MARTE:
the new OMG standard UML profile for RTE modeling

MARTE Workshop (DATE’2008), Munich, Germany, March 14\textsuperscript{th}, 2008

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

(This keynote reused materials extracted from the full MARTE tutorial realized by CEA (French Energy Atomic Agency), Thales Research & Technology and INRIA.

\[ \rightarrow \text{http://www.omgmarте.org/} \]

- CEA LIST (contact: sebastien.gerard@cea.fr)
  - Huascar Espinoza, Sébastien Gérard, Safouan Taha and Frédéric Thomas.

- INRIA (contact: Robert.De_simone@sophia.inria.fr)
  - Charles André, Robert de Simone, Pierre Boulet

- Thales TRT (contact: Laurent.rioux@thalesgroup.com)
  - Madeleine Faugère, Laurent Rioux, Sebastien Demathieu
Agenda

- MDD for ES & MARTE in a nutshell
- **Non-Functional Properties and Time**
- CBSE & other high-level modelling for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
Model: first citizen of MDD

*Phil Bernstein, "A Vision for Management of Complex Systems". A model is a complex structure that represents a design artifact such as a relational schema, an interface definition (API), an XML schema, a semantic network, a UML model or a hypermedia document.

*OMG, "UML Superstructure". A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. This purpose determines what is included in the model and what is relevant. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model at the appropriate level of detail.

*OMG, "MDA Guide". A formal specification of the function, structure and/or behavior of an application or system.

*Steve Mellor, et al., “UML Distilled” A model is a simplification of something so we can view, manipulate, and reason about it, and so help us understand the complexity inherent in the subject under study.

*Anneke Kleppe, et. al. "MDA Explained" A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.

*Chris Rastrick et al., "Model Driven Architecture with Executable UML" A formal representation of the function, behavior, and structure of the system we are considering, expressed in an unambiguous language.

*J. Bézivin & O. Gerbé, “Towards a Precise Definition of the OMG/MDA Framework” A simplification of a system built with an intended goal in mind: The model should be able to answer questions in place of the actual system.

One possible definition
- A reduced/abstract representation of some system that highlights the properties of interest from a given viewpoint.
- The viewpoint defines concern, scope and detail level of the model.
A Bit of Modern Software…

SC_MODULE(producer)
{
    sc_outmaster<int> out1;
    sc_in<bool> start; // kick-start
    void generate_data ()
    {
        for(int i =0; i <10; i++) {
            out1 =i ; //to invoke slave;
        }
    }
    SC_CTOR(producer)
    {
        SC_METHOD(generate_data);
        sensitive << start;}
}

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

SC_MODULE(top) // container
{
    producer *A1;
    consumer *B1;
    sc_link_mp<int> link1;
    SC_MODULE(top)
    {
        A1 = new producer(“A1”);
        A1.out1(link1);
        B1 = new consumer(“B1”);
        B1.in1(link1);}};

Can you spot the architecture?

(Extracted from B. Selic presentation during Summer School MDD For DRES 2004 (Brest, September 2004)
…and its UML Model

Can you spot the architecture?

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

- Models can be refined continuously until the specification is complete

```
void generate_data()
{for (int i=0; i<10; i++)
{out1 = i;}}
```
Model-Driven Style of Development (MDD)

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

(1) ABSTRACTION

Realm of modeling languages

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

(2) AUTOMATION

Realm of tools

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

(Extracted from B. Selic presentation during Summer School MDD For DRES 2004 (Brest, September 2004)
Two Kinds of OMG Modeling Languages

- **General Purpose Language (GPL)**
  - E.g., UML2, SysML and CCM

- **Domain Specific Language**
  - E.g. LwCCM, UML profile for SoC and MARTE

**M2 Level**

**GPL (General Purpose Language)**
- UML2 meta-model
- «profile» UML profile for SysML
- «reference»

**DSL (Domain Specific Language)**
- «profile» UML profile for SoC
- «profile» UML profile MARTE
- «reference»
Basics of UML profiling

- Lightweight Extension / Specialization of the UML metamodel
  - For particular application/domain concerns
- Stereotypes
  - Extension / Specialization of existing metaclasses
  - Contains a set of “tagged values” (i.e. domain specific properties)
- Constraints
  - On the usage of stereotypes
  - On the usage of constructs provided by the source metamodel
- Notation options (i.e. icons, figures)
- Example: A UML profile for C++
The ProMARTE Team

- Public OMG web site for MARTE
  - www.omgmarте.org

- Partners
  - Industrials
    - Alcatel*, France Telecom, Lockheed Martin* and Thales*
  - Tool vendors
  - Academics
    - Carleton University, Commissariat à l’Energie Atomique, ESEO, ENSIETA, INRIA, INSA from Lyon, Software Engineering Institute (Carnegie Mellon University), Universidad de Cantabria

* Submitter to OMG UML Profile for MARTE RFP
OMG Standardization Process

MARTE FTF: Comments due for 22 December 2007!

