Specific Model-based ADL: the Standards-based Approach

PhD. Sébastien Gérard
CEA LIST Senior Expert
Laboratory of model driven engineering for embedded systems (LISE)
CEA Techshow, Tokyo, June 10th, 2010.

Sebastien.Gerard@cea.fr
• Thanks to Sara, Chokri and Arnaud for their contributions:

• And thanks to Bran Selic:
  ▪ Some of the slides of this presentation have been extracted from the following keynotes:

  ▪ Sébastien Gérard and Bran Selic, "On a Specific Model-based Architecture Description Language: the Standards-Based Approach.", invited keynote at to the affiliated Artist workshop "Formalisms for architectural description and visualization" held within the CPS Week 2010, Stockholm, Sweden, March 18-19, 2010.
• **What?**
  - Development of Real-Time Embedded System of Systems (RTE-SoS)
    - Complex and heterogeneous systems responding to real-world events.

• **Beget!**
  - Multiple engineering disciplines
    - Many different methods, languages and tools.
  - QoS-constrained systems
    - E.g., Resource, energy, and time constraints.
  - Complex and often contradictory requirements/concerns
• Introduction
  ▪ The thesis
Going further for developing modern complex systems of systems requires new advanced and innovative methods!

A Standards-, Architecture-, and Model-based Approach
Solution: a system-level approach is needed

- Design the system as a whole rather than as an aggregate of separately designed sub-systems
  - Ensures system integrity
  - Requires a “big picture” approach; i.e., an **architecture**
- **One definition of Architecture** [IEEE Standard 1471]:
  “The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”
- **Architecture does help in designing RTE-SoS, because:**
  - It improves stakeholder communication
    - Concrete/tangible representation used as a focus of discussion by stakeholders of the system development
  - It enables team working
    - Used to distribute the tasks along working teams
    - Used to drive integration of its implemented subsystems
  - It reduces development risks by enabling early analysis/validation
    - Used for validation to know whether the system can meet its non-functional requirements => very important result for RT/E!
  - It fosters large-scale reuse
    - The architecture description is one key stone of product lines
Components for architecture description

- **Definition**
  - "A component-based application consists of components interacting via connected ports/interfaces.

- Component-based approaches are then ideal candidates for promoting architecture usage in development process.

- **Components are available at both design- and run-times.**
  - Foster homogenous architecture refinement through the dev. Process.

- **Components get two complementary facets**
  - An external view (or “black-box” view)
    - Publicly visible features (operations & properties).
    - Behavior may be attached to interface, port or/and to the component itself.
    - Component wiring via assembly-connectors between ports.
  - An internal view (or “white-box” view)
    - Private properties and realizing classifiers.
    - External and internal mapping using delegation connectors.
    - More detailed behavior specifications.
But why modeling?

- ARCHITECTURE [IEEE 1471]: “The fundamental organization...”
  ⇒ Architectural specifications abstract out non-fundamental detail
  ⇒ i.e., they are models!

  “To architect is to model”

- Characteristics of useful engineering models
  - Purposeful (i.e., intended for specific purposes/viewpoints/domains/audiences)
  - Abstract (i.e., they leave out inessential detail)
  - *Understandable* (easy to comprehend for intended audience)
  - *Accurate* (i.e., faithfully represent elements of interest)
  - Predictive (i.e., can be used to predict key system characteristics)
  - Significantly easier and cheaper to construct than the system they represent

- Accuracy and understandability, in particular, impose important requirements on modeling languages for describing architectures (*architectural description languages* → ADL)
Finally, why standards?

Standards have traditionally provided major boosts to technological progress!

- 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

- The Object Management Group (OMG) has created the Model-Driven Architecture initiative:
  - A comprehensive set of standards in support of MBE including standard modeling languages: **UML2, MARTE and SysML**.
• Originally intended for modeling software-intensive systems:
  - UML models capture different views of a software system (information model, run-time structure/behavior, packaging, deployment, etc.)
  - Inspired primarily by the concepts from object-oriented languages (class, operation, object, etc.)

• However, the general nature of its concepts made UML2 suitable for extensions to any specific domains.
UML2, a family of modeling languages

**UML2 Structure Diagram**
- Class Diagram
- Component Diagram
- Object Diagram
  - Composite Structure Diagram
  - Deployment Diagram
  - Package Diagram

**UML2 Behavior Diagram**
- State Machine Diagram
- Activity Diagram
- Use Case Diagram
  - Sequence Diagram
  - Communication Diagram
  - Timing Diagram
  - Interaction Overview Diagram
But UML is also customizable!

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

- However, the general nature of its concepts made UML suitable for extensions to other domains.

Domain Specific Modeling by profiling the UML2!
**UML Profile**
- A special kind of package containing stereotypes, modeling rules and model libraries that, in conjunction with the UML metamodel, define a group of domain-specific concepts and relationships.

**Profiles can be used for two different purposes:**
- To define a domain-specific modeling language.
- To define a domain-specific viewpoint.

**Minimal benefits of profile usage are:**
- Correctly defined profiles allow direct and effective reuse of the extensive support structure provided for UML (e.g., Tools, methods, experience, training...).
- DSMLs based on UML profiles share a common semantic foundation which can greatly reduce the language fragmentation problem.
Profile definition
(Language definition level)

- Specific notation
  - « metaclass »
    - `UML::Class`
  - « stereotype »
    - `Semaphore`
    - `limit: Integer`
    - `getSema: Operation`
    - `relSema: Operation`
  - Usage constraint
    - « Constraint »
      - `limit < UpperLimit`

Profile application
(User model level)

- « semaphore »
  - `SpeedDataLock`
  - `SpeedDataLock`

Ps: Slides credited to Bran Selic
• **Systems Modeling Language (www.SysML.org)**
  - General-purpose systems modeling language
    - Specification, analysis, design, verification and validation of a broad range of complex systems

• **UML-compatible systems modeling language**
  - For supporting the exchange of information using standardized notations and semantics that are understood in precise and consistent ways.

