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
Acknowledgments

- 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 the affiliated Artist workshop "Formalisms for architectural description and visualization" held within the CPS Week 2010, Stockholm, Sweden, March 18-19, 2010.
The Domain (bis)

- **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
Going further for developing modern complex systems of systems requires new advanced and innovative methods!

A Standards-, Architecture-, and Model-based Approach

Standards

Component Paradigm

Model-based Engineering
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.
  - 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.

Component-based approaches are then ideal candidates for promoting architecture usage in development process.
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.**
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
UML2 is a chameleon!

- 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.

Domain Specific Modeling by profiling the UML2!
Domain specific modeling with UML

- **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.
On UML profiles in one slide!

**Profile definition**
(Language definition level)

- **Specific notation**
  - « metaclass »
    - UML::Class

- **Specific properties**
  - « stereotype »
    - Semaphore
    - Limit: Integer
    - getSema: Operation
    - relSema: Operation

- **Usage constraint**
  - « Constraint »
    - Limit < UpperLimit

**Profile application**
(User model level)

- « semaphore »
  - SpeedDataLock

Ps: Slides credited to Bran Selic
Outlines of SysML

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

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

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

• **Aligned with the ISO AP-233 standard and connected to Modellica**
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.

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!
• **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.
An example of architecture view

- **Atomic flow ports**

- **Non-atomic flow ports** typed by flow specification!
Two reminders on UML

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

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

  ![Graphical mark denoting a behavior port.](image-url)
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.)

In behavior flow port conveying integer values

Activity denoting the classifier behavior of Regulator

AcceptEventAction with a trigger based on a « DataEvent ».
Pull semantics of MARTE’s flow port

- 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.

![Diagram of Pull semantics of MARTE’s flow port]
Semantics view of MARTE’s flow port: Pull semantics (seq.)

- 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 »

- « metaclass » Property
- « enumeration » DataPoolOrderingKind
  - FIFO
  - LIFO
  - UserDefined

ordering : DataPoolOrderingKind [1] = FIFO

- « stereotype » DataPool
- « metaclass » Behavior

insertion
[0..1] selection
[0..1]

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

• Example of usage

- CarSpeedRegulator
  -spm:Speedometer [1]

- regulator
  - rgm: Regulator [1]

- « dataPool »
  - currentSpeed: Integer [*]

- « dataPool » ordering = UserDefined
  - insertion = Replace
  - selection = LastIsBest

Replace

input
write input
on currentSpeed

LastIsBest
read currentSpeed
output
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
• 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»</td>
</tr>
<tr>
<td>Thread</td>
</tr>
<tr>
<td>«storageResource»</td>
</tr>
<tr>
<td>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>
<tr>
<td>«allocated» {kind=application} SpeedController</td>
</tr>
<tr>
<td>«allocated» {kind=application} CarSpeed</td>
</tr>
</tbody>
</table>

- A platform model

<table>
<thead>
<tr>
<th>OperatingSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>«allocated» {kind=executionPlatform} «schedulableResource» Thread</td>
</tr>
<tr>
<td>«allocated» {kind=executionPlatform} «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
    - «allocate»
      - `{nature=timeScheduling}
      - Thread`
    - «allocate»
      - `{nature=spatialDistribution}
      - 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
MARTE alternative for allocation: Assignement

Allocation example with « allocate »

ApplicationModel

- «allocated» {kind = application}
- «computingResource»
  Processor

PlatformModel

- «allocated» {kind = executionPlatform}
- «computingResource»
  CPU

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

Allocation example with « assign »

ApplicationModel

- «allocated» {kind = application}
- «computingResource»
  Processor

AssignmentsModel

- «assign» {nature=timeScheduling}

PlatformModel

- «allocated» {kind = executionPlatform}
- «computingResource»
  CPU

- 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>

- **Step 2:** Instance specification and values

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

How to specify units and other qualifiers?

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 \( n^{\text{th}} \) and the \( n^{\text{th}}+10 \) 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
Annotating NFPs in Tagged Values

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.
Defining NFPs in Slots

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.
Defining NFPs in Constraints

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.
• **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 defines extended data types

Declaration example...