- **WP**
  - yes
  - no

- **RFI**
  - yes
  - no

- **RFP**
  - yes
  - no

- **Prop-1**
  - yes

- **Prop-n**

- **Prop**
  - yes
  - no

---


Final Adopted Specification Publication: 6 August 2007

Comments Due: 22 December 2007

Recommendations and Report Deadline: 4 July 2008

---

Beta 1

FTF

MARTE v1.0
SPT was the first OMG’s UML profile for Real-Time Systems:
- Support for Schedulability Analysis with RMA-type techniques
- Support for Performance Analysis with Queuing Theory and Petri Nets
- A rich model for “metric” Time and Time Mechanisms

Several improvements were required:
- Modeling HW and SW platforms, Logical Time, MoCCs, CBSE…
- Alignment to UML2, QoS&FT, MDA,…
- SPT’s constructs were considered too abstract and hence hard to apply
- ...

Hence, a Request For Proposal for a new profile was issued.
Design Pattern Adopted for MARTE

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

Stage 2 → Mapping of MARTE domain models towards UML2
- Purpose: MARTE domain models design as a UML2 extensions
- Techniques: UML2 profile

« metamodel »
Marte domain model

« profile »
Marte profile

« refine »

« reference »
UML2 metamodel
MARTE Overview

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

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

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

Extracted from S.Gerard (ECRTS07)
MARTE Foundations

- **MARTE foundations**
  - Define a set of basic concepts for MDD of DRES
  - Intended to be used for as basic layer at OMG for later extensions dedicated to RT/E domain

- **Consists of 5 sub-profiles**

```
« profile »
MARTE_Foundations

« profile »
NFPs
(Non-FunctionalProperties)

« profile »
GRM
(GeneralResourceModeling)

« profile »
Alloc
(Allocation)

« profile »
Time

« profile »
GCM
(GeneralComponentModel)
```
Agenda of the Tutorial

- MDD for ES & MARTE in a nutshell
- Non-Functional Properties and Time
- CBSE & other high-level modelling for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
Non-Functional Properties (NFPs)

Non-functional properties denote quantitative aspects of system’s behaviors
(E.g., performance, memory usage and power consumption)

- Nature of NFPs
  - Quantitative: magnitude + unit (E.g., energy, data size and duration)
  - Qualitative (E.g., periodic or sporadic event arrival patterns)

- NFP values need to be qualified
  - E.g. source, statistical measure, precision,…

- NFPs need to be parametric and derivable
  - Variables: placeholders for unknown values
  - Expressions: math. and time expressions

- NFPs need clear semantics
  - Predefined NFPs (E.g., end-to-end latency, processor utilization)
  - User-specific NFPs (but still unambiguously interpreted!)
Introduction to the MARTE’s NFPs Framework

UML lacks modeling capabilities for NFPs!!

Value qualifiers?

Measures?

Annotation mechanism?

NFP Libraries?

And UML expression syntax is also not sufficient!!

Variables?

Structured Values?

Complex time expressions?

Data Type System?

Value Specification Language (VSL)
The MARTE’s NFP sub-profile

- Relies on the ability to put additional information on models
- Three possible model-based annotation mechanisms in UML
  - Value of stereotype properties
  - Slots value of classifier instance
  - Constraints
Annotating NFPs in Tagged Values

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

2) Define NFP-like extensions
   - Define stereotypes and their attributes using NFP types

3) Specify NFP values
   - Apply stereotypes and specify their attribute values using VSL
Annotating NFPs in Slots

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

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

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

Model-specific NFPs

- Declare NFP types
- Declare NFPs
- Specify NFP values
Annotating NFPs in Constraints

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

2) Declare NFPs
- Define classifiers and their attributes using NFP types

3) Specify NFP values
- Create Constraints to define assertions on NFP values using VSL

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

- Declare NFPs
  - Define classifiers and their attributes using NFP types

- Specify NFP values
  - Create Constraints to define assertions on NFP values using VSL
The MARTE’s NFP Modeling Framework

- Three main language extensions to the UML syntax
  1) Rich grammar for extended expressions
  2) Stereotypes for extended expressions
  3) Complex time expressions

Value Specification Language (VSL)
## Basic Textual Expressions in VSL

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

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

Sd DataAcquisition

:Controller

Duration constraint

:Sensor

Variable declaration denoting a duration (UML DurationObservation)

Bounded instant interval constraint

Variable declaration denoting an instant (UML Time Observation)

Bounded duration interval constraint
MARTE time models relies on partial ordering of instants

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

Defines literals used to specify the discrete or dense nature of a time value.