• **SysML will have to be customized to model domain specific applications**
  - Space, Automotive, Aerospace, Communications...

• **Aligned with the ISO AP-233 standard and connected to Modelica**
SysML static views

Similar to UML2 composite structure diagram but based on Blocks.

SysML Structure Diagram

Enable to denote mathematical relationships between physical parameter of a system.

Internal Block Diagram

Block Definition Diagram

Parametric Diagram

Similar to UML2 class diagram but manipulating Block instead of Class.

SysML Cross-cutting Diagram

Used to formally model and organize textual requirements, but also support for requirement traceability analysis.

New

Requirements Diagram
Minor extensions of UML2 activity diagram:
- Two types of object/data flow:
  - continuous or discrete.
- Probability on edges of the graph.

Note: SysML excludes Communication Diagram, Interaction Overview Diagram, and Timing Diagram!
Complex systems may be real-time & embedded

- The good ingredients for carrying out the modeling of a complex system successfully are then:
  - **UML2 as basis modeling language**
    - Broad acceptance and tooling, rich set of concepts, and customizable to fit specific domains/concerns.
  - **SysML for its support of systems engineering concerns**
    - Requirement engineering, mathematical expression description, continue/discrete behaviors, and a very good acceptance level in industry (stronger than UML itself!)

- But which standards for RTE concerns?
The four pillars of MARTE

- **Pillar 1: Architecture Modeling**
  - **GCM**: for architecture modeling based on components interacting by either messages or data.
  - **Alloc**: for specifying allocation of functionalities to entities realizing them.

- **Pillar 2: QoS-aware Modeling**
  - **HLAM**: for modeling high-level RT QoS, including qualitative and quantitative concerns.
  - **NFP**: for declaring, qualifying, and applying semantically well-formed non-functional concerns.
  - **Time**: for defining time and manipulating its representations.
  - **VSL**: the Value Specification Language is a textual language for specifying algebraic expressions.

- **Pillar 3: Platform-based Modeling**
  - **GRM**: for modeling of common platform resources at system-level and for specifying their usage.
  - **SRM**: for modeling multitask-based design
  - **HRM**: for modeling hardware platform

- **Pillar 4: Model-based QoS Analysis**
  - **GQAM**: for annotating models subject to quantitative analysis.
  - **SAM**: for annotating models subject of scheduling analysis.
  - **PAM**: for annotating models subject of performance analysis.
Pillar 1: Architecture Modeling

- **MARTE’s GCM in a nutshell**
  - Introduced to cope with various RTE component models
    - AADL, Autosar, EAST-ADL2, Lightweight-CCM, XP-ACT and SysML.
  - Precise semantics enabling various models of computation and communication
    - Especially on relationships between structural and behavioral aspects.
  - Relies on UML structured class, and only extend UML Port

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

Support for “classic” OO message-based communication schema.

- « stereotype » FlowPort
- « metaclass » UML2::Ports::Port
- « stereotype » ClientServerPort
An example of architecture view

- **Atomic flow ports**

  - **Out flow port conveying integer data values**

  - **In flow port conveying integer data values**

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

  - **In flow property**
    - « interface »
    - « flowSpecification »
    - SpeedSensorFS
    - cSpeed: SpeedDT
    - cTime : Time

  - **Non-atomic inout flow port**
    - « flowPort »
    - outSpeed : SpeedSensorFS

- **Out flow property**

- **Typed by flow specification**
  - typed by flow specification!
Two reminders on UML

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

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

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

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

MARTE modeling rule: A model owning in or inout behavior flow ports with delegation connectors is considered to be ill-formed.
Semantics view of MARTE’s flow port: Push semantics (seq.)

Activity denoting the classifier behavior of Regulator

In behavior flow port conveying integer values

AcceptEventAction with a trigger based on a « DataEvent ».
Pull semantics is relying on a specific modelling pattern defined in:


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

<table>
<thead>
<tr>
<th>CarSpeedRegulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>spm:Speedometer [1]</td>
</tr>
<tr>
<td>« flowPort »</td>
</tr>
<tr>
<td>inSpeed: Integer [1]</td>
</tr>
<tr>
<td>rgm:Regulator [1]</td>
</tr>
<tr>
<td>currentSpeed: Integer [1]</td>
</tr>
</tbody>
</table>
Possible other standard storing policies defined using the stereotype: « **DataPool** »

**Two standard policies defined via the property ordering:**
FIFO and LIFO
Semantics view of MARTE’s flow port: Pull semantics (seq.)

• Details of the stereotype « DataPool »

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

• Example of usage
Outline of the Allocation concept

- **Purpose**
  - Provide support for denoting the "mapping/association" of the functional parts of a system onto its computing resources.

