

# Applying Model Driven Engineering to RTES: Technologies, Standards and Experiences

TiMoBD Workshop - Embedded Systems Week  
Oct 9-14 2011, Taipei, Taiwan

**PhD. Sara Tucci-Piergiovanni**  
CEA LIST Laboratory of model driven  
engineering  
for embedded systems (LISE)

 [Sara.Tucci@cea.fr](mailto:Sara.Tucci@cea.fr)

**Introduction to RTES and Model-Driven Engineering**

**Focus on the OMG Standard Language MARTE**

**Experience with MARTE in the European Project  
INTERESTED**

**Conclusion**

# Introduction

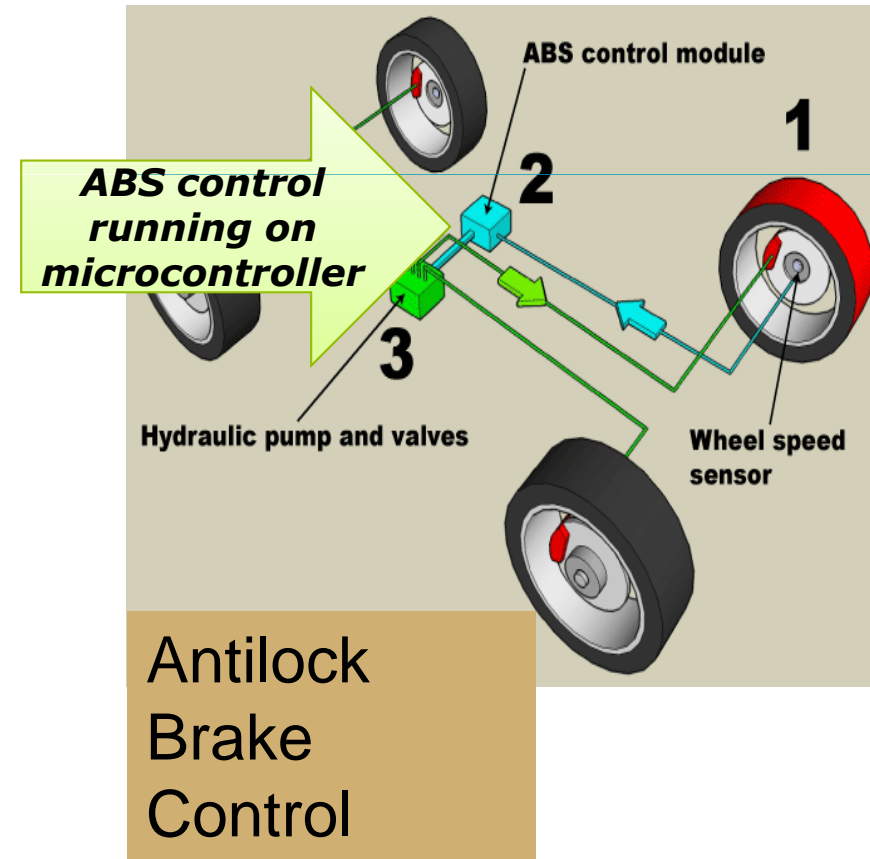
In 1969, the Japanese engineer Tetsuro Mori defined the term 'mechatronic system' (MS), for a system whose main functionality is:

- *control and adaptation to complex real-world phenomena*
- *implemented through the interaction of software, electronic, electric and mechanical parts or sub-systems*

The 'magic mixture' was defined to deliver sophisticated functionality to clients, keeping production profitable

In MS, the control logics, subject to stringent timing and safety constraints, is *implemented in software* then *deployed on some dedicated hardware*

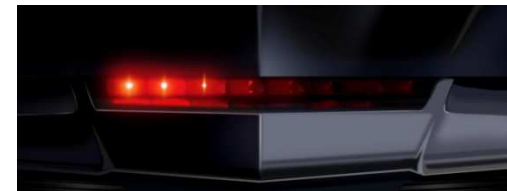
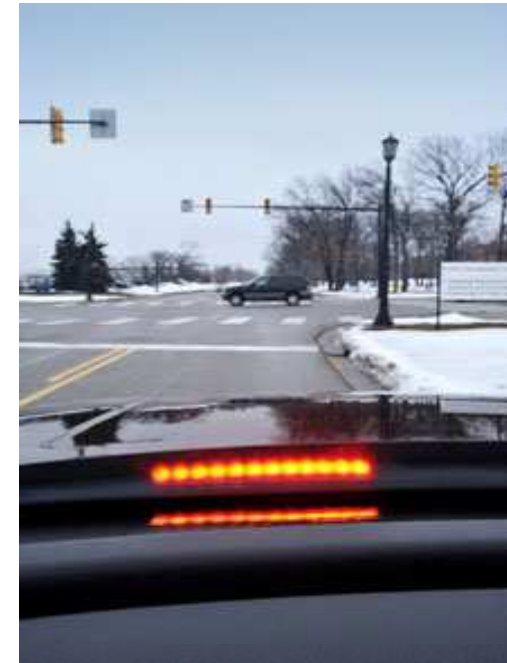
The control part of the system is a *Real-Time Embedded (sub) System*



Upcoming MS capabilities require a massive introduction of complex RTE sub-systems.

Volvo vision for 2020 *'Cars that talk to one another, that recognize road signs, predict crashes; prevent them and prepare for the worst case scenario'*

**Next RTES:** Advanced controls, peer-to-peer communication, and sophisticated sensing capabilities; running on distributed networks of controllers

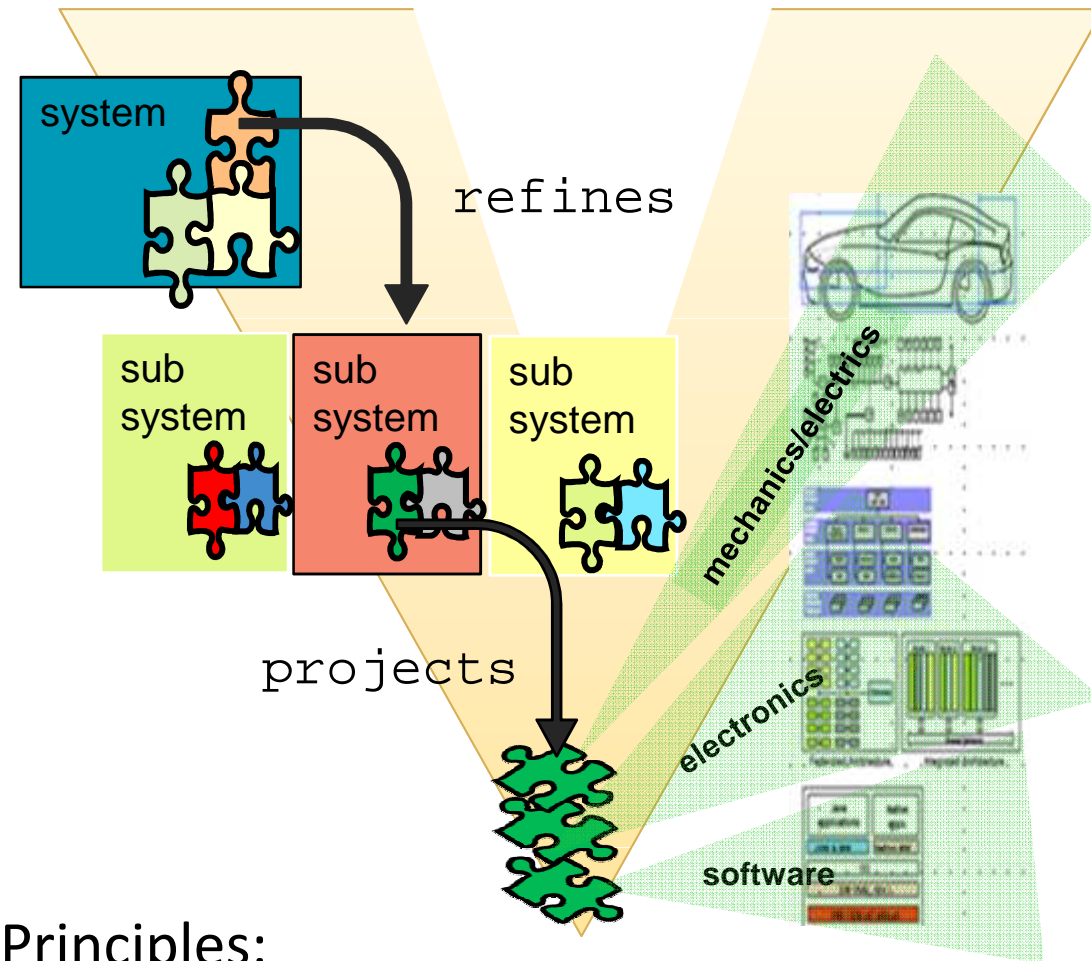


# MS development in industry today

A **rigorous** and **time-effective** development process is a key factor to produce the right system, in the right way, at the right time

A fundamental step in this direction is the introduction of so-called architecture frameworks (e.g. DoDAF, MODAF, UPDM, TOGAF).

**Holistic view** of the **system** inside the enterprise and in its eco-system specified through a process and associated architecture views.



