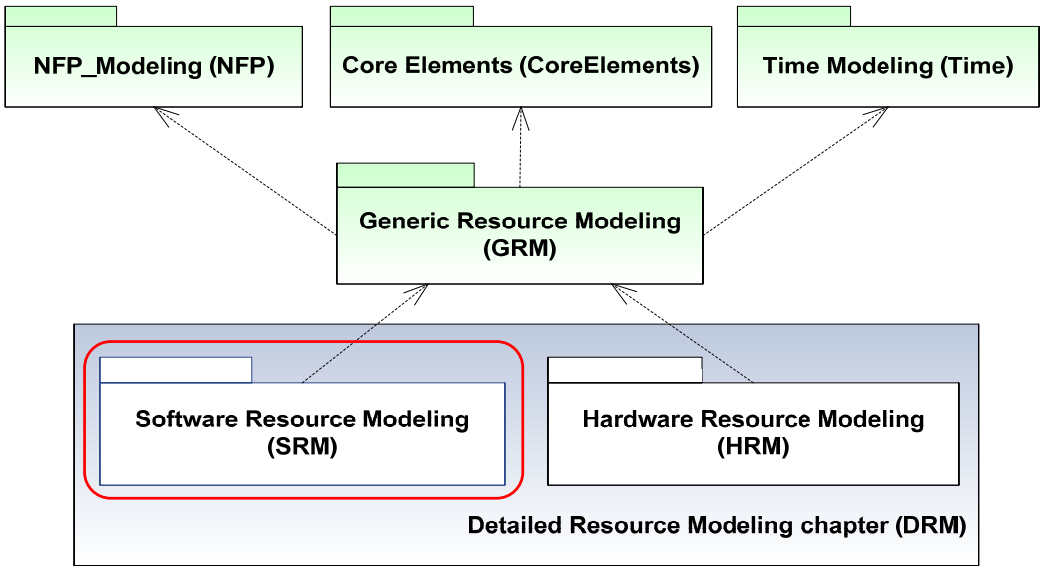
*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE



# Detailled Resource Modeling within MARTE

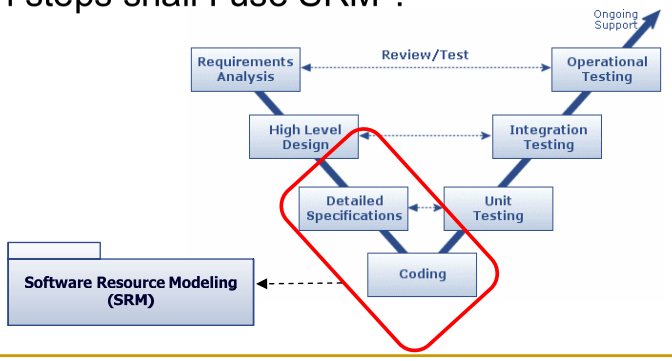


## What is the Software Resource Modeling Profile (SRM) ?

- 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 = a unified mean to describe such existing or proprietary APIs**

- In which steps shall I use SRM ?

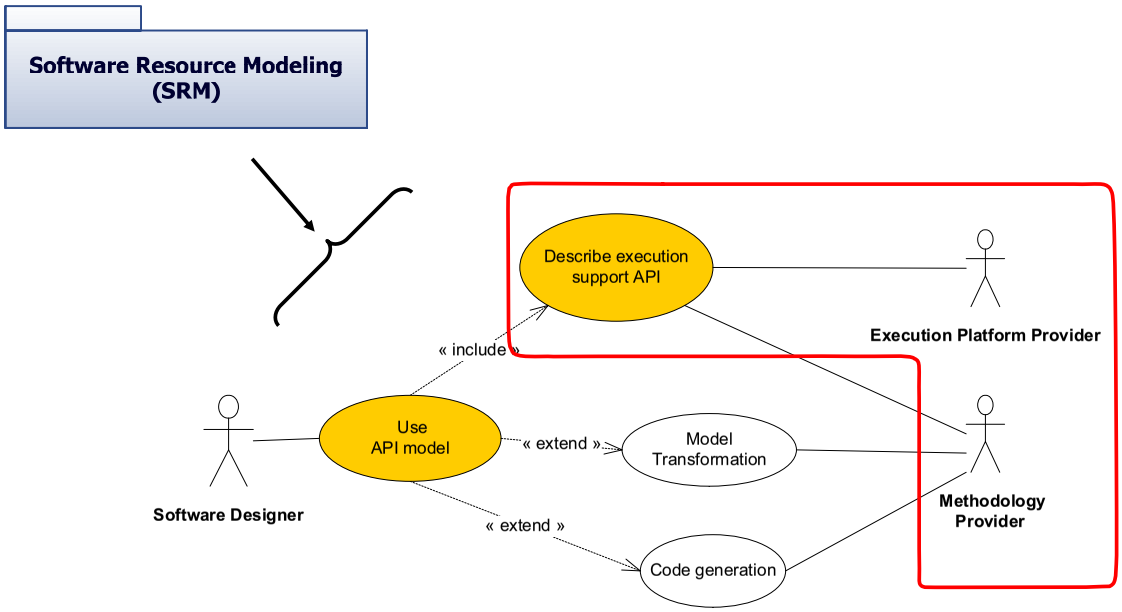





## Items shown in next slides

- SRM overview
- Details of SRM through the OSEK/VDX case study
- Interests to use SRM through examples
  - Model of multitask design
    - E.g., a robotic case study
  - Generation of OS configuration file
    - E.g., OSEK OIL configuration file generation
  - Porting existing RTE application models
    - E.g., from OSEK to ARINC

# Main expected use cases of 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 relies 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

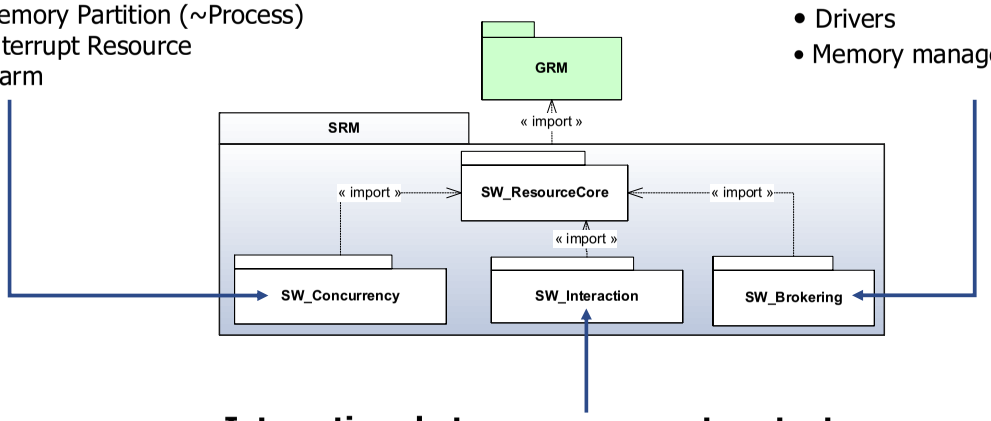
# What is supported by the SRM profile ?

**Concurrent execution contexts**:

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

**Hardware and software resources brokering:**

- Drivers
- Memory management



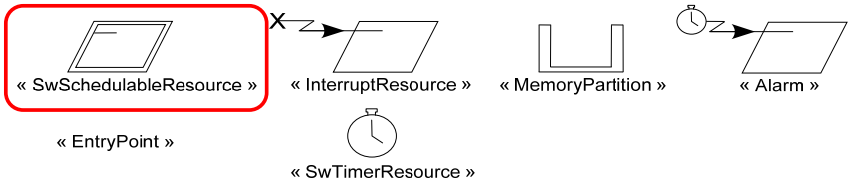

**Interactions between concurrent contexts**:

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

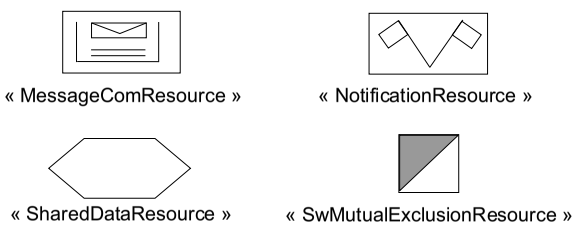


# Snapshot of the UML extensions provided by SRM

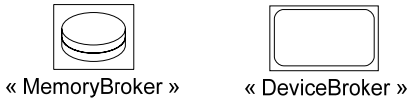
**SRM::SW_Concurrency**




**SRM::SW_Interaction**




**SRM::SW_Brokering**



# The OSEK/VDX case study

- OSEK/VDX standard (http://www.osek-vdx.org)
  - Automotive industry a 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 characterisics
  - 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

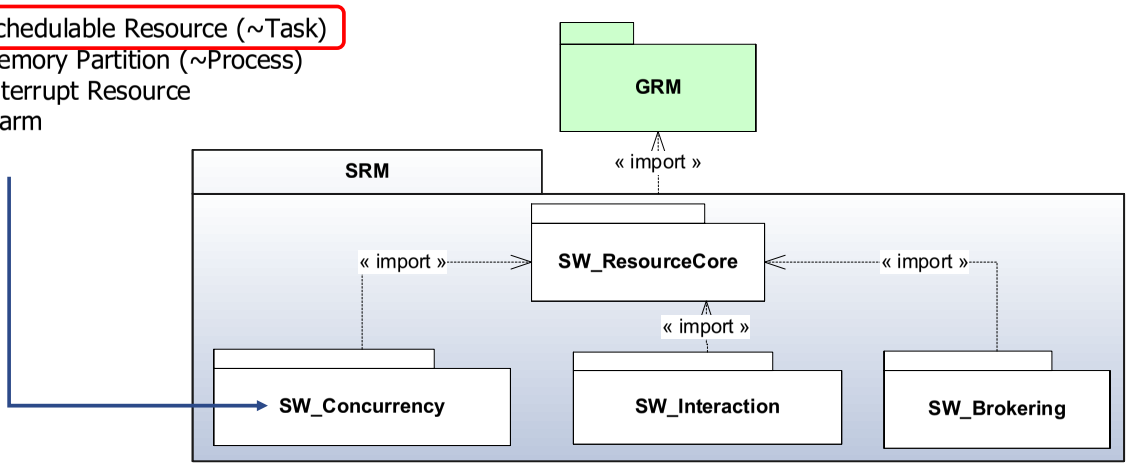
- Semantic
  - An OSEK-VDX task provides the framework for computing application functions. A scheduler will organize the sequence of task executions.
- Example of properties
  - **Priority: UINT32**
    - Priority execution of the task
  - **StackSize: UINT32**
    - Stack size associated to the execution 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



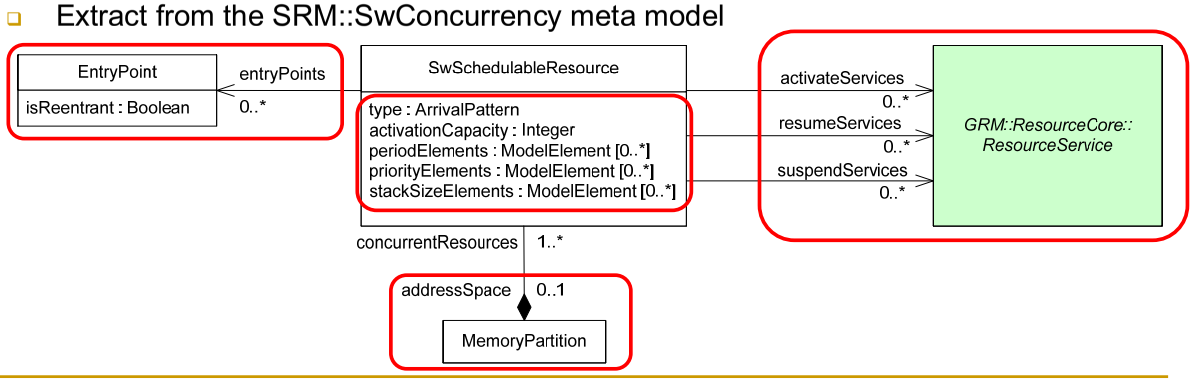
# Details of «SwSchedulableResource»

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

  **==> SRM artifacts for modeling OSEK-VDX Task!**

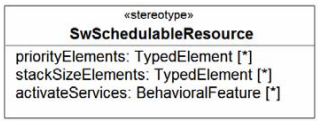
- Main features
  - Owns an entry point referencing the application code to execute
  - May be restricted to execute in a given address space (i.e. a memory partition)
  - Owns properties
    - E.g., Priority, Deadline, Period and StackSize
  - Provides services
    - E.g., activate, resume and suspend

- Extract from the SRM::SwConcurrency meta model

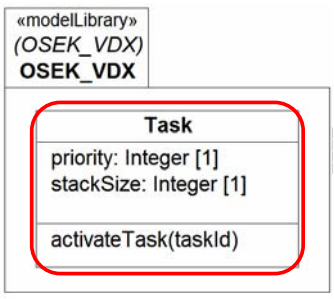




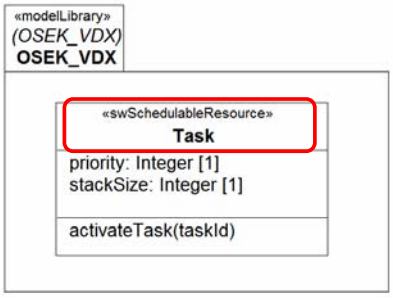

# Model of an OSEK Task with «SwSchedulableResource»



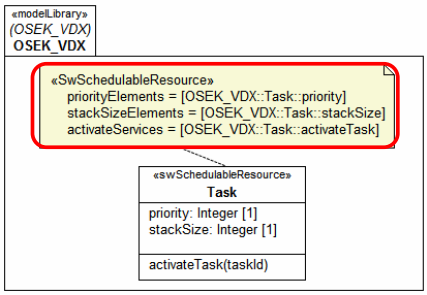

1. Define a UML model for OSEK_VDX::Task
   a. Add model library applying the SRM profile
   b. Add a class and defines its features (properties and operations)
2. Applying the «SwSchedulableResource» stereotype
3. Fullfill the tagged values of the applied stereotype




(Step 1)




(Step 2)




(Step 3)

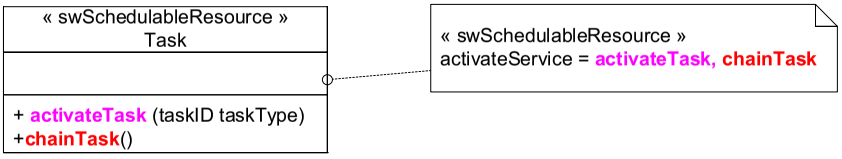
Models have been realized with the Papyrus Eclipse-based open-source tool for UML2:
http://www.papyrusuml.org

# SRM modelling facilities

- How to model multiples candidates for the same semantic ?
  - 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

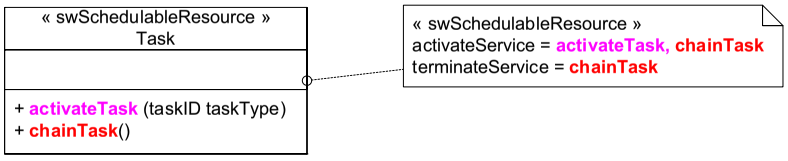

      - Both *activateTask* and *chainTask* operations are task activating services



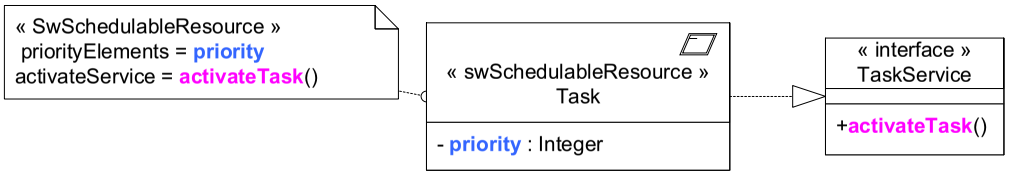



# SRM modeling facilities (seq.)

- How to model a feature which have multiple semantic ?
  - 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



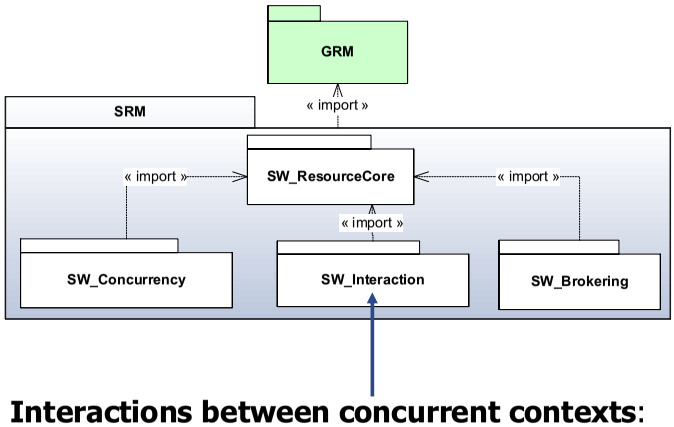

- SRM allows multiple usages
  - User can use constraints, such as OCL rules, to limit those possibilities

# Focus on the OSEK/VDX Event definition

- Semantic :
  - The event mechanism is a means of synchronisation 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?



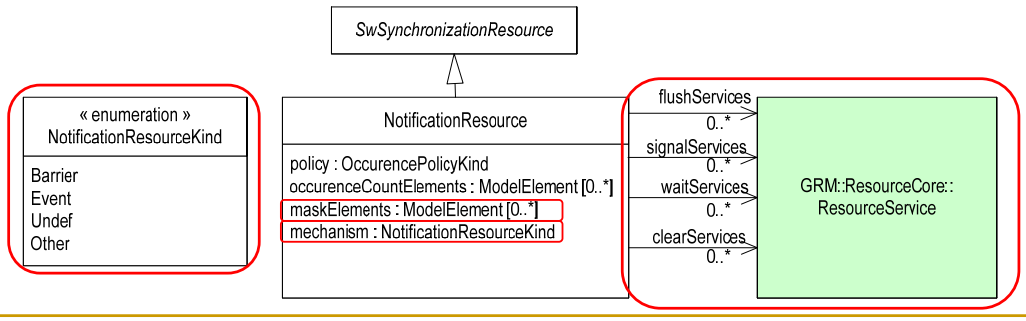

**Interactions between concurrent contexts**:

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

# Details of «NotificationResource»

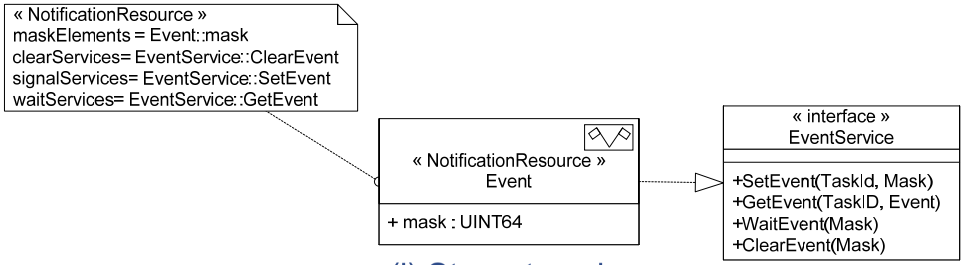
- Semantic
  - *NotificationResource* supports control flow by notifying the occurrences of conditions to awaiting concurrent resources

  ==> 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

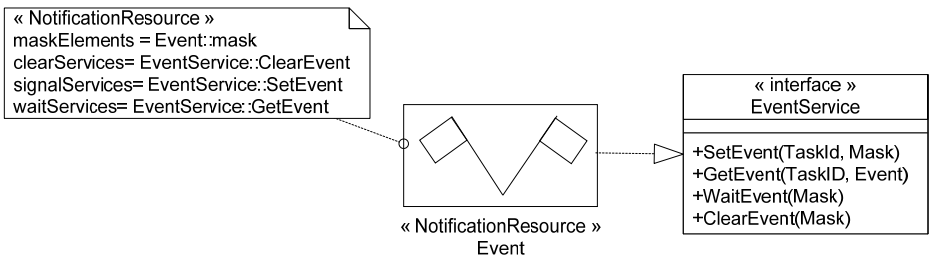





# OSEK/VDX Event as a NotificationResource

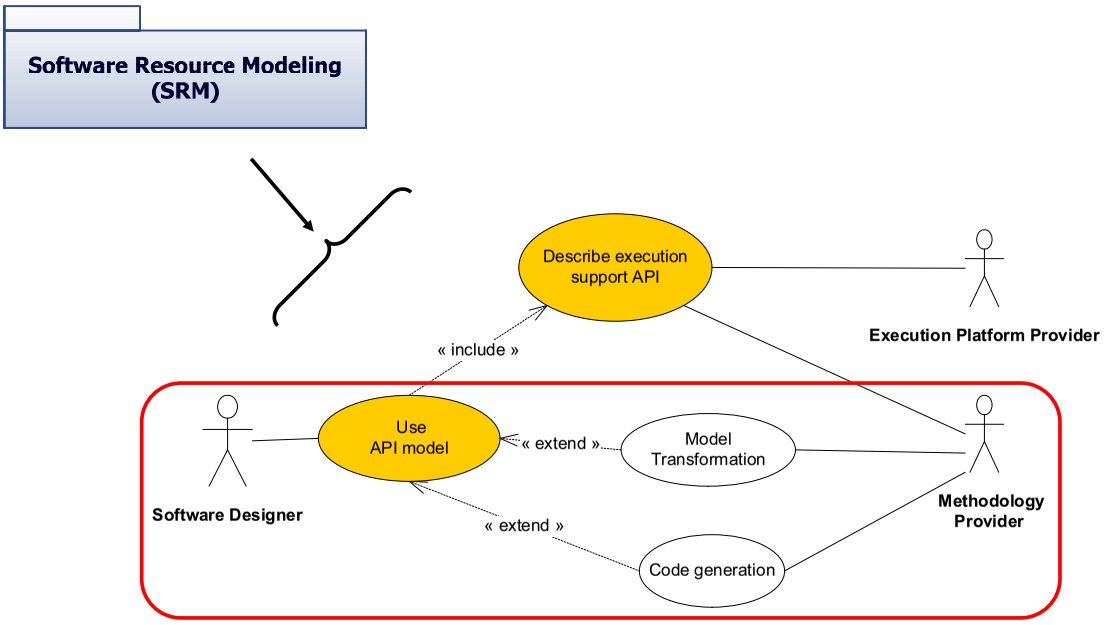



(i) Stereotype icon




(ii) Stereotype shape

# In which typical cases shall I use SRM ?



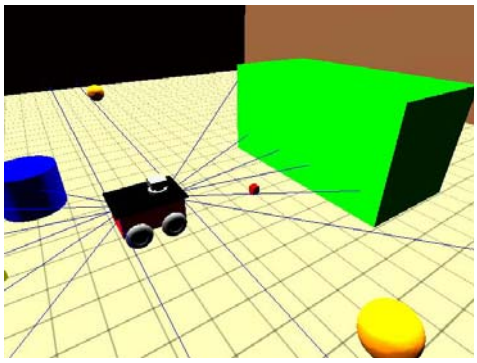



# Use examples of one RTOS modeled with SRM

- 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

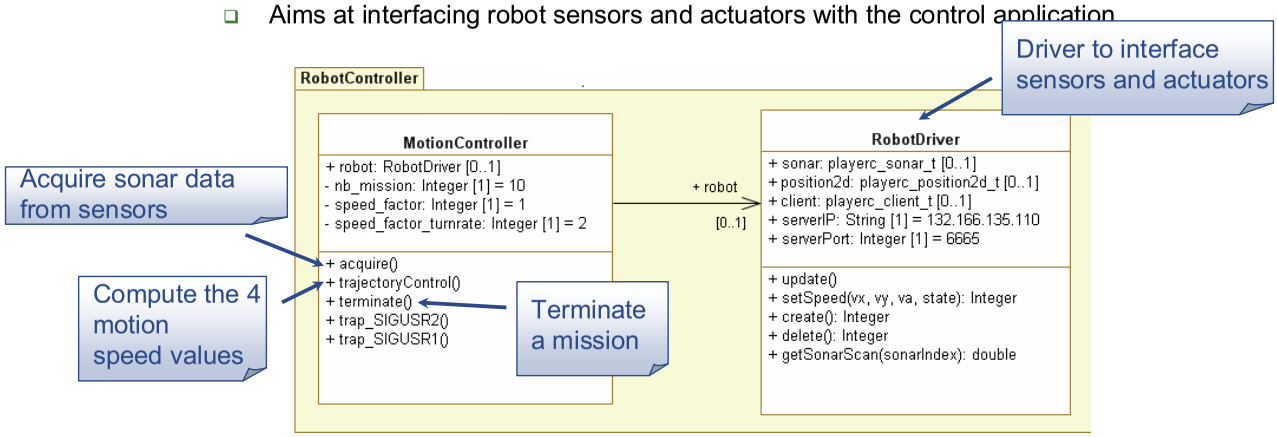


# Purpose and context of the example 1

- 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



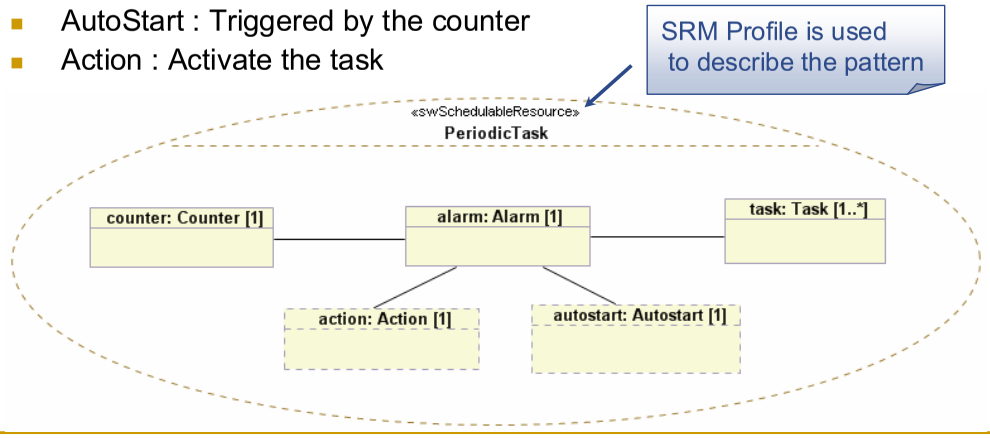



# Principles of the applied multitask design

- 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

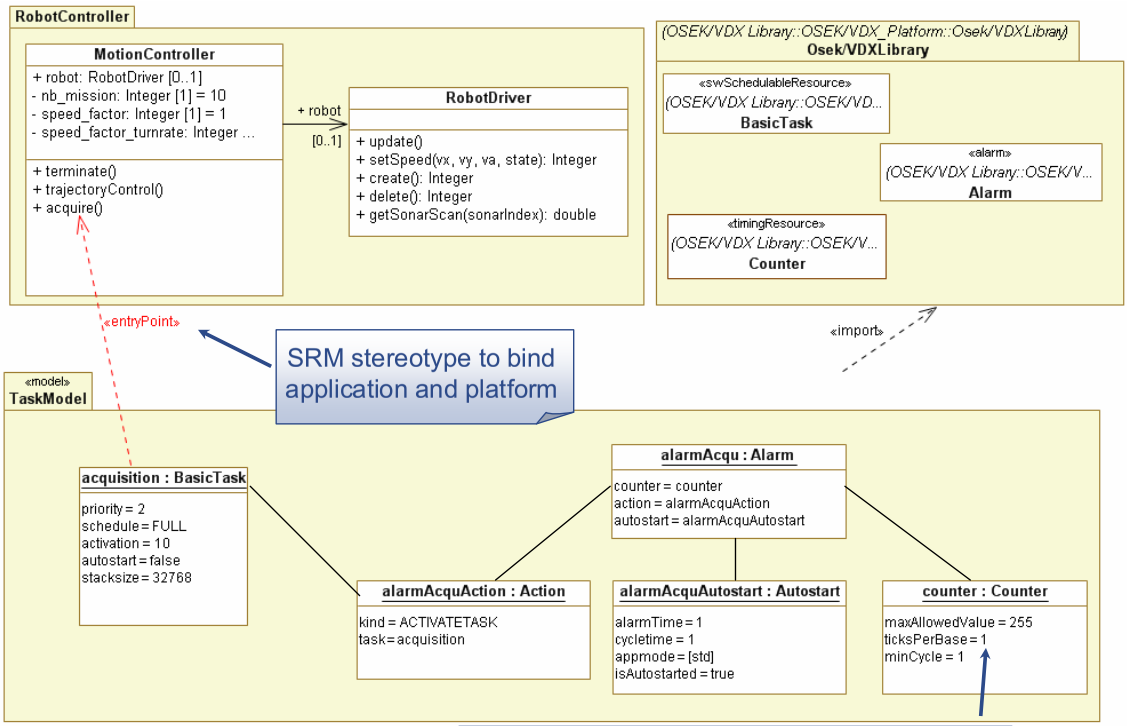
# Periodic task in OSEK/VDX

- 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



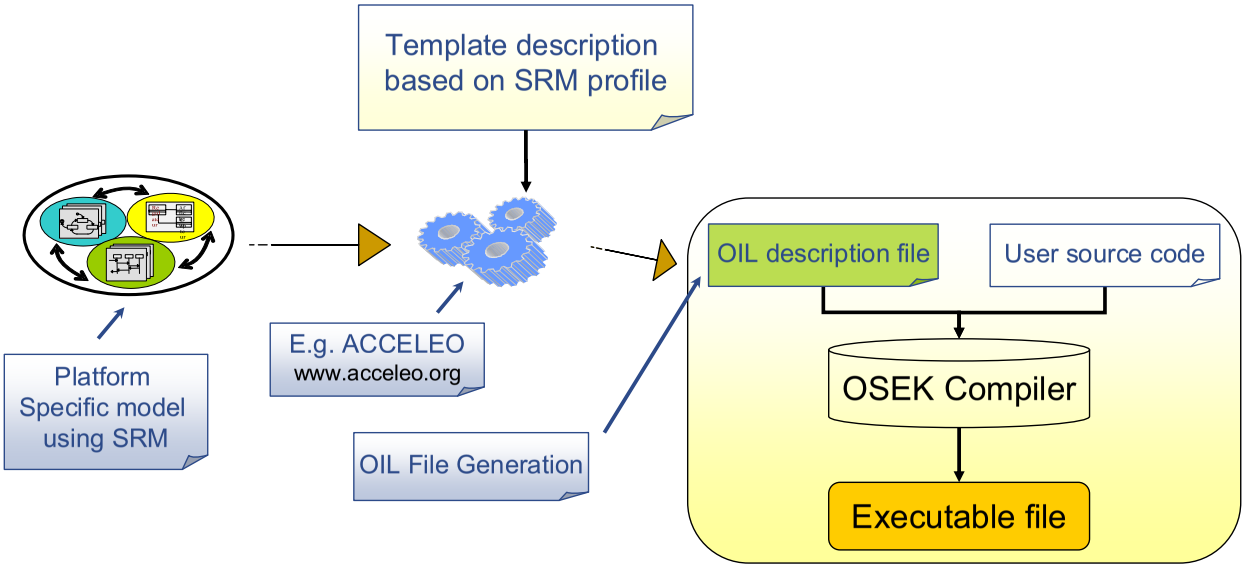
# Example 2: OSEK Configuration File generation