- **Similar to the SysML Allocation**
  - More restrictive than in SysML
    - "Model the application allocation on its related platform"
  - 2 use cases of allocation modeling
    - Spatial distribution aspect
      - e.g., a variable allocated to a given memory resource.
    - Temporal scheduling aspect
      - e.g., a function computed on a given processor resource.
  - Possibly several allocations description with different non-functional constraints
Outline of the Allocation stereotype

«metaclass»
UML::Abstraction

«stereotype»
Allocate

kind : AllocationKind
nature : AllocationNature

«enumeration»
AllocationKind
structural
behavioral
hybrid

impliedConstraint

«stereotype»
NFP_Modeling::NfpConstraint

«enumeration»
AllocationNature
spatialDistribution
timeScheduling
Defining the roles of model elements within allocation

- **MARTE allocation rely on the distinction between:**
  - Model elements of the application
  - Model elements of the platform

- **Stereotype « allocated »**

```plaintext
UML

```
```plaintext
MarTE allocation rely on the distinction between:
- Model elements of the application
- Model elements of the platform

Stereotype « allocated »
```
• **An application Model**

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg : SpeedRegulatorSystem</td>
</tr>
<tr>
<td>SpeedController</td>
</tr>
</tbody>
</table>

• **A platform model**

<table>
<thead>
<tr>
<th>OperatingSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>«schedulableResource» Thread</td>
</tr>
<tr>
<td>«storageResource» VirtualMemory</td>
</tr>
</tbody>
</table>
Allocation example: identification of allocation roles

- **An application Model**

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg : SpeedRegulatorSystem</td>
</tr>
</tbody>
</table>

- **A platform model**

<table>
<thead>
<tr>
<th>OperatingSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>«allocated» {kind=executionPlatform}</td>
</tr>
<tr>
<td>«scheduledResource» Thread</td>
</tr>
<tr>
<td>«allocated» {kind=executionPlatform}</td>
</tr>
<tr>
<td>«storageResource» VirtualMemory</td>
</tr>
</tbody>
</table>
Allocation example: model allocations

- My application Model

  Application

  reg : SpeedRegulatorSystem

  «allocated»
  {kind=application}
  SpeedController

  «allocated»
  {kind=application}
  CarSpeed

- My platform model

  OperatingSystem

  «allocated»
  {kind=executionPlatform}
  «schedulableResource»
  Thread

  «allocated»
  {kind=executionPlatform}
  «storageResource»
  VirtualMemory
Application and Platform roles are relative concepts

Application

- reg : SpeedRegulatorSystem
  - «allocated» {kind=application}
    - sc: SpeedController
  - «allocated» {kind=application}
    - cs: CarSpeed

OperatingSystem

- «allocated» {kind=both}
  - «schedulableResource»
    - th1: Thread
- «allocated» {kind=both}
  - «storageResource»
    - vm1: VirtualMemory

OperatingSystem

- «allocated» {kind=executionPlatform}
  - «computingResource»
    - cpu1: CPU
- «allocated» {kind=executionPlatform}
  - «storageResource»
    - mem1: Memory
Limitation of the MARTE/SysML allocation

- **Semantics of the allocation:**
  - An allocation is NOT always a dependency
  - An allocation is NOT always an abstraction

- **In MARTE domain view: explicit distinction between allocation and abstraction/refinement.**
  - Abstraction relates two concepts of different abstraction levels
  - Often allocation relates two concepts at the same level but of a different nature (application vs. execution platform)

- **Requirements: define a general allocation mechanism in MARTE**
  - The allocated elements and the targets can be refined
  - We do not necessarily need to modify the allocated element
  - We may even want to prevent a modification (read_only)
MARTE alternative for allocation: Assignement

Allocation example with « allocate »

- Straightforward modeling concept.
- Dependency between both application and platform models.

Allocation example with « assign »

- Need an additional model
+ No dependencies between both application and platform models.
The four pillars of MARTE

- **Pillar 1: Architecture Modeling**
  - **GCM**: for architecture modeling based on components interacting by either messages or data.
  - **Alloc**: for specifying allocation of functionalities to entities realizing them.

- **Pillar 2: QoS-aware Modeling**
  - **HLAM**: for modeling high-level RT QoS, including qualitative and quantitative concerns.
  - **NFP**: for declaring, qualifying, and applying semantically well-formed non-functional concerns.
  - **Time**: for defining time and manipulating its representations.
  - **VSL**: the Value Specification Language is a textual language for specifying algebraic expressions.

- **Pillar 3: Platform-based Modeling**
  - **GRM**: for modeling of common platform resources at system-level and for specifying their usage.
  - **SRM**: for modeling multitask-based design
  - **HRM**: for modeling hardware platform

- **Pillar 4: Model-based QoS Analysis**
  - **GQAM**: for annotating models subject to quantitative analysis.
  - **SAM**: for annotating models subject of scheduling analysis.
  - **PAM**: for annotating models subject of performance analysis.
The issue: explicit semantics in models

Step 1: Type and properties definition

<table>
<thead>
<tr>
<th>CAN_Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>transmMode: TransmModeKind</td>
</tr>
<tr>
<td>speedFactor: Real</td>
</tr>
<tr>
<td>capacity: Integer</td>
</tr>
<tr>
<td>packetT: Real</td>
</tr>
</tbody>
</table>

« instanceof »

<table>
<thead>
<tr>
<th>myCan: CAN_Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>transmMode = Half-Duplex</td>
</tr>
<tr>
<td>speedFactor = 0.8</td>
</tr>
<tr>
<td>Capacity = 4</td>
</tr>
<tr>
<td>packetT = 64</td>
</tr>
</tbody>
</table>

How to specify units and other qualifiers?

How these numbers were obtained? Calculated/measured/estimated?

The MARTE NFP/VSL Modeling Framework is the solution.
The solution: MARTE for semantics-aware models

The system requires the capacity of the CAN_BUS can1 to be at a maximum of 4 kHz.

It has been calculated that the time for sending packet via can1 is 64 ms.