Principles:

1. Decomposition
2. Abstraction
3. Multiple view points

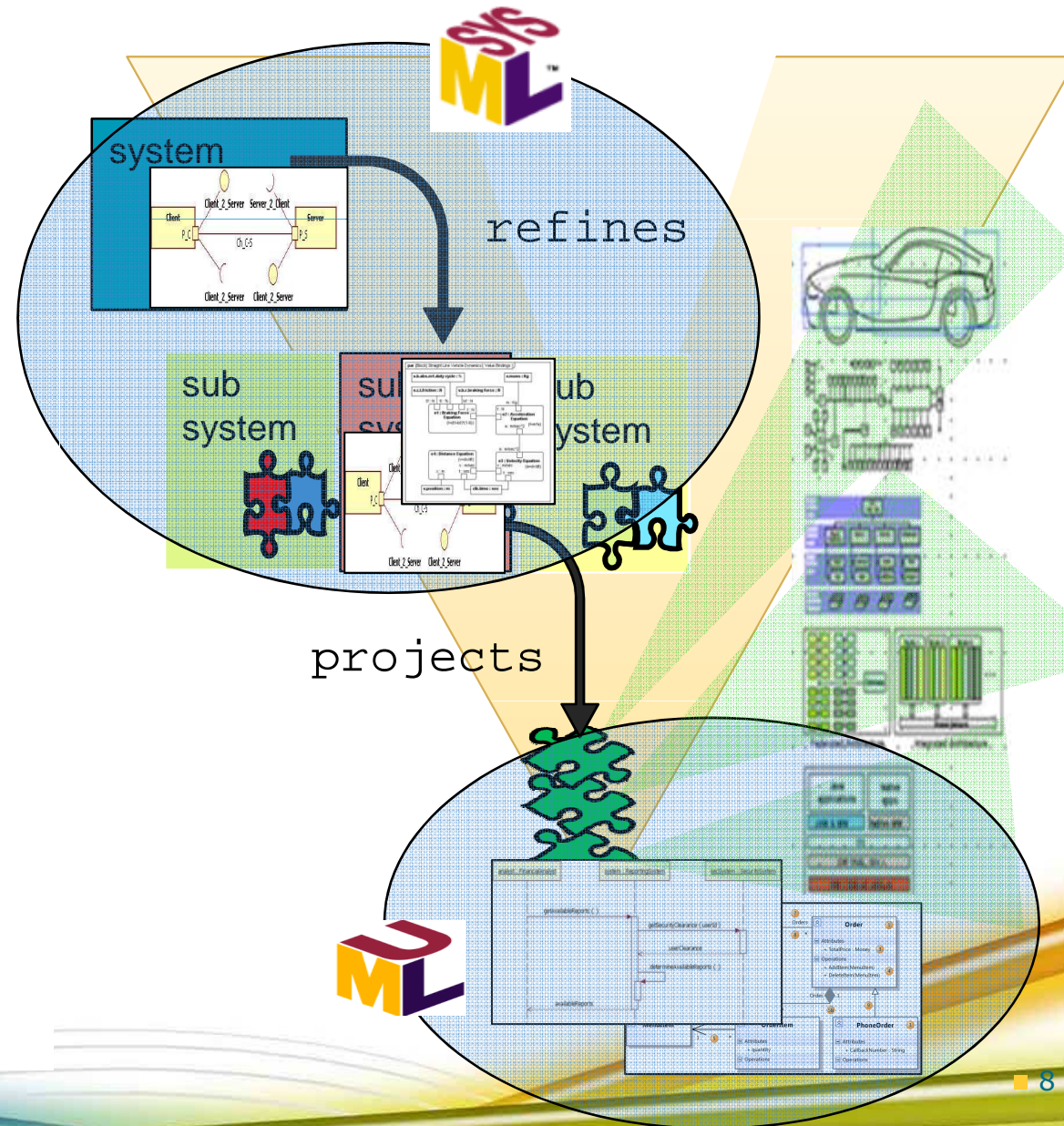
Architecture-based development has opened the door to the use of **modeling languages**, as:

a modeling language (e.g. SysML, UML) has the capability of

- *expressing the concepts of architecture, decomposition, abstraction and view*
- *establishing explicit relationships between elements at different abstraction levels and projected in different views*



**Model is holistic**

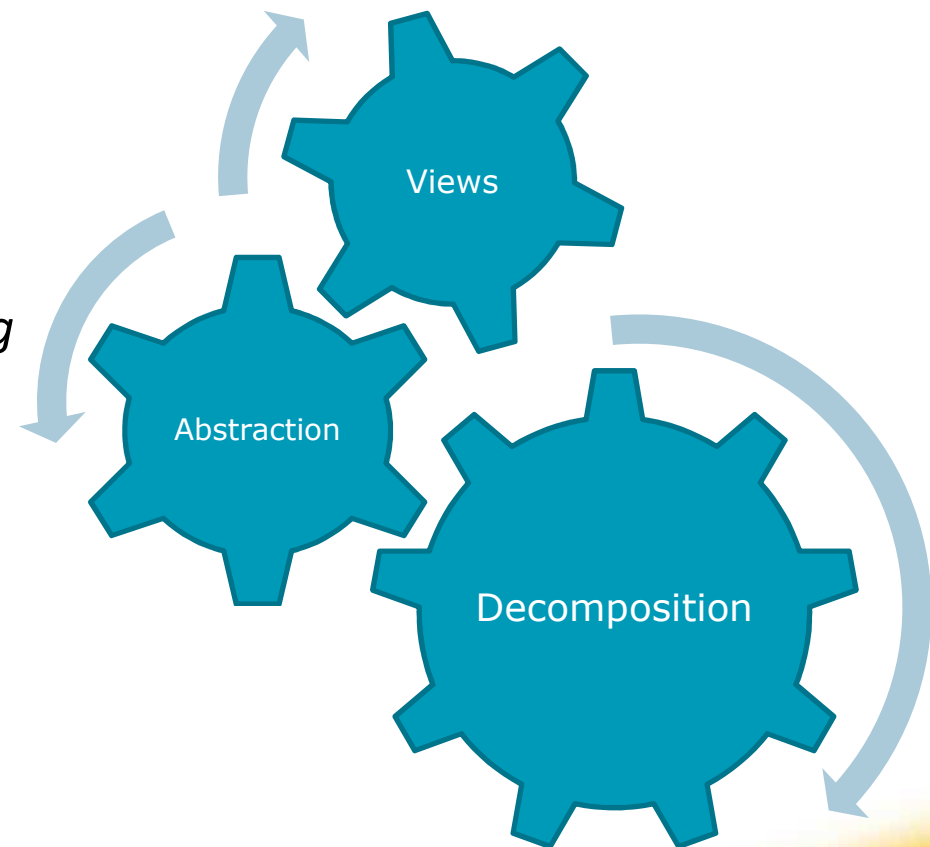




But how to **manage** (create, modify, cancel) **formal relationships in a holistic model?**  
**How to keep the coherence?**

**The model-driven engineering (MDE)** discipline, focuses on this problem. It mainly studies:

- 1. model transformations towards successive refinements under the same language, managing formal relationships between elements at different abstraction levels*  
-- SEMANTICS COHERENCE is OK--
- 1. meta-models mappings, managing formal relationships between models expressed in different languages*  
-- SEMANTIC COHERENCE is THE ISSUE-

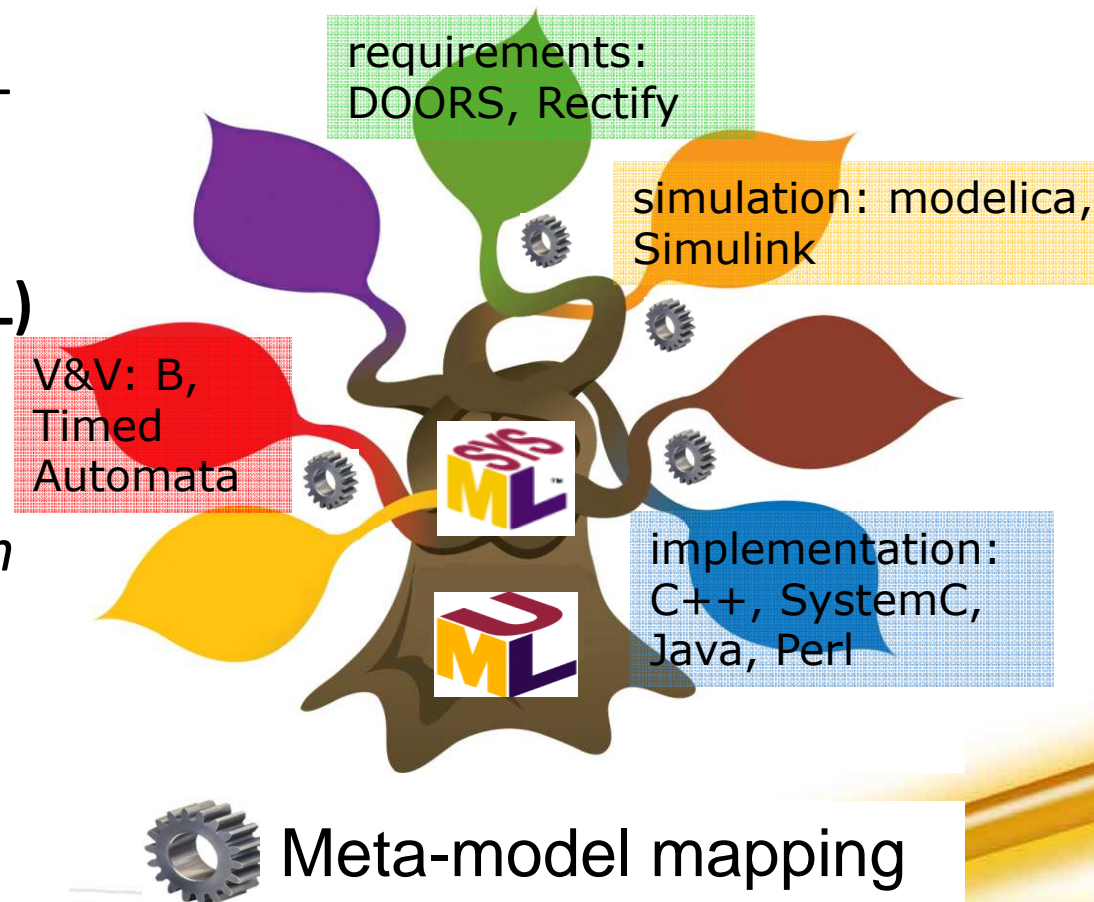


Today, **MDE is a sufficiently mature technology**, offering methods, techniques, standards and tools to **concretely implement** a full architecture-based development

**Standard** modeling languages as SysML and UML are the **backbone of the architecture-based development**, but **domain specific languages (DSL)** may **co-habit** especially for specific view points.

*Views connect to the backbone through meta-model mappings.*

*A backbone language 'includes' the semantics of connected DSLs*



## **Standards reduce the heterogeneity, improving:**

education of engineers

communication/exchanges between various stakeholders involved in  
process developments

interoperability

## **But standards enable also vendor independence**

Users have a choice of different vendors (no vendor “tie-in”)

Forces vendors into competing and improving their products

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

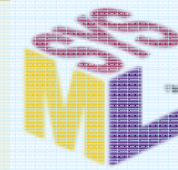
A comprehensive set of standards in support of MDE including standard modeling languages as **UML2 and SysML**.