- **TimeNatureKind**
  - discrete
  - dense

Defines literals used to specify the way to interpret a time expression: either as a duration or as an instant.

- **TimeInterpretationKind**
  - duration
  - instant

Define time related extensions of UML

- **modelLibrary**
  - TimeTypesLibrary

Defines time related facilities for user models

- **modelLibrary**
  - TimeLibrary

- **profile**
  - Time
« ClockType »

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

```plaintext
{ nature = discrete, unitType = TimeUnitKind, resolAttr = resolution, getTime = currentTime } ChronometricClock
resolution: Real {readOnly}
currentTime( ) : Real

{ nature = dense, unitType = TimeUnitKind, getTime = currentTime } IdealClock
currentTime( ) : Real
```
« Clocks »

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

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

currentTime( ): Real
```

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

currentTime( ): Real
```
Examples of «TimedInstantObservation» and «TimedDurationObservation»

currentTime( ): Real

currentTime( ): Real

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

IdealClock

« clock »
{ unit = s }

idealClk: IdealClock

Sd DataAcquisition

:Controller

start()

@t1

« TimedInstantObservation »
{ on = idealClk, obsKind = send }

« TimedDurationObservation »
{ on = idealClk }

&d1
d1<=(1, ms)

acquire()

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

ack()

sendData (data)

:Sensor
Agenda

- MDD for ES & MARTE in a nutshell
- Non-Functional Properties and Time
- CBSE & other high-level modelling for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
The MARTE General Component Model

- Defined within MARTE to support CBSE and to provide a common denominator among various existing component models, which in principle do not target exclusively the RTE domain.

- Introduce no new component-related concepts
  - Has to cope with various component models
    - E.g., UML2, SysML, Spirit, AADL, Lightweight-CCM, EAST-ADL2, Autosar

- Main features
  - Mainly refined UML structured classes, on top of which a support for SysML blocks has been added
  - But also compatible with a support for Lightweight-CCM, AADL and EAST-ADL2, Spirit and Autosar
  - Shortcuts for UML2 modeling of components/composites diagrams
The MARTE GCM Sub-profile

- « profile » GeneralComponentModel

- « metaclass » InvocationAction
- « stereotype » SendFlowAction

- « metaclass » Property
- « metaclass » BehavioralFeature

- « metaclass » Port
- « stereotype » FlowPort

- « metaclass » Interface
- « stereotype » MessagePort

- « metaclass » FlowSpecification
- « stereotype » FlowBFeature

- « metaclass » BFeatureSpecification
- « stereotype » BFeatureKind

- « enumeration » DirectionKind
- « enumeration » BFeatureKind

: concepts for data-based flow communication

: concepts for message-based flow communication

The MARTE GCM Sub-profile: concepts for data-based flow communication; concepts for message-based flow communication
Example of a MARTE component model

SpeedRegulatorInterfaces

- `import` rgm: Regulator
- `messagePort` rp: RegInterface
  - `flowPort` outSpeed: Integer [1]
  - `messagePort` regOn: Start [1]

SignalDeclarations

- `signal` Start
  - targetSpeed: Integer [1]

CarWithSpeedRegulator

- `import` « import »

CarSpeedRegulator

- `flowPort` spm: Speedometer [1]
- `messagePort` regOn: Start [1]
- `flowPort` inSpeed: Integer [1]
- `messagePort` rgmtRegulator [1]
- `messagePort` engineCmd: ECInterface [1]
RTE Model of Computation & Communication

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

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

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

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

### « stereotype » PpUnit
- Generalization of UML2 Passive Objects
  - Concurrency policy specified either locally or globally
  - Processing is either immediateRemote or deferred
Quantitative RT features modeling

« RealTimeFeature »

« stereotype »
rtf

utility: UtilityType
occKind: ArrivalPattern
tRef: TimedInstantObservation
relDl: NFP_Duration
absDl: NFP_DateTime
boundDl: NFP_BoundedDuration
rdTime: NFP_Duration
miss: NFP_Percentage
priority : NFP_Integer

« metaclass »
UML2::
BehavioralFeature,
Action,
Message,
Signal

« stereotype »
RteConnector

InternalStructures::Connector

« metaclass »
MARTE_Library::MARTE_DataTypes::
TransmModeKind

« enumeration »
simplex
half-duplex
full-duplex

« stereotype »
RteConnector

bandwidth: NFP_DataTxRate
packetT: NFP_Duration
blockT: NFP_Duration
transmMode: TransmModeKind
Dynamics aspects of « RtUnit » and « PpUnit »

- **Services specification**

  ```
  « metaclass » UML2::BehavioralFeature
  
  while (queueSchedPolicy = SchedPolicyKind) 
  queueSize : Integer
  msgMaxSize : NFP_DataSize
  
  « metaclass » BehavioralFeature
  ```