Annotating models with formal NFP is a key point for efficient model-based engineering:

- Automatics model evaluation
Examples of Non-Functional Specifications

- Max. delay of a control loop (sensor to actuator) = 10 ms
- Max. deviation of a nominally periodic event (jitter) = 5.3 ms
- Delay between the n\textsuperscript{th} and the n\textsuperscript{th}+10 occurrence of an event = 125 ms iff generated data is greater than 10 KB
- Dynamic power consumption of an electronic component = 12 mW
- Number of occurrences of an event in a 50 ms interval time = 10
- Ignition deadline according to the zero-position of a flywheel = 25 °CAM
- Maximum processor utilization of P1 = 50 % of P2 utilization (math expressions, variables!)
- ...
On the MARTE framework for QoS-aware modeling

- **Tenets of the NFPs sub-profile**
  - Measurements: magnitude + unit (e.g., energy and duration)
  - Value Qualifiers: Value source, statistical measure, precision...
  - Core mechanisms for the characterization of modeling artifacts with non-functional information

- **Value Specification Language (VSL)**
  - Mathematical expressions (arithmetic, logical...)
  - Variables: placeholders for unknown analysis parameters.
  - Extended system of data types: tuples, collections, intervals...
  - Time expressions (delays, periods, trigger conditions...)
  - Formal textual syntax for specifying values of NFPs

- **Why do we need this level of formalization?**
  - For enabling tool-assisted V&V.
  - For fostering common & unambiguous understanding by stakeholders
Outlines of the NFP sub-profile

- The NFP Sub-profile is a framework for defining semantically rich model annotations of the QoS.

- Only five stereotypes:
  - Nfp, NfpType, NfpConstraint, Unit, Dimension

- A predefined library of Units, Dimensions and NFP Types
  - E.g. Power, Frequency, DataSize, Duration, and BoundDuration
  - A set of generic qualifiers for these NFP Types

- Three mechanisms to specify NFP values:
  - UML ValueSpecification of InstanceSpecification Slots
  - Stereotype attributes
  - Constraints
1) Declare NFP types:
   i. Define measurement units and conversion parameters.
   ii. Define specific NFP types with qualifiers.

2) Define NFP-like extensions:
   i. Define stereotypes and their attributes using previously defined NFP types.

3) Specify NFP values:
   i. Apply stereotypes and specify their tag values using VSL.
1) Declare NFP types:
   i. Define measurement units and conversion parameters.
   ii. Define NFP types with qualifiers.

2) Declare NFPs in user models:
   i. Define classifiers and their attributes using NFP types.
   ii. Attributes are then tagged as «nfp».

3) Specify NFP values:
   i. Instantiate classifiers and specify their slot values using VSL.
1) Declare NFP types:
   i. Define measurement units and conversion parameters.
   ii. Define NFP types with qualifiers.

2) Declare NFPs:
   i. Define classifiers and their attributes using NFP types.

3) Specify NFP values:
   i. Create Constraints to define assertions on NFP values using VSL.

- If the utilization of the processor is greater than 90%, the clock frequency must be equal to 60 Mhz.
- Otherwise, it must be equal to 20 Mhz.
Summary on the MARTE 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
The Value Specification Language (VSL): An expression language for values, functions, variables, and (basic) time expressions.
• **The Value Specification Language (VSL)**
  - The expression language for values, functions, variables, and (basic) time expressions
  - Provides a concrete and formal textual syntax for NFP typed values
• **Formally defines:**
  - A set of stereotypes extending UML::DataTypes
  - A Grammar (EBNF) for the VSL textual syntax
• **An extended system of data types**
  - Composite types: Tuples, Collection, Choice, Interval types
  - Subtypes: bounded subtype
• **An extended language for complex expressions**
VSL Extended Data Types

- **BoundedSubtype**
- **IntervalType**
- **CollectionType**
- **TupleType**
- **ChoiceType**

**Examples::DataTypesUse**

**MyClass**
- length: Long
- priorityRange: IntegerInterval
- position: IntegerVector
- shape: IntegerMatrix
- consumption: Power
- arrival: ArrivalPattern

**cl: MyClass**
- length = 212333
- priorityRange = [0.2]
- position= {2,3}
- shape = {{2,3},{1,5}}
- consumption = (c, exp=x*v1, unit= mW, source= calc)
- arrival: periodic (period= 10, jitter= 0.1)
## Basic Textual Expressions in VSL

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

<table>
<thead>
<tr>
<th>Value Spec.</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Number</td>
<td>1.2E-3 //scientific notation</td>
</tr>
<tr>
<td>DateTime</td>
<td>#12/01/06 12:00:00# //calendar date time</td>
</tr>
<tr>
<td>Collection</td>
<td>(1, 2, 8, 5, 2) //sequence, bag, ordered set..</td>
</tr>
<tr>
<td></td>
<td>{{1,2,3}, (3,2)} //collection of collections</td>
</tr>
<tr>
<td>Tuple and choice</td>
<td>(value=2.0, unit= ms) //duration tuple value</td>
</tr>
<tr>
<td></td>
<td>periodic(period=2.0, jitter=3.3) //arrival pattern</td>
</tr>
<tr>
<td>Interval</td>
<td>[1..251] //upper opened interval between integers</td>
</tr>
<tr>
<td></td>
<td>[$A1..$A2] //interval between variables</td>
</tr>
<tr>
<td>Variable declaration &amp;</td>
<td>io$var1 //input/output variable declaration</td>
</tr>
<tr>
<td>Call</td>
<td>var1 //variable call expression</td>
</tr>
<tr>
<td>Arithmetic Operation</td>
<td>+(5.0, var1) //&quot;add&quot; operation on Real datatypes</td>
</tr>
<tr>
<td>Call</td>
<td>5.0+var1 //infix operator notation</td>
</tr>
<tr>
<td>Conditional Expression</td>
<td>((var1&lt;6.0)?(10^6):1) //if true return 10 exp 6,else 1</td>
</tr>
</tbody>
</table>
**Duration expression between two successive occurrences**

- `constraint1 = (t0[i+1] - t0[i]) > 100 ms`
- `constraint2 = (t3 when data < 5.0) < t2 + 30 ms`

**Jitter constraint**

- `start() { jitter(t0) < 5 us }

**Extended duration intervals with bound « [ ] » specification**

- `{ t1..t1+(8, ms) }

**Constraint in an observation with condition expression**

- `{ d1..30*d1 }

**Duration Observation**

- `ack()`
Conclusions on MARTE pillar 2

- The NFP/VSL modeling framework of MARTE provides:
  - A generic and open support for characterization of modeling artifacts with non-functional information.
  - A formal syntax for specifying values of aforementioned non-functional properties.