RTES development focuses on software running on dedicated hardware

entry point: **System-level component interfaces**

to refine towards an **Application Software Architecture Model ...**

...to refine and refine, and to end-up with **code**



## System level

System scope and requirement analysis  
Functional and technical architecture  
(definition of component interfaces)

## Software Level

Application software architecture  
Implementation Architecture (target Platform level)  
Code

## **Key capability for RTES modeling**

**languages** is the power of expressing:

- Real-time design paradigms (e.g. data flows, active objects)
- Detailed Platforms and Allocations
- Models for Performance  
/Schedulability Prediction/Simulation

## **UML is not tailored to fit this capability**

At the same time in RTES domain too many specific approaches, languages and tools...

Sometimes redundant, but often complementary

**Few capabilities of interoperability**



**Necessity of a standard integrator language as UML, but specific to RTES**

MARTE is a **OMG standard UML profile** with the following goals:

To be the universal translator towards domain-specific languages in RTES and provide tools interoperability

To be a modelling language establishing relationships among elements at each level of abstraction in the development process

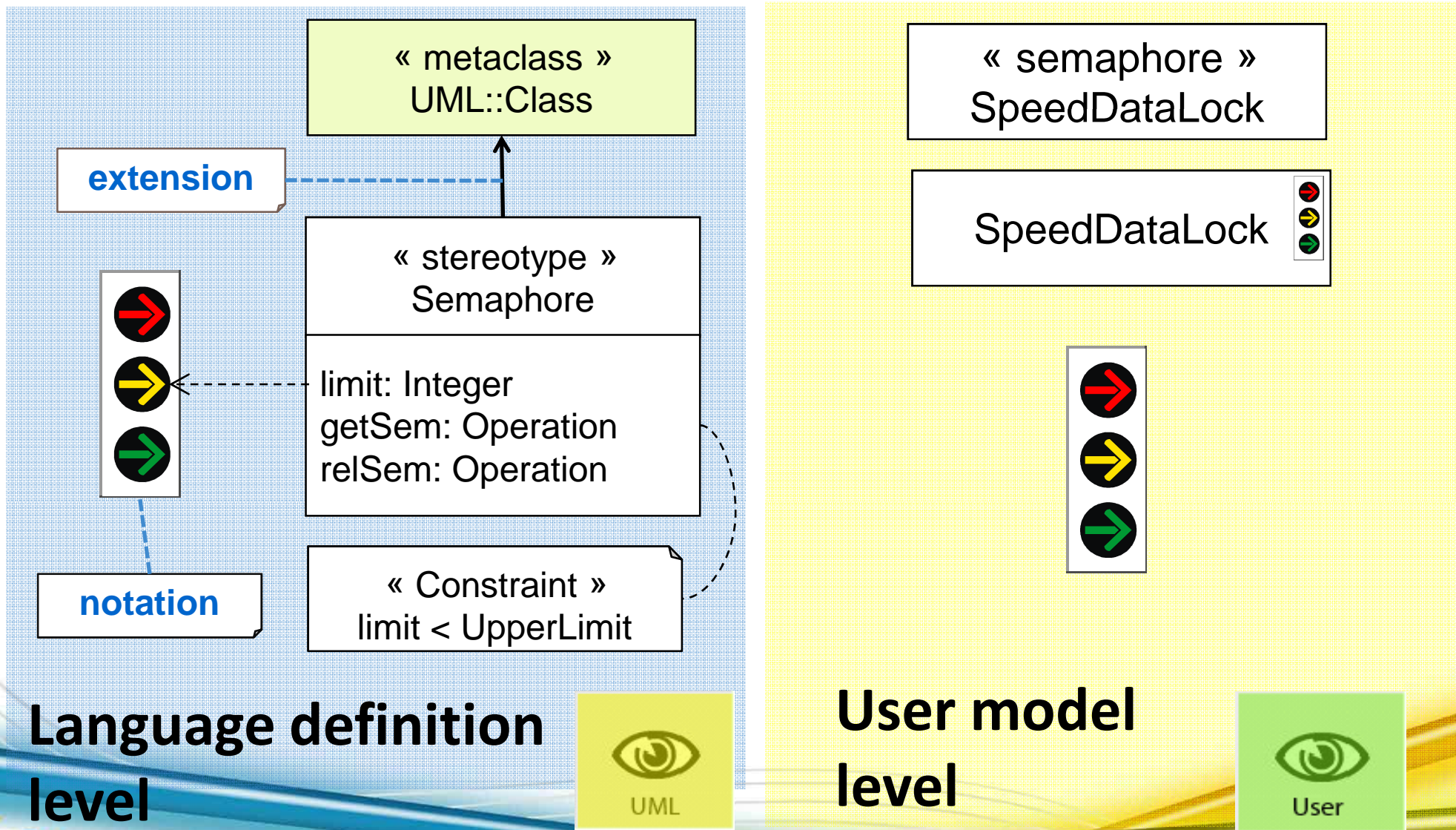
To ease capitalising, exchanging expertise



# A pleasant stroll through the MARTE specification

~720 pages – the first version finalised end 2009 after 4 years of work 😊

*“A special kind of package containing stereotypes, modeling rules and model libraries that, in conjunction with the UML meta-model, define a group of domain-specific concepts and relationships.”*



**Language definition level**



**User model level**





# The three pillars of MARTE

Pillar 1: Platform Independent Modeling

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, based on active objects*

Pillar 2: Detailed Resource Modeling

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

Pillar 3: Annotations for quantitative analysis

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

**Value Specification Language:** *textual language for specifying algebraic expressions*

# The three pillars of MARTE

Pillar 1: Platform Independent Modeling

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, based on active objects*

Pillar 2: Detailed Resource Modeling

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

Pillar 3: Annotations for quantitative analysis

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

**Value Specification Language:** *textual language for specifying algebraic expressions*

# MARTE Generic Component Model

## General component model

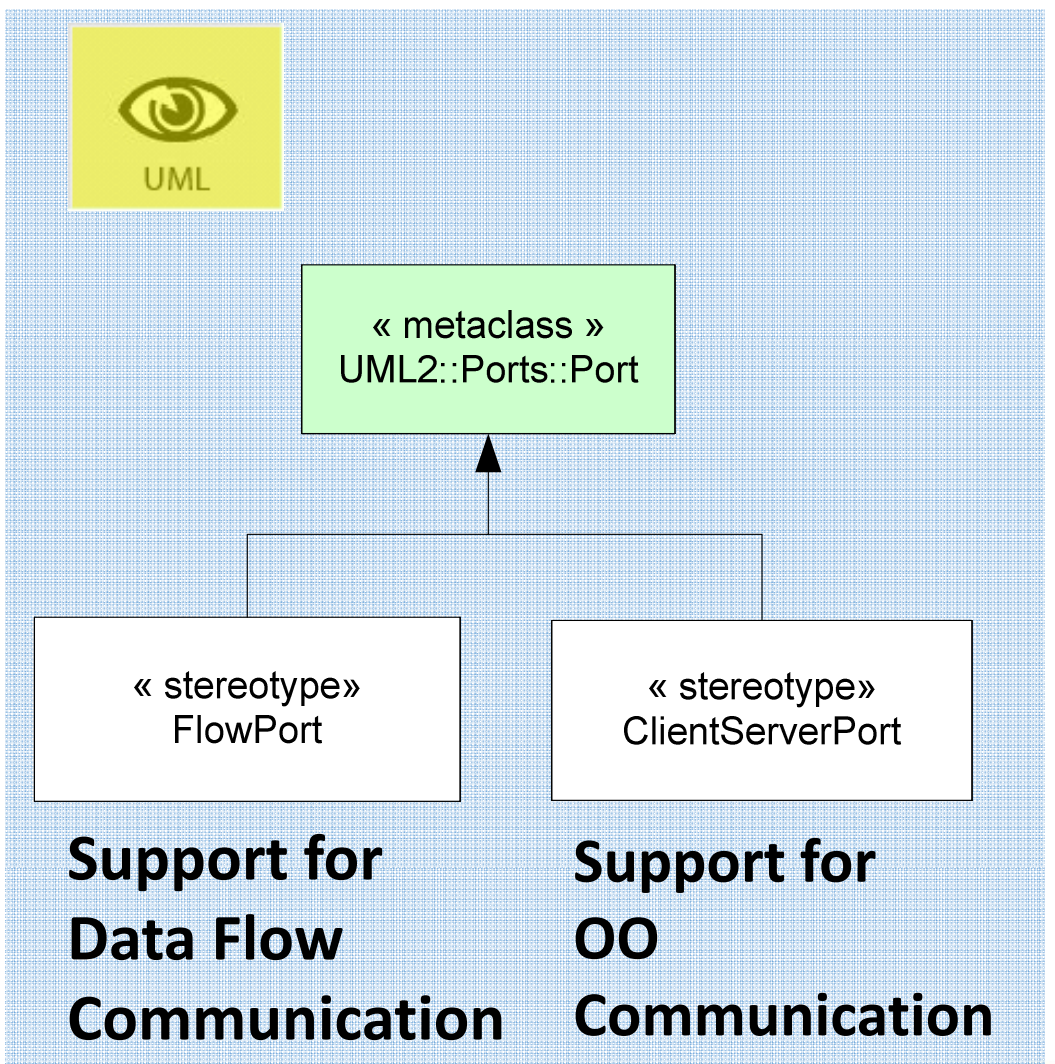
*Compatible with well-known component models, such as AADL, Autosar, EAST-ADL2, Lightweight-CCM, and SysML.*

*Offering a rich semantics enabling various models of computation and communication*

## MARTE Component: UML

BehavioredClassifier: contains behavior and structured data

## MARTE Port: extends UML Port



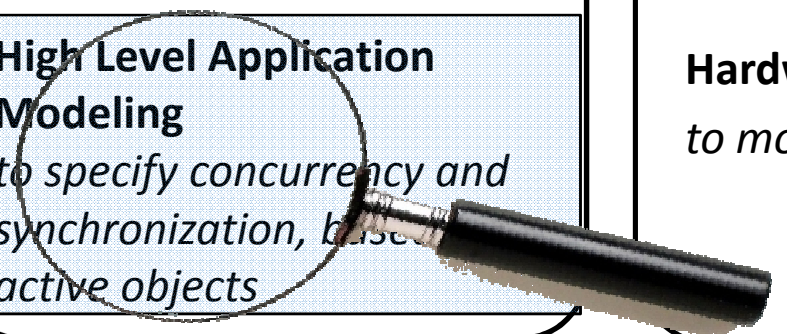
*possibility of specifying pull and push semantics*