- Purpose
  - Generation of the OSEK OIL configuration files from the multi-task design of the robot controller
- OIL: **O**SEK **I**mplementation **L**anguage
  - 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

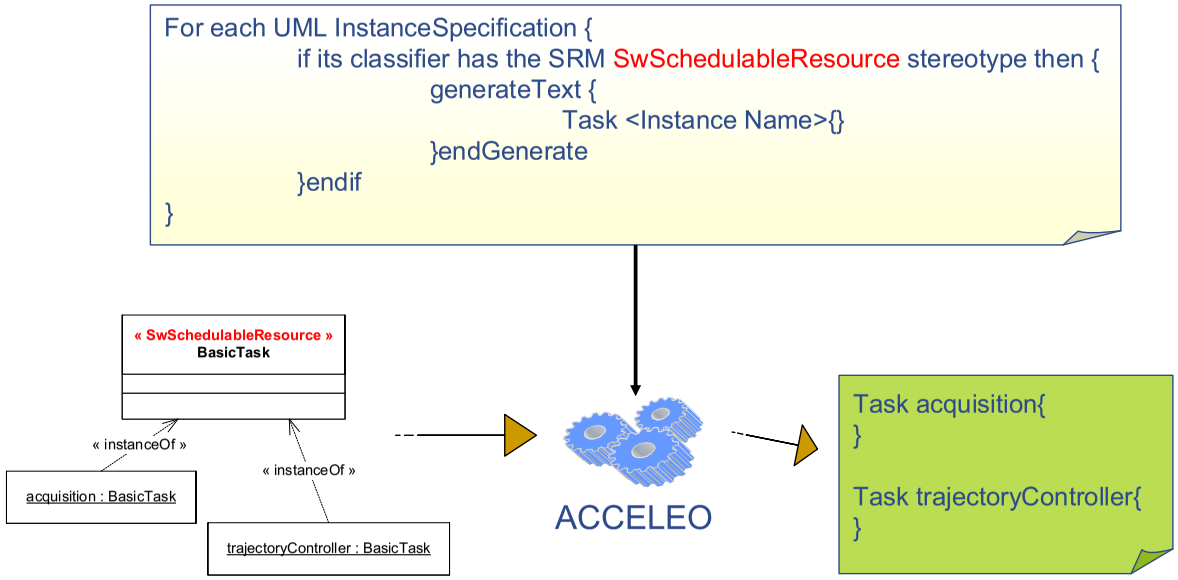
```
OIL_VERSION = "2.5" : "RobotController" ;

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 ;
  };
...
```



# Principle to go from a UML model to an OIL file

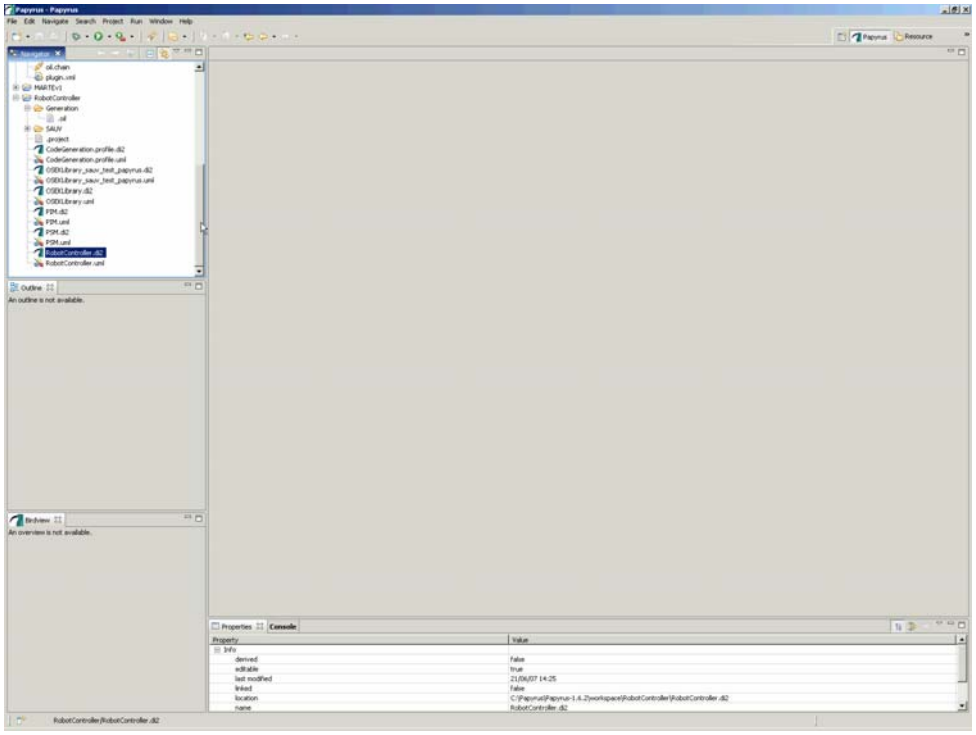


# Example of an OIL generation template



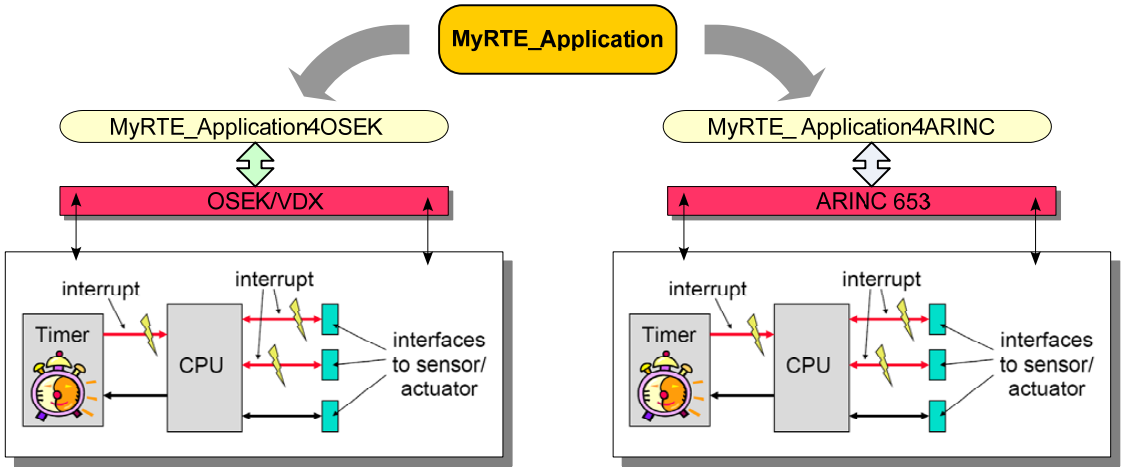



# Generation of the OIL file in the Papyrus UML Tool



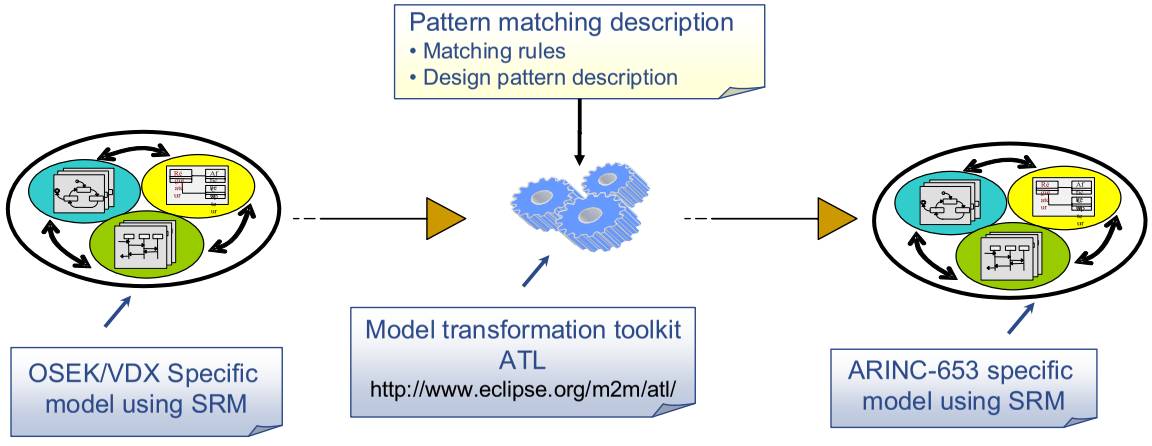
# Example 3: Assist user to port multitask designs

- 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.



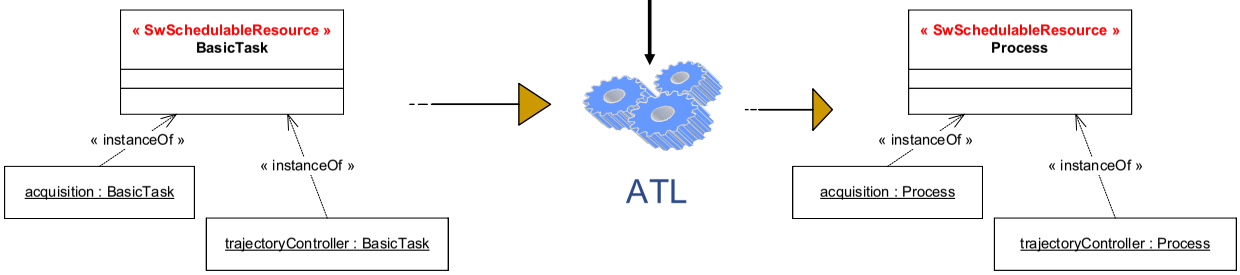



# Principles of the model transformation



# Matching pattern example (1/2)

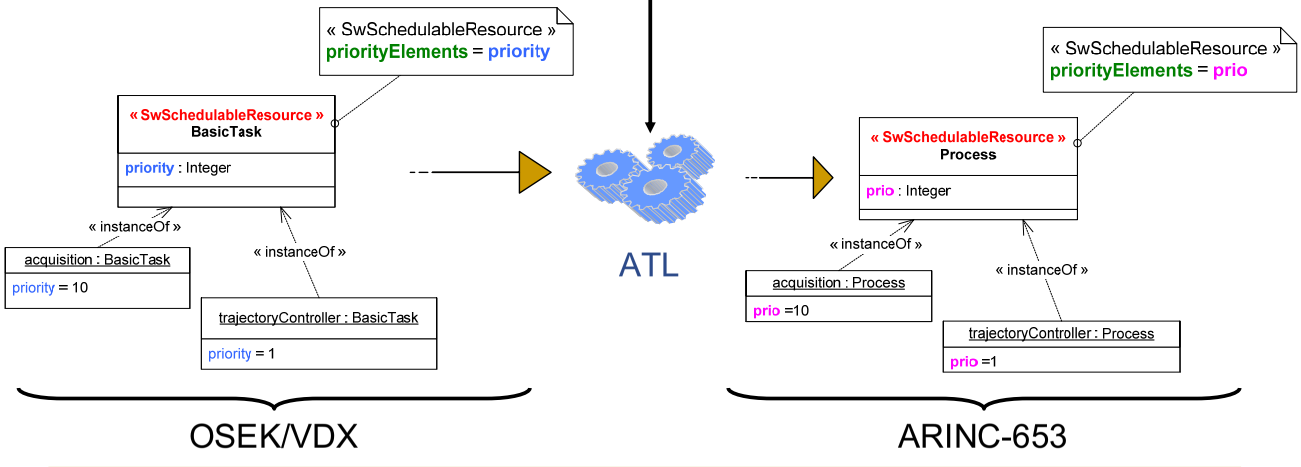
For each UML InstanceSpecification {
if its classifier has the SRM SwSchedulableResource stereotype then {
- generate a new Instance Specification;
- its target classifier is that which is stereotyped
**SwSchedulableResource in the target execution support;**
}endGenerate
}endif
}



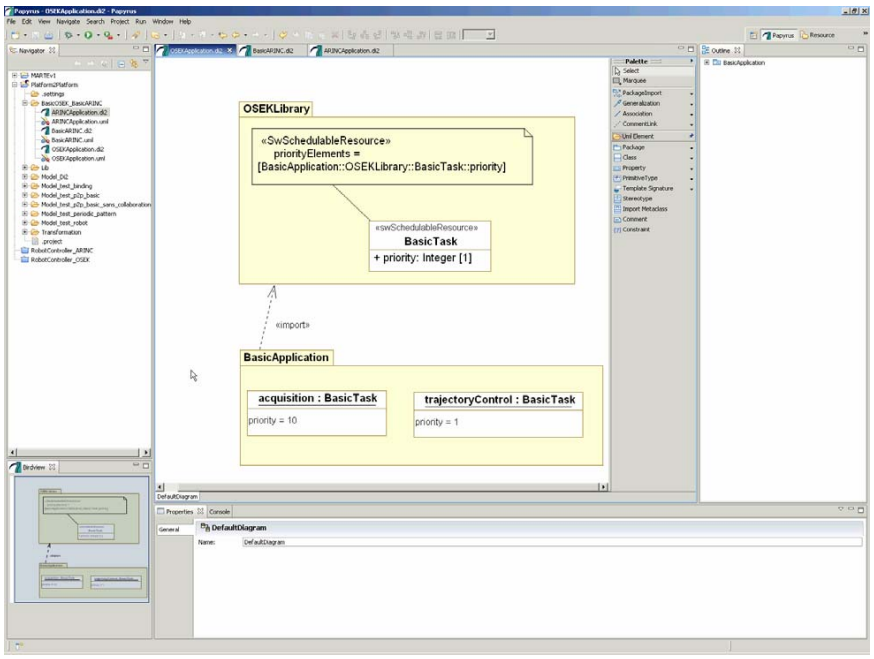



# Matching pattern example (2/2)

For each UML InstanceSpecification {
if its classifier has the SRM SwSchedulableResource stereotype then {
- …
- each source priorityElements match one target priorityElements
}endGenerate
}endif
}



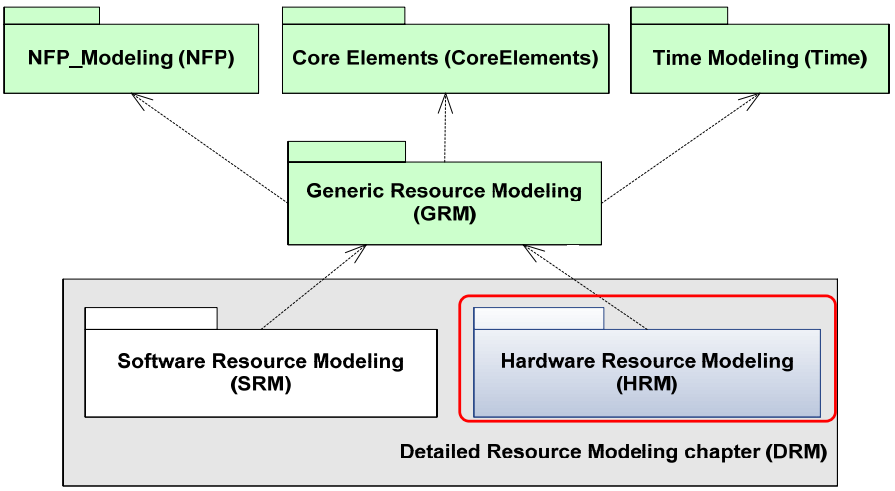
# Assist user to port multi-task designs in the Papyrus UML tool : a basic example



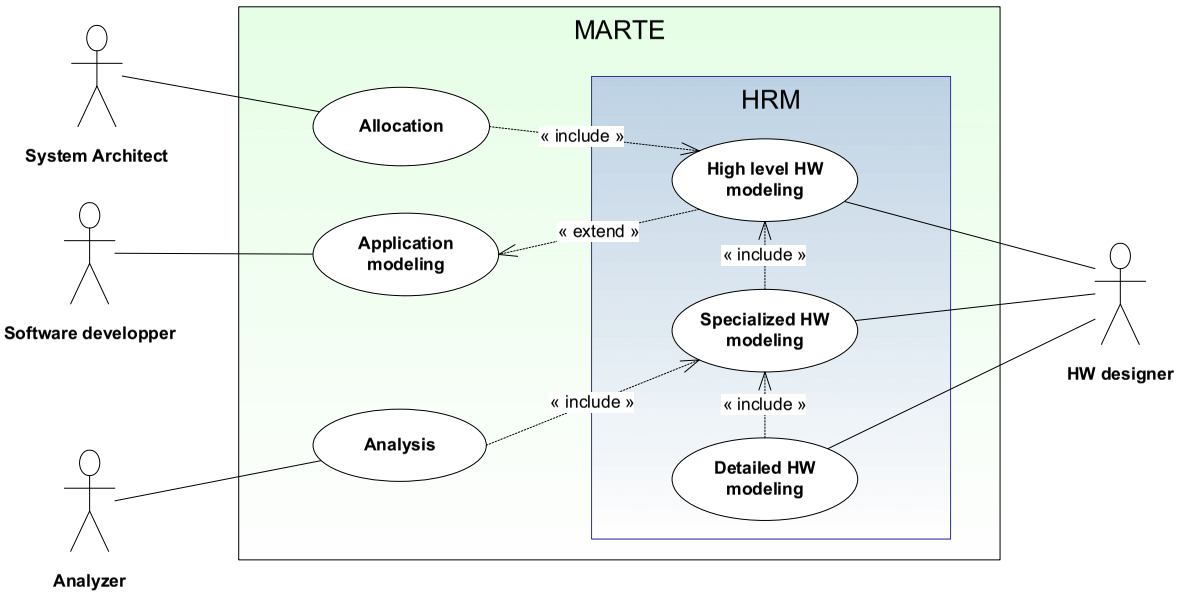



# Hardware Resource Modeling with MARTE

- Specialization of GRM
  - HRM and SRM share a common structure
    - It eases HW/SW allocation!



# 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

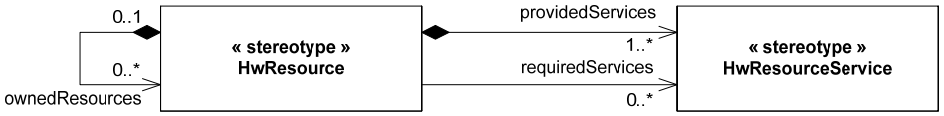


## HRM use cases -- Detailed hardware modeling

- 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)*



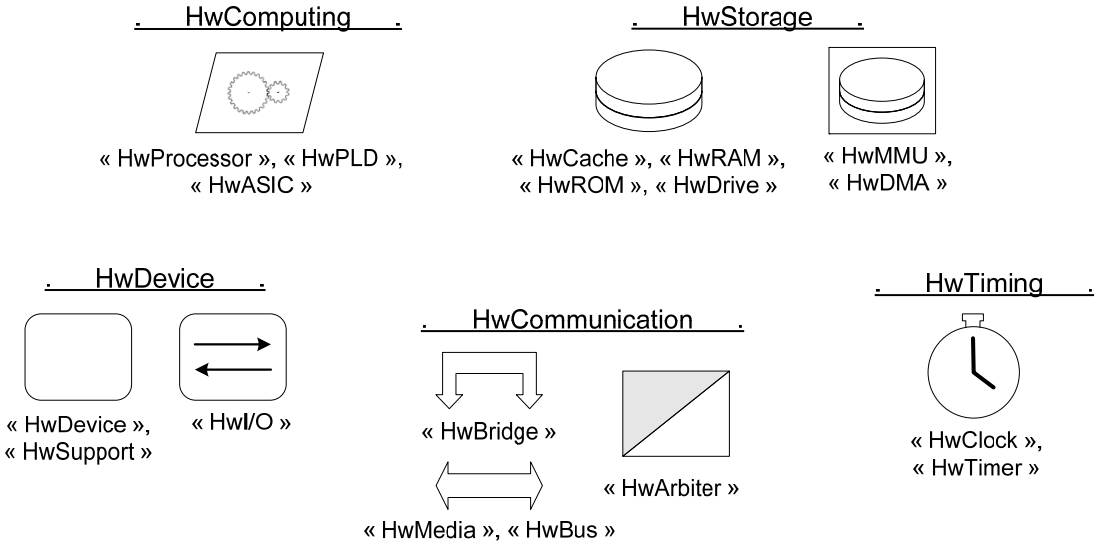

- **Two** modeling **views** to separate concerns

**Logical / Physical**



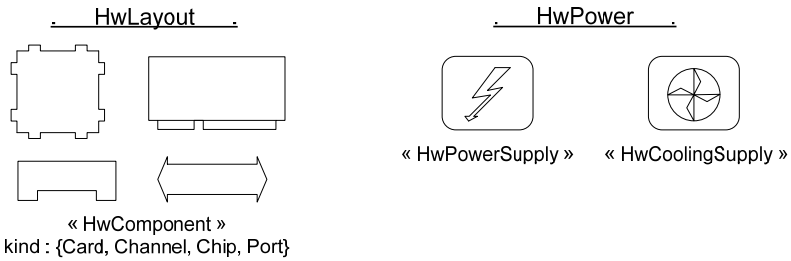
# HRM structure -- Logical modeling

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



# HRM structure -- Physical modeling

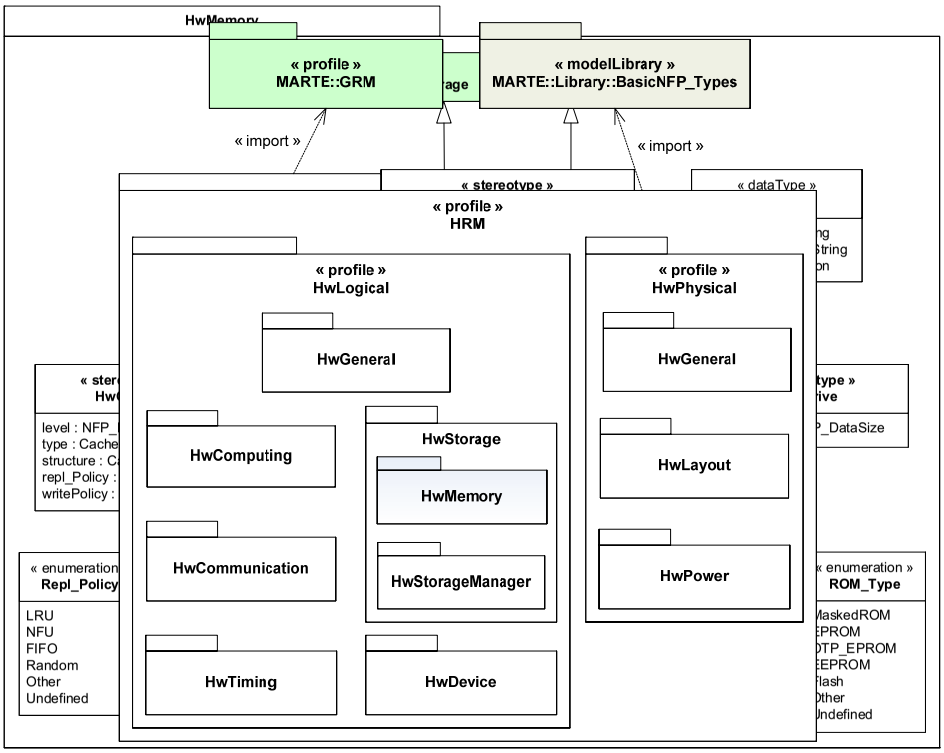
- Provides a **physical** properties description
- Based on both following packages
  - HwLayout
    - Forms: Chip, Card, Channel…
    - Dimensions, area and arrangement mechanism within rectilinear grids
    - Environmental conditions: e.g. temperature, vibration, humidity…
  - HwPower
    - Power consumption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