- It is shared by almost all other MARTE sub-profiles
  - Understanding its principles is important to fully benefit from the expressive power of MARTE.
High-level modeling concepts for RT/E design
- Qualitative aspects
  - E.g. concurrency and behavior
- Quantitative aspects as real-time feature
  - E.g. deadline or period

Allows expressing real-time constraints on component interfaces and connectors
- Applicable whether component are active or passive

For active components, introduces specific models of computation
- Alternative MoCC can be defined
- "Common" active objects MoCC
Details of the « RtUnit » and « PpUnit »

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

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

RT units can manage a pool of computing resources:

Refers to a behavior denoting an possible operational modes and their related transitions.
Example of RtUnit and PpUnit

CruiseControlSystem

- CruiseController
  - tgSpeed: Speed
  - «rtService» {exeKind=deferred} start()
  - «rtService» {exeKind=deferred} stop()

- ObstacleDetector
  - startDetection()
  - stopDetection()

- Speedometer
  - getSpeed(): Speed

- Speed

isMain = true
main = start

isDynamic = false
isMain = false
poolSize = 10
poolPolicy = create
Example of RtUnit and PpUnit (seq.)

```
stim « modeBehavior » CruiseControlModes

CruiseControlModes « mode » NominalMode

CruiseControlModes « mode » DegradedMode

CruiseControlModes « modeTransition » [NodeCrash]/ReconfigToDegraded
```

```
CruiseControlSystem

rtUnit » CruiseController

tgSpeed: Speed

rtUnit » ObstacleDetector

startDetection()
stopDetection()

rtUnit » Speedometer

getSpeed(): Speed
```

```dataType »

Speed
```

```ppUnit »

{concPolicy=guarded}

Speedometer
```

```rtUnit »

CruiseController
```

```rtUnit »

ObstacleDetector
```

```rtUnit »

Speedometer
```

```ppUnit »

Speed
```

isMain = true
main = start
operationalMode = CruiseControlModes

isDynamic = false
isMain = false
poolSize = 10
poolPolicy = create
The four pillars of MARTE

- **Pillar 1: Architecture Modeling**
  - **GCM**: for architecture modeling based on components interacting by either messages or data.
  - **Alloc**: for specifying allocation of functionalities to entities realizing them.

- **Pillar 2: QoS-aware Modeling**
  - **HLAM**: for modeling high-level RT QoS, including qualitative and quantitative concerns.
  - **NFP**: for declaring, qualifying, and applying semantically well-formed non-functional concerns.
  - **Time**: for defining time and manipulating its representations.
  - **VSL**: the Value Specification Language is a textual language for specifying algebraic expressions.

- **Pillar 3: Platform-based Modeling**
  - **GRM**: for modeling of common platform resources at system-level and for specifying their usage.
  - **SRM**: for modeling multitask-based design
  - **HRM**: for modeling hardware platform

- **Pillar 4: Model-based QoS Analysis**
  - **GQAM**: for annotating models subject to quantitative analysis.
  - **SAM**: for annotating models subject of scheduling analysis.
  - **PAM**: for annotating models subject of performance analysis.
Platform-based design: the classical "Y-Chart" scheme

Application Model with Non-Functional Constraints

- Structure and Behavior
- Timing Requirements
- Time- and resource properties and constraints

Platform Models (Libraries)

- RTOS Libraries
- HW Component Models

Allocation

Validation of Non-Functional Properties

Integrated and Validated System Model

Generated Code

Input Files for Analysis

Analysis Tools
Basic pattern for modeling platforms

Object concern

ResourceInstance

context 1
exeServices 0..*

ResourceServiceExecution

Classifier concern

Resource

ownedElement 0..*
owner 0..1

type 1..*

resMult: Integer [0..1]

context 1

ResourceService

instance 0..*
type 1..*
pServices 1..*
What is the Software Resource Modeling Profile (SRM)?

- A UML profile for modeling APIs of RTE software platforms
  - Real Time Operating Systems (RTOS)
  - Dedicated Language Libraries (e.g. ADA)
- BUT it is NOT a new API standard dedicated to the RT/E domain!
  - SRM is the result of a very deep state of the art and of the practices including but not limited to:
    - POSIX, ARINC 653, SCEPTRE, Linux RT, ...
  
  → SRM = a unified mean to describe such existing or proprietary APIs

- In which steps shall I use SRM?
Why shall I use SRM for modeling RTOS APIs?

- **RTOS API modeling with UML is already possible**
  - But, generics UML is lacking RTE native artifacts!
    - No modeling artifacts to describe specific concepts such as tasks, semaphores and mailboxes.
  - Consequently, models relies only on naming conventions
    - Not possible to define tool-assistants manipulating such models!

- **Hence, SRM profile allows:**
  - To model precise multitasking designs
  - To be able to describe generic generative tools
  - To describe SW platform in an unified and standard way
    - SRM profile is a sub-profile of the MARTE GRM
What is supported by the SRM profile?