# The three pillars of MARTE

**Pillar 1: Platform Independent Modeling**

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, base active objects*



**Pillar 2: Detailed Resource Modeling**

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

**Pillar 3: Annotations for quantitative analysis**

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

**Value Specification Language:** *textual language for specifying algebraic expressions*

# High-Level Application Modeling

*Language support to express*

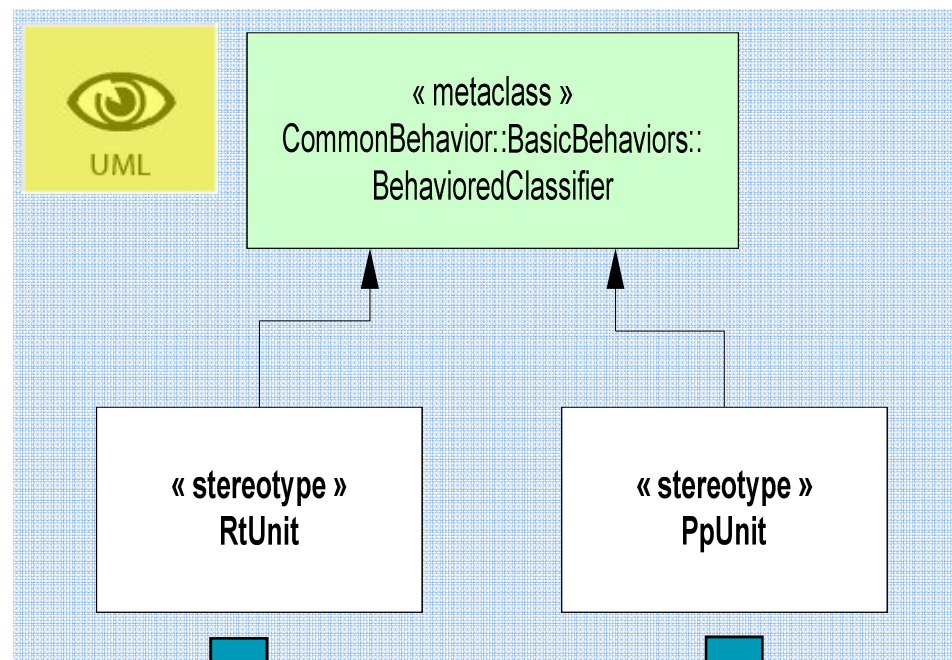
*models of computation and*

*concurrency: "Common" active objects, but alternative models can be defined*

*real-time constraints: allows expressing real-time constraints as deadline or period on component interfaces and connectors*

**MARTE active object : «RtUnit»**  
*generalization of UML active object*

**MARTE passive object: : «PpUnit»**  
*generalization of UML passive object*



*owns at least one execution thread. Threads are managed either statically (pool) or dynamically. May have operational mode description. Offers several (de) queuing policies*

*To execute requires threads. Supports different concurrency policies: sequential, guarded or concurrent.*

# The three pillars of MARTE

Pillar 1: Platform Independent Modeling

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, based on active objects*

Pillar 2: Detailed Resource Modeling

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

Pillar 3: Annotations for quantitative analysis

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

**Value Specification Language:** *textual language for specifying algebraic expressions*

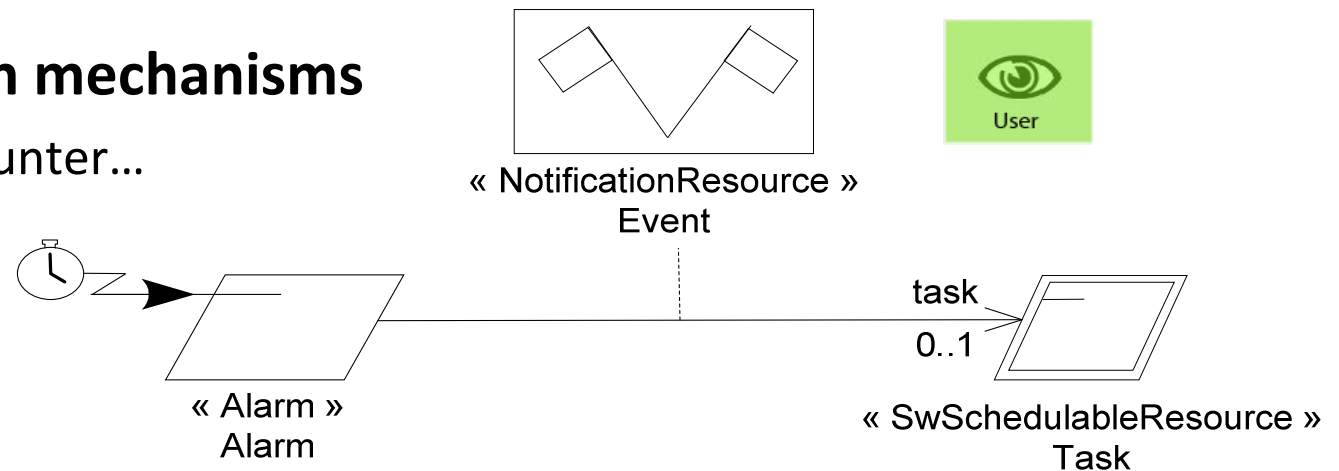
## Software Resource Modeling: define constructs for modeling multitask design

Real-Time Operating Systems (e.g. POSIX, OSEK/VDX and ARINC 653)

Real-Time language libraries (e.g. ADA)

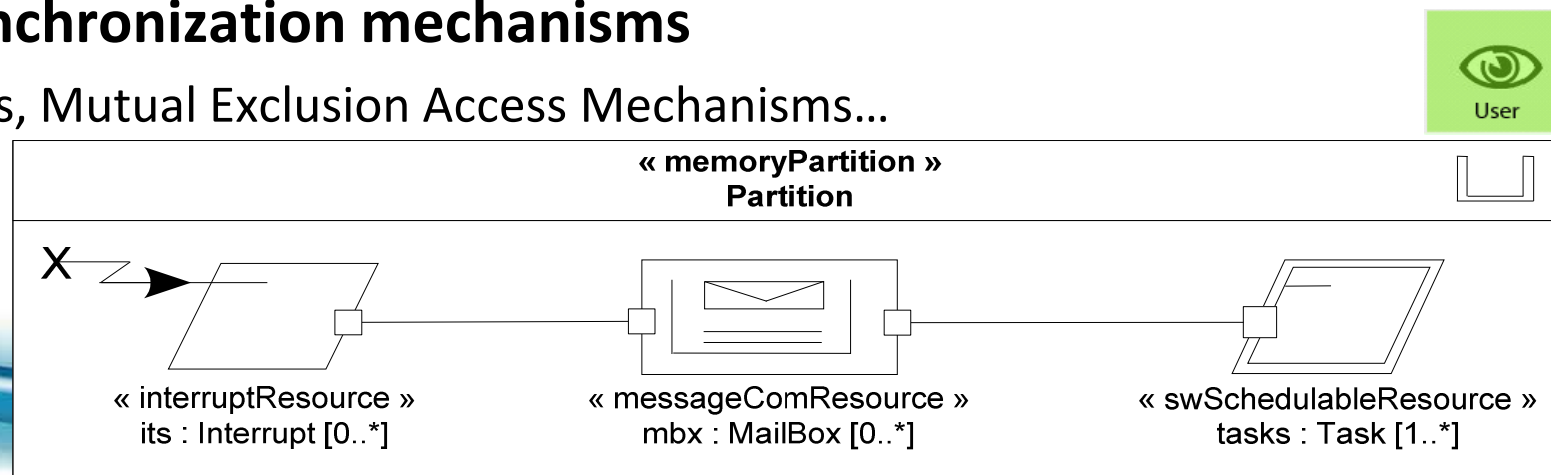
### Ex.1: Concurrent execution mechanisms

Task, Interrupt, Alarm & Counter...



### Ex2.: Synchronization mechanisms

Events, Mutual Exclusion Access Mechanisms...