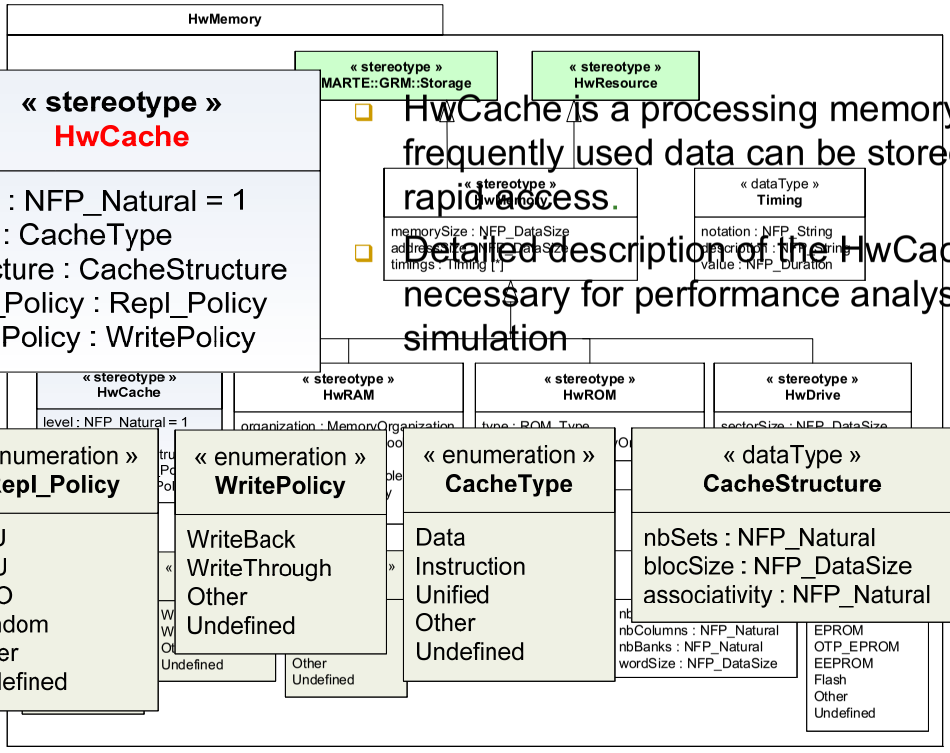

# HRM profile overview

# HRM profile -- HwMemory






# HRM profile -- HwMemory -- 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

# HRM profile -- HwMemory -- HwCache

| « stereotype » HwCache |
|---|
| level : NFP_Natural = 1<br>type : CacheType<br>structure : CacheStructure<br>repl_Policy : Repl_Policy<br>writePolicy : WritePolicy |

- Specifies the cache level.
- Default value is 1

| « enumeration » Repl_Policy |
|---|
| LRU<br>NFU<br>FIFO<br>Random<br>Other<br>Undefined |

| « enumeration » WritePolicy |
|---|
| WriteBack<br>WriteThrough<br>Other<br>Undefined |

| « enumeration » CacheType |
|---|
| Data<br>Instruction<br>Unified<br>Other<br>Undefined |

| « dataType » CacheStructure |
|---|
| nbSets : NFP_Natural<br>blocSize : NFP_DataSize<br>associativity : NFP_Natural |



# HRM profile -- HwMemory -- HwCache

| « stereotype » HwCache |
|---|
| level : NFP_Natural = 1<br>type : CacheType<br>structure : CacheStructure<br>repl_Policy : Repl_Policy<br>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*

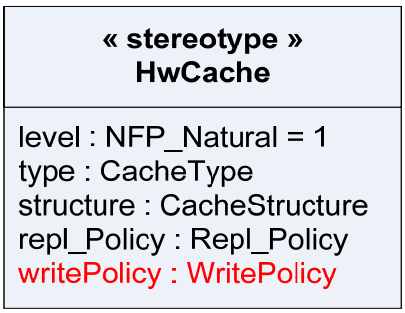
| « enumeration » Repl_Policy |
|---|
| LRU<br>NFU<br>FIFO<br>Random<br>Other<br>Undefined |

| « enumeration » WritePolicy |
|---|
| WriteBack<br>WriteThrough<br>Other<br>Undefined |

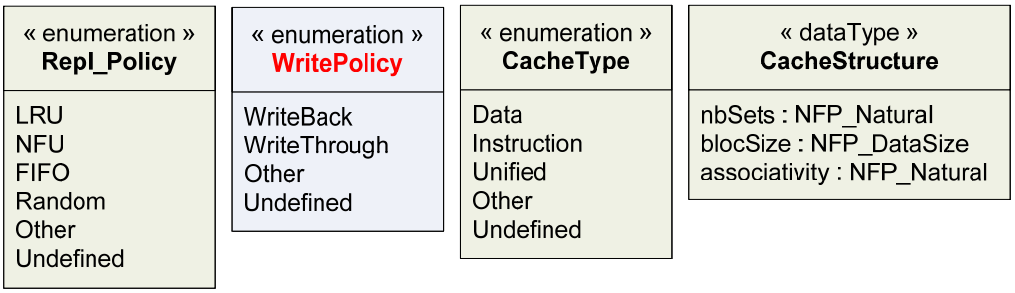
| « enumeration » CacheType |
|---|
| Data<br>Instruction<br>Unified<br>Other<br>Undefined |

| « dataType » CacheStructure |
|---|
| nbSets : NFP_Natural<br>blocSize : NFP_DataSize<br>associativity : NFP_Natural |

# HRM profile -- HwMemory -- HwCache




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



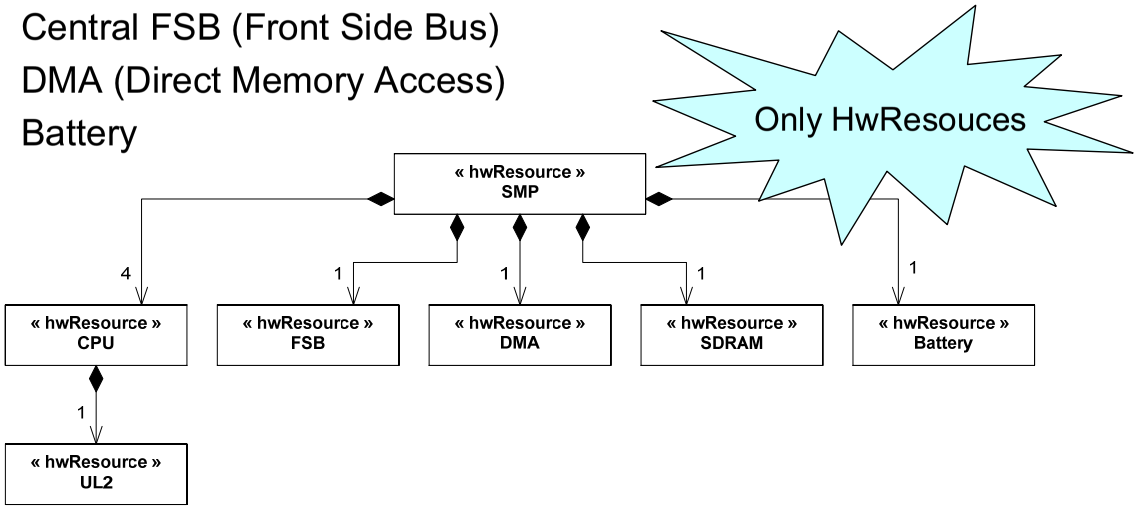



# HRM usage

- 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
  - Tag definitions 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)*

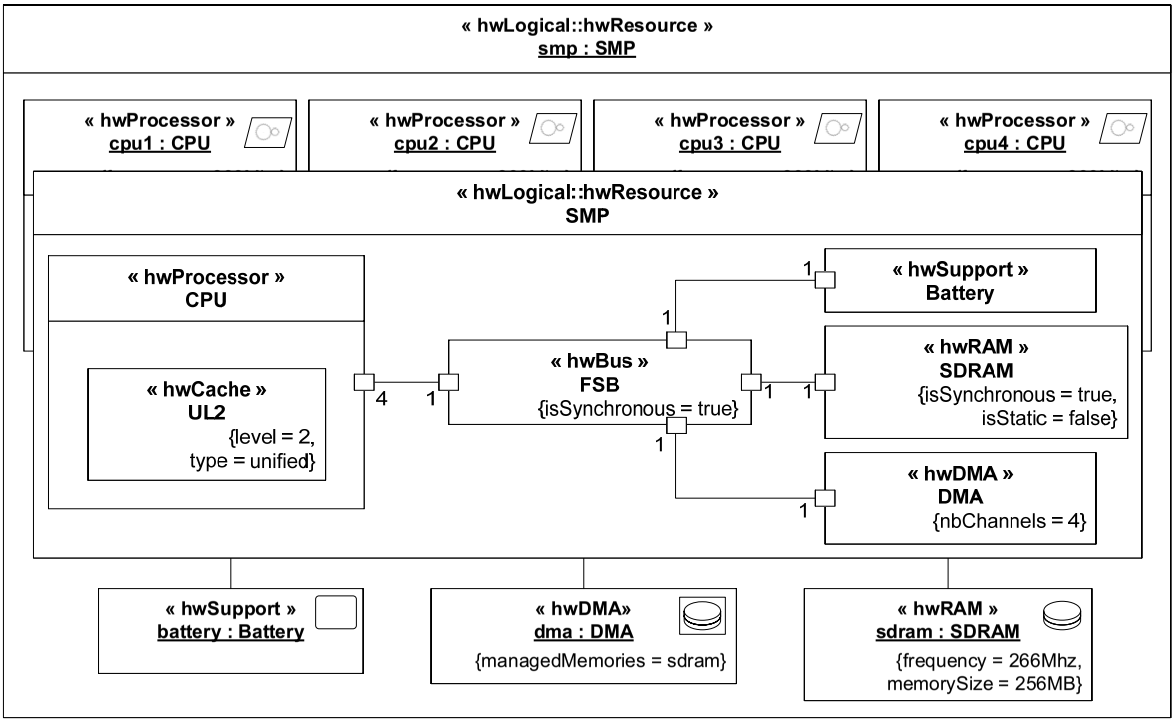
# Very early Hw Architecture Description

- 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

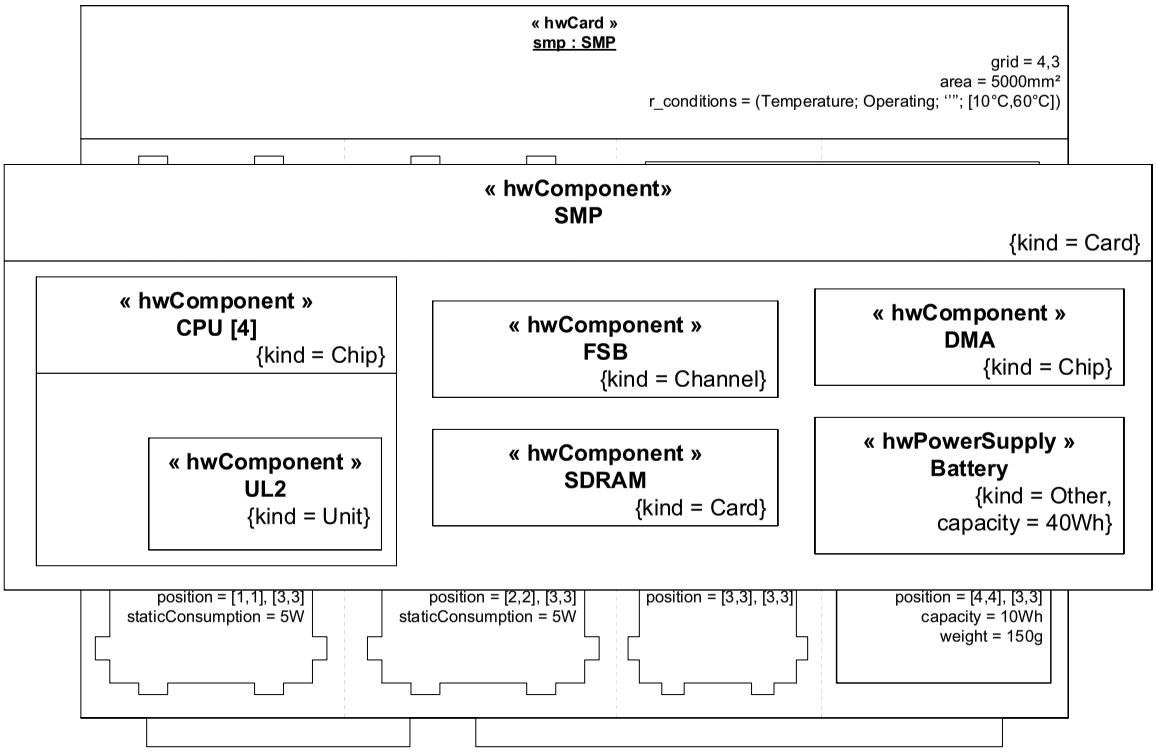





# HRM usage example: Logical view



# HRM usage example: Physical view






# HRM case study -- TC1796 (µController)

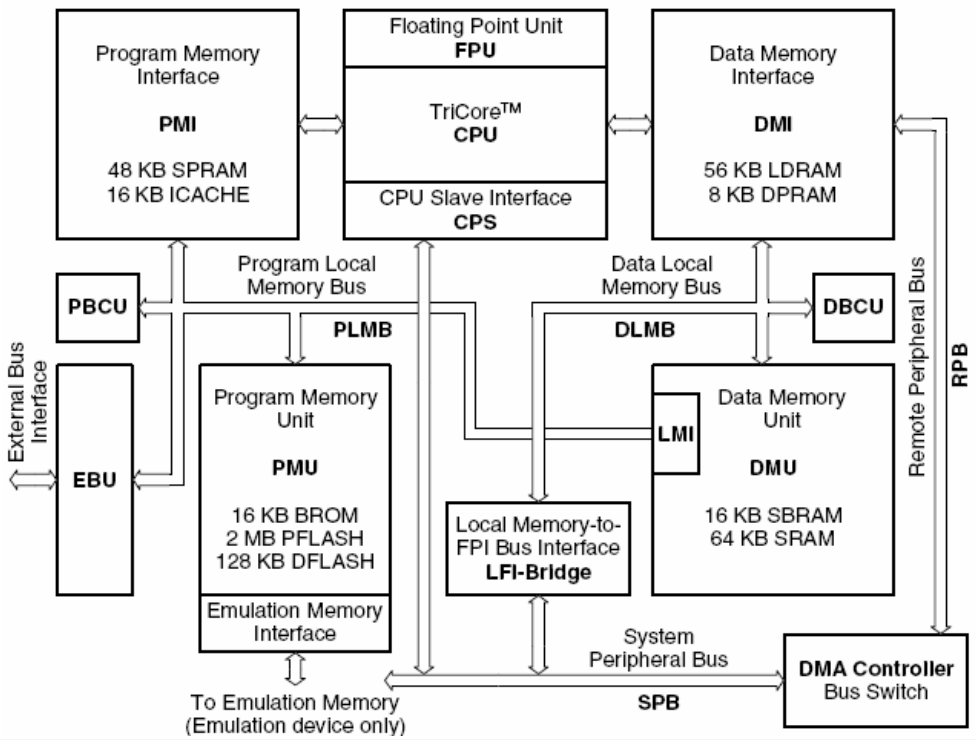
- Advanced 32-bit TriCore™-based Next Generation Microcontroller for Real-Time Embedded systems
  - Automotive control systems
  - Industrial robotic control
- Features
  - Super-scalar TriCore CPU
    - Superior real-time performance
      - Efficient interrupt handling
    - 4 stage pipeline
    -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

**DEMO**

(Papyrus)



# 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:
  1. **Describe** the HW models in UML using HRM
  2. **Parse** and Capture all the required HW properties
  3. **Verify** coherency and completion
  4. **Generate** the configuration file for the target emulation tool
  5. **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

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE

# Design rationale for GQAM

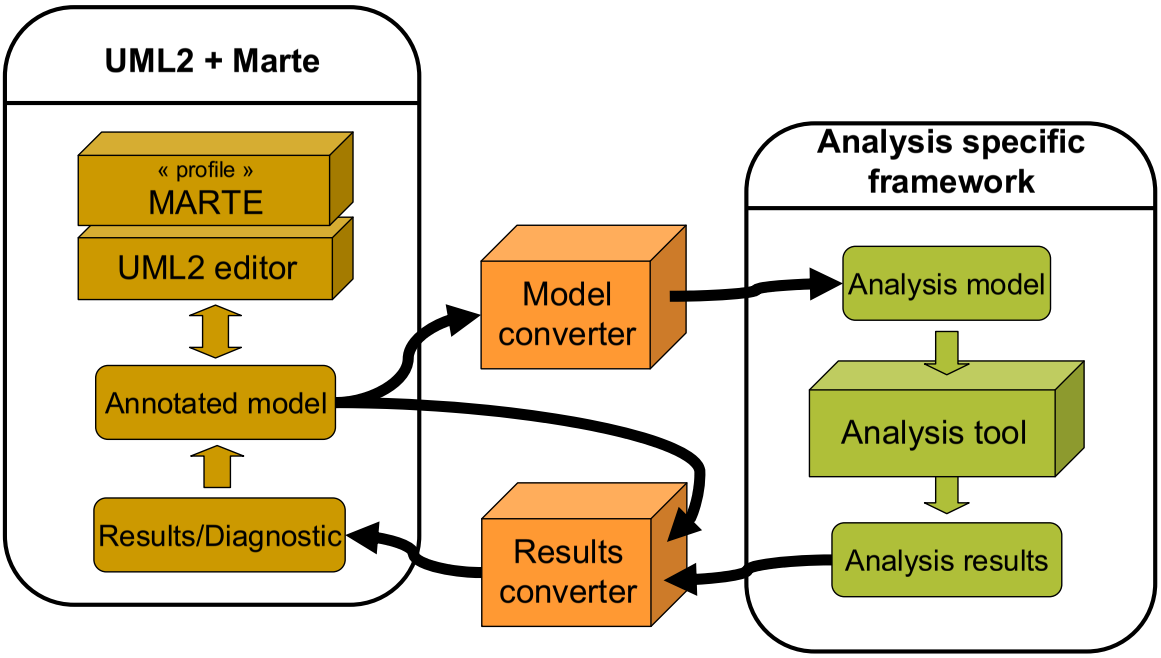
- Updates SPT
  - Alignment to UML2
  - Harmonization between the two SPT analysis sub-profiles: schedulability and performance
  - Extension of timing annotations expressiveness
    - Overheads (e.g. messages passing)
    - Response times (e.g. BCET & ACET)
    - Timing requirements (e.g. miss ratios and max. jitters)
- Provides support for methodological aspects
  - Sensitivity analysis and design-space exploration
  - Modeling reuse and component-based design
  - Support of the MDA approach



# GQAM overview

- Generic Quantitative Analysis Modeling (GQAM) expresses foundation concepts and NFPs shared by different quantitative analysis domains – e.g., schedulability and performance - that have their own terminology, concepts and semantics
  - core GQAM concepts describe how the system behavior uses resources over time
  - NFPs used in quantitative analysis techniques:
    - "input NFPs" - taken as input data
      - e.g., request or trigger arrival rates, execution demands, deadlines, QoS targets
    - "output NFPs" - computed as results
      - e.g., response times, deadline failures, resource utilizations, queue sizes
- Different analysis goals:
  - 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 of some output results to some input parameters
  - Scalability analysis: how the system performs when the problem size or the system size grow.

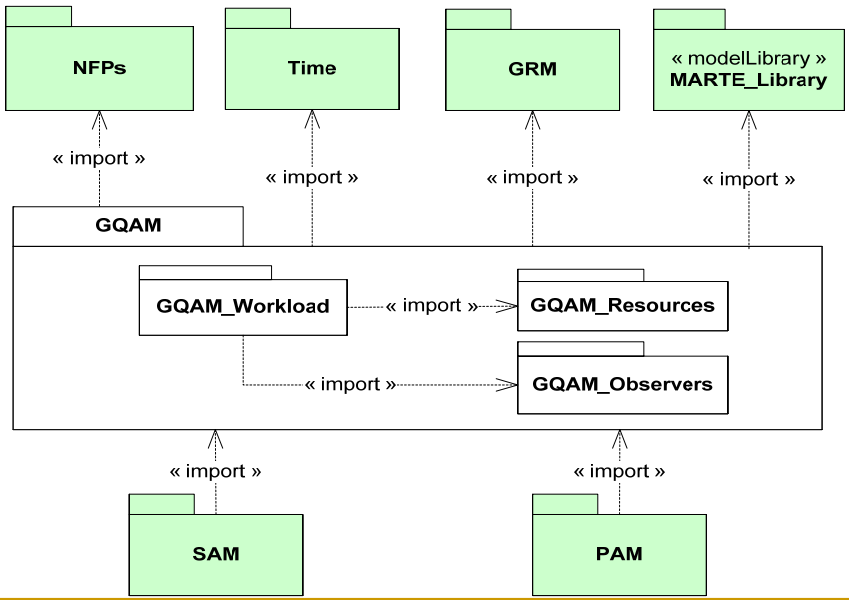
# Processing schema for model-based analysis





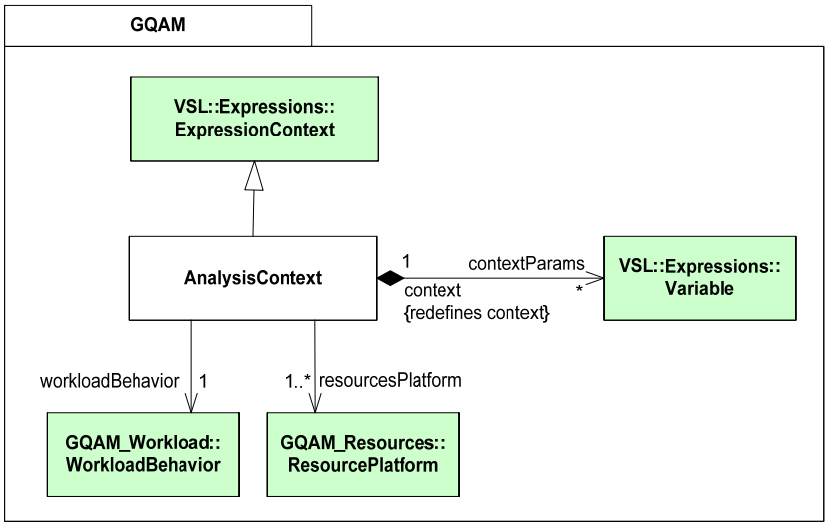

# GQAM dependencies and architecture

- **GQAM** (Generic Quantitative Analysis Modeling): Common concepts for analysis
- **SAM:** Modeling support for schedulability analysis techniques.
- **PAM:** Modeling support for performance analysis techniques.



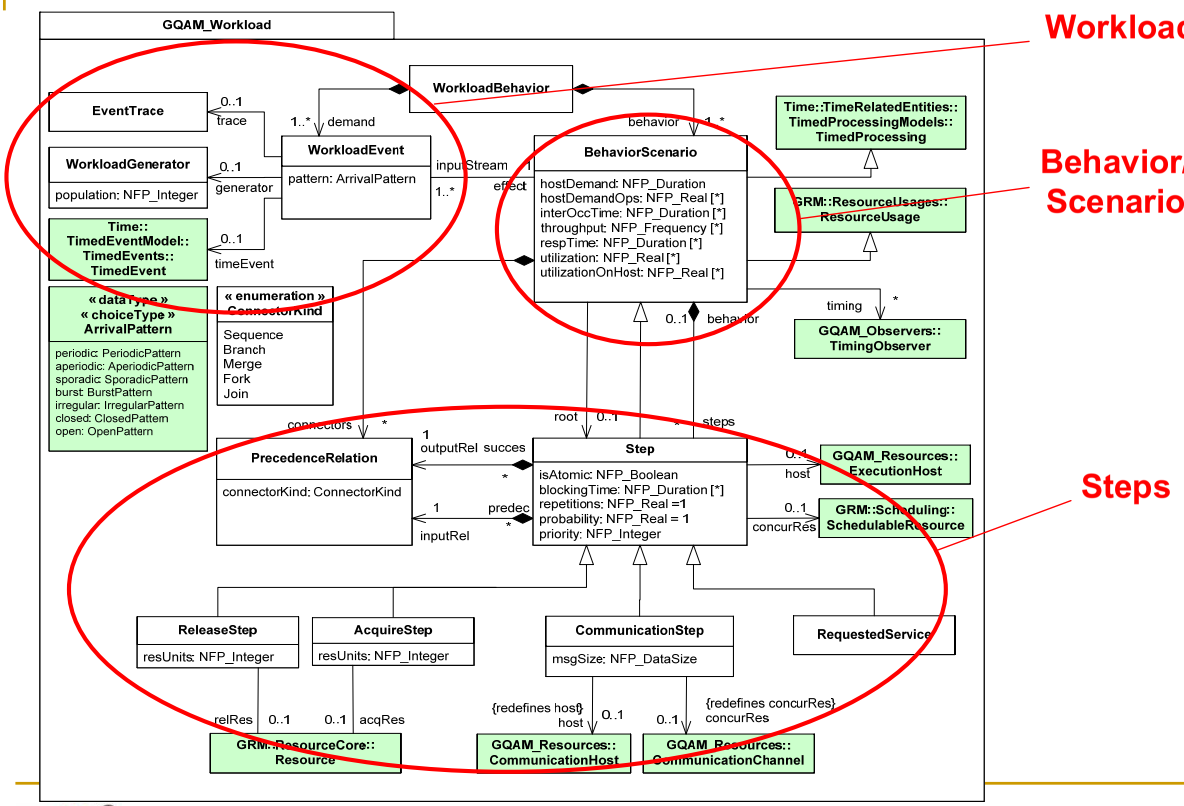
# GQAM domain model – top package

- Main concepts common for quantitative analysis:
  - Resources
  - Behavior
  - Workload
  - All embedded in an analysis context (may have analysis parameters)






# GQAM domain model: Workload and Behavior



# Workload concepts

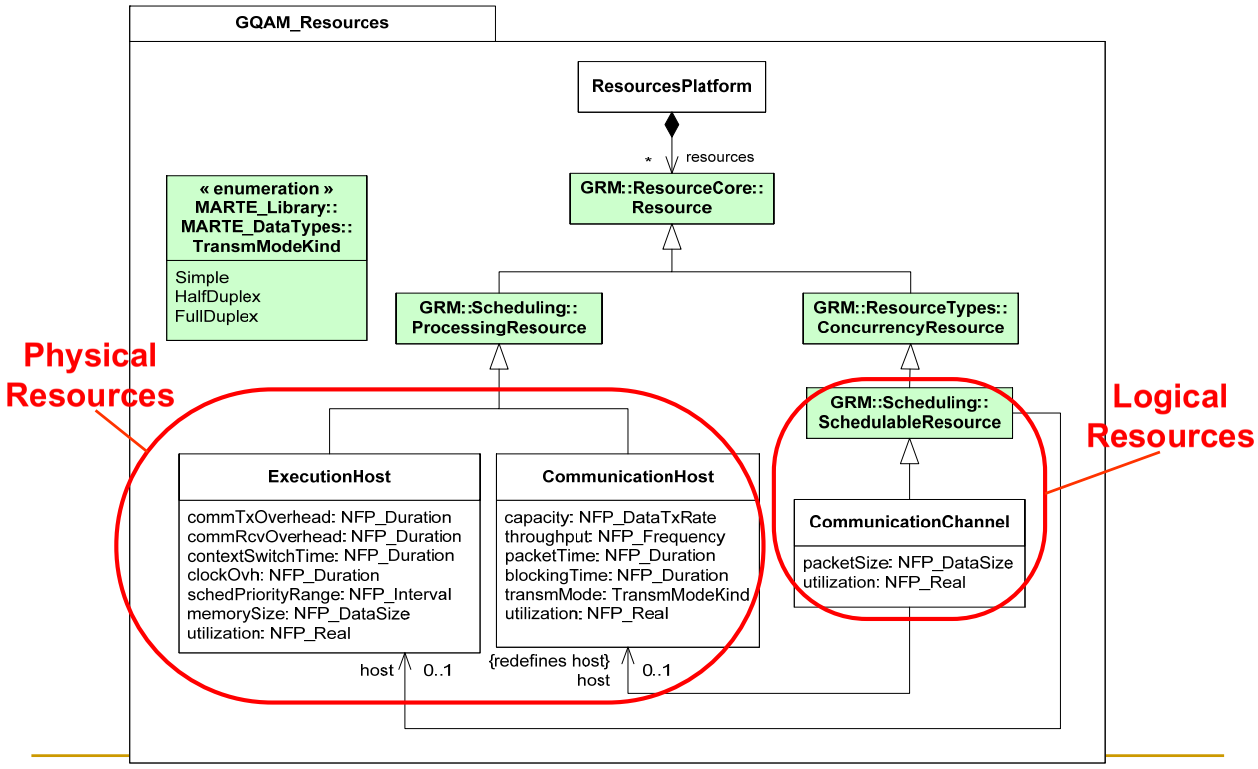
- Different workloads correspond to different operating modes:
  - takeoff, in-flight and landing of an aircraft
  - peak-load and average-load of an enterprise application.
- A workload is represented by a stream of triggering events, WorkloadEvent, generated in one of the following ways:
  - by a timed event
  - by a given arrival pattern
    - periodic, aperiodic, sporadic, burst, irregular, open, closed
  - by a generating mechanism named Workload Generator (may be modeled as a state-machine)
    - multiple independent identical generating mechanisms may exist (their number is called its "population“)
  - from a trace (EventTrace) stored in a file.