- **boundedSubtype**
  - baseType: Integer
  - minValue = -480000
  - maxValue = +480000
  - Long

- **dataType**
  - **intervalType**
  - intervalAttrType = bound
  - IntegerInterval
  - bound: Integer [2]

- **collectionType**
  - collectionAttrType = vectorElement
  - IntegerVector
  - vectorElement: Integer [0..*]

- **collectionType**
  - collectionAttrType = matrixElement
  - IntegerMatrix
  - matrixElement: IntegerVector [0..*]

- **tupleType**
  - Power
  - value: Real
  - expr: VSL_Expression
  - unit: PowerUnitKind
  - source: SourceKind

- **choiceType**
  - ArrivalPattern
    - periodic: PeriodicPattern
    - sporadic: SporadicPattern

Usage example...

**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}, {1,5}
  - shape = {2,3}, {1,5}
  - consumption = (\( e^{-x^2} \) \cdot 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><strong>Real Number</strong></td>
<td>1.2E-3 //scientific notation</td>
</tr>
<tr>
<td><strong>DateTime</strong></td>
<td>#12/01/06 12:00:00# //calendar date time</td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td>{1, 2, 88, 5, 2} //sequence, bag, ordered set..</td>
</tr>
<tr>
<td></td>
<td>{{1,2,3}, {3,2}} //collection of collections</td>
</tr>
<tr>
<td><strong>Tuple and choice</strong></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><strong>Interval</strong></td>
<td>[1..251] //upper opened interval between integers</td>
</tr>
<tr>
<td></td>
<td>[$A1..$A2] //interval between variables</td>
</tr>
<tr>
<td><strong>Variable declaration &amp; Call</strong></td>
<td>io$var1 //input/output variable declaration</td>
</tr>
<tr>
<td></td>
<td>var1 //variable call expression.</td>
</tr>
<tr>
<td><strong>Arithmetic Operation Call</strong></td>
<td>+(5.0,var1) //&quot;add&quot; operation on Real datatypes</td>
</tr>
<tr>
<td></td>
<td>5.0+var1 //infix operator notation</td>
</tr>
<tr>
<td><strong>Conditional Expression</strong></td>
<td>((var1&lt;6.0)?(10^6):1) //if true return 10 exp 6,else 1</td>
</tr>
</tbody>
</table>
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.
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 ?**
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 »
- « InterruptResource »
- « MemoryPartition »
- « Alarm »
- « SwTimerResource »

**SRM::SW_Interaction**

- « MessageComResource »
- « NotificationResource »
- « SharedDataResource »
- « SwMutualExclusionResource »

**SRM::SW_Brokering**

- « MemoryBroker »
- « DeviceBroker »
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

Diagram:

- `SRM` imports `GRM`
- `GRM` imports `SW_ResourceCore`
- `SW_ResourceCore` imports `SW_Concurrency`, `SW_Interaction`, `SW_Brokering`
Details of «SwSchedulableResource»

- **Semantic** (from MARTE::SRM::Concurrence package)
  - Resource which executes, periodically or not, concurrently to other concurrent resources
  
  \[
  \Rightarrow \text{SRM artifacts for modeling OSEK-VDX Task!}
  \]

- **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)
HRM structure -- Logical modeling

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

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

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

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

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

  - **HwTiming**
    - « HwClock », « HwTimer »
HRM structure -- Physical modeling

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

```
  - HwLayout
    - HwComponent
      - kind : {Card, Channel, Chip, Port}
  - HwPower
    - HwPowerSupply
    - HwCoolingSupply
```
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.**

### HwCache

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

### Repl_Policy

- LRU
- NFU
- FIFO
- Random
- Other
- Undefined

### WritePolicy

- WriteBack
- WriteThrough
- Other
- Undefined

### CacheType

- Data
- Instruction
- Unified
- Other
- Undefined

### 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

```
+-----------------+        +-----------------+        +-----------------+        +-----------------+        +-----------------+
|                 |        |                 |        |                 |        |                 |        |
| CPU             | 4      | FSB             | 1      | DMA             | 1      | SDRAM            | 1      | Battery         |
|                 |        |                 |        |                 |        |                 |        |                 |
| « hwResource »  |        |                 |        |                 |        | « hwResource »  |        |                 |
| UL2             | 1      |                 |        |                 |        | SDRAM            | 1      |                 |
```