# Hardware platform-based Modelling

## Hardware Resource Modelling: to describe structure of hardware platforms

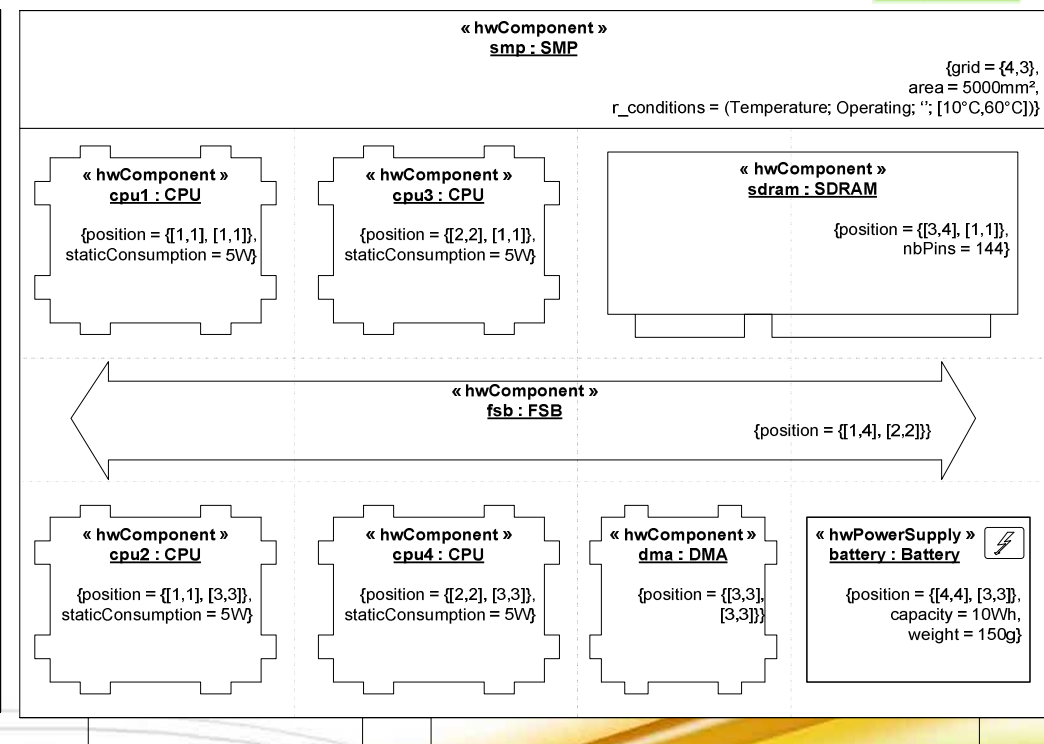
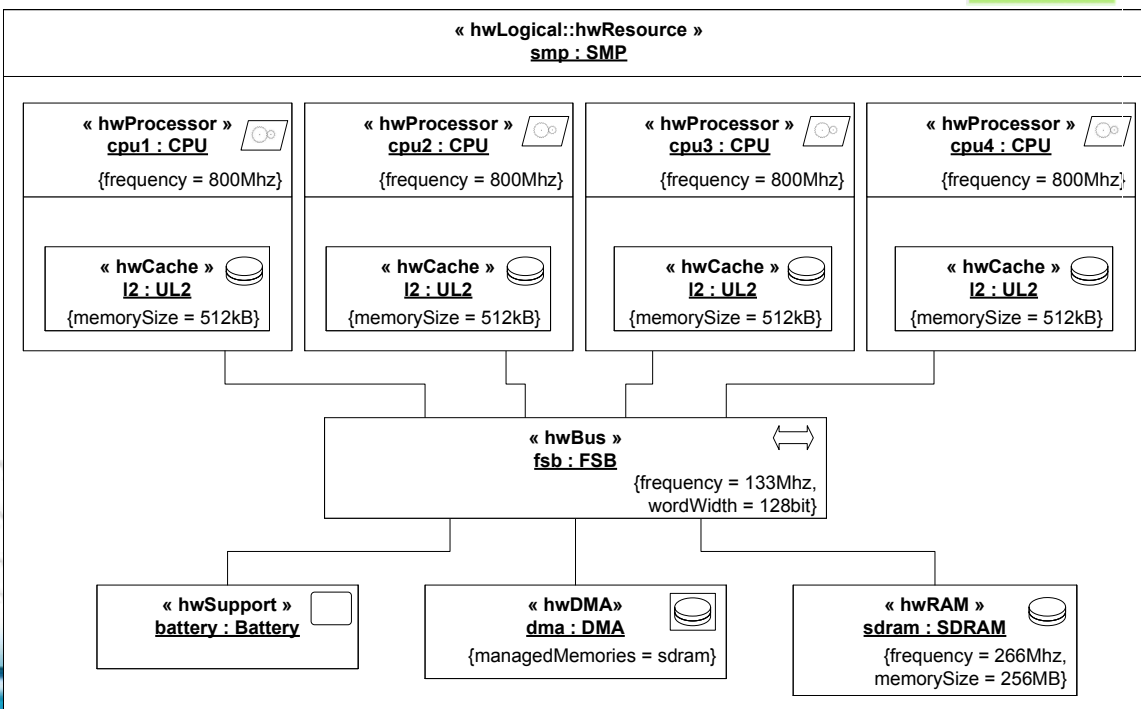
Different abstraction levels for: processor simulation, power consumption calculation, WCET analysis, block diagrams

Two sub-views:

### Logical view (functionality)



### Physical view (layouts)





# The three pillars of MARTE

Pillar 1: Platform Independent Modeling

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, based on active objects*

Pillar 2: Detailed Resource Modeling

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

Pillar 3: Annotations for quantitative analysis

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

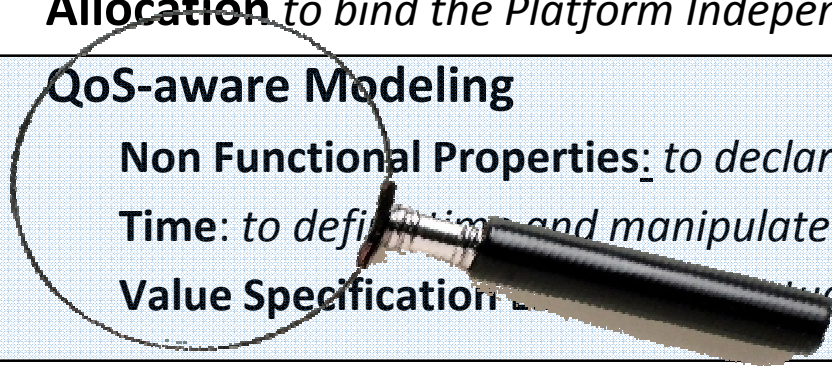
**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

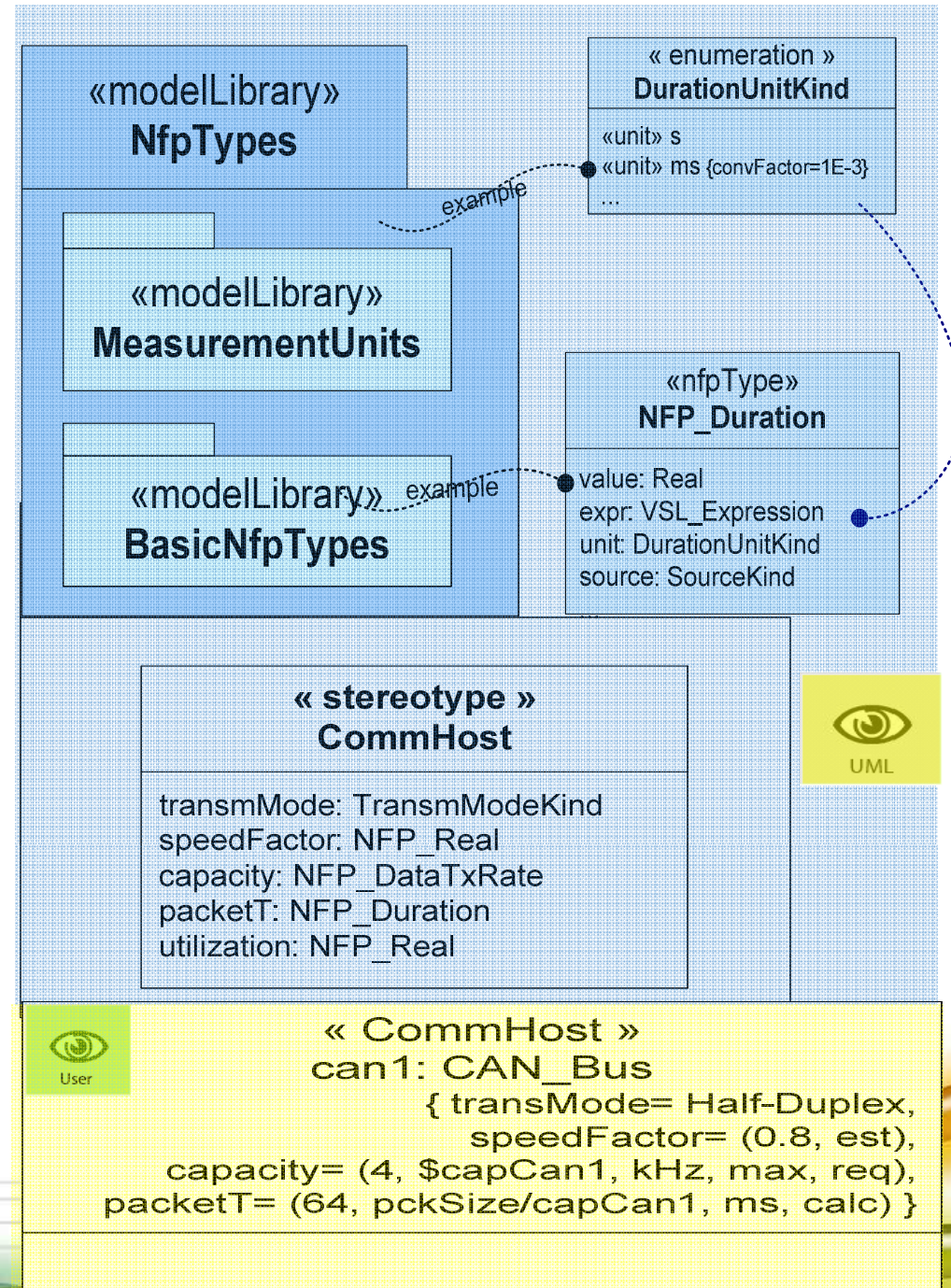
**Value Specification:** *to provide a formal language for specifying algebraic expressions*



## NFP types have the capability of modeling

1. Value as unit and dimension
2. characteristics on the way the value has been obtained, i.e. the nature of the measurement (e.g. statistical, empirical), its precision, the value's source, e.g. Worst Execution Time as estimated.

## Rich MARTE library of predefined types as Power, Frequency, DataSize, Duration



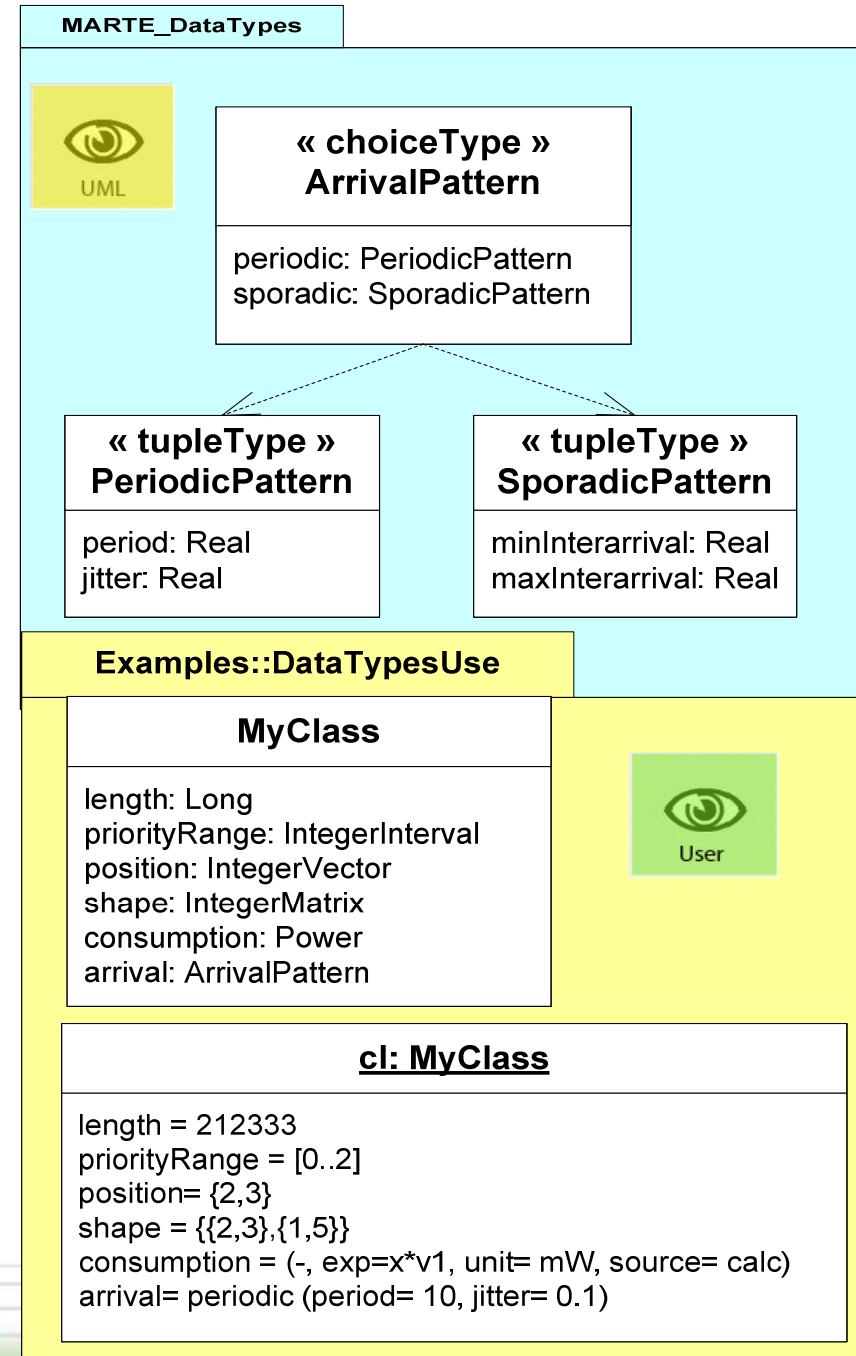
## The Value Specification Language (VSL) defines the formal textual syntax for specifying values of NFPs