# Behavior scenario concepts

- BehaviorScenario describes a behavior triggered by an event
  - composed of sub-operations called Steps
  - any Step may be refined as another BehaviorScenario
- A BehaviorScenario captures any system-level behavior description or any operation and attaches resource usage to it in different ways:
  - each primitive Step executes on a host processor
  - a Step implicitly uses a SchedulableResource (process, thread or task)
  - a Step may be a specialized AcquireStep or ReleaseStep to acquire or release a Resource
  - BehaviorScenarios and Steps may use other kind of resources, so BehaviorScenario inherits from ResourceUsage which links resources with concrete usage demands.
    - a few concrete forms of usage defined in GQAM: memory, CPU execution time, energy from a power supply and size of messages.
- Predecessor-successor relationship between Steps:
  - sequence, branch, merge, fork, join
- A CommunicationStep defines the conveyance of a message
- Services are provided by resources and by subsystems
  - a subsystem service associated with an interface operation provided by a component may be identified as a RequestedService
  - RequestedService is a subtype of Step - may be refined by a BehaviorScenario.

# GQAM domain model: Resources



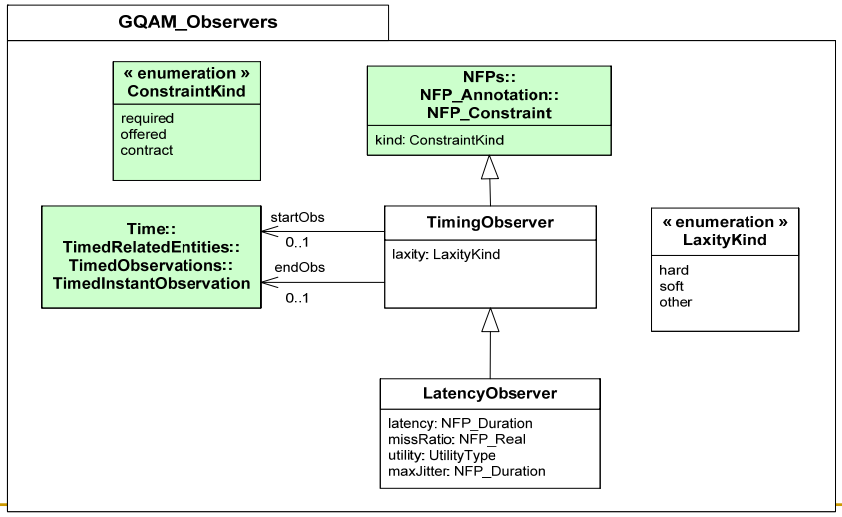



# Resource concepts

- ResourcesPlatform - the top class in the GQAM_Resource package represents a logical container for all the resources
- The viewpoint of resources is based on the abstract Resource class from the GRM package
  - common features include a scheduling discipline, multiplicity, services
- From an analysis viewpoint, four types of resources are important:
  - ExecutionHost: a processor or other device that executes operations specified in the model. It has a host role relative to the processes and the Steps that execute on it.
  - CommunicationsHost: hardware links between devices, with the role of host to the conveyance of a message.
  - SchedulableResource: a schedulable service like a process or thread pool, which is a software resource managed by the OS.
  - CommunicationChannel : a middleware or protocol layer that conveys messages.
- There are also other concurrency resources, such as mutual exclusion resources from the GRM chapter
  - e.g., critical section; semaphores and locks; a finite buffer pool; pool of admission control tokens.

# GQAM domain model: Observers

- Timing Observers are conceptual entities that collect timing requirements and predictions related to a pair of user-defined observed events startObs and endObs.
  - annotate and compare timing constraints against timing predictions provided by analysis tools
  - can be used as predefined and parameterized patterns or by means of more elaborate expressions (e.g., in OCL or VSL)
- LatencyObserver specifies a duration observation, with a miss ratio assertion (percentage), a utility function placing a value on the duration, and a jitter constraint.






# Common NFP Attributes for Analysis (1)

| NFP | For Resource | For Scenario and Step | For Workload Event |
|---|---|---|---|
| repetitions: NFP_Real[*] | N/A | the number of times the Step is repeated, once triggered (default = 1). | N/A |
| probability: NFP_Real[*] | N/A | the probability that the step is executed, following its predecessor (for conditions) | N/A |
| hostDemand:NFP_Duration[*], hostDemandOps:NFP_Real[*] | composite demand across all services of the Resource, in terms of time and in terms of processor operations | For a Step, the CPU demand on the host of the process that executes the Step. For a Scenario, the sum of all demands for all its Steps. | N/A |
| priority : NFP_Integer[*] | N/A | For a Step, priority on its host | N/A |
| respTime: NFP_Duration[*] | response time, composite average response time across all services offered by the resource | total delay from the trigger event until completion of the Step or Scenario | required value for the Scenario |
| execTime : NFP_Duration[*] | execution time | respTime minus any scheduling delays | N/A |

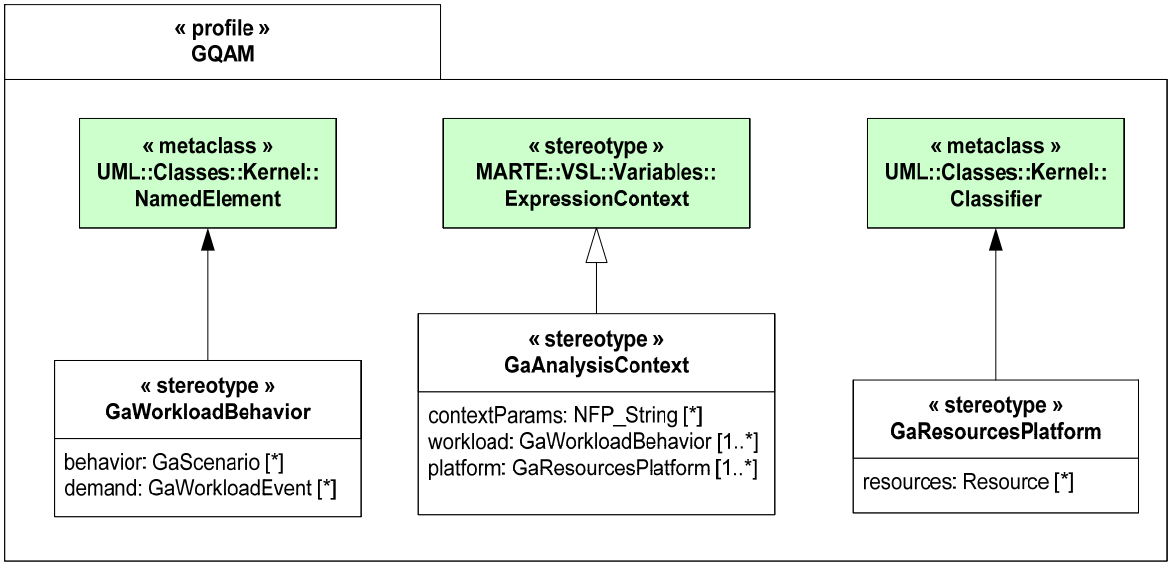
# Common NFP Attributes for Analysis (2)

| NFP | For Resource | For Scenario and Step | For Workload Event |
|---|---|---|---|
| interOccTime: NFP_Duration[*] | inter-occurrence time, interval between successive requests for services | interval between initiations | interval between trigger events |
| throughput : NFP_Frequency[*] | frequency of requests for all services | frequency of initiations | frequency of the trigger event |
| utilization : NFP_Real[*] | fraction of time the resource is active (has an active service). For a multiple resource, the mean number of busy units. | fraction of time the BehaviorScenario is active (between its trigger event and its completion) | N/A |
| utilizationOnHost: NFP_Real[*] | N/A | fraction of time the host is busy executing the BehaviorScenario. If it has multiple hosts, this is a set of values. | N/A |
| blockingTime: NFP_Duration[*] | blocking time | a pure delay which is part of the behavior of the Step or Scenario | N/A |

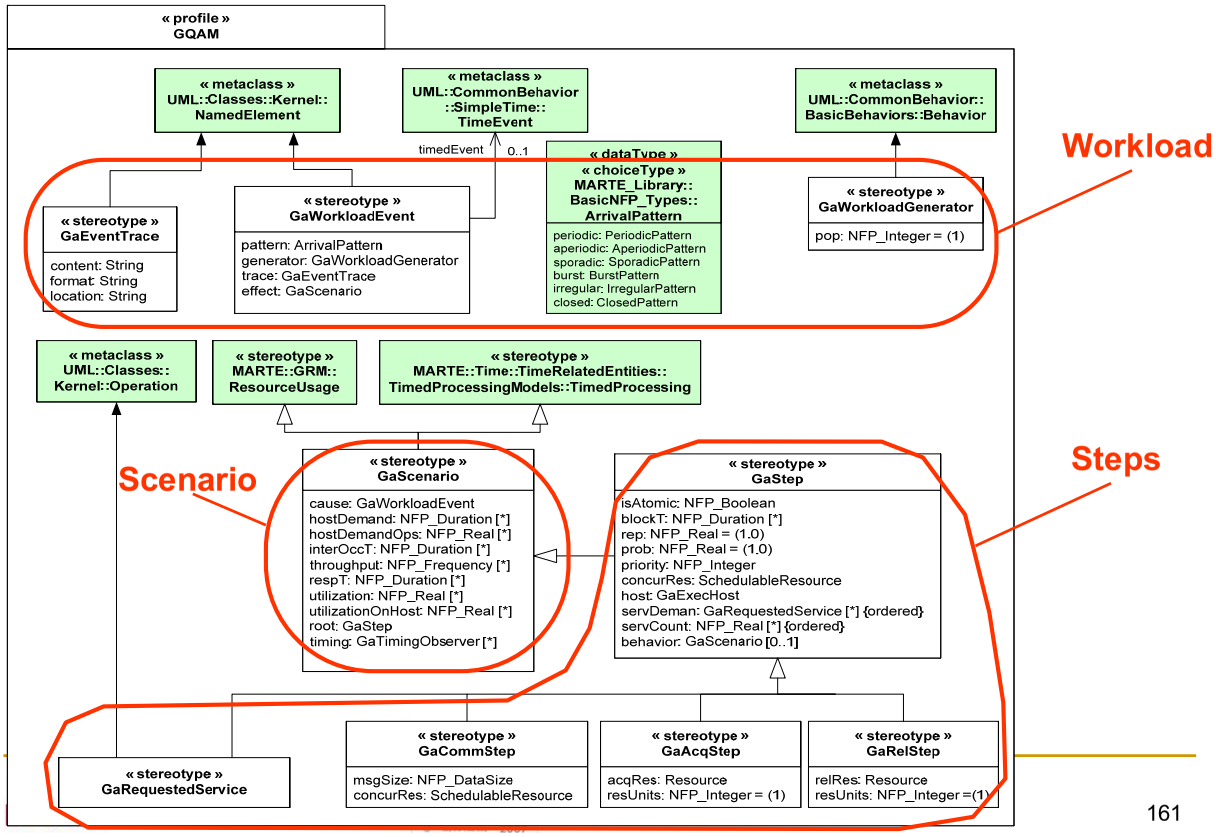


# GQAM Profile: top-level stereotypes

- The UML extensions for the GQAM sub-profile are presented in four figures related to corresponding domain model packages:
  - GQAM, GQAM_Workload, GQAM_Resources and GQAM_Observers.



# GQAM Profile: Workload and Behaviour



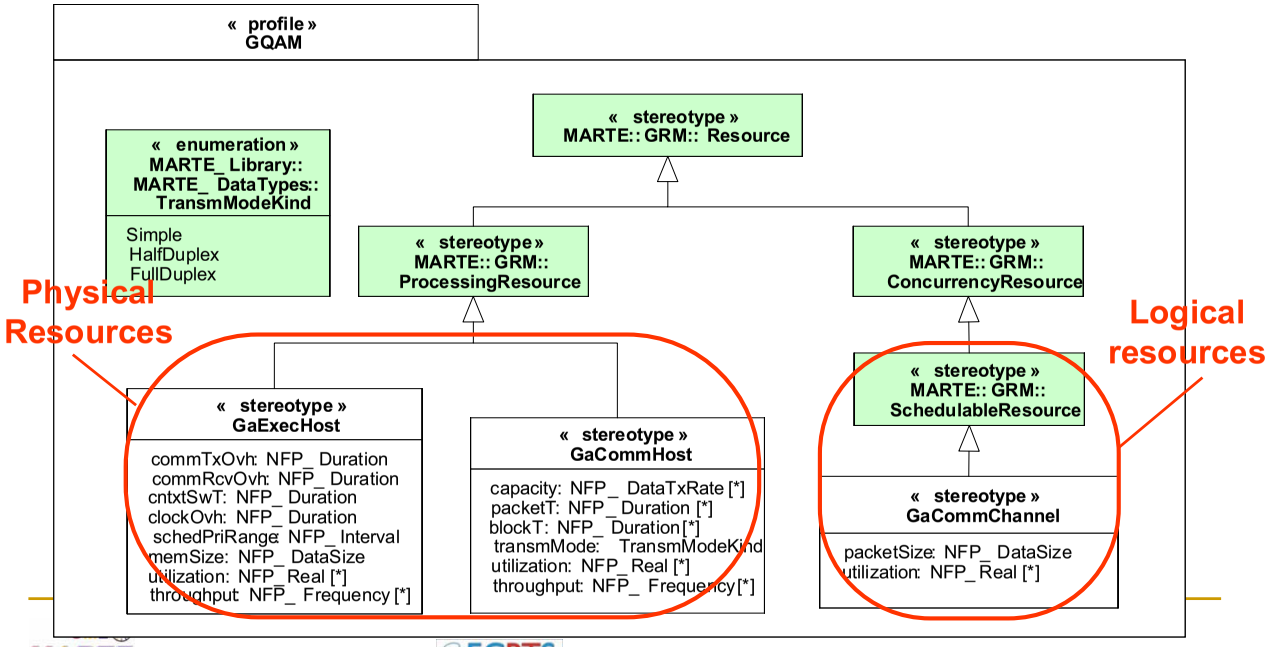



# Applying behavior-related stereotypes

- *GaScenario* and *Step* stereotypes inherit from:
  - *TimeModels::TimedProcessing* - which in turn extends *Behavior, Message, Actions*
  - and from *GRM::ResourceUsage* - which extends *NamedElement*
- Therefore, Scenario and Step stereotypes can be applied to a wide set of behavior-related elements covered by the UML 2 metaclass NamedElement, such as:
  - *Operations*, *Actions*
  - *Messages* that initiate *Operations* or *Actions*
  - *Transitions* and *States* in state machine diagrams
  - *Signals* that trigger state machine transitions
  - *Events*, *ExecutionOccurrenceSpecifications* and *InteractionFragments* in interaction diagrams
  - *InputPins* in activity diagrams
  - *UseCases*.

# GQAM Profile: Resources

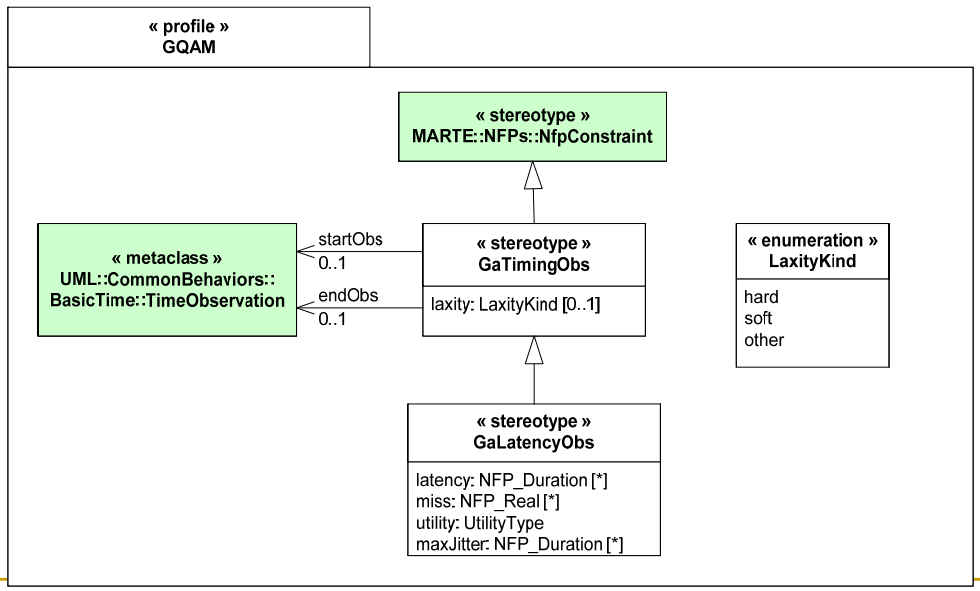
- In general, resource-related stereotypes extend the UML metaclasses *Classifier*, *InstanceSpecification* and *Property*
- Therefore, resource stereotypes can be applied to all kinds of classes, instances, components, parts and deployment nodes.






# GQAM Profile: Observers

- A timing observer may be attached to a start and end observed events, or to a behavior element (whose execution start and end represent the observed events)
- Such modeling constructs are useful for complex end-to-end flows, providing flexible means to annotate analyzer-defined timing constraints.



# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE



# Key reasoning for the use of UML in the scheduling analysis of RTE Systems

- UML is a standard semi-visual language for conceptual modeling that enables the usage of a Model Based approach for software and system engineering.
- MDD and UML have been broadly introduced and used in principle by the software engineering community, and have reached a significant number of practitioners and tool support.
- To take benefit of this in the RTE domain, they need to be capable of supporting the necessary (at least timing) verifications.
- This leads to the necessity of model based scheduling analysis techniques.
- As well as the necessity to have the modeling elements to describe the platform, the interacting environment and the timing requirements.

# Topics

- A flash of schedulability analysis:
  - RMA: the foundations
  - Offset-based techniques: high level models.
- Structural and behavioral modeling elements in MARTE.
- Examples of the notation used.



# Options for managing time

- Compile-time schedules:
  - Known as cyclic executives
  - Predictability through static schedule
  - Logical integrity often compromised by timing structure
  - Difficult to maintain, even with model based strategies
- Run-time schedules:
  - Priority-based schedulers
  - Preemptive or non preemptive
  - Fixed priority or dynamic priority
  - Analytical methods needed for predictability
  - Separates logical structure from timing
- Server-based frameworks:
  - Combine static or dynamic schedules with budgets and periods enforced at run-time

# Fixed-Priority Scheduling

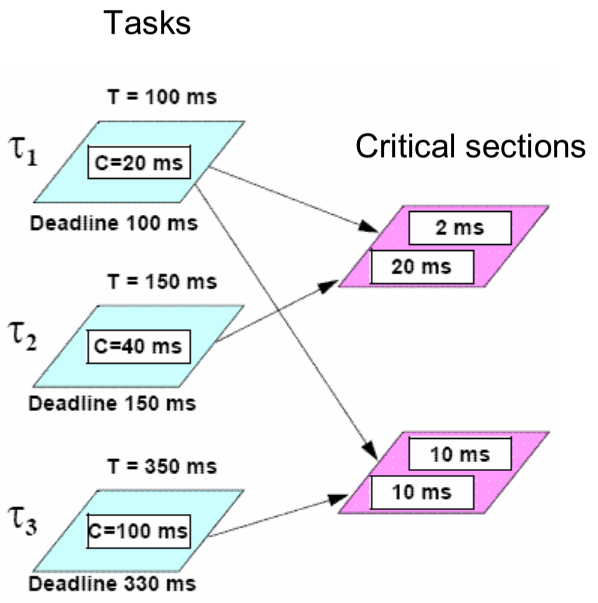
- Fixed-priority preemptive scheduling is very popular for practical applications, because:
  - Timing behavior is simpler to understand
  - Treatment of transient overload is easier to predict
  - A complete analytical technique exists
  - All standard concurrent languages or operating systems specify fixed-priority scheduling
- FP analysis works also for non-preemptible tasks or non-preemptible sections



# Rate Monotonic Analysis (RMA)

- Rate monotonic analysis is an engineering basis for analyzing and designing real-time systems.
- Provides guidelines for optimum priority assignment
- Provides an analytical framework for verifying timing requirements
- Helps to identify timing bottlenecks and errors
- Is very popular, and is supported by concurrent languages and real-time operating systems.

# A simple example




(Slide credit: M.Gonzalez Harbour & the SEI RMA tutorial)



# Why Are Deadlines Missed?

- For a task to meet its deadline, it must accommodate
  - preemption: time waiting for higher-priority tasks
  - execution: time to do its own work
  - blocking: time delayed by lower-priority tasks
- The task is schedulable if the sum of its preemption, execution, and blocking is less than its deadline for the worst possible case:
  - Execution is unavoidable (unless requirements change).
  - Preemption can be minimized by choosing an optimum priority assignment
  - Main focus: identify and reduce blocking

# Concepts and Definitions (periodic tasks)

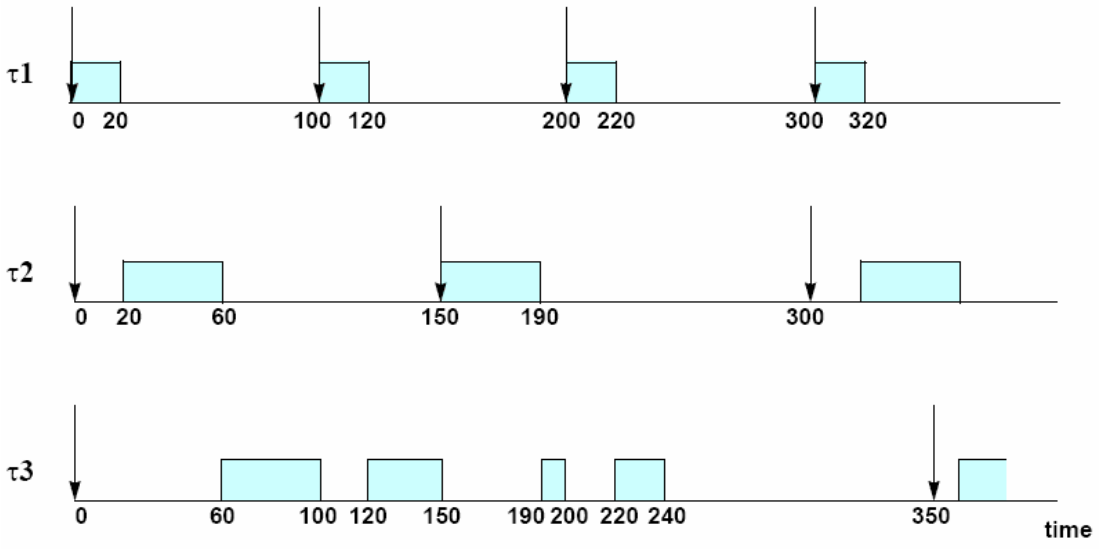
- Periodic task
  - initiated at fixed intervals
  - must finish before start of next cycle
- Task's CPU utilization: $U_i = C_i / T_i$
  - $C_i$ = compute time (execution time) for task ti
  - $T_i$ = period of task $t_i$
  - $P_i$ = priority of task $t_i$
  - $D_i$ = deadline of task $t_i$
  - $f_i$ = phase of task $t_i$
  - $R_i$ = response time of task $t_i$
- CPU utilization for a set of tasks: $U = U_1 + U_2 + \ldots + U_n$



# Basic Principles of RMA

- Two concepts help to build the worst-case condition:
  - Critical instant. The worst-case response time for all tasks in the task set is obtained when all tasks are activated at the same time
  - Checking the first deadline. When all tasks are activated at the same time, if a task meets its first deadline, it will always meet all of its deadlines
- Based on these concepts, several results arise:
  - Optimality of rate monotonic priorities
  - Utilization bound test
  - Exact test

# Critical instant




(Slide credit: M.Gonzalez Harbour & the SEI RMA tutorial)



# Utilization and Response Time tests

- Utilization Bound(UB) Test: A set of n independent periodic tasks, with deadlines at the end of the periods, scheduled by the rate monotonic algorithm will always meet its deadlines, for all task phasings, if: U <= U(n) = $n(2^{1/n} - 1)$
- Completion time test:
  - For a number the tasks according to priority (highest priority=$t_1$, lowest priority=$t_n$)
  - Under a critical instant condition, the amount of work Wi(t) of tasks at priority Pi or higher started before t is

$$W_i(t) = \left\lceil \frac{t}{T_1} \right\rceil C_1 + \ldots + \left\lceil \frac{t}{T_{i-1}} \right\rceil C_{i-1} + C_i$$

# Response time evaluation

- *R*i may be computed by the following iterative formula:

$$a_0 = C_1 + C_2 + \ldots + C_i$$

$$a_{k+1} = W_i(a_k) = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + \ldots + \left\lceil \frac{a_k}{T_{i-1}} \right\rceil C_{i-1} + C_i$$

- The iteration ends when:

$$a_{k+1} = a_k = R_i$$

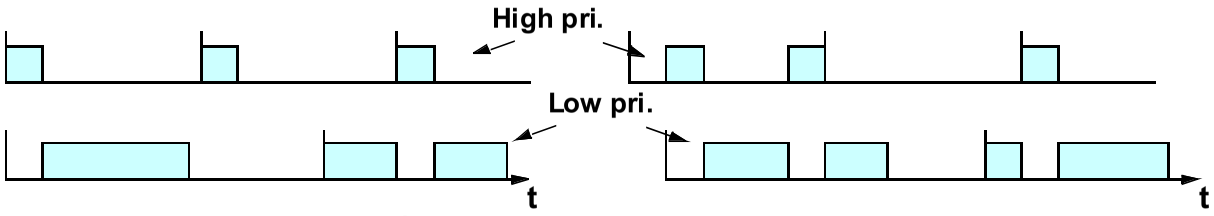
- Task $t_i$ is schedulable if:

$$R_i \leq D_i$$



# Effects of jitter

- Periodic events with jitter have an arrival time which may be early or late, within a bounded interval:
  - events arrive at $t_o + nT +/- J$
- Jitter may have a delay effect on lower priority tasks



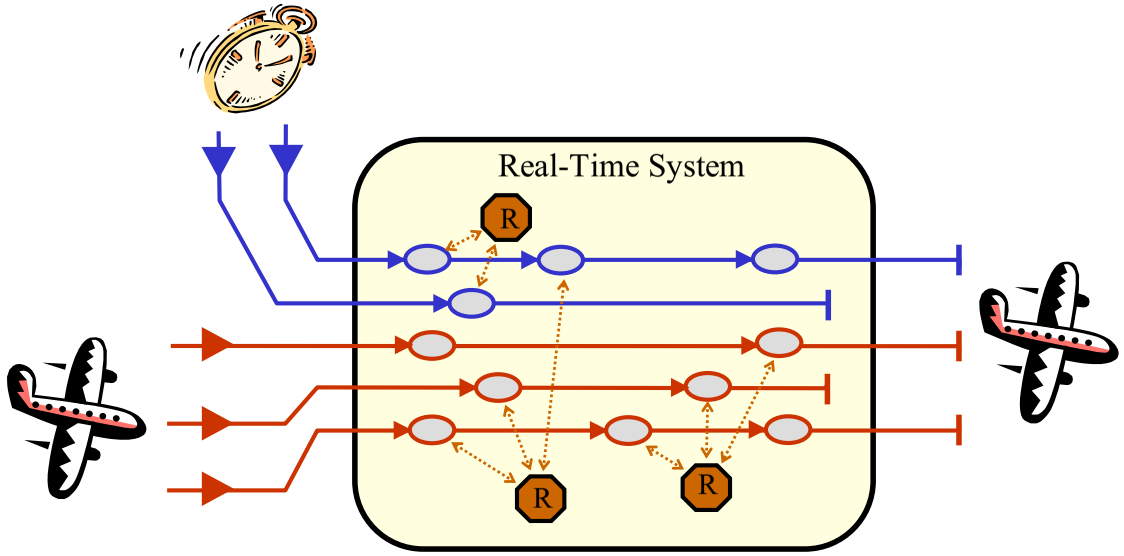

**Execution sequence for two periodic tasks. Worst case is one preemption**

**Execution sequence with jitter. Worst case is two preemptions**

(Slide credit: J.C. Palencia & M.Gonzalez Harbour)