Only HwResources
HRM usage example: Logical view 2

```
< hwLogical:hwResource >
  smp : SMP

< hwProcessor >
  cpu1 : CPU
    {frequency = 800Mhz}

< hwProcessor >
  cpu2 : CPU
    {frequency = 800Mhz}

< hwProcessor >
  cpu3 : CPU
    {frequency = 800Mhz}

< hwProcessor >
  cpu4 : CPU
    {frequency = 800Mhz}

< hwCache >
  l2 : UL2
    {memorySize = 512kB}

< hwCache >
  l2 : UL2
    {memorySize = 512kB}

< hwCache >
  l2 : UL2
    {memorySize = 512kB}

< hwCache >
  l2 : UL2
    {memorySize = 512kB}

< hwBus >
  fsb : FSB
    {frequency = 133Mhz, wordWidth = 128bit}

< hwSupport >
  battery : Battery

< hwDMA >
  dma : DMA
    {managedMemories = sdram}

< hwRAM >
  sram : SDRAM
    {frequency = 266Mhz, memorySize = 256MB}
```
<table>
<thead>
<tr>
<th>«hwComponent»</th>
<th>«hwComponent»</th>
<th>«hwComponent»</th>
<th>«hwComponent»</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU [4]</td>
<td>FSB</td>
<td>DMA</td>
<td>Battery</td>
</tr>
<tr>
<td>{kind = Chip}</td>
<td>{kind = Chip}</td>
<td>{kind = Chip}</td>
<td>{kind = Other, capacity = 40Wh}</td>
</tr>
<tr>
<td>UL2</td>
<td>SDRAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{kind = Unit}</td>
<td>{kind = Card}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMP</td>
<td></td>
</tr>
<tr>
<td>{kind = Card}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HRM usage example: Physical view 2

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

- **`` hwCard ``**
  - **smp : SMP**

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

- **`` hwChip ``**
  - **cpu1 : CPU**
    - **position = [1,1], [1,1]**
    - **staticConsumption = 5W**
  - **cpu3 : CPU**
    - **position = [2,2], [1,1]**
    - **staticConsumption = 5W**
  - **cpu2 : CPU**
    - **position = [1,1], [3,3]**
    - **staticConsumption = 5W**
  - **cpu4 : CPU**
    - **position = [2,2], [3,3]**
    - **staticConsumption = 5W**
  - **dma : DMA**
    - **position = [3,3], [3,3]**

- **`` hwPowerSupply ``**
  - **battery : Battery**
  - **position = [4,4], [3,3]**
  - **capacity = 10Wh**
  - **weight = 150g**
  - **flash icon**
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 \( \Rightarrow \) less expensive to fix!
MARTE’s General Quantitative Analysis Modeling (GQAM)

- **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.
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”.

**Computed assisted model conversion**

Results conversion back to models
• **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

- **End-to-end flows and their deadlines**
- **Workload events with their arrival patterns**
- **Worst case execution time on the host**

- **End-to-end flow**: end2EndD = (60, ms)
- **Workload event**: arrivalPattern = periodic(60, ms)
- **Step execution time**: execTime = (10, ms), host=HECU
- **Step**: execTime = (15, ms), host=HECU
- **Fault diagnosis flow**: arrivalPattern = aperiodic(100, ms)
- **Step**: execTime = (10, ms), host=HECU
The schedulability analysis context stereotype is applied to a UML view describing the mapping of functional steps to tasks and their synchronization.

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

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

### SaTaskMapping

- « saAnalysisContext »
  - workload=ElectronicBrakeControlWorkload
  - platform=SaResources
  - isSched=true

- « saEndToEndFlow »
  - isSched=true
  - schSlack=108,59%
  - end2EndT=35ms

- « saEndToEndFlow »
  - isSched=true
  - schSlack=362,50%
  - end2EndT=50ms

Shared resource

**Task Mapping**

- task1=fp(20)
  - acquisitionForAbs
  - DataProcessingBehavior
  - AntiLockBehavior

- task2=fp(10)
  - acquisitionForDiagnosis
  - DiagnosisBehavior
  - AntiLockBehavior

**Workload Events**

- « gaWorkloadEvent »
  - arrivalPattern = periodic(60, ms)
- « gaWorkloadEvent »
  - arrivalPattern = aperiodic(200, ms)

**Schedule Parameters**

- « saStep »
  - execTime = (10, ms)
  - host=HECU
  - sharedRes=[AntiLockBehavior]
Towards a analysis-driven design process

Construct a multitask OSEK/VDX design model
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.
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