Mathematical expressions  
(arithmetic, logical)

Variables: placeholders for a-priori unknown parameters.

Extended system of data types  
(tuples, collections, intervals)

Time expressions (delays, periods, jitters)



# The three pillars of MARTE

Pillar 1: Platform Independent Modeling

**General Component Model**  
*to support component based designs*

**High Level Application Modeling**  
*to specify concurrency and synchronization, based on active objects*

Pillar 2: Detailed Resource Modeling

**Software Resource Modeling**  
*to model OS execution support*

**Hardware Resource Modeling**  
*to model hardware platforms*

Pillar 3: Annotations for quantitative analysis

**Generic Quantitative Analysis Modeling**  
*to enable quantitative analysis*

**Schedulability and Performance Analysis Modeling**

## Foundations:

**Allocation** *to bind the Platform Independent Model to the Platform Model*

## QoS-aware Modeling

**Non Functional Properties:** *to declare and apply well-formed non-functional concerns*

**Time:** *to define time and manipulate its representations*

**Value Specification Language:** *textual language for specifying algebraic expressions*

# MARTE for Model-based QoS Analysis

**GQAM** A foundational framework for model-based quantitative analysis

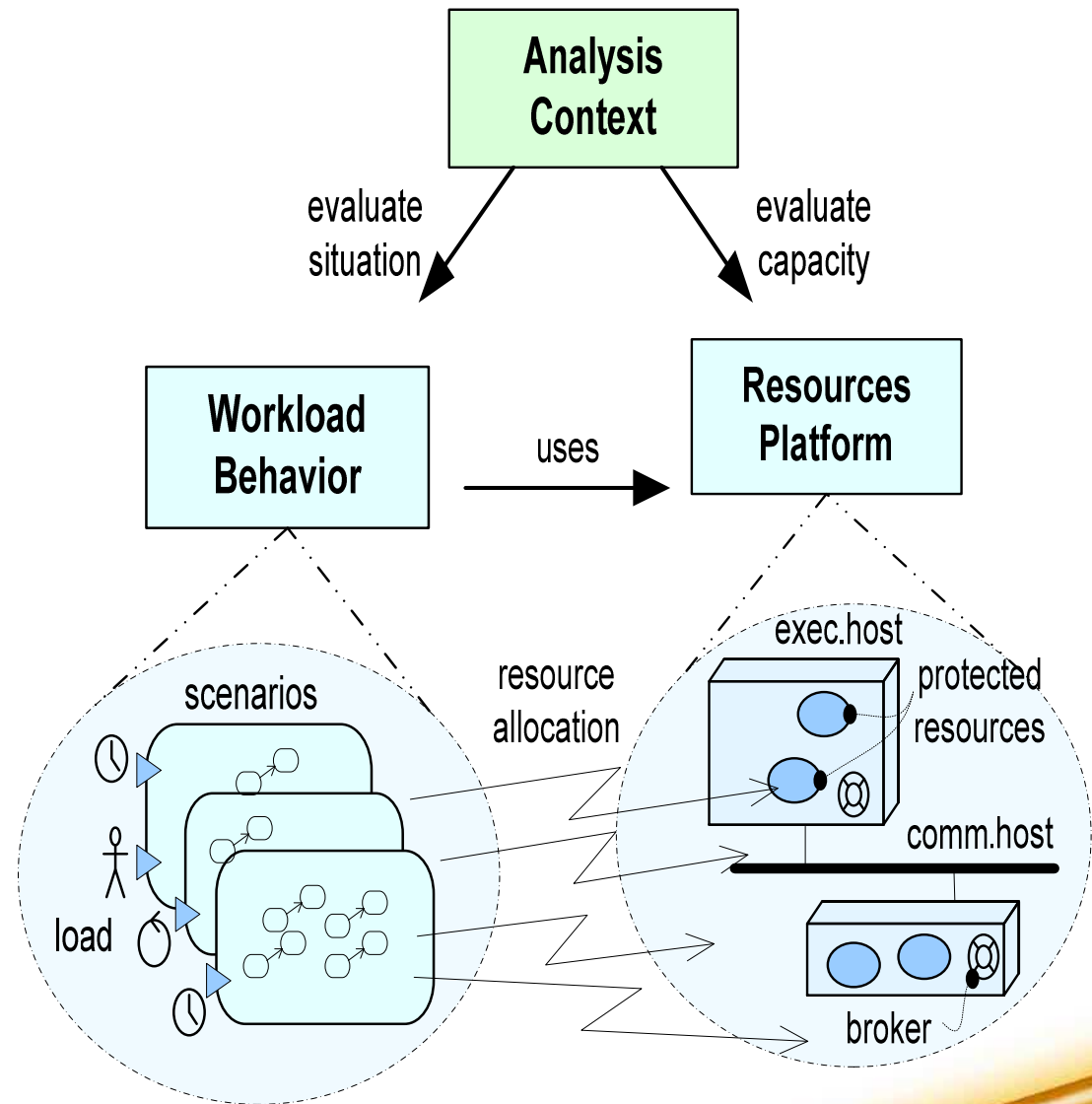
## Schedulability Analysis

### Modeling

*Supports most common scheduling analysis techniques RMA-based, EDF –based, holistic techniques*

## Performance Analysis Modeling

*Supports most of common performance analysis techniques queuing networks, petri nets, and simulation*



# Model-driven experience with MARTE– Interested Project

Now we focus on a Interested Project workpackage that explores the application of a model-driven approach, based on the standard language MARTE, to:

- Automotive Domain
- AUTOSAR software architecture
- Development of an RTES sub-system

**Goal:** Verification of the Deployment Model (application architecture to OS tasks and OS tasks on ECUs) from a Schedulability point of view before actual Deployment, through **an integrated tool-chain**

## Tools:



MARTE modeler



RT-DRUID for schedulability analysis

System-Desk AUTOSAR designer

# Verification of the Deployment model in AUTOSAR

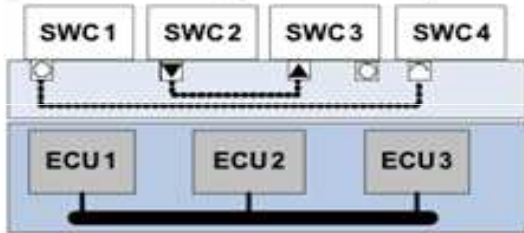
## System-level Specification



## Software Level

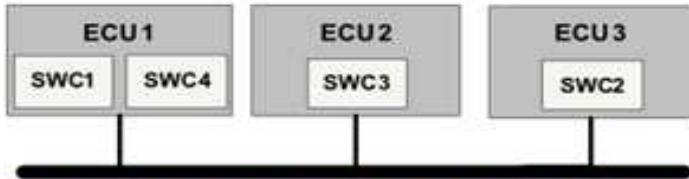
### AUTOSAR Methodology

#### System Configuration Input



SWCs and Connections  
 ECUs and topology  
 Mapping Constraints

#### System Configuration Description



Mapping of SWC to ECU  
 Component Internal Behavior (runnables)

#### Extract ECU Specific Information



RTE configuration

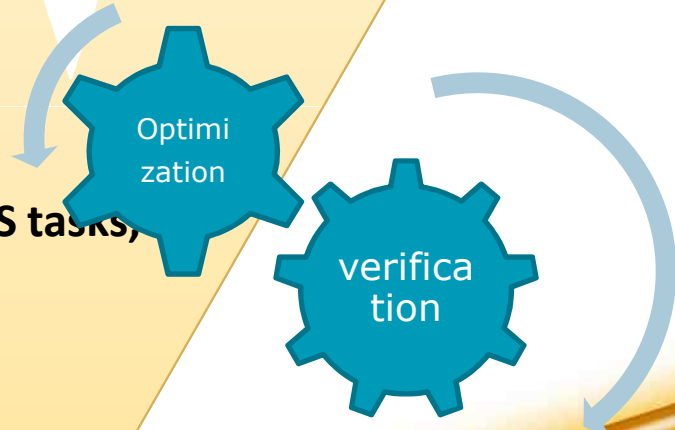
#### ECU Configuration



mapping of runnables on OS tasks,  
 BSW configuration

#### ECU Executable Generation

```
0110101100011
0101111100110
0000011100110
1010101010100
1101010110100
0001111011111
```