# More general approach

**Scenario(instance) based, distributed, control-flow dependencies**

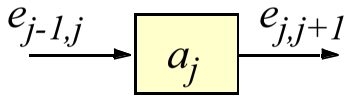



(Slide credit: J.M. Drake)



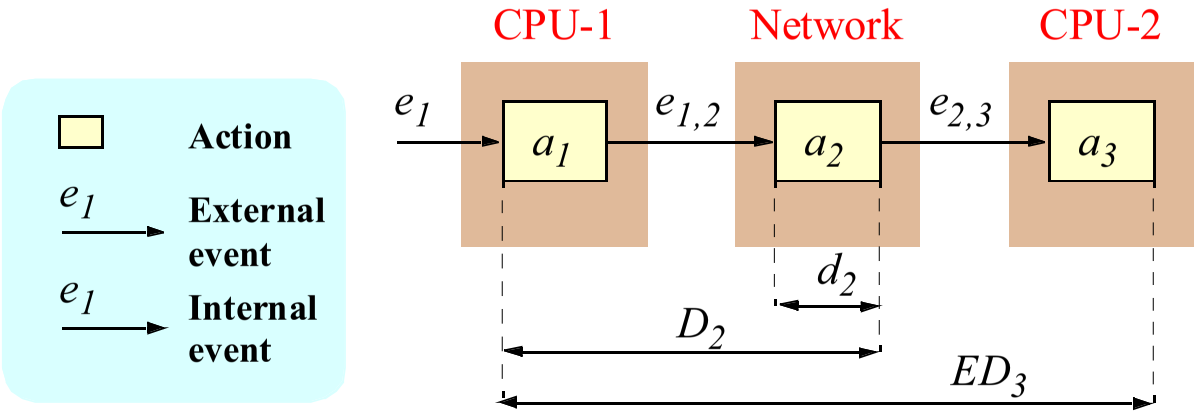
# Distributed system model

Linear Action:



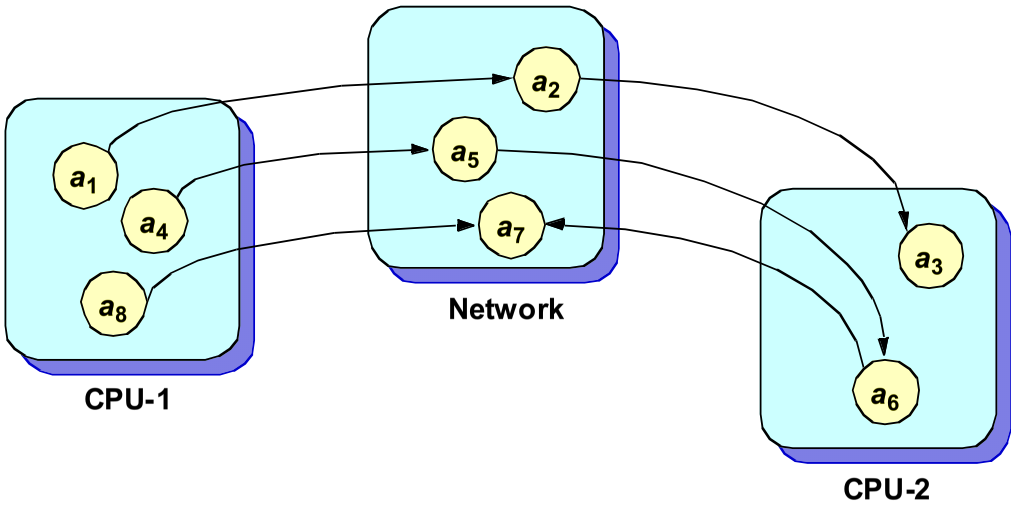

$$T_{j-1,j} = T_j = T_{j,j+1}$$

Linear Response to an Event:




(Slide credit: J.C. Palencia & M.Gonzalez Harbour)

# Jitter in distributed systems

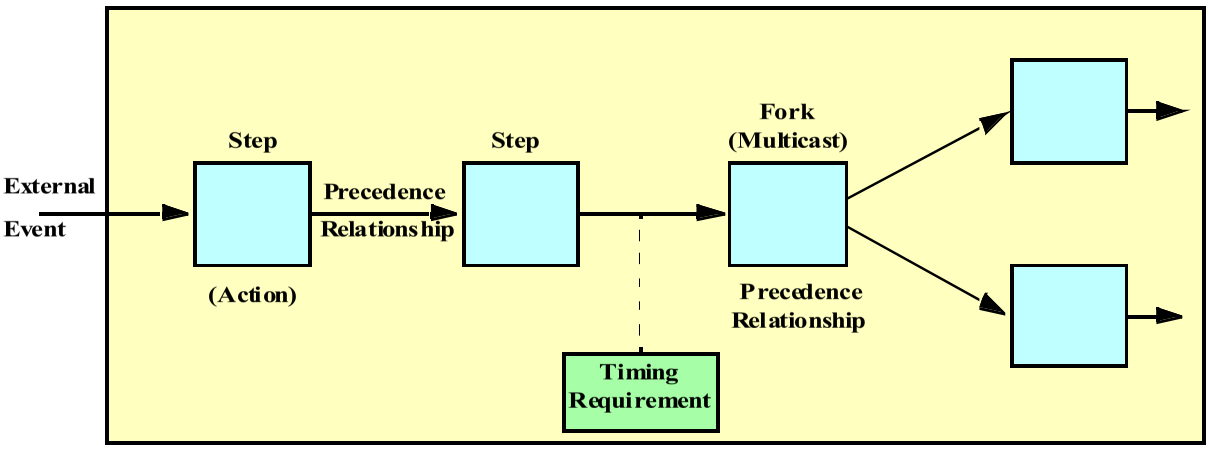



- Jitter in one processing resource depends on the response times in other processing resources
- Response times depend on jitters

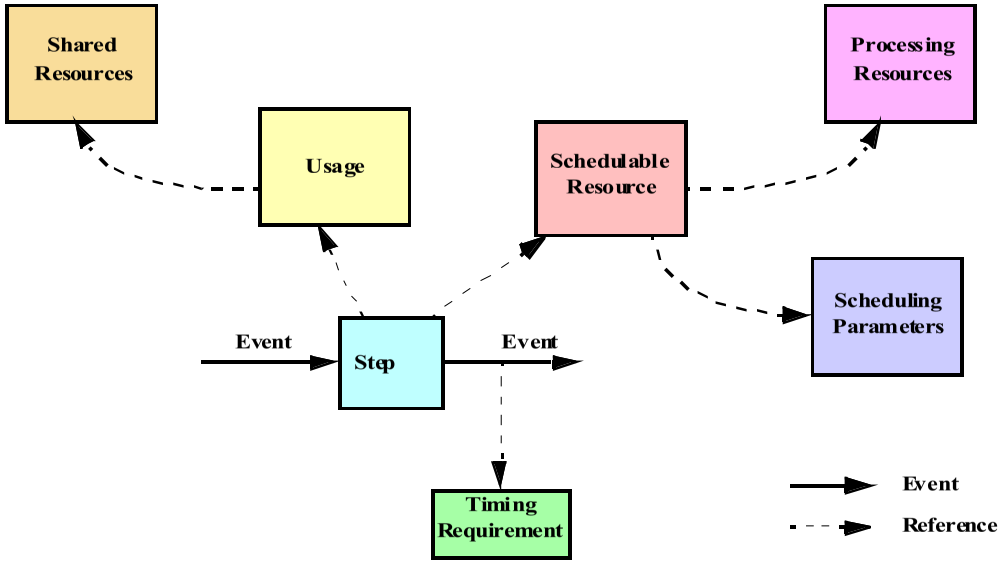
(Slide credit: J.C. Palencia & M.Gonzalez Harbour)



# Transactions

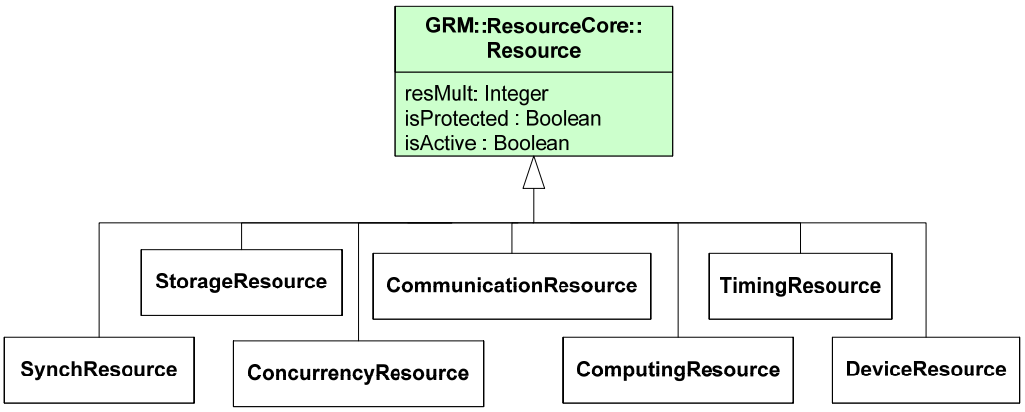


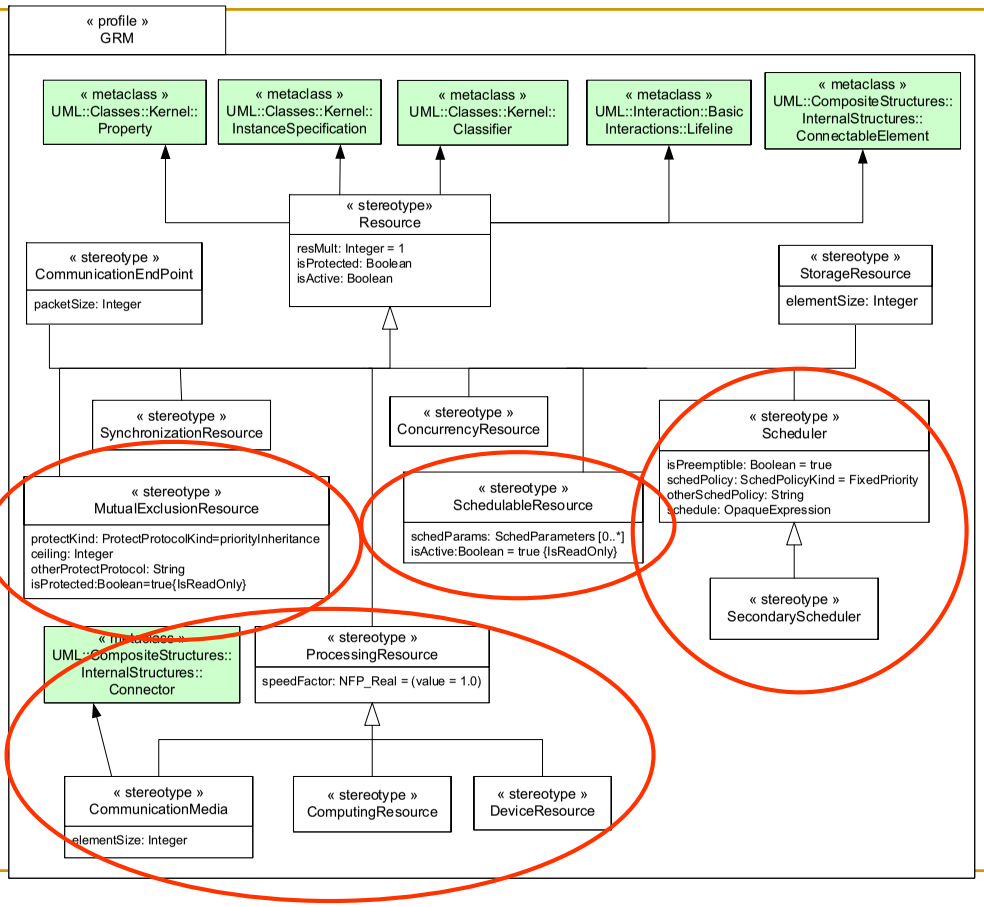
# Transactions





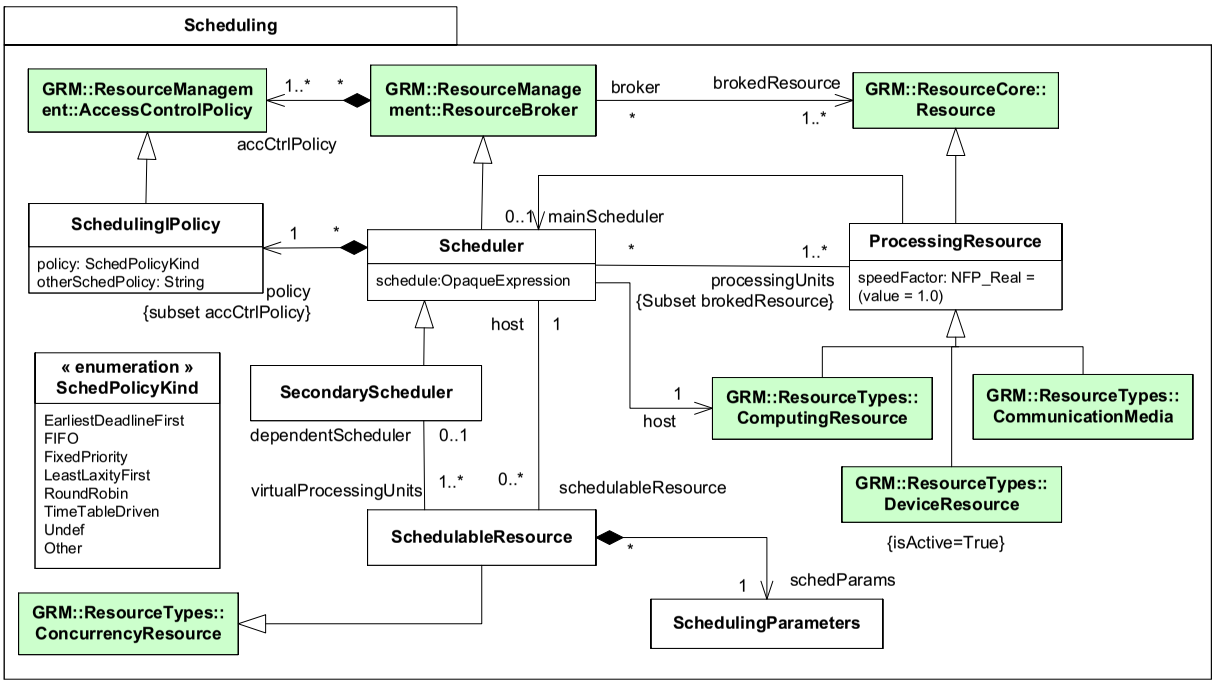

# Resources



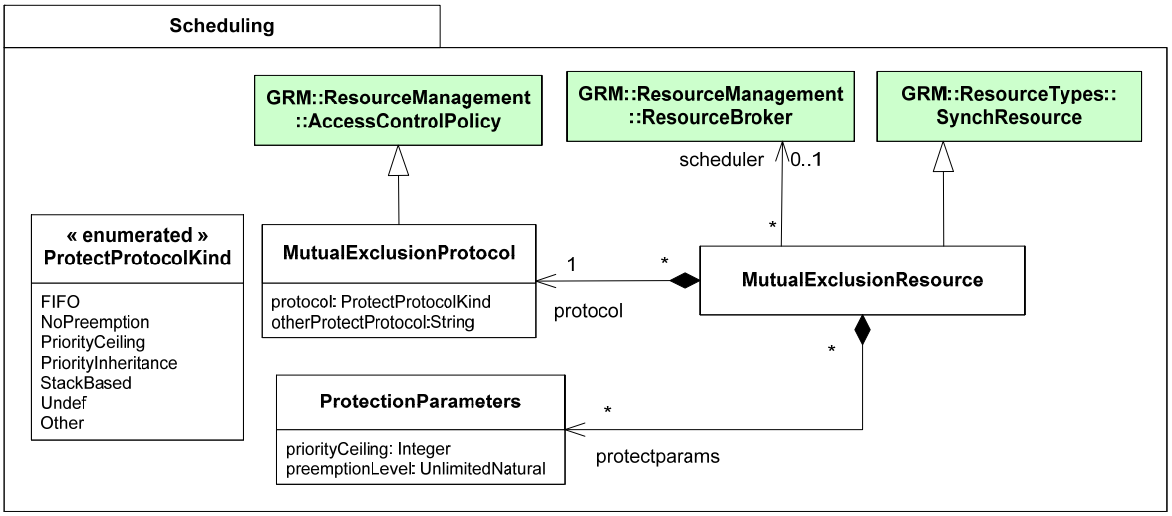





# Domain model for Schedulable resources

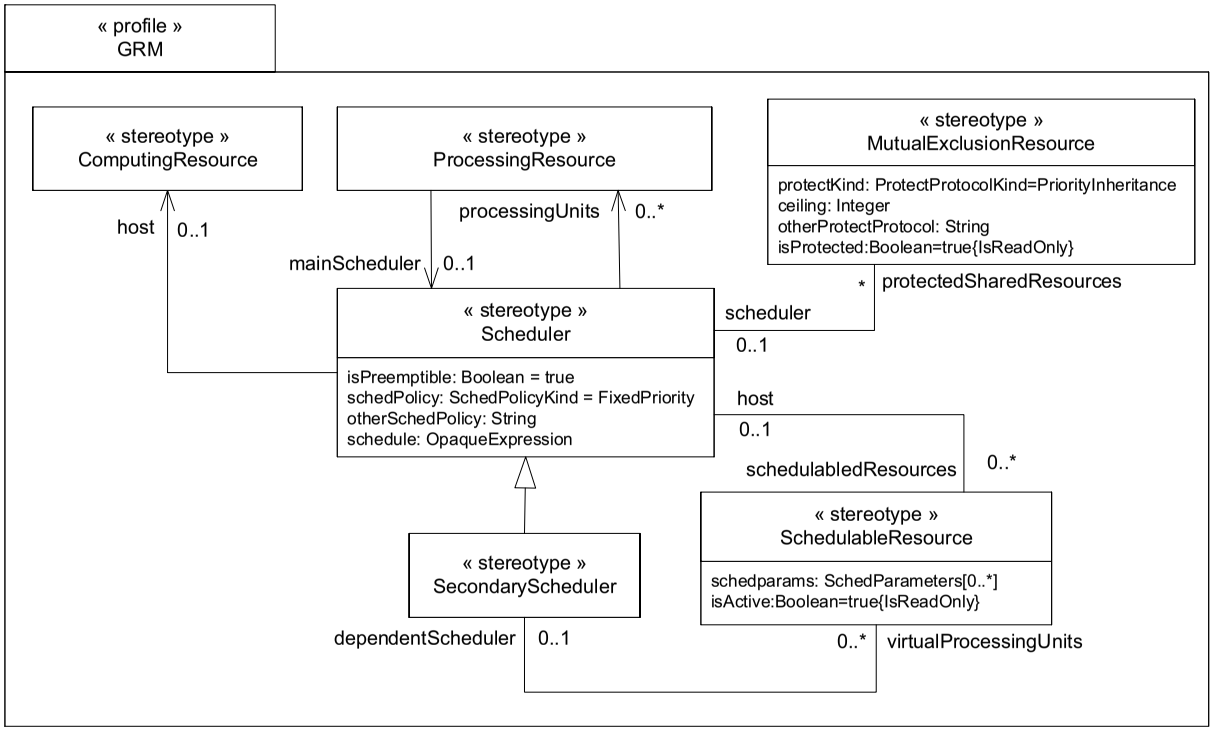


# Shared resources

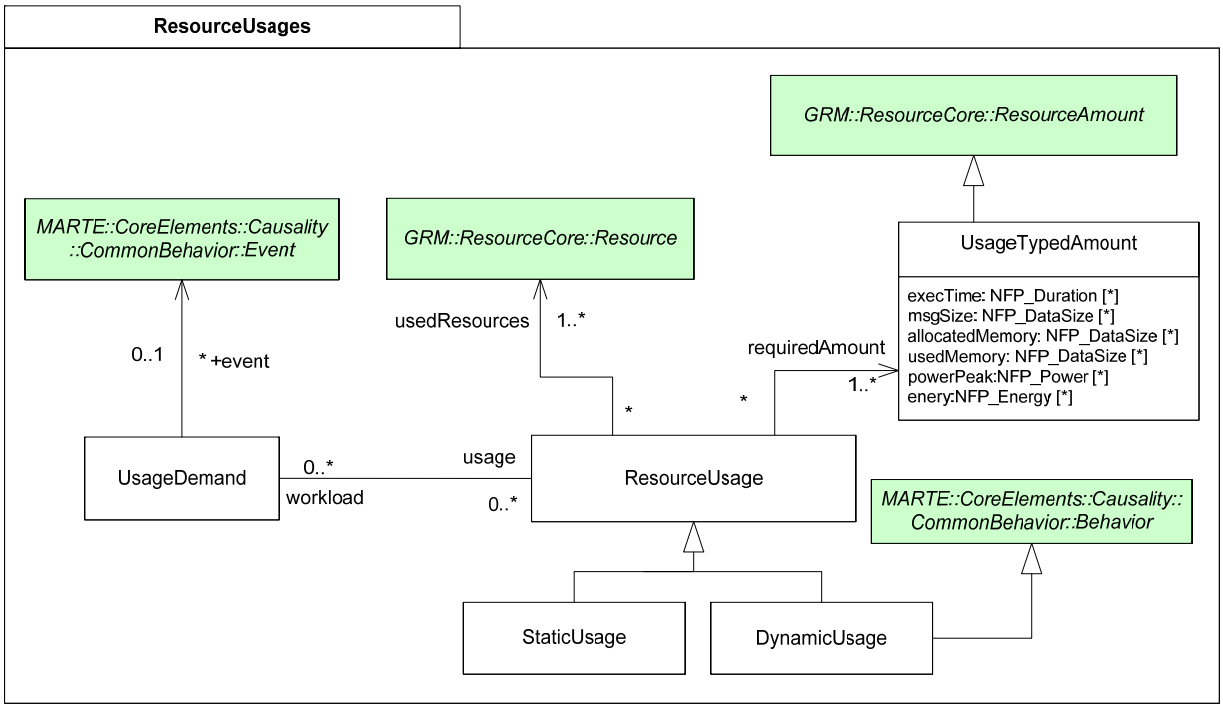




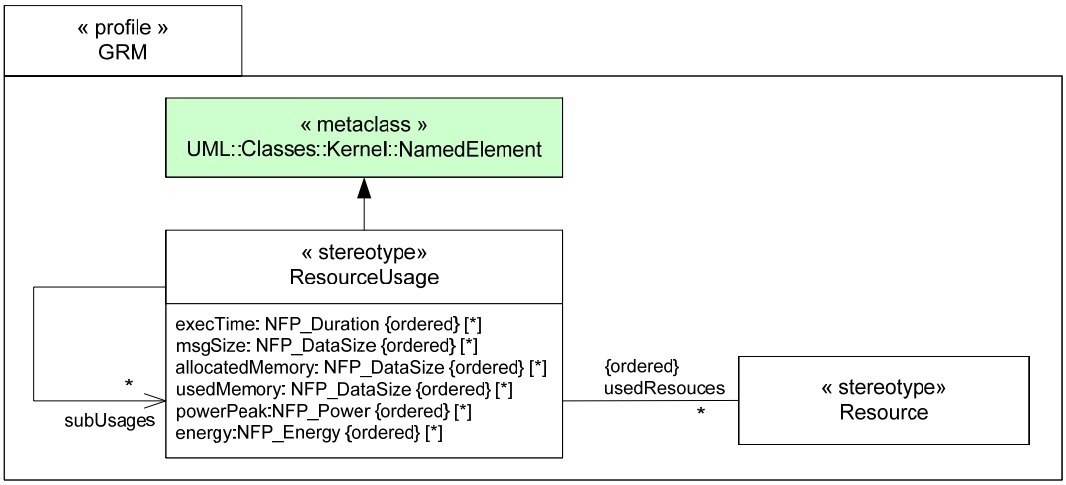

# Extensions for scheduling



# Resource Usage

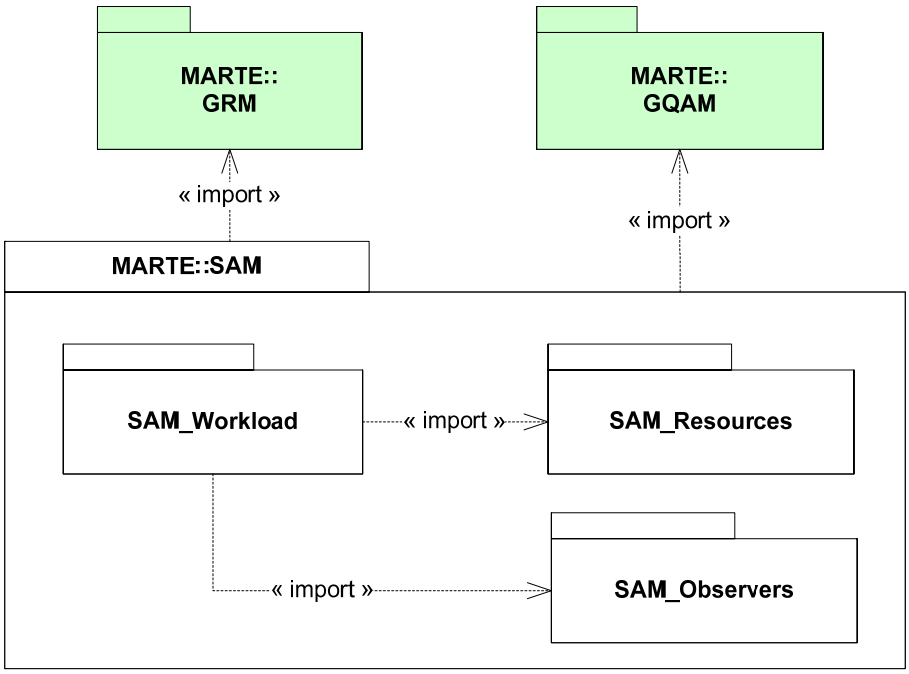




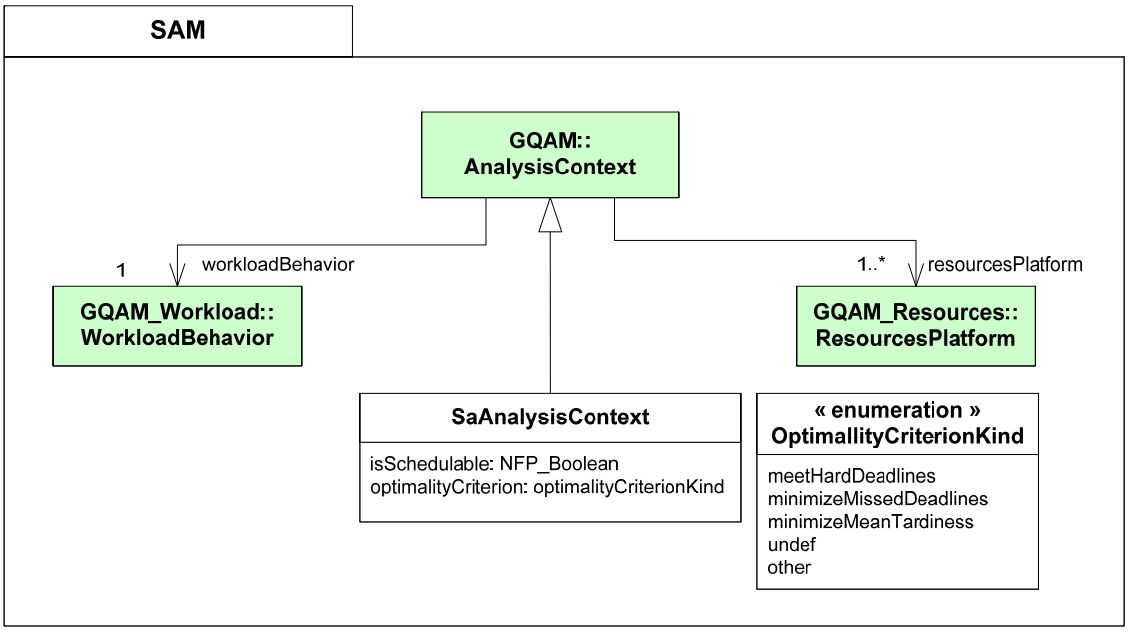

1. If the list usedResources is empty the list subUsages should not be empty and viceversa..
2. If the list usedResources has only one element, all the optional lists of attributes refer to this unique Resource and at least one of them must be present.
3. If the list usedResources has more than one element, all of the optional lists of attributes that are present, must have that number of elements, and they will be considered to match one to one.
4. If the list subUsages is not empty, and any of the optional lists of attributes is present, then more than one annotation for the same resource and kind of usage may be expressed. In this case, if the annotations have also the same source and statistical qualifiers they will be considered in conflict, and hence the ResourceUsage inconsistent.