**Concurrent execution contexts:**
- Schedulable Resource (~Task)
- Memory Partition (~Process)
- Interrupt Resource
- Alarm

**Interactions between concurrent contexts:**
- Communication
  - Shared data
  - Message (~Message queue)
- Synchronization
  - Mutual Exclusion (~Semaphore)
  - Notification Resource (~Event mechanism)

**Hardware and software resources brokering:**
- Drivers
- Memory management
Snapshot of the UML extensions provided by SRM

**SRM::SW_Concurrency**

- SwSchedulableResource
- EntryPoint
- SwTimerResource
- InterruptResource
- MemoryPartition
- Alarm

**SRM::SW_Interaction**

- MessageComResource
- NotificationResource
- SharedDataResource
- SwMutualExclusionResource

**SRM::SW_Brokering**

- MemoryBroker
- DeviceBroker
OSEK/VDX standard ([http://www.osek-vdx.org](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 ECUs
- OSEK-NM layer: Network Management
  - Configuration determination and monitoring
- OSEK-OS layer: Operating System
  - API specification of RTOS for automotive ECU
Focus on the OSEK/VDX Task concept

- **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»

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

(Step 1)

(Step 2)

(Step 3)
Example of the OSEK/VDX Event as a NotificationResource

- **Stereotype shown through its icon**

```plaintext
« NotificationResource »
maskElements = Event::mask
clearServices= EventService::ClearEvent
signalServices= EventService::SetEvent
waitServices= EventService::GetEvent

« NotificationResource »
Event
+ mask : UINT64

« interface »
EventService
+ SetEvent(TaskId, Mask)
+ GetEvent(TaskId, Event)
+ WaitEvent(Mask)
+ ClearEvent(Mask)
```

- **Stereotype shown through its shape**

```plaintext
« NotificationResource »
maskElements = Event::mask
clearServices= EventService::ClearEvent
signalServices= EventService::SetEvent
waitServices= EventService::GetEvent

« NotificationResource »
Event

« interface »
EventService
+ SetEvent(TaskId, Mask)
+ GetEvent(TaskId, Event)
+ WaitEvent(Mask)
+ ClearEvent(Mask)
```
Hardware platform modeling with MARTE

- **Hierarchical taxonomy of hardware concepts**
  - 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 view & Physical view
• **Provides a functional description**

• **Based on a functional classification of hardware resources:**

```plaintext
- **HwComputing**
  - « HwProcessor », « HwPLD », « HwASIC »

- **HwStorage**
  - « HwCache », « HwRAM », « HwROM », « HwDrive »
  - « HwMMU », « HwDMA »

- **HwDevice**
  - « HwDevice », « HwSupport »
  - « HwI/O »

- **HwCommunication**
  - « HwBridge »
  - « HwMedia », « HwBus »
  - « HwArbiter »
  - « HwClock », « HwTimer »

- **HwTiming**
```
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

```
* 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
```

```
```

```
```

kind: {Card, Channel, Chip, Port}
HRM profile -- HwMemory
Details of the Memory subprofile of HRM
HRM profile -- HwMemory – HwCache (seq.)

- **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.

<table>
<thead>
<tr>
<th>stereotype</th>
<th>HwCache</th>
</tr>
</thead>
<tbody>
<tr>
<td>level : NFP_Natural = 1</td>
<td></td>
</tr>
<tr>
<td>type : CacheType</td>
<td></td>
</tr>
<tr>
<td>structure : CacheStructure</td>
<td></td>
</tr>
<tr>
<td>repl_Policy : Repl_Policy</td>
<td></td>
</tr>
<tr>
<td>writePolicy : WritePolicy</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enumeration</th>
<th>Repl_Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td></td>
</tr>
<tr>
<td>NFU</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Undefined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enumeration</th>
<th>WritePolicy</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteBack</td>
<td></td>
</tr>
<tr>
<td>WriteThrough</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Undefined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enumeration</th>
<th>CacheType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Instruction</td>
<td></td>
</tr>
<tr>
<td>Unified</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Undefined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dataType</th>
<th>CacheStructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbSets : NFP_Natural</td>
<td></td>
</tr>
<tr>
<td>blocSize : NFP_DataSize</td>
<td></td>
</tr>
<tr>
<td>associativity : NFP_Natural</td>
<td></td>
</tr>
</tbody>
</table>
• Specifies the cache level.
  ▪ Default value is 1
• 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
  ▪ \( \text{memorySize} = \text{nbSets} \times \text{blocSize} \times \text{associativity} \)
HRM profile -- HwMemory -- HwCache (seq.)

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

```
<< stereotype >>
HwCache

level : NFP_Natural = 1
type : CacheType
structure : CacheStructure
repl_Policy : Repl_Policy
writePolicy : WritePolicy
```

```
<< enumeration >>
Repl_Policy

LRU
NFU
FIFO
Random
Other
Undefined

<< enumeration >>
WritePolicy

WriteBack
WriteThrough
Other
Undefined

<< enumeration >>
CacheType

Data
Instruction
Unified
Other
Undefined

<< data type >>
CacheStructure

nbSets : NFP_Natural
blocSize : NFP_DataSize
associativity : NFP_Natural
```
Very early Hw Architecture Description

- **SMP (Symmetric MultiProcessing) hardware platform**
  - 4 identical processors
    - Unified Level 2 cache for each
  - Shared main memory (SDRAM)
  - Central FSB (Front Side Bus)
  - DMA (Direct Memory Access)
  - Battery
HRM usage example: Logical view 1