**Methods:** model-driven approach with  
*SySML/MARTE as backbone language*  
*DSLs to connect: AUTOSAR, RT-DRUID*

## Source meta-model **AUTOSAR**

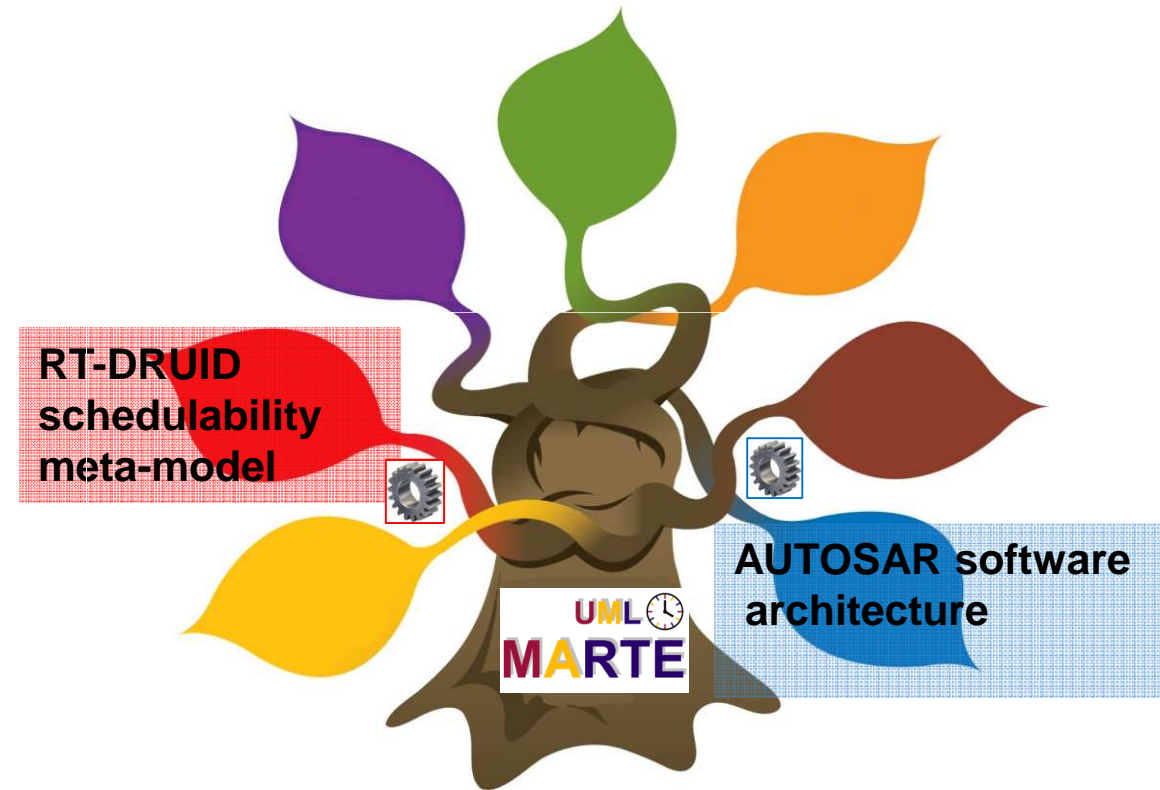


AUTOSAR-MARTE mapping



MARTE-RT-druid mapping

Target meta-model **Rt-Druid**



## Integration Technologies

Eclipse Modeling Framework



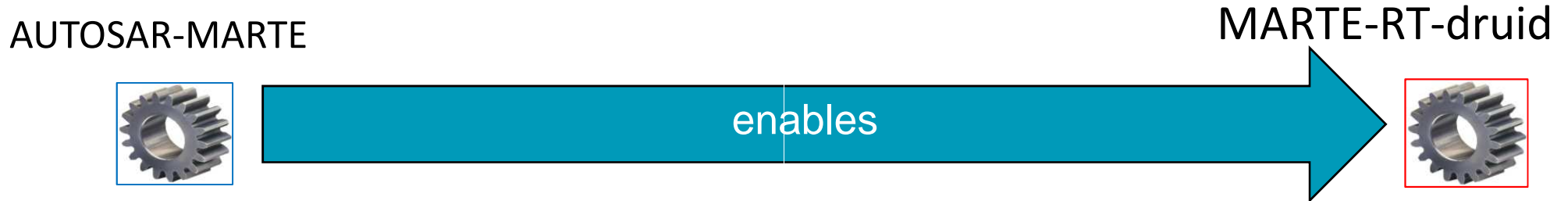
ATL as language to specify meta-model mappings



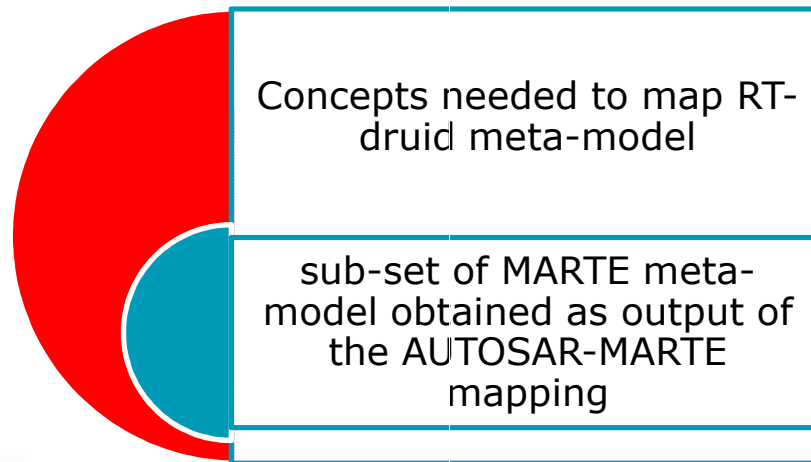
# Goal of the AUTOSAR-MARTE-RT-druid mappings

**Goal of the AUTOSAR-MARTE mapping: Enabling the MARTE-RT-DRUID Mapping**

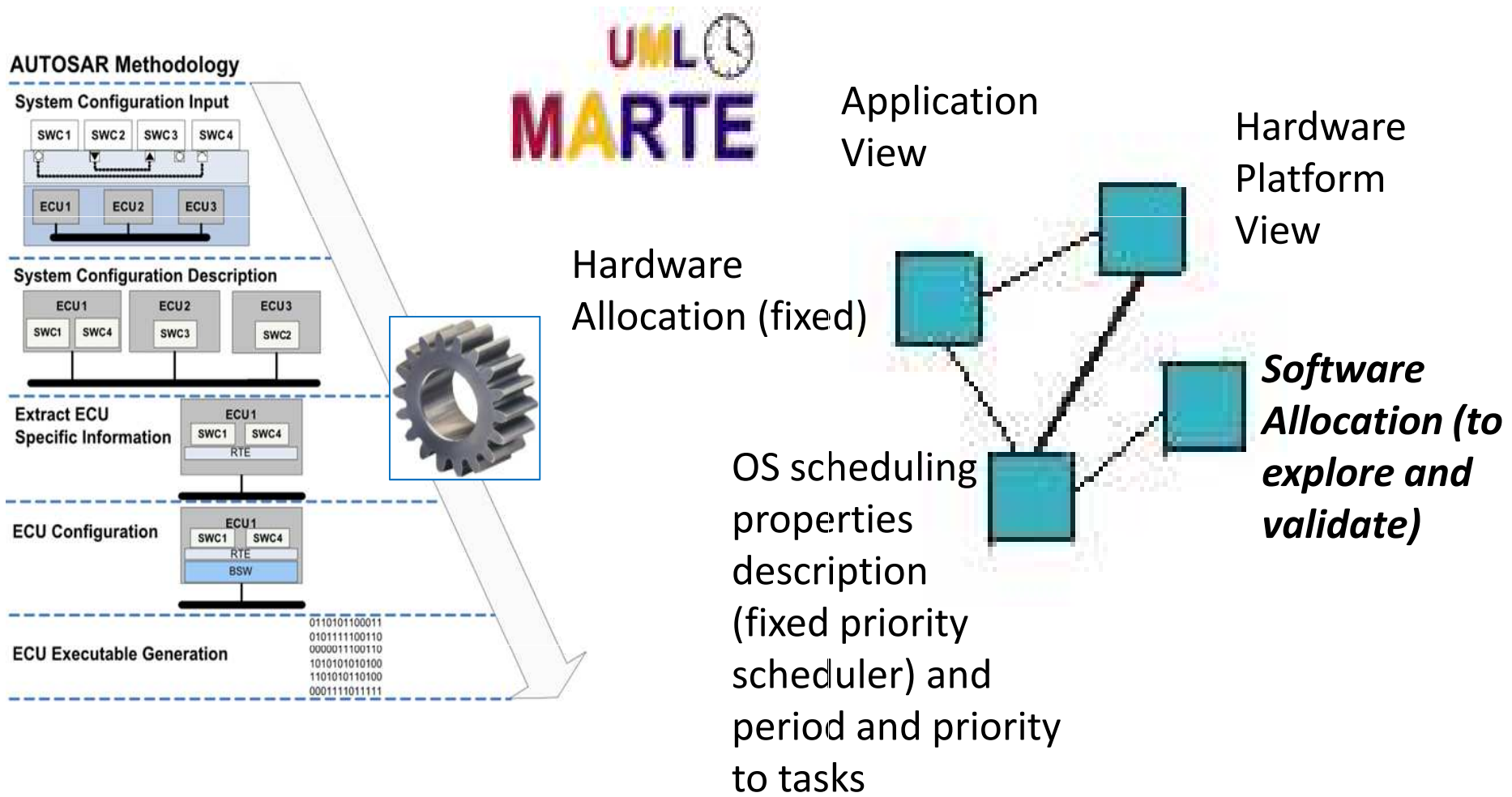
*obtained MARTE meta-model must contain **all and only the concepts** relevant to perform verification of timing constraints through schedulability analysis*



*.. but something was missing...*



# AUTOSAR-MARTE mapping



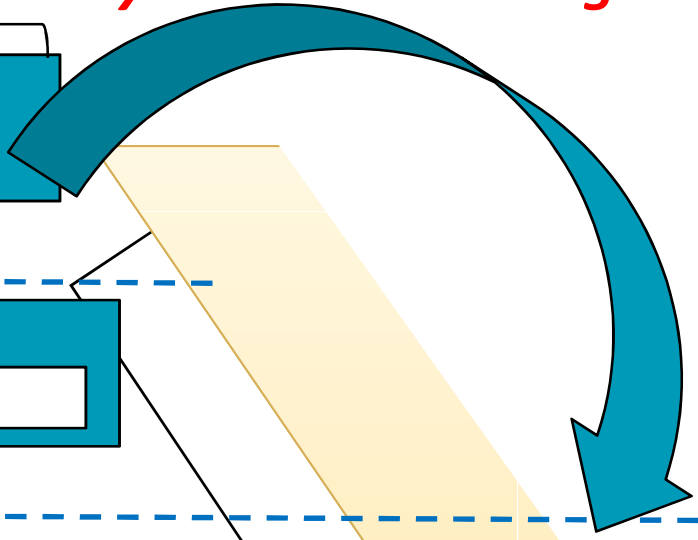
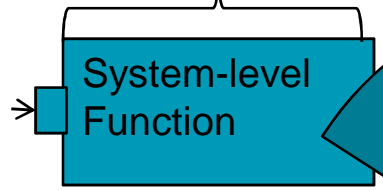
## So what is missing?