# Structure of SAM

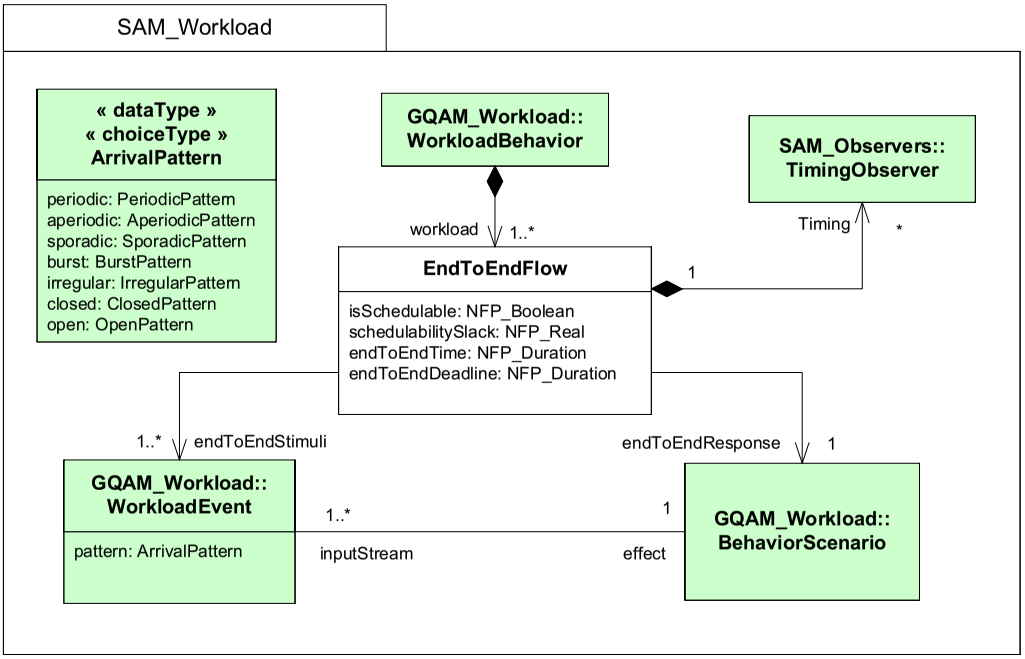





# SAM Overview

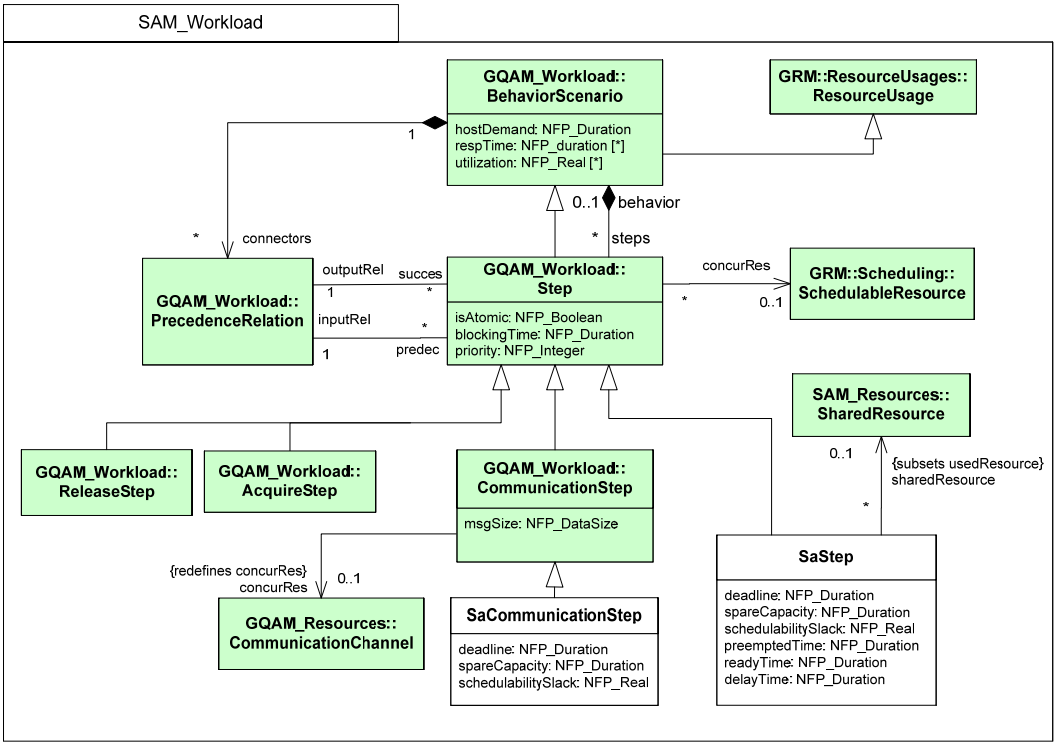


# SAM_Workload: End-to-endFlow

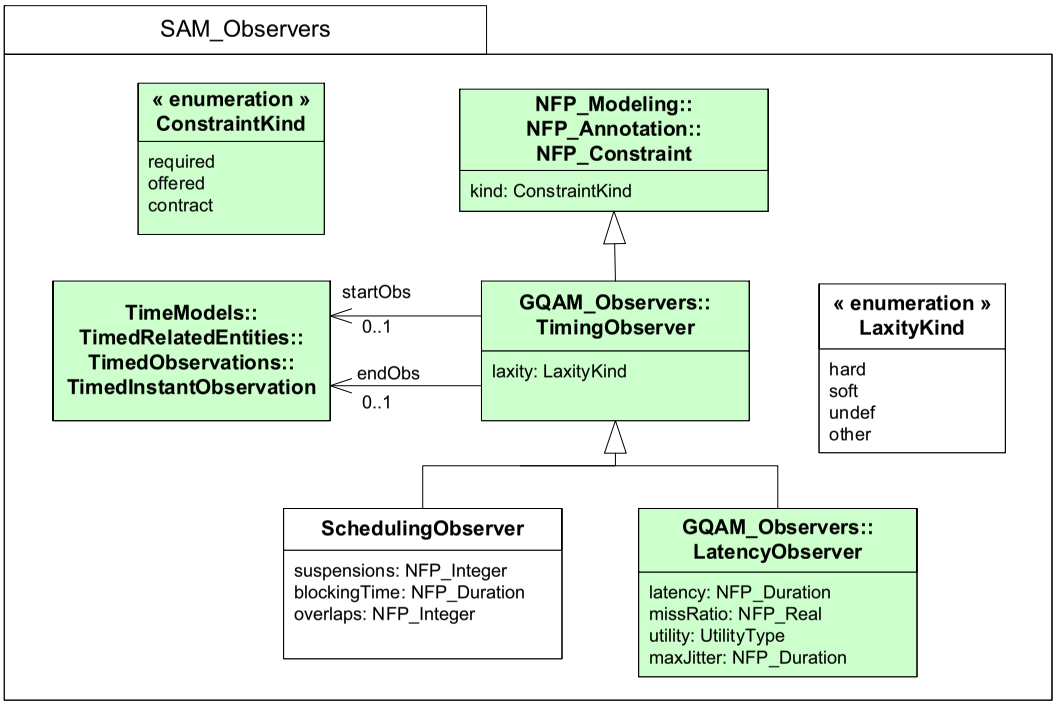





# SAM_Workload: BehaviorScenario

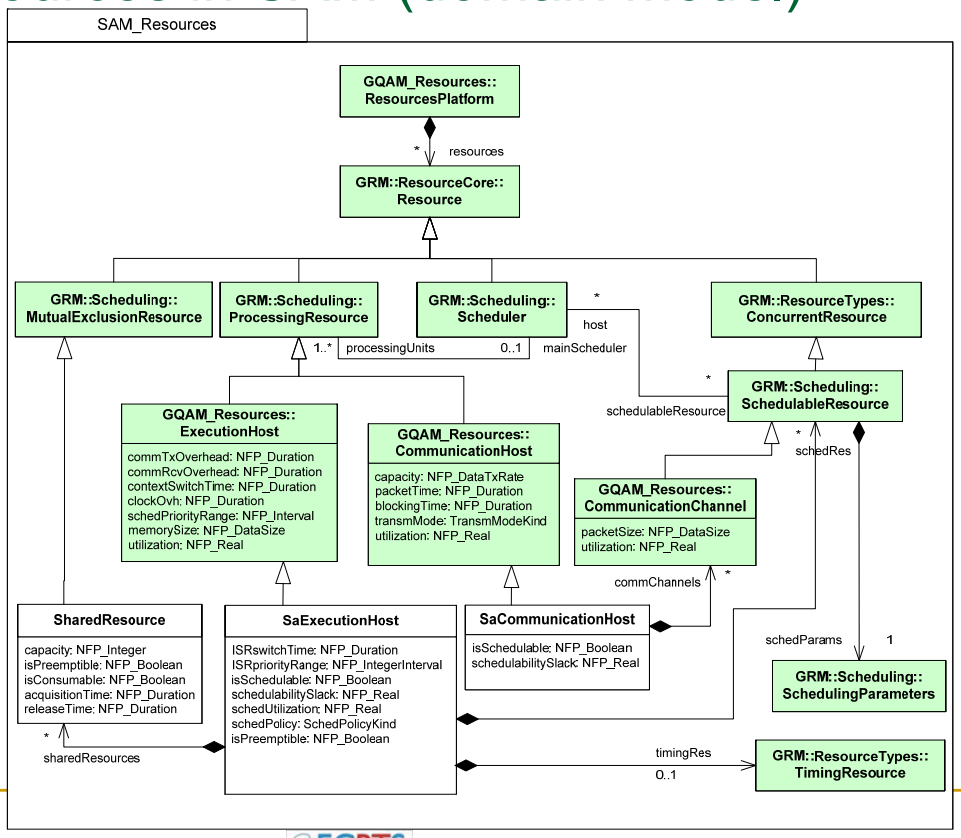


# SAM_Observers

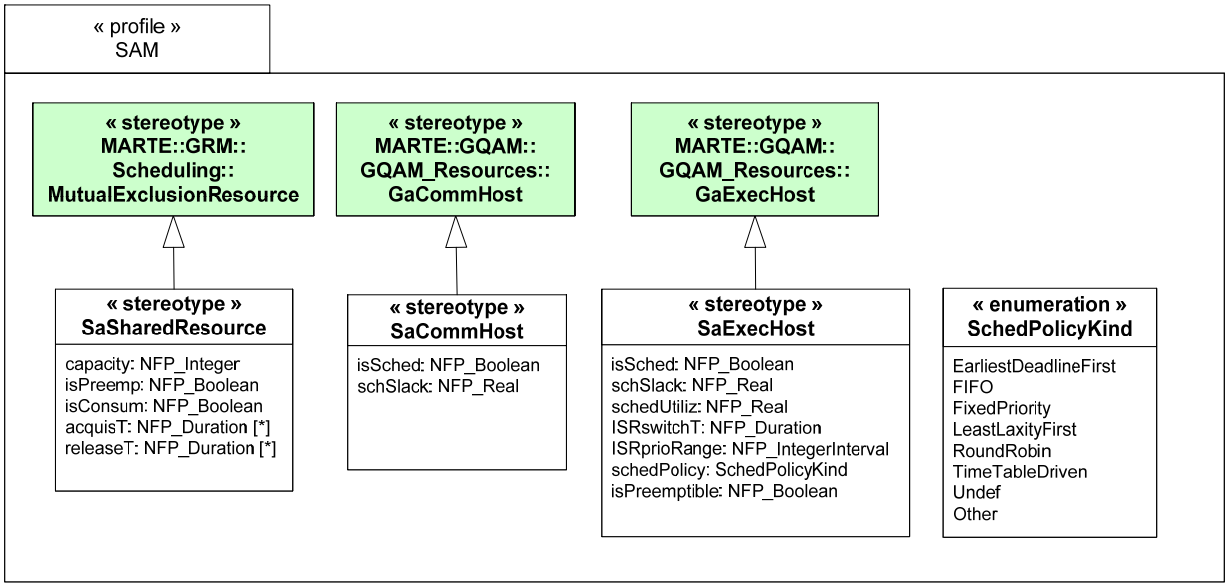





# Resources in SAM (domain model)

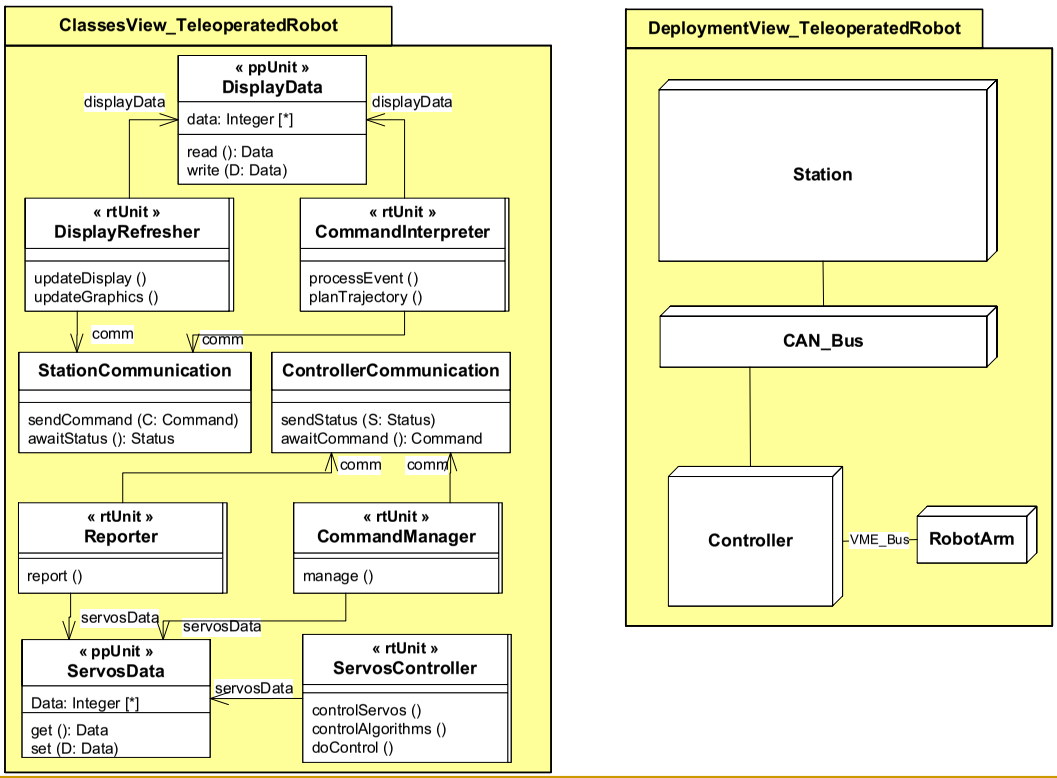


# Resources in SAM (profile)

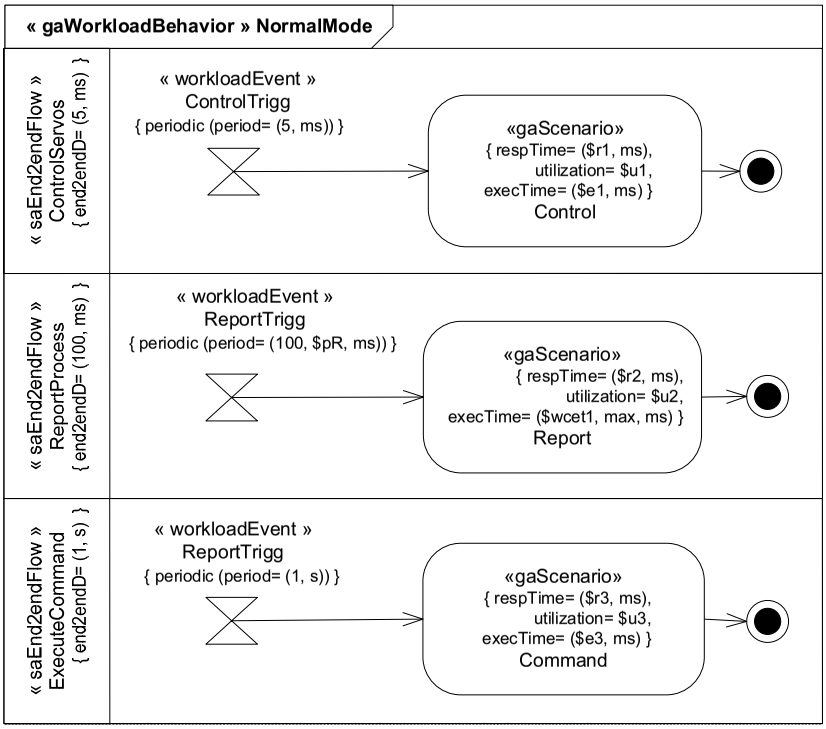





# Example

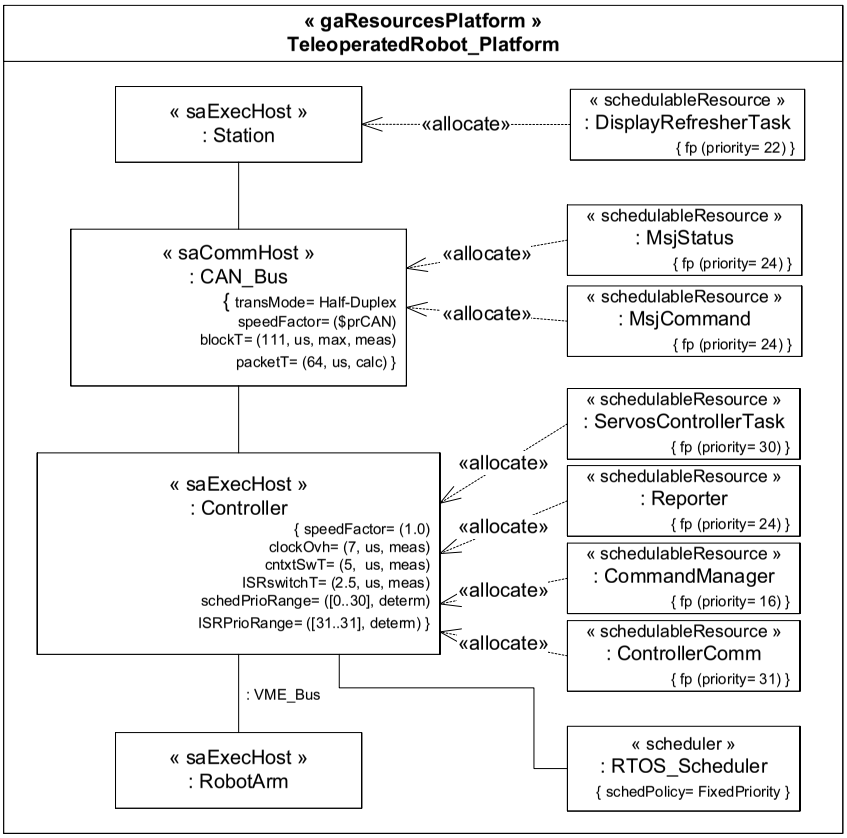


# Real-time situation

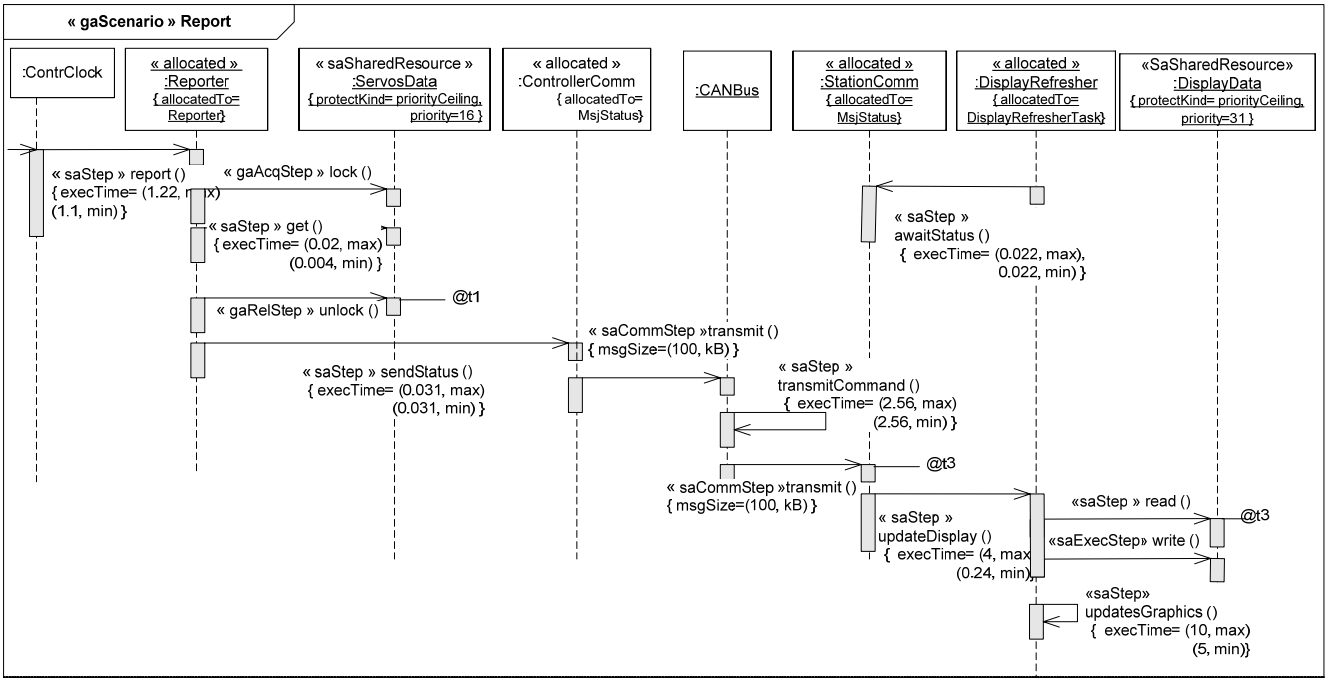





# Platform

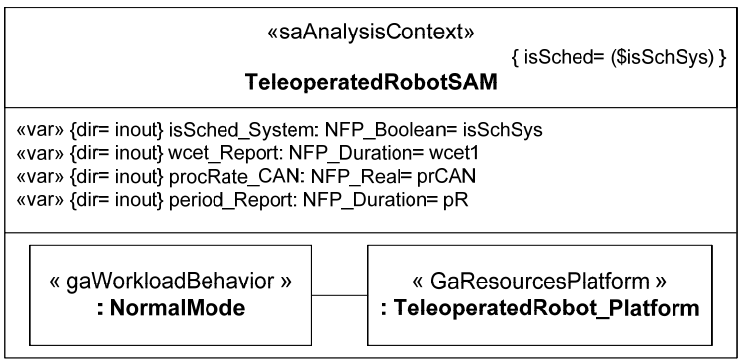


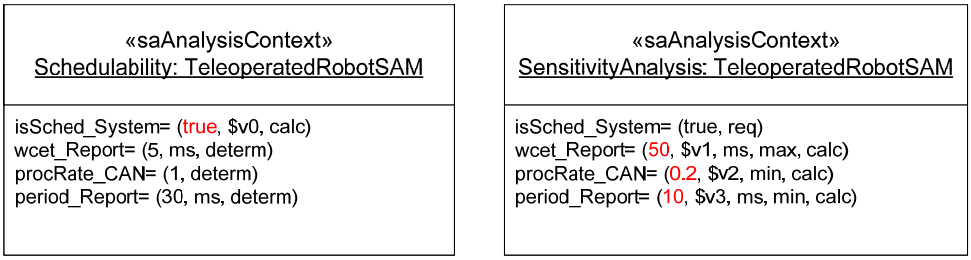
# A behavior scenario






# Parametric analysis contexts



# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

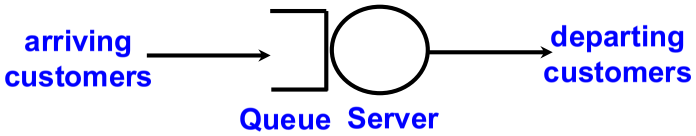
**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE



# Performance modeling formalisms

- Different performance models developed over time in the performance evaluation field may be used for performance analysis of UML models
  - model transformation from UML+MARTE to such target performance models are needed
- Classification of performance models
  - Analytic models
    - Queueing Networks (QN)
      - capture well contention for resources
      - efficient analytical solutions exists for a class of QN (“separable” QN)
      - Layered Queueing Networks (QN extension) are used as an example
    - Stochastic Petri Nets
      - good flow models, but not as good for resource contention
      - analytical solutions suffer from state space exponential explosion
    - Stochastic Process Algebra
      - introduced recently by merging Process Algebra and Markov Chains
  - Simulation models
    - less constrained in their modeling power, can capture more details
    - harder to build and more expensive to solve (run the model repeatedly).

# Queueing Network (QN): Single Service Center



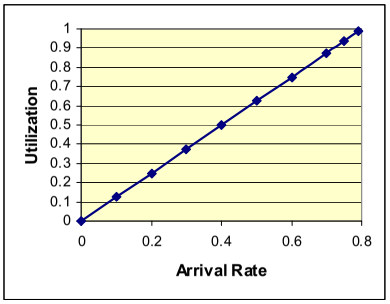

- Parameters
  - scheduling policy (FIFO, priority, etc.)
  - workload intensity - arrival process
    - e.g., Poisson arrival with rate 0.5/second
  - service demand per customer
    - e.g., exponential distribution with mean of 1.25 seconds
- Performance measures
  - utilization = proportion of time the server is busy
  - residence time = average time spent at the service center by a customer, both queueing and receiving service
  - queue length = average number of customers at the service center
  - throughput = rate at which customers pass through the service center.



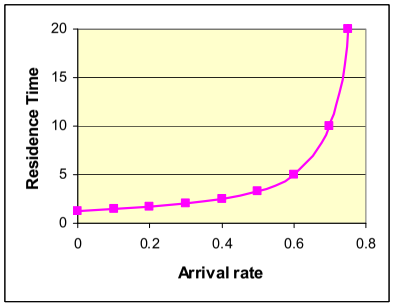
# Single Service Center: Performance Results

- Typical non-linear behaviour for queue length and waiting time
  - server reaches saturation at a certain arrival rate (utilization close to 1)
  - at low workload intensity: an arriving customer meets low competition, so its residence time is roughly equal to its service demand
  - as the workload intensity rises, congestion increases, and the residence time along with it
  - as the service center approaches saturation, small increases in arrival rate result in dramatic increases in residence time.