- « hwLogical::hwResource »
  - SMP

  - « hwProcessor »
    - CPU

  - « hwCache »
    - UL2
      - {level = 2, type = unified}

  - « hwBus »
    - FSB
      - {isSynchronous = true}

  - « hwSupport »
    - Battery

  - « hwRAM »
    - SDRAM
      - {isSynchronous = true, isStatic = false}

  - « hwDMA »
    - DMA
      - {nbChannels = 4}
HRM usage example: Logical view 2

```
« hwLogical::hwResource »
smp : SMP

« hwProcessor »
cpu1 : CPU
{frequency = 800Mhz}

« hwCache »
L2 : UL2
{memorySize = 512kB}

« hwProcessor »
cpu2 : CPU
{frequency = 800Mhz}

« hwCache »
L2 : UL2
{memorySize = 512kB}

« hwProcessor »
cpu3 : CPU
{frequency = 800Mhz}

« hwCache »
L2 : UL2
{memorySize = 512kB}

« hwProcessor »
cpu4 : CPU
{frequency = 800Mhz}

« hwCache »
L2 : UL2
{memorySize = 512kB}

« hwBus »
fsb : FSB
{frequency = 133Mhz, wordWidth = 128bit}

« hwSupport »
battery : Battery

« hwDMA »
dma : DMA
{managedMemories = sdrm}

« hwRAM »
sdrm : SDRAM
{frequency = 266Mhz, memorySize = 256MB}
```
## HRM Usage Example: Physical View 1

<table>
<thead>
<tr>
<th>« hwComponent »</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP</td>
<td>« hwComponent »</td>
<td>« hwComponent »</td>
<td>« hwComponent »</td>
</tr>
<tr>
<td>{kind = Card}</td>
<td>CPU [4]</td>
<td>FSB</td>
<td>DMA</td>
</tr>
<tr>
<td>{kind = Chip}</td>
<td></td>
<td>{kind = Channel}</td>
<td>{kind = Chip}</td>
</tr>
<tr>
<td></td>
<td>« hwComponent »</td>
<td>« hwComponent »</td>
<td>« hwPowerSupply »</td>
</tr>
<tr>
<td>UL2</td>
<td>FSB</td>
<td>SDRAM</td>
<td>Battery</td>
</tr>
<tr>
<td>{kind = Unit}</td>
<td>{kind = Channel}</td>
<td>{kind = Card}</td>
<td>{kind = Other, capacity = 40Wh}</td>
</tr>
</tbody>
</table>
HRM usage example: Physical view 2

- **hwCard**
  - `smp : SMP`
  - `grid = 4,3`
  - `area = 5000mm²`
  - `r_conditions = (Temperature; Operating: ""; [10°C,60°C])`

- **hwChip**
  - `cpu1 : CPU`
    - `position = [1,1], [1,1]`
    - `staticConsumption = 5W`
  - `cpu3 : CPU`
    - `position = [2,2], [1,1]`
    - `staticConsumption = 5W`

- **hwCard**
  - `sdram : SDRAM`
    - `position = [3,4], [1,1]`
    - `nbPins = 144`

- **hwChannel**
  - `fsb : FSB`
    - `position = [1,4], [2,2]`

- **hwChip**
  - `cpu2 : CPU`
    - `position = [1,1], [3,3]`
    - `staticConsumption = 5W`
  - `cpu4 : CPU`
    - `position = [2,2], [3,3]`
    - `staticConsumption = 5W`

- **hwChip**
  - `dma : DMA`
    - `position = [3,3], [3,3]`

- **hwPowerSupply**
  - `battery : Battery`
    - `position = [4,4], [3,3]`
    - `capacity = 10Wh`
    - `weight = 150g`
The four pillars of MARTE

• Pillar 1: Architecture Modeling
  ▪ **GCM**: for architecture modeling based on components interacting by either messages or data.
  ▪ **Alloc**: for specifying allocation of functionalities to entities realizing them.

• Pillar 2: QoS-aware Modeling
  ▪ **HLAM**: for modeling high-level RT QoS, including qualitative and quantitative concerns.
  ▪ **NFP**: for declaring, qualifying, and applying semantically well-formed non-functional concerns.
  ▪ **Time**: for defining time and manipulating its representations.
  ▪ **VSL**: the Value Specification Language is a textual language for specifying algebraic expressions.

• Pillar 3: Platform-based Modeling
  ▪ **GRM**: for modeling of common platform resources at system-level and for specifying their usage.
  ▪ **SRM**: for modeling multitask-based design
  ▪ **HRM**: for modeling hardware platform

• Pillar 4: Model-based QoS Analysis
  ▪ **GQAM**: for annotating models subject to quantitative analysis.
  ▪ **SAM**: for annotating models subject of scheduling analysis.
  ▪ **PAM**: for annotating models subject of performance analysis.
On how model-based analysis reduces risk

- Repeated evaluation of architectural models (using simulation, formal and informal analyses)
  - Early experience with the design
  - Early detection of potential design flaws $\implies$ less expensive to fix!
• Supports predictive or model-based quantitative analysis
  ▪ Predictive analysis: to detect potentially unfeasible real-time architectures and/or implementations before the realization phase
  ▪ Later analysis: to validate non functional requirements on the final system
• Supports sensitivity analysis
  ▪ to explore and evaluate different design alternatives
• Supports the “Y-chart” approach
  ▪ Application model vs. platform model
• Improves modeling reuse and component-based design
• Rich set of predefined model-based analyses supported
  ▪ Performance and schedulability models
An analysis context is the root concept used to collect relevant quantitative information for a specific analysis scenario.
Stereotypes define “analysis” abstractions:
workload events, scenarios,
schedulable resources, protected resources,
exection and communication hosts.
**Stereotype attributes** define pre-defined NFPs
- event arrival patterns,
- host demand, resource usage, speed factor

Pattern = periodic (value = 50, unit = ms)

Host demand = (value = 5, unit = ms)

Speed factor = 1.0
Sensitivity/Exploration analysis support

- Definition of multiple Analysis Contexts
- Possibility of using NFP VARIABLES
- Possibility of back annotating Results