# AUTOSAR-MARTE mapping

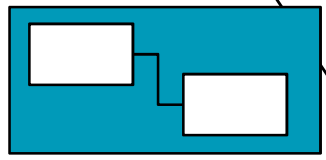
**Missing information: description of timing requirements!**

**System-level Timing Constraint to validate...**

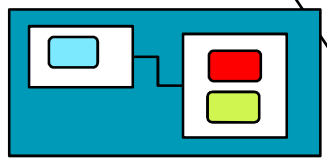
Triggering Stimulus



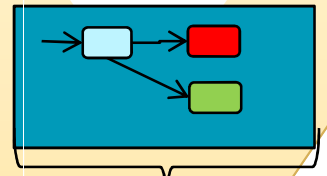
Software decomposition



Internal Behavior definition



Triggering Stimulus



**deadline**

**...refines also during software design to...**

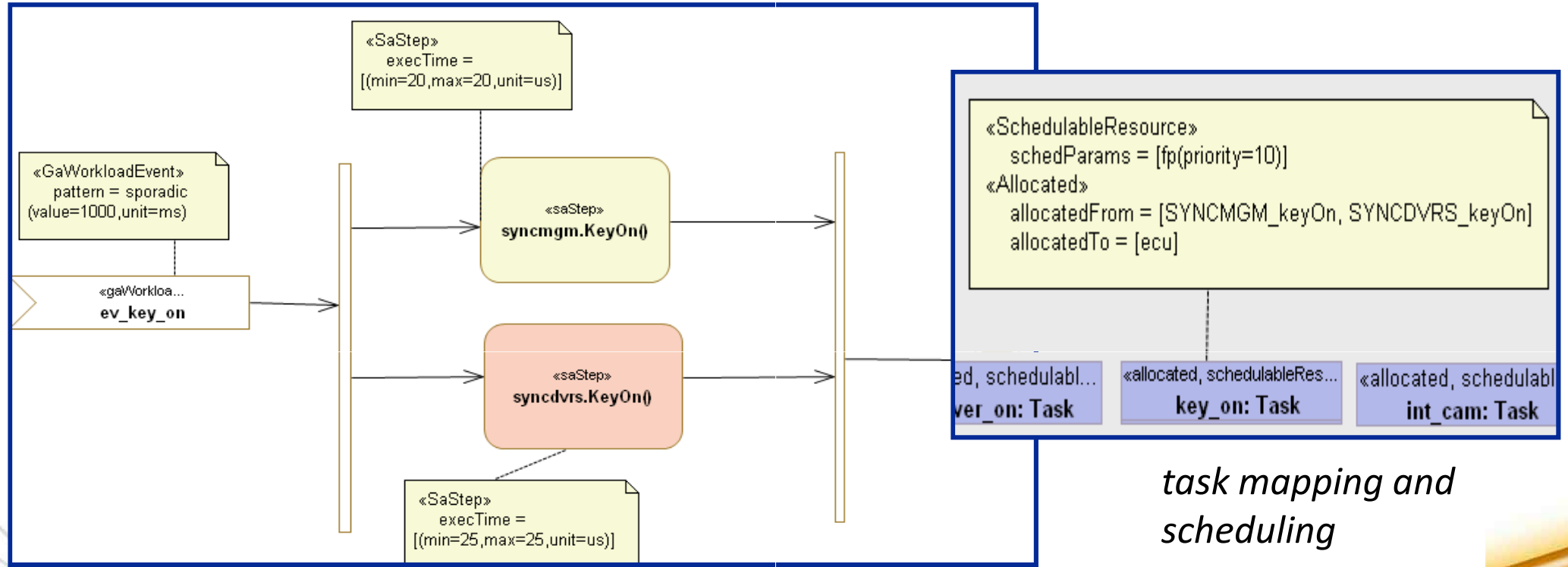
**...end-to-end runnable's activation chains descriptions with deadlines and stimuli arrival patterns!**

*Note: in the newest release of AUTOSAR (v4) it has been added the capability of expressing timing constraints (Timing View), but it was not available at the beginning of the project and will discuss this point in conclusions*

# (manual) completion of the model in MARTE

*End-to-end runnable's activation chains descriptions with deadlines and stimuli arrival patterns*

**runnables to tasks mapping → to validate against well-formed constraints**

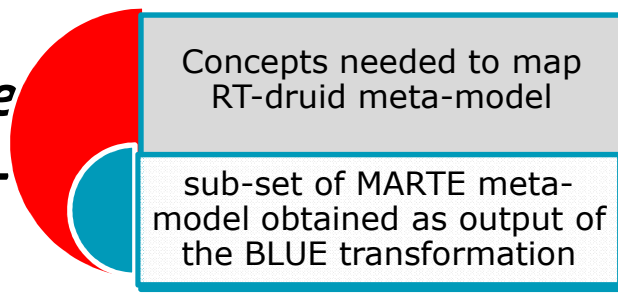


*task mapping and scheduling parameters*

**MARTE « end-to-end flow », attribute *end2endD=1000***

# The MARTE-RT-DRUID Mapping

*The mapping assumes the 'augmented' source meta-model*

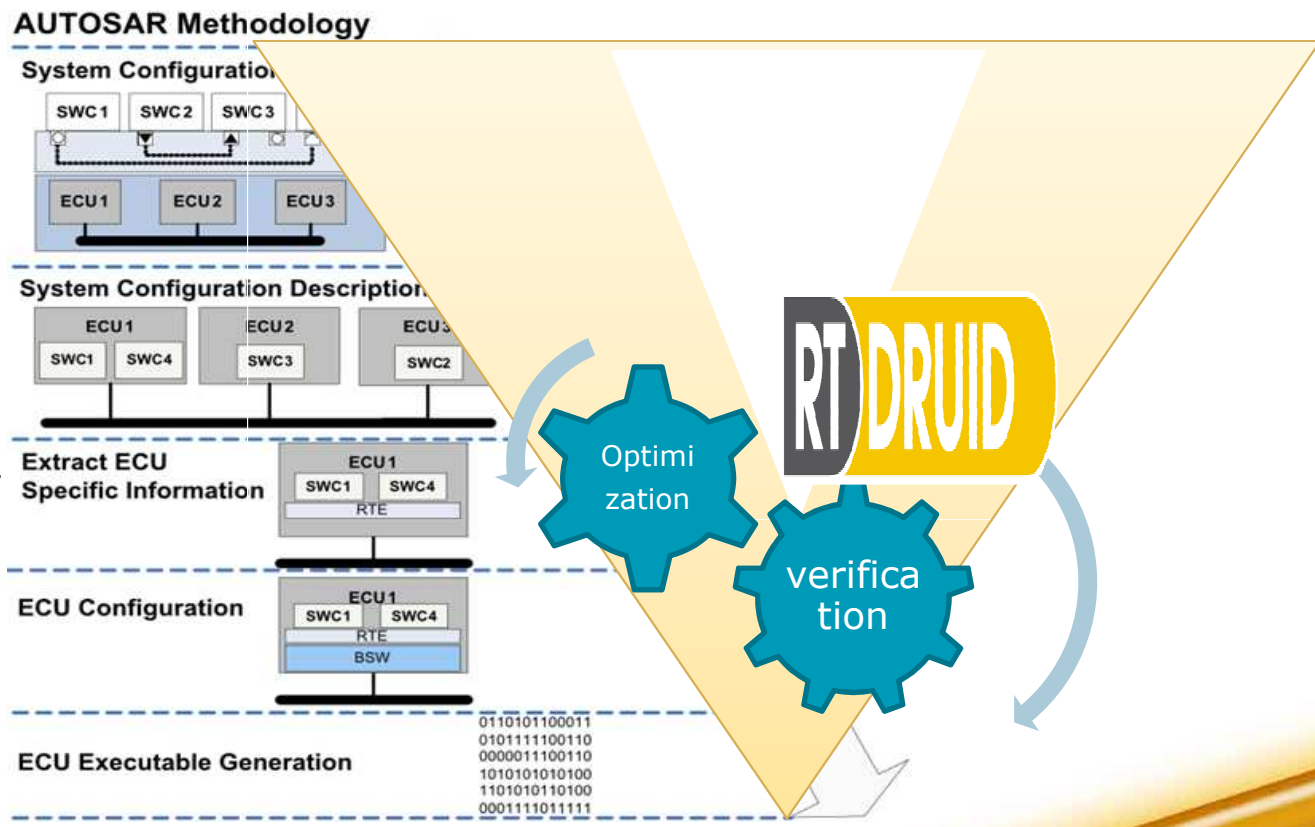


Meta-model

*Now we can automatically generate the entry model for RT-DRUID*

*Perform timing analysis in RT-DRUID: try different runnable-to-task mapping options, semantically valid (precedence order of runnables respected)*

*Re-import valid task mapping and obtained results in the MARTE model and in AUTOSAR*



## Lesson learnt

- *The Interested experience succeeded in proving the fitness of MARTE as a BACKBONE LANGUAGE for RTES development*
- *Although AUTOSAR v4 has introduced a Timing View, AUTOSAR has a limited scope and it has not been designed to be a modeling language. It cannot be used as backbone of the development process (remember the holistic view?).*

## Future Work and Issues

- *The missing piece in the Interested experience has been (1) the lack of formal representation of timing requirements at system level and (2) the proof that MARTE can keep formal relationships and coherence between system- and software -level elements*
- *Keeping coherence with system-level is the big issue. **The SysML/MARTE integration is a good choice as they share the same core (UML).***

**Acknowledgments: sebastien gerard, francois terrier, bran selic.**

## Useful links:

- **MARTE Current version : MARTE 1.1** (<http://www.omg.org/spec/MARTE/1.1/>)

**first version was MARTE 1.0**

- More than 4 years of work
- ~720 pages – Finalised end 2009

- **Papyrus is an Eclipse's Modelling::MDT component**  
<http://www.eclipse.org/papyrus>





# Questions?