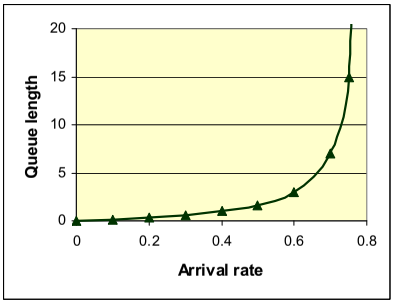
Utilization




Residence Time

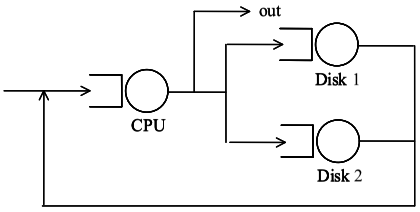


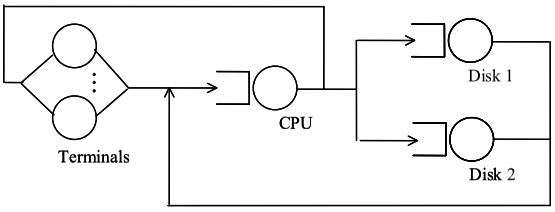

Queue length



# Network of queues

- Open and closed QN models



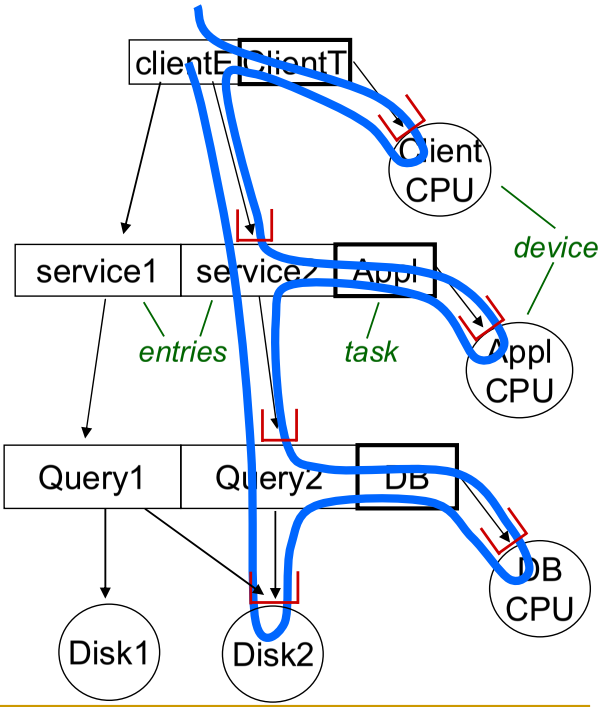

- Additional parameters needed to describe the movement of customers through the network
  - e.g., after a CPU service, a customers moves with given probabilities or visit ratios to Disk1, Disk2 or out
- Multiple customer classes
  - each class has its own workload intensity, service demands and visit ratios
  - combination of open and closed classes is possible
  - the performance results will be obtained by class
- Bottleneck service center - may be different for different classes.



# Layered Queueing Network (LQN) model

http://www.sce.carleton.ca/rads/lqn/lqn-documentation

- LQN is a extension of QN
  - models both software tasks (rectangles) and hardware devices (circles)
  - represents nested services (a server is also a client to other servers)
  - software components have entries corresponding to different services
  - arcs represent service requests (synchronous and asynchronous)
  - multi-servers used to model components with internal concurrency
- What we get from the LQN solver:
  - Service time (mean, variance) including nested services
  - Waiting time
  - Throughput
  - Utilization
  - Probability of missing a deadline (obtained only by simulation)



# Performance Analysis in MARTE

- In MARTE, "performance modeling" describes the analysis of temporal properties of best-effort systems and soft-real-time embedded systems
  - e.g., information processing systems, web-based applications and services, enterprise systems, multimedia, telecommunications.
- Performance measures (analysis outputs) are statistical, such as:
  - mean throughput and delay (response time)
  - mean queue length and queueing delay
  - probability of missing a target response time
  - resource utilization
- Input parameters to the analysis may also be probabilistic, such as:
  - random arrival process,
  - random execution time for an operation
  - probability of a cache hit.
- Common performance analysis techniques include:
  - simulations
  - queueing networks and extended queueing networks
  - discrete-state models such as Stochastic Petri Nets or Stochastic Process Algebra.
  - behavior is often regarded as non-terminating for the purposes of analysis (steady state behavior rather than transient behavior).

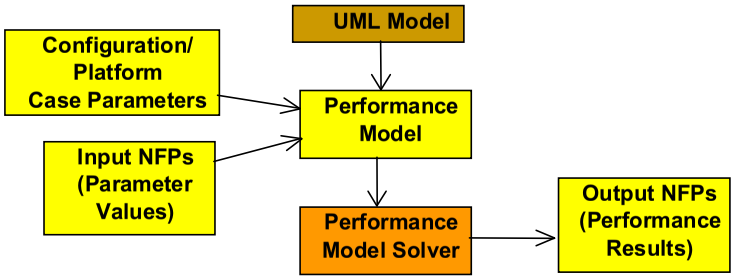


# Performance Analysis Model (PAM) overview

- PAM domain model employs and extends the GQAM domain model.
- Workload:
  - PAM employs features such as the WorkloadEvent description of a stream of arriving events
  - focus on some workload types: open and closed arrivals, workload generators and traces
- Behavior Scenario:
  - PAM uses the behavior-causality model of Scenarios and Steps
  - extends the properties of a Step to include more kinds of operation demands during a step
  - adds the possibility of an asynchronous (non-synchronizing) parallel operation, which is forked but never joins
  - other Step extension: a PassResource step which indicates the passing of a shared resource from one process to another
- Resources:
  - Important resources include hardware ExecutionEngines, concurrent process threads (ScheduledResources), and LogicalResources defined by the software
    - logical resource examples: semaphore, lock, buffer pool, critical section
  - introduces a special concept of RunTimeObjectInstance as an alias for a process or thread pool resource identified in behavior specifications by other entities (such as lifelines and swimlanes)
- Observers: uses the GQAM observer concepts.

# AnalysisContext parameters

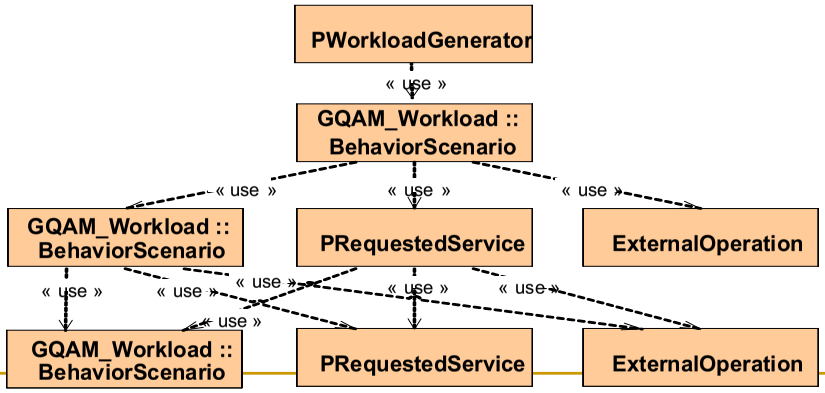
- AnalysisContext (from GQAM) corresponds to the scope of an evaluation.
  - combines the system represented by its behaviour and resources with one or more workloads.
  - gives a set of model parameters for defining the range of variation of cases that will be analysed
- One UML specification may give rise to several performance models, due to variations in system usage, workload, allocation, deployment, and configuration→ different models cases.
- Parameterized NFPs are supported by the use of:
  - variables global to the AnalysisContext
  - variable names in place of numeric values of input properties for model elements.
  - functional dependencies of the input properties on the global variables, to define values.



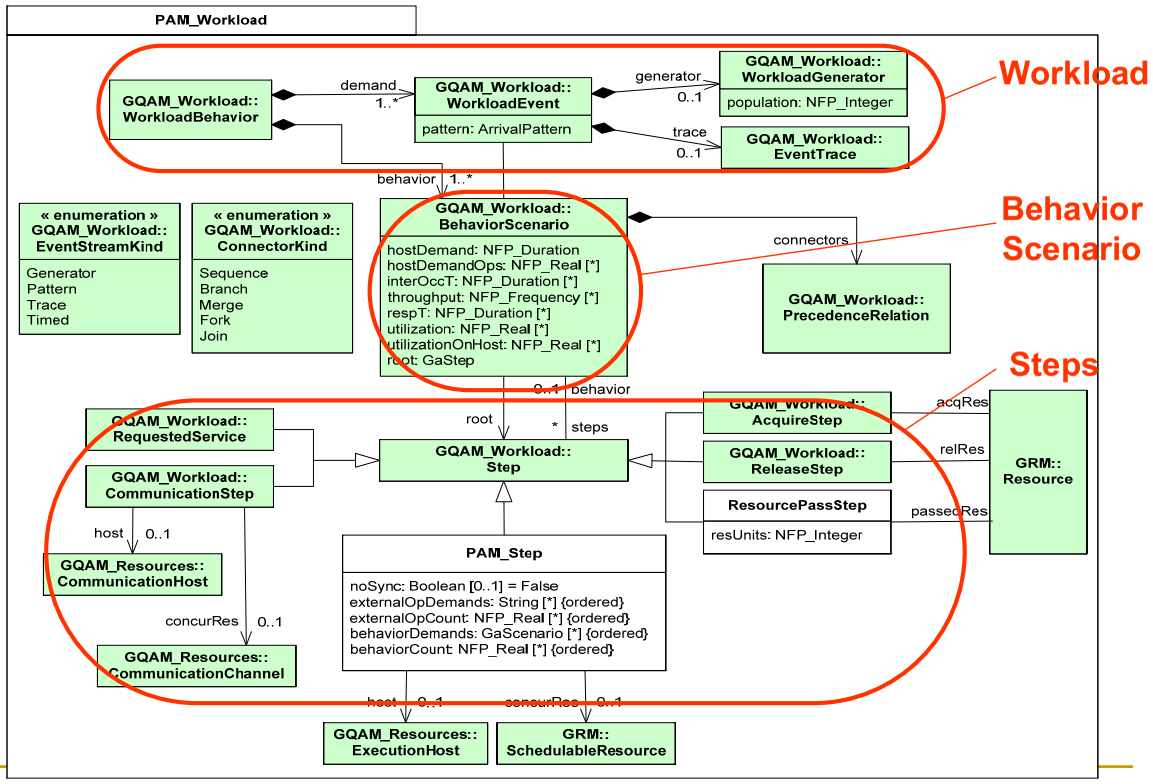



# Informal view of Services and Behavior

- Service is an important concept in performance
  - requests for service may have a required quality of service
  - an actual service is defined by a BehaviorScenario, with a provided quality of service
- Services may be incorporated into the modeled behavior in three ways:
  - by making a serviceDemand from a Step to a RequestedService, representing an operation offered at some interface, which is in turn defined by a BehaviorScenario
  - by making a behaviorDemand from a Step to directly invoke a BehaviorScenario
  - by making an extOpDemand from a Step to request an external service, which is defined in the performance environment outside the UML model.



# PAM Domain Model: Workload and Behavior



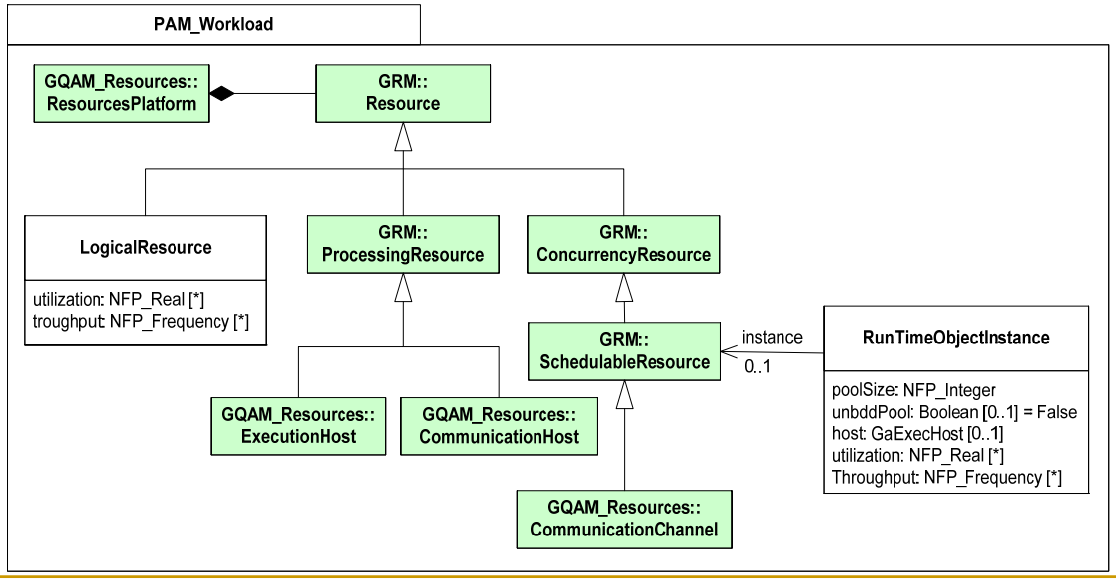



# PAM Step extensions

- External resource demands
  - Resources which are not modeled within the software design may have an impact on performance (e.g., disk access).
  - PAM domain model identifies "external operations" by such resources by a name (a string), so they can be modeled in the performance model.
  - Demands by a Step for external operations are described by the pair of properties externalOpDemand and externalOpCount:
    - externalOpDemand is an ordered list of operation names (strings)
    - externalOpCount is an ordered list (in the same order) of the number of demands made during one execution of the Step (may be an integer, an average value or a probability distribution)
- Direct demands to a BehaviorScenario
  - Demands by a Step directly to a BehaviorScenario are described by the pair of properties behavDemand and behavCount
- “noSync” attribute
  - folowing a fork due to an asynchronous message or par operand, “noSync” explicitly indicate that the respective parallel branch does not join
  - behaviour using noSync may provide increased concurrency and increased performance.

# PAM Domain Model: Resources

- LogicalResource: defined by the software - has to be modeled as a resource in the performance model because it introduces delays
- RunTimeObjectInstance: alias for a process or thread pool resource identified in behavior specifications by lifelines and swimlanes






# Communication Channel

- A message between two objects is conveyed by some mechanism:
  - if the objects are in the same execution thread (process), it is conveyed by the language runtime
  - if the objects are in different execution threads (processes) in the same node (ProcessingHost) it is conveyed by the operating system
  - if the objects are on different nodes, it is conveyed by a system layer named here CommChannel, which may be:
    - a middlware layer (CORBA connection, Java Remote Method Invocation, web services connection, etc.)
    - a Message-Passing Interface connection in a grid
    - a socket or secure socket connection
    - a more complex infrastructure such as a publish-and-subscribe system.

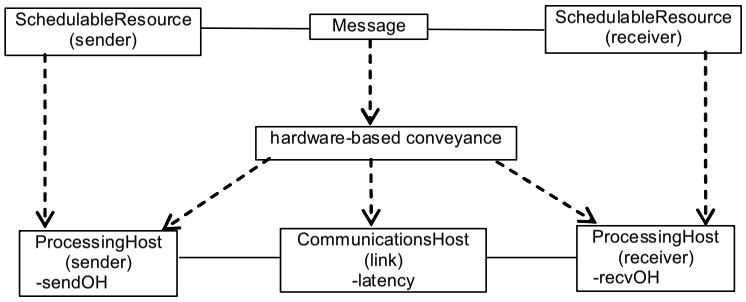
# Modeling Communication Channels

1. Within the same node, language runtime and operating system costs are considered part of the scenario.
   - the inter-process communication cost per byte of the host may be used to calculate the hostDemand for communication.
2. Between nodes the default is:
   - determine node hostDemands from the sending and receiving overhead on the nodes
   - insert the latency of the link (an attribute of the connecting CommunicationsHost)
3. Between nodes the conveyance of the message may also be modeled as an external operation, invoking a submodel of the communications layer
   - this may be an attractive option for modeling the behaviour of the Internet
4. Between nodes a communications layer such as CORBA may be defined as a UML StructuredClass offering send and receive operations to the two end-point processes
   - such a layer is denoted as a CommChannel with a conveyance operation demanded by the CommunicationsStep in the scenario
   - its service is defined by a BehaviorScenario, which may involve directory look-ups, authorization and redirection of requests
5. Between nodes a complex communications protocol can be modeled by a pure BehaviorScenario not associated with a system component, but describing a collaboration of the hosts.

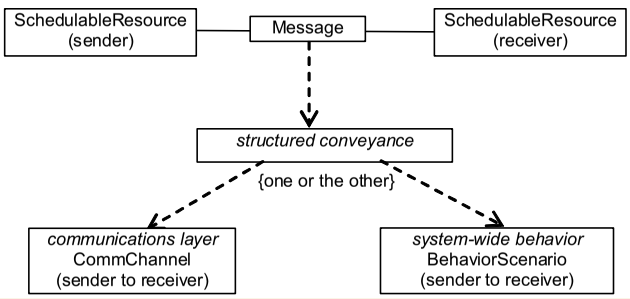


# Communication channels at different detail levels

Detail Level 2, conveyance modeled at the hardware level

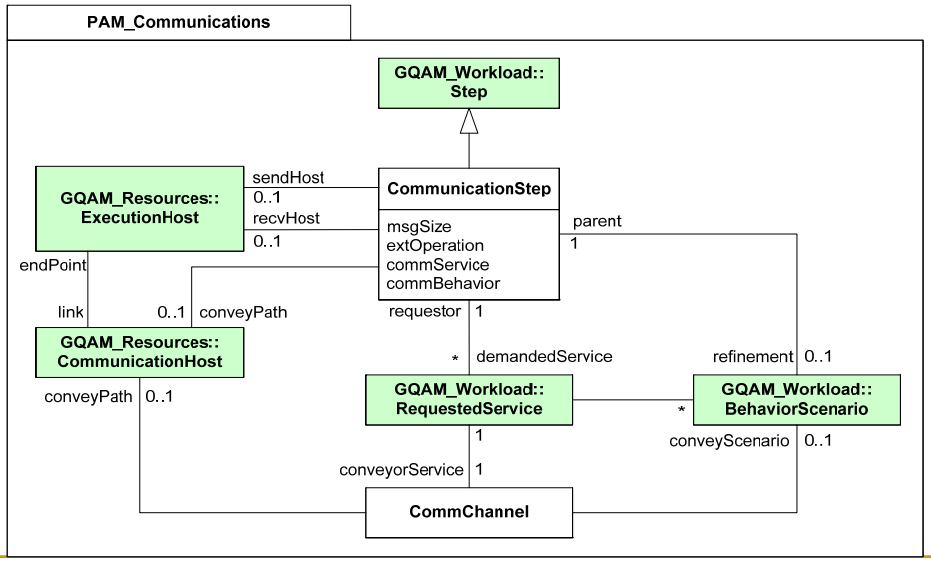



Detail levels 4 and 5: using a communications layer



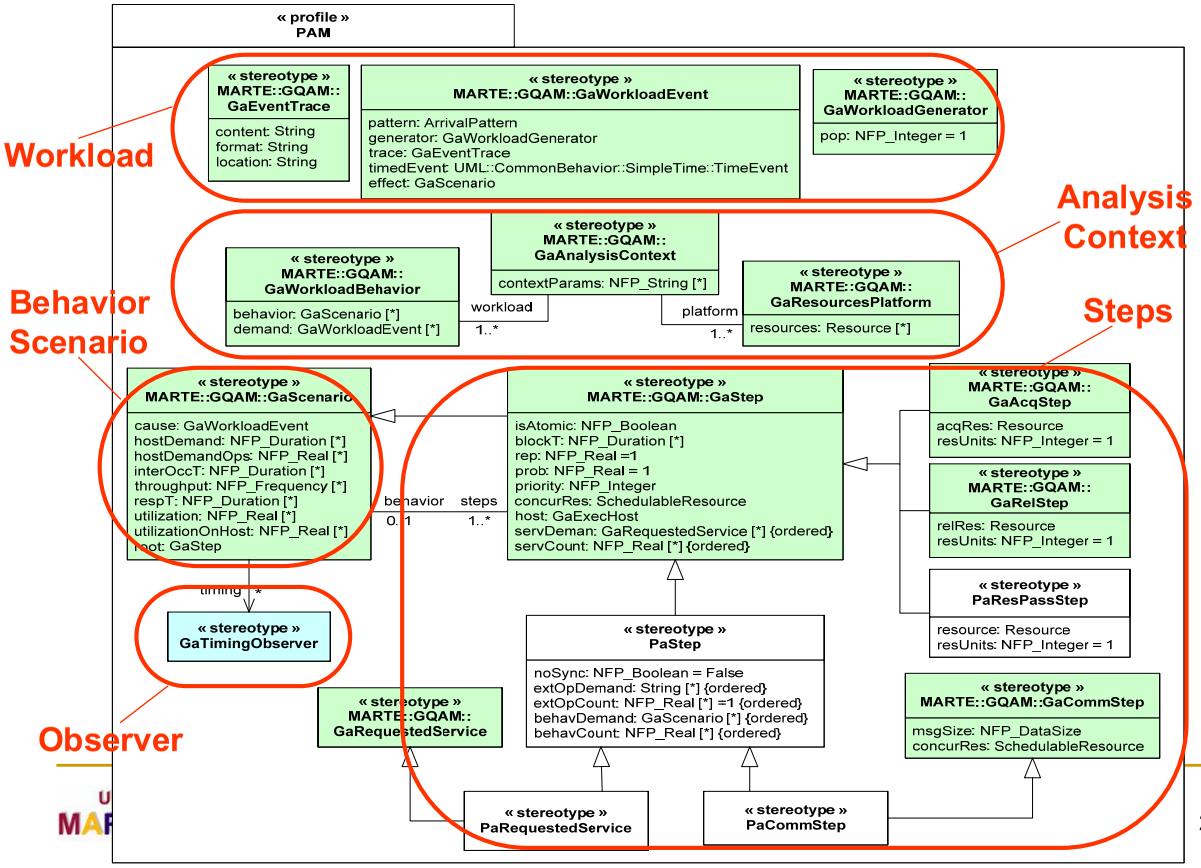
# PAM Domain Models: Communication

- The domain model to support communications modeling contains:
  - CommChannel: mechanism for conveying messages
  - CommunicationStep: specialization of Step handling message communication