Resources Platform

exec.host

speedFactor = 2.0

comm.host

broker

Workload Behavior

scenarios

pattern = periodic(value=50, unit=ms)

workload events

hostDemand = (value=5, unit=ms)
Sensitivity/Exploration analysis support

- Definition of multiple Analysis Contexts
- Possibility of using VARIABLES
- Possibility of back annotating Results

workload events

scenarios

pattern=periodic(value=X,unit=ms)

workload, events

hostDemand=(value=Y, unit=ms)
Sensitivity/Exploration analysis support

- Definition of multiple Analysis Contexts
- Possibility of using VARIABLES
- Possibility of back annotating results

\[
\text{pattern}=\text{periodic}( \text{value}=60, \text{unit}=\text{ms})
\]

\[
\text{hostDemand} = (\text{value}=5, \text{unit}=\text{ms})
\]
• GQAM sub-profiles specialize concepts of GQAM to support the “analysis model viewpoint” w.r.t. a specific analysis
  ▪ Stereotypes and attributes to capture specific analysis concepts
  ▪ Special attributes to back annotate analysis results
  ▪ “constraints” to construct analyzable models

• MARTE Schedulability Analysis Model (SAM)
  ▪ Specialized stereotypes, e.g.:
    ▪ End-to-end flows specializing the concept of scenario with an end-to-end deadline property.
    ▪ Schedulability execution hosts that specialize the concept of processing resource with a scheduling algorithm.
  ▪ Specialized attributes
    ▪ Hard end-to-end deadlines for scenarios.
    ▪ Scheduling parameters such as priorities.
    ▪ Worst case execution times.
    ▪ Protected resources access protocols.
  ▪ Specialized attributes to back annotate schedulability analysis results
    ▪ E.g., attribute isSched to back annotate the results of the analysis in the analysis context.
Towards a Schedulability Analysis Model

- Analysis context evaluates the Workload Behavior against a set of available resources
- Schedulability Analysis works on Tasks
Analysis-driven design of real-time embedded SoS

• Scheduling and performance analysis at early stages of architectural design
  ▪ Early detection of unfeasible real-time architectures
  ▪ Reduction costs of errors fixing, Architecture optimization
  ▪ Impact analysis of adding new functionalities

• Integration of commonly used design/component models with analysis tools ➔ Automate the generation of analysis models and the analysis of the results!

Pb1: Input models are often incomplete from analysis point of view
  ➔ Provide tooled guidelines for completion of design model based on MARTE as pivot.

Pb2: Different choices should be allowed, different analysis on same models
  ➔ Enable variability expression of analysis parameters in MARTE.
  ➔ Tooled methodology for “what to describe” & “how to interpret results”.
• **Classic sensor-controller-actuator system**

• **Functional view:**
  - Components and their interactions
  - Information about the three functions on a single processing resource

HECU

Processing data coming from sensor.

Diagnosis function that disables the anti-locking function in case a fault in the subsystem is detected.

Anti-locking brake function calculating the command to send to the actuator
Electronic Brake Control System Workload Model

- Workload events with their arrival patterns
- End-to-end flows and their deadlines
- Worst case execution time on the host
The schedulability analysis context stereotype is applied to a UML view describing the mapping of functional steps to tasks and their synchronization.

- Task mapping of functions
- Task scheduling parameters
  - task1=fp(20)
  - task2=fp(10)

For schedulability analysis a blocking time for synchronizing the access to a shared functional step has to be counted.
Results of the schedulability analysis

- The mapping is schedulable
- End-to-end response times
- Slack for both end-to-end flows

- Task mapping of functions
- Task scheduling parameters
  - task1=fp(20)
  - task2=fp(10)

- Workload events with their arrival patterns

```
« saEndToEndFlow »
isSched=true
schSlack=108.59%
end2EndT=35ms
```

```
« saEndToEndFlow »
isSched=true
schSlack=362.50%
end2EndT=50ms
```
Towards a analysis-driven design process

Functional Model

Optimum Schedulability Analysis Results

OSEK/VDX SRM Model Library

OSEK/VDX multitask design model

Construct a multitask OSEK/VDX design model
Two OSEK extended tasks corresponding to the two tasks provided by Optimum with their scheduling parameters.

OSEK events corresponding to workload events of the schedulability analysis results.

Tasks are automatically started and are put in the "ready" state awaiting for the event to occur.

AntiLock shared resource
• **OIL: OSEK Implementation Language (http://osek-vdx.org)**
  - Provides 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.

```plaintext
OIL_VERSION = "2.5" ;
IMPLEMENTATION OSEK {
}
CPU hecu {
  APPMODE nominalMode {};
  RESOURCE antiLock {
    RESOURCEPROPERTY = STANDARD;
  }
  TASK task1 {
    PRIORITY = 20 ;
    SCHEDULE = FULL ;
    ACTIVATION = 1 ;
    AUTOSTART = TRUE {
      APPMODE = nominalMode ;
      CYCLETIME = 1 ;
    };
    RESOURCE = antiLock;
    EVENT = aquisitionForABS;
  }
  TASK task2 {
    PRIORITY = 10 ;
    SCHEDULE = FULL ;
    ACTIVATION = 1 ;
    AUTOSTART = TRUE {
      APPMODE = nominalMode ;
      CYCLETIME = 1 ;
    };
    RESOURCE = antiLock;
    EVENT = aquisitionForDiagnosis;
  }
  ...
```
www.eclipse.org/papyrus

New version released in July 14th, 2010!
For specific questions on Papyrus, MARTE and SysML, do not hesitate to contact me:

Sebastien.Gerard@cea.fr

This work was partly sponsored by both next projects of EC 7th framework program:

- ADAMS Action Support, www.adams-project.org
- INTERESTED, www.interested-ip.eu