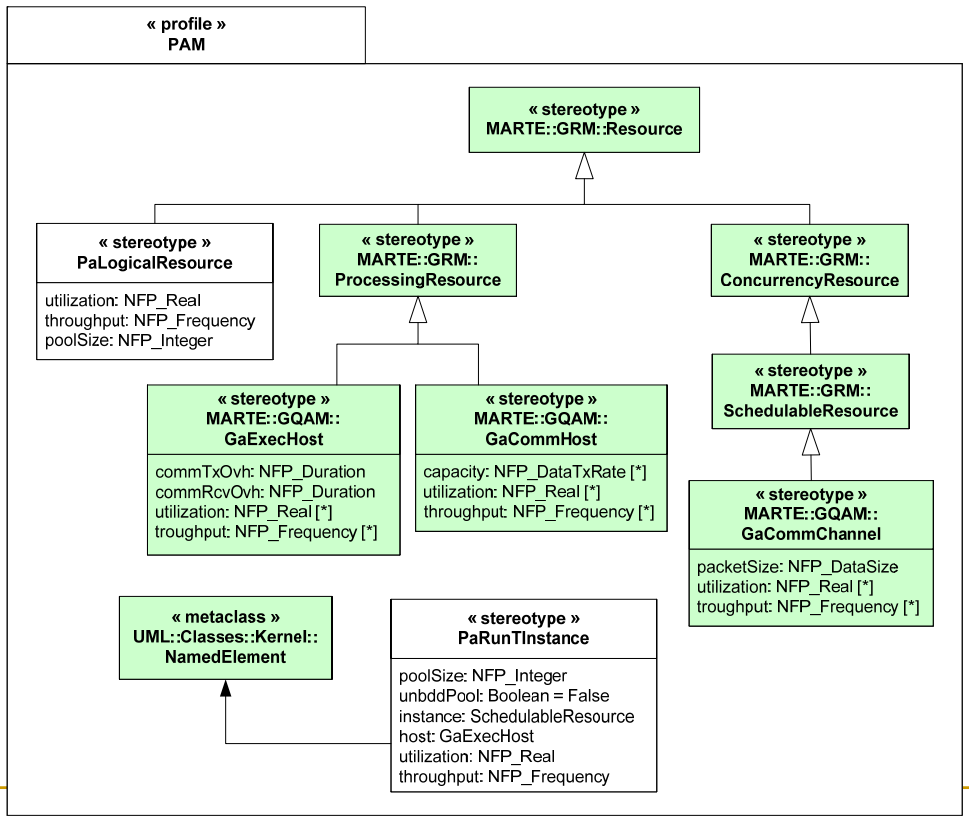





# PAM Profile: Workload, Behavior, Observer

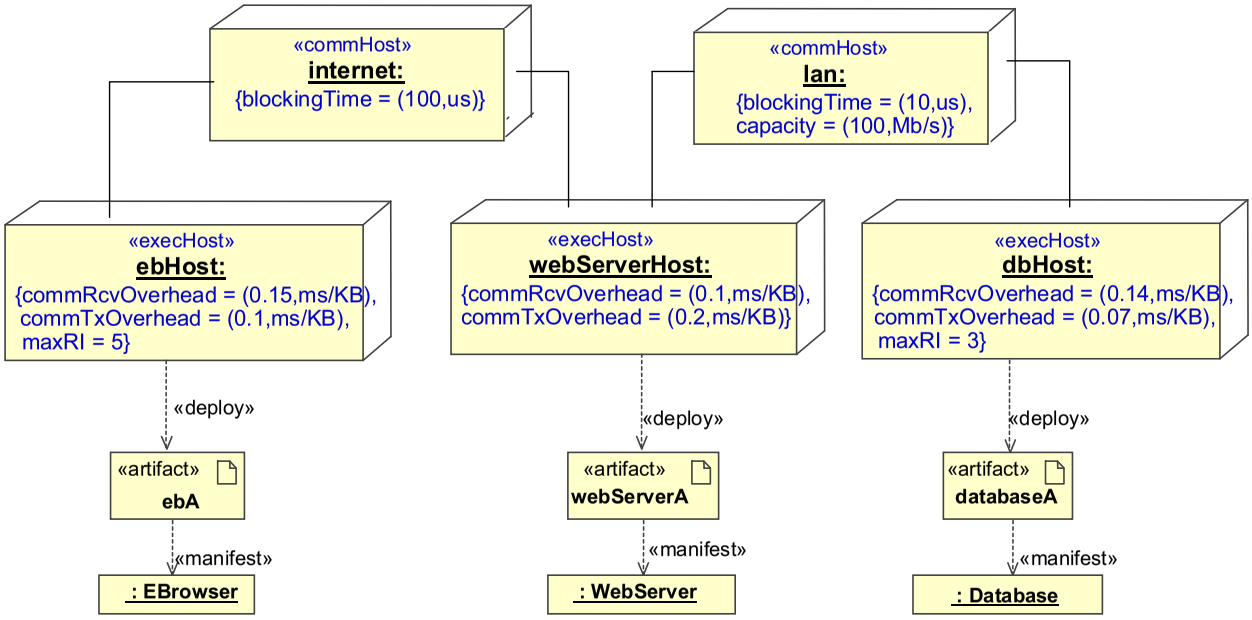


# PAM Profile: Resources

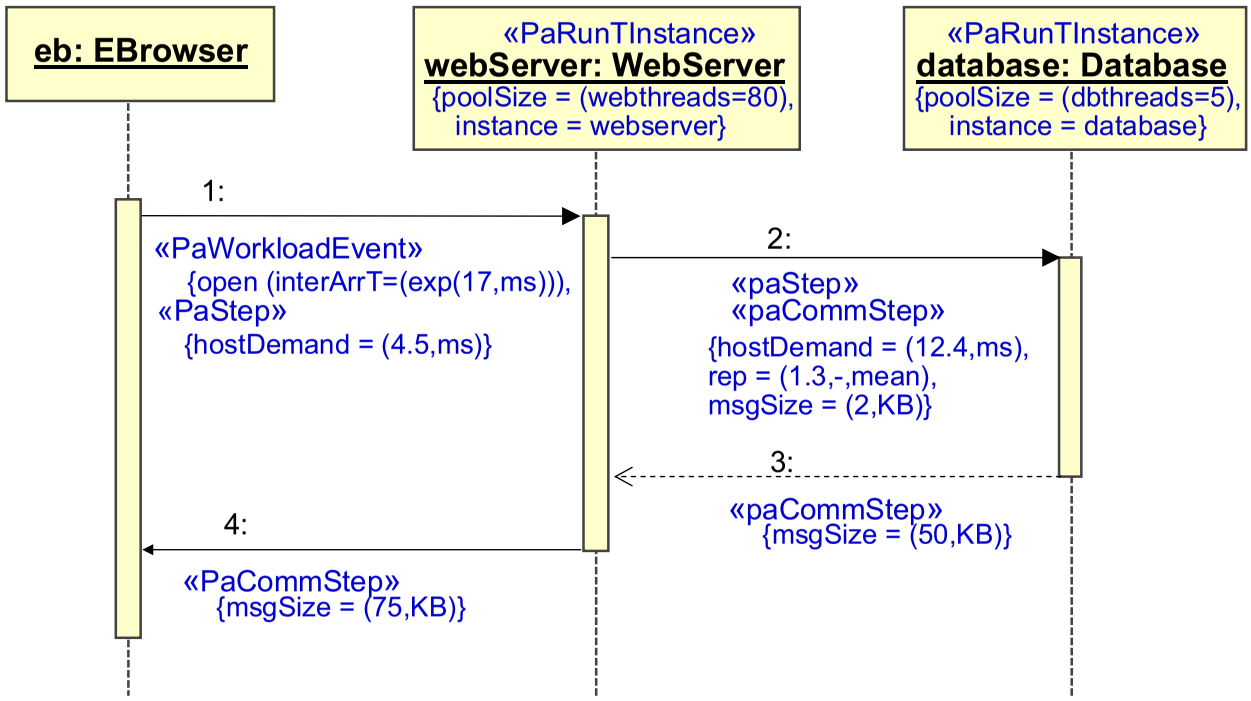





# Example 1: TCP-W deployment

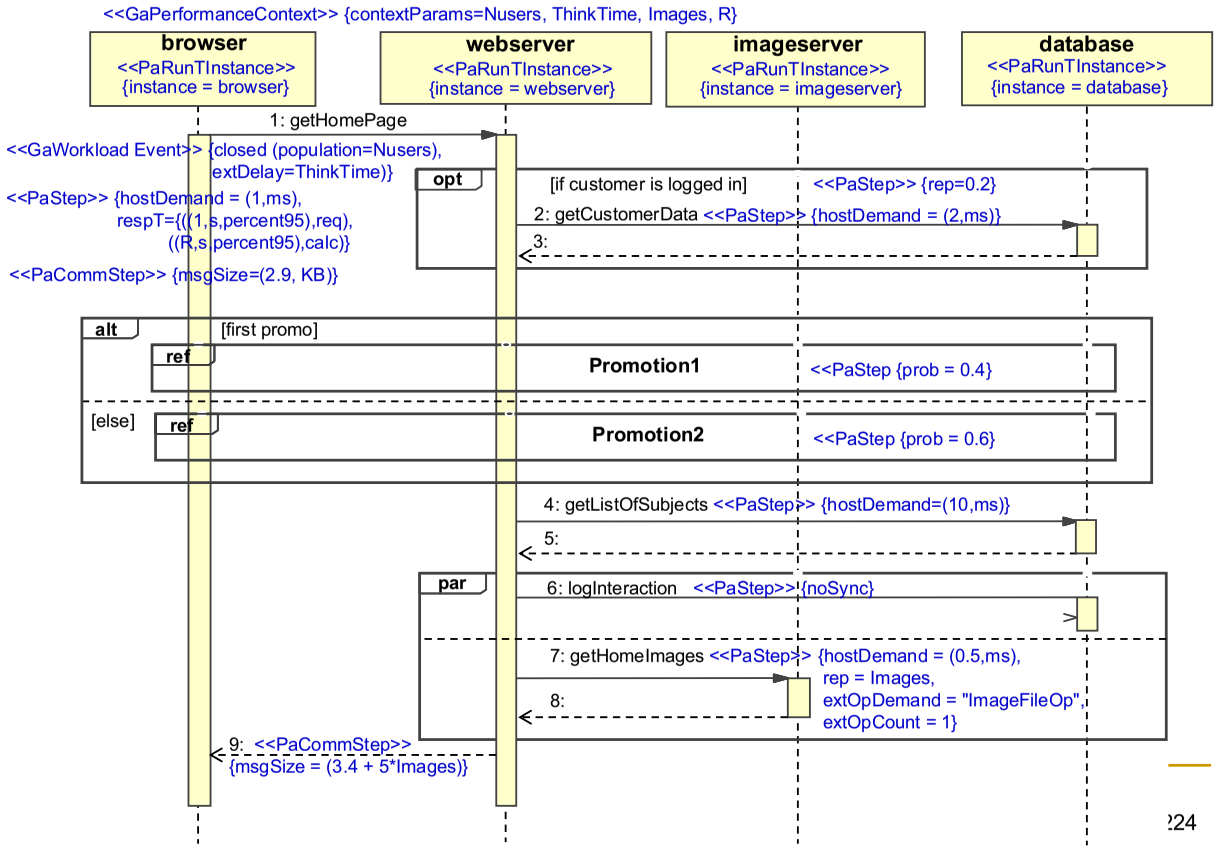


# Example 1: simple scenario

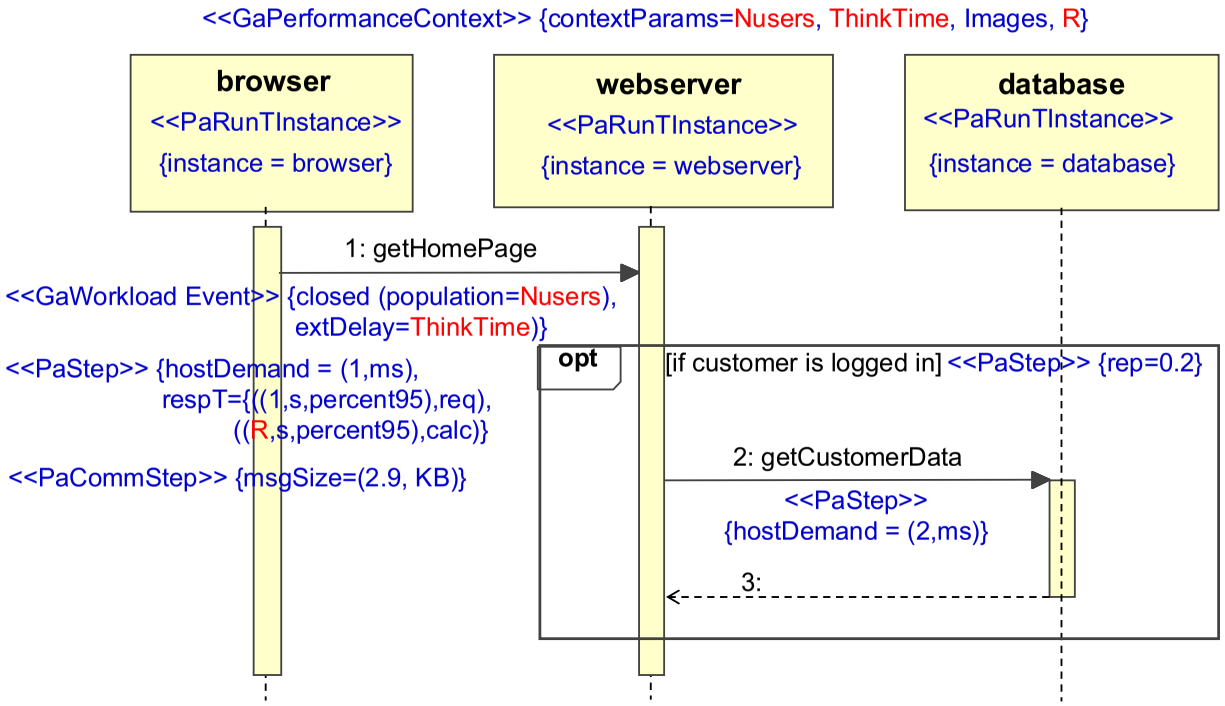





# Example 1: GetHomePage scenario

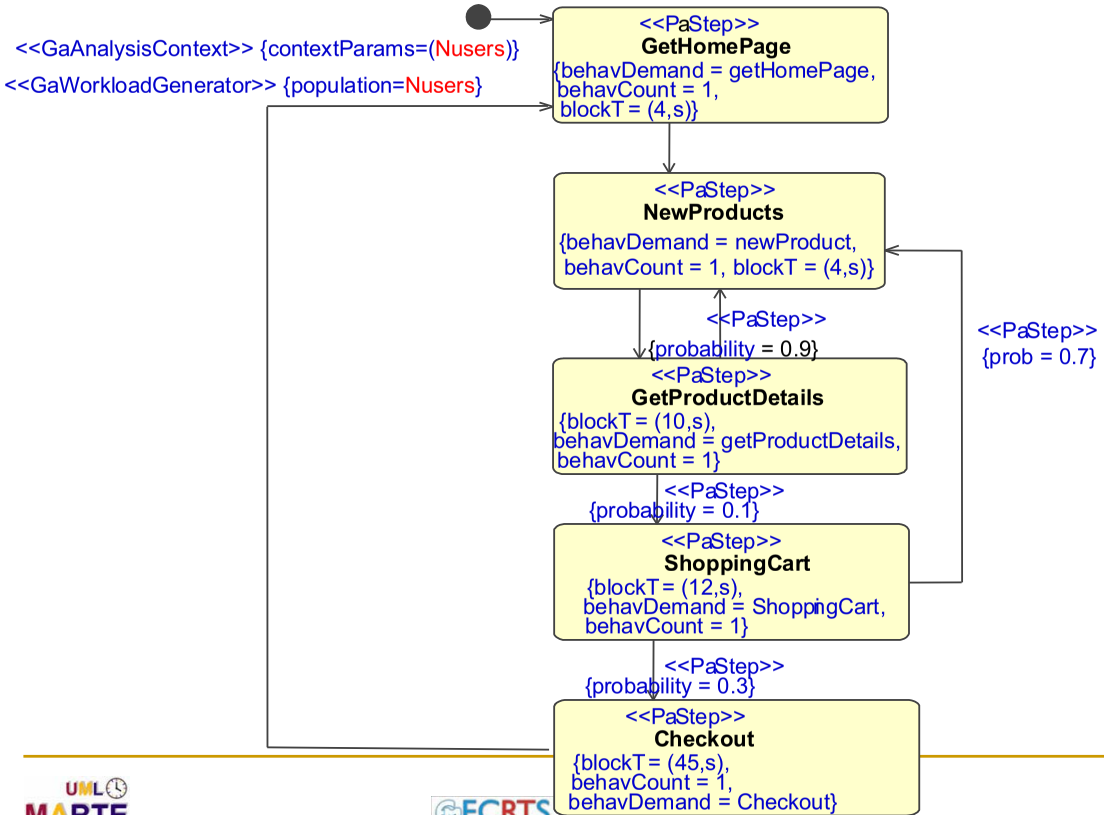


# Example 1: GetHomePage scenario (fragment)



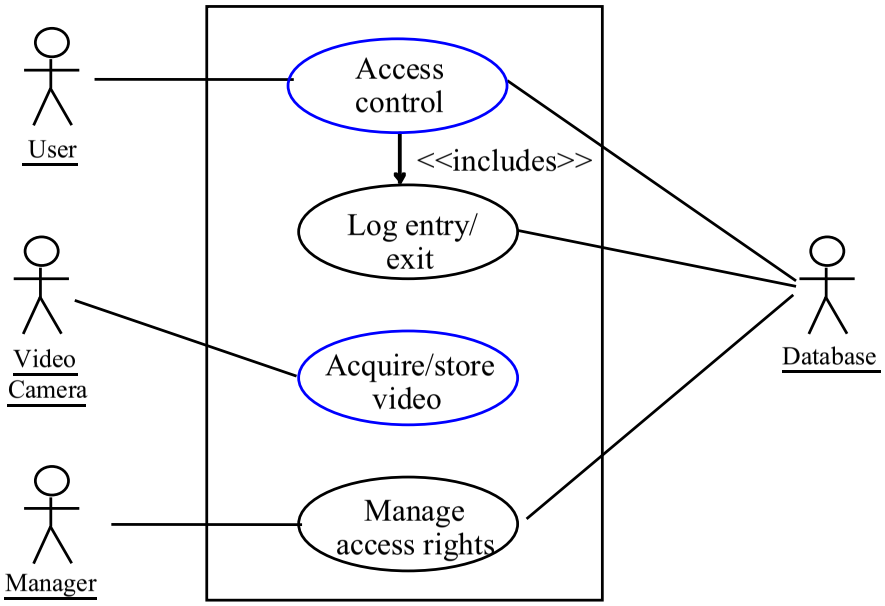



# Workload generator for TPC-W



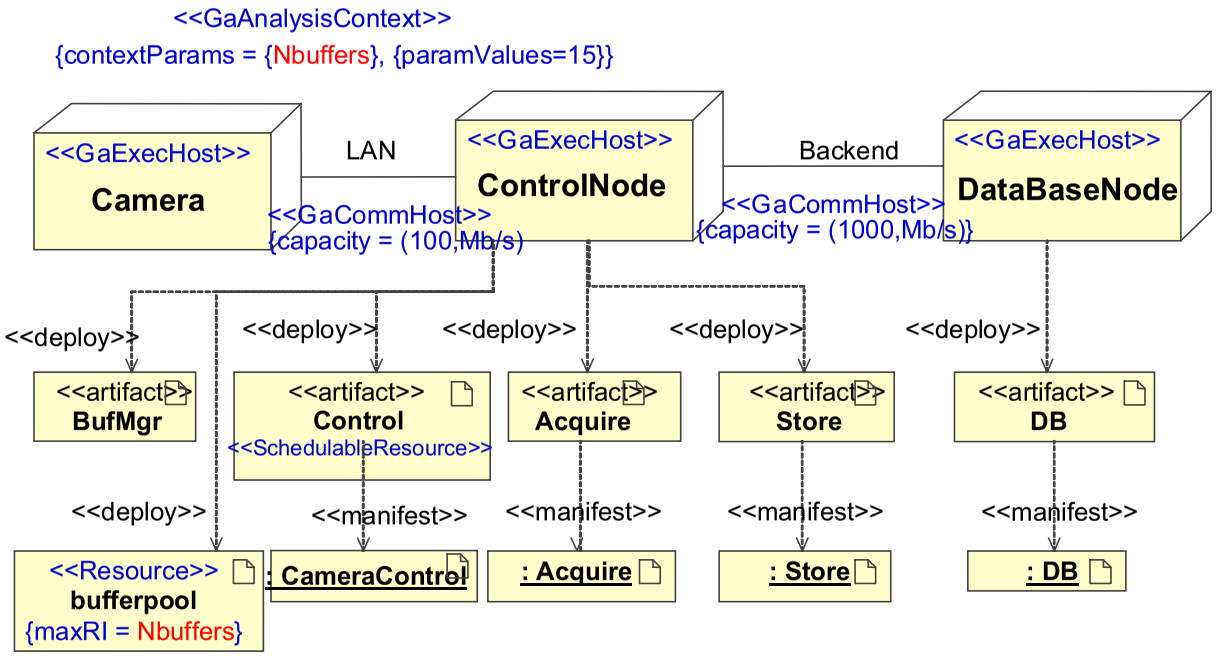
# Example 2: Building Surveillance System

- Two frequently executed scenarios are chosen for performance analysis

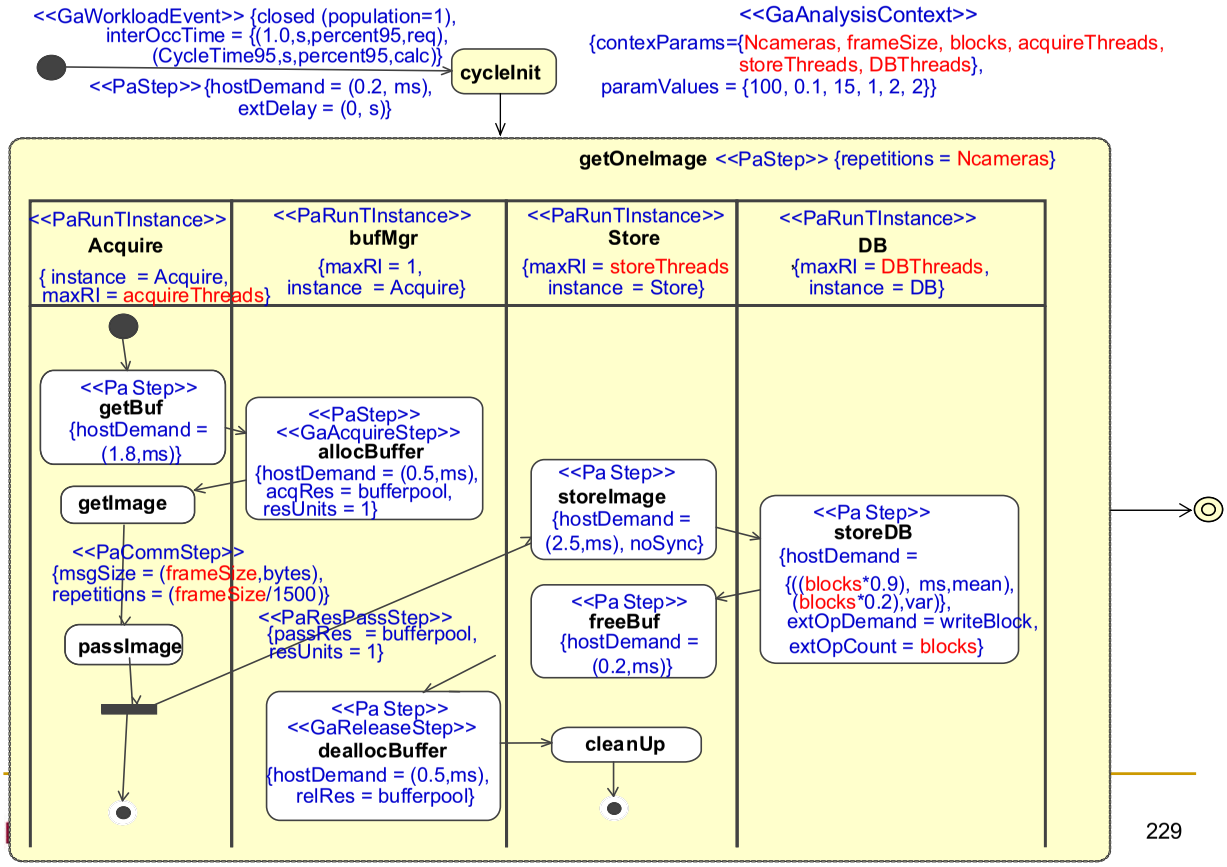





# Building Surveillance System: deployment



# Building Surveillance System: AcquireVideo scenario






# From UML+MARTE to performance models

- What is needed for performance analysis in a UML model extended with MARTE annotations:
  - key use cases described by representative scenarios
    - frequently executed, with performance constraints
  - resources used by each scenario
    - resource types: active or passive, physical or logical, hardware or software
    - quantitative resource demands for each scenario step
      - how much, how many times?
  - workload intensity for each scenario
    - open workload: arrival rate of requests for the scenario
    - closed workload: number of simultaneous users

# Direct UML to LQN Transformation: our first approach

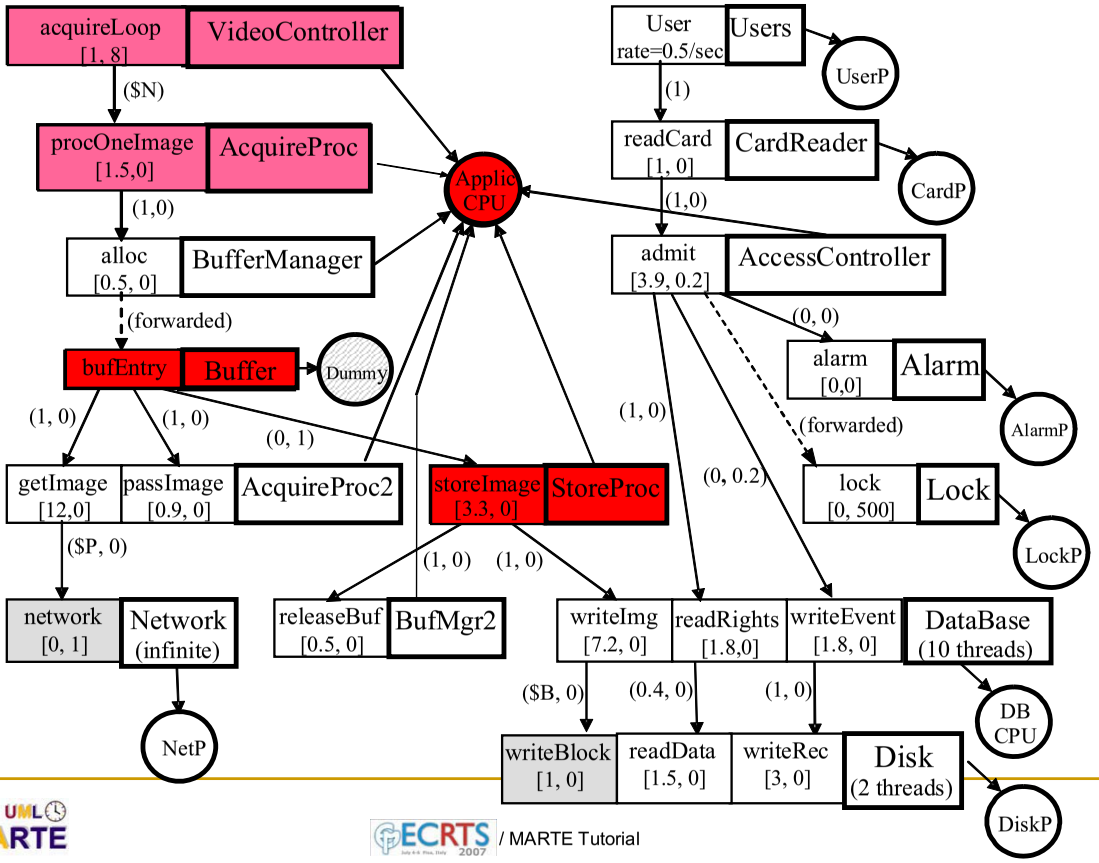
- Generate LQN model structure (tasks, devices and their inter-connections) from:
  - UML model of the high-level software architecture
  - UML deployment diagram
- Generate LQN detailed elements (entries, phases, activities and their parameters) from:
  - UML models of key scenarios with performance annotations
  - Scenarios can be represented in UML by the software designers as:
    - interaction diagrams
    - activity diagrams



# UML → LQN Transformation Algorithm

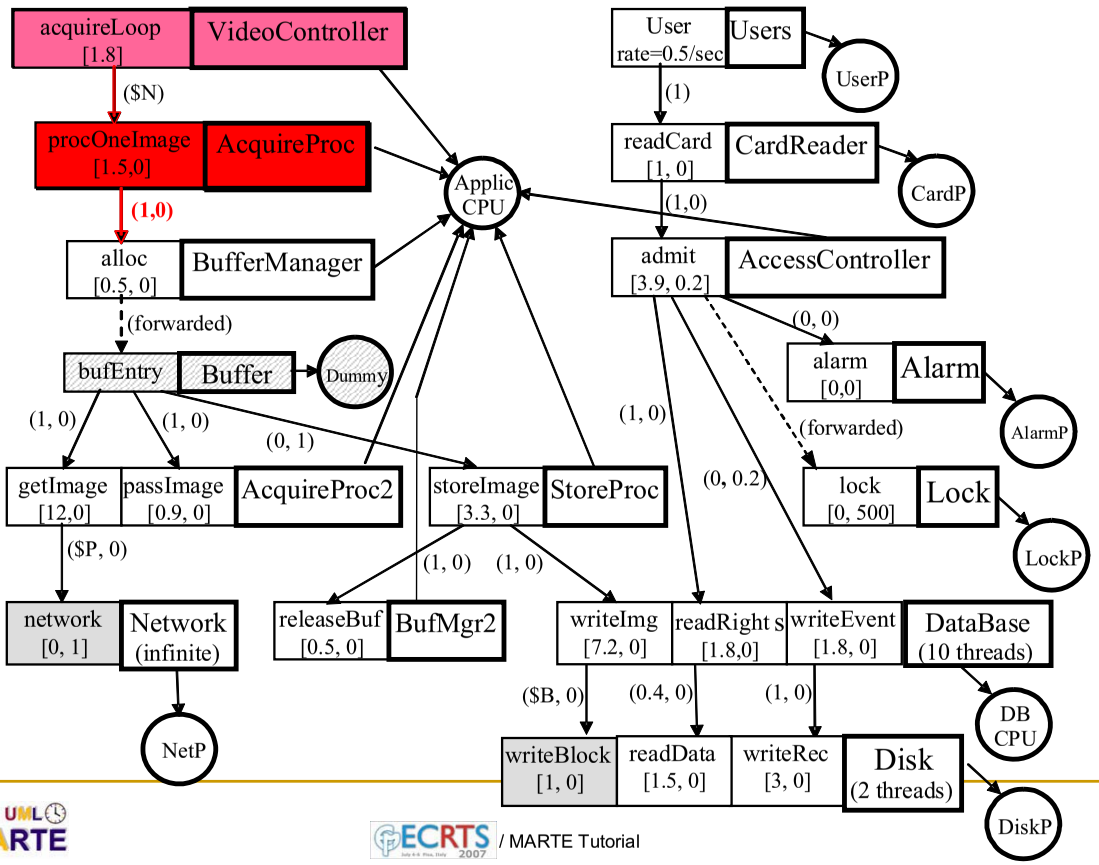
1. Generate the LQN model structure
   - 1.1. determine LQN software tasks from high-level components
   - 1.2. determine LQN hardware devices from deployment diagram
2. Generate LQN entries, phases, activities from scenarios
   - 2.1. for each scenario, process the activity (interaction) diagram(s)
     - 2.1.1. match inter-component communication style from pattern with messages between components from the activity diagram
     - 2.1.2. divide the activity diagram into sub-graphs corresponding to different LQN entries, phases, and activities; create the respective LQN elements and compute their parameters;
   - 2.2. merge the LQN submodels corresponding to different scenarios;
3. Traverse the LQN graph and write out textual model description.

# LQN model of a Building Surveillance System






# LQN model of a Building Surveillance System



# Using the LQN Model for Improvements

- Base case system capacity: 20 cameras
  - problem: *software bottleneck* at the buffers
- Adding software resources:
  - 4 Buffers and 2 StoreProc threads
  - result: 40 cameras supported, performance improvement 100%
  - next problem: *hardware bottleneck* at the processor
- Replicating the processor:
  - Dual Application CPU
  - result: 50 cameras supported, performance improvement 150%
- Increasing software concurrency level
  - use asynchronous messages – raise concurrency level
  - result: 100 cameras supported, performance improvement 400%



# Agenda

**09:00 am to 10:15 am**
- Introduction to MDD for RT/E systems
- MARTE foundations

*10:15 am to 10:45 am: Break*

**10:45 am to 12:00 am**
- High-level modeling constructs
- Detailed software and hardware platforms modeling

*13:00 pm to 14:30 pm: Lunch*

**14:30 pm to 15:00 pm**
- Introduction to model-based RTE analysis

**15:00 pm to 15:45 pm**
- Model-based schedulability analysis

*15:45 pm to 16:15 pm: Break*

**16:15 pm to 17:00 pm**
- Model-based performance analysis

**17:00 pm to 17:30 pm**
- Conclusions and perspectives about MARTE

# 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!**



# Questions ?

- www.martes.org
  - Workshop on model-based design and validation for RTES
  - Held in co,njunction with trhe Models 2007 conference
- www.promarte.org
  - The MARTE web site
- www.papyrusuml.org
  - On open source Eclipse plug-in for UML2 graphical modeling
  - MARTE implementation available end of July within the V1.7 relalease 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/...