- **Classifier behavior specification**

  ```
  « stereotype » RtBehavior
  
  « metaclass » CommonBehavior:: BasicBehaviors:: Behavior
  ```

  ```
  « enumeration » SynchronisationKind
  
  isAtomic : Boolean [1] = false
  concPolicy : ConcurrencyKind
  exeKind : ExecutionKind
  synchKind : SynchronizationKind
  
  « enumeration » ConcurrencyKind
  reader
  writer
  parallel
  ```

  ```
  « enumeration » ExecutionKind
  deferred
  remoteImmediate
  localImmediate
  ```

  ```
  « enumeration » SchedPolicyKind
 EarliestDeadlineFirst
LIFO
FixedPriority
LeastLaxityFirst
RoundRobin
TimeTableDriven
Undef
Other
```
Usage examples of the RTEMoCC extensions (1)

CruiseControlSystem

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

- « rtUnit » ObstacleDetector
  - startDetection()
  - stopDetection()

- « ppUnit » {concPolicy=guarded} Speedometer
  - getSpeed(): Speed

- « dataType » Speed

- isMain = true
  - main = start

- isDynamic = false
  - isMain = false
  - poolSize = 10
  - poolPolicy = create

act start

- @t0 {kind=startAction}

- « rtf »
  - tgSpeed = spm->getSpeed()

- occKind = aperiodic()
  - value = (tRef=t0, relDl=(10, ms), miss=(1, %, max))
Usage examples of the RTEMoCC extensions (2)

- `CruiseControlStart`:
  - `occKind = aperiodic ()`
  - `value = (tRef=t0, relDl=(10, ms), miss=(1, %, max))`

- `Speedometer`:
  - `occKind = periodic (period=(10, ms), jitter=(2, us))`
  - `value = (tRef=t0, relDl=(10, ms), miss=(1, %, max))`

- `Speed`:
  - `occKind = periodic (period=(10, ms), jitter=(2, us))`
  - `value = (tRef=t0, relDl=(10, ms), miss=(1, %, max))`
Usage examples of the RTEMoCC extensions (3)

- **computeTrajectory**
  - `computeTrajectory` event at `t0`:
    - `priority=1`
    - `occKind = periodic (period=(10,ms), jitter=(2,us))`
    - `relDl=(3,ms)`
    - `tRef=t0`
    - `miss=(1, %, max)`
    - `syncKind=delayedSynchronous`

- **getLocation**
  - `getLocation` event:
    - `priority=1`
    - `occKind = periodic (period=(10,ms), jitter=(2,us))`
    - `relDl=(4,ms)`
    - `tRef=t0`
    - `miss=(1, %, max)`
    - `syncKind=synchronous`

- **getFlightPlan**
  - `getFlightPlan` event:
    - `priority=1`
    - `occKind = periodic (period=(10,ms), jitter=(2,us))`
    - `relDl=(1,ms)`
    - `tRef=t0`
    - `miss=(1, %, max)`
    - `syncKind=synchronous`
Agenda

- MDD for ES & MARTE in a nutshell
- Non-Functional Properties and Time
- CBSE & other high-level concepts for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
Platforms modeling concern within MARTE
Essence of the GRM Package

Object concern

- ResourceInstance
  - context 1
  - exeServices 0..*

- ResourceServiceExecution
  - instance 0..*

Classifier concern

- Resource
  - resMult: Integer [0..1]
  - context 1
  - ownedElement 0..*
  - owner 0..1

- ResourceService
  - pServices 1..*
  - type 1..*
Allocation of applications on platforms

- « profile » MARTE_Foundations
  - « profile » NFPs (Non-FunctionalProperties)
  - « profile » GRM (GeneralResourceModeling)
    - « profile » Alloc (Allocation)
  - « profile » Time
  - « profile » GCM (GeneralComponentModel)
Allocation & Refinement

**Basic ideas**

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

**Inspired by the SysML allocation, with two additional features:**

- Restricted to allocation of application to execution platform
  - SysML is more general.
- Allocation may own NFPs
A two step process for allocation modeling

- Identify possible sources and targets of allocations

- Define allocation relationship and its features
Allocation example (1)

**Application**

<table>
<thead>
<tr>
<th>mySpeedRegulator : SpeedRegulatorSystem[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpeedController</td>
</tr>
</tbody>
</table>

**RealTimeOperatingSystem**

<table>
<thead>
<tr>
<th>« schedulableResource »</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS_Task</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>« storageResource »</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualMemory</td>
</tr>
</tbody>
</table>
Allocation example (2)

**Application**

<table>
<thead>
<tr>
<th>mySpeedRegulator : SpeedRegulatorSystem [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>« app_allocated » SpeedController</td>
</tr>
<tr>
<td>« app_allocated » CarSpeed</td>
</tr>
</tbody>
</table>

**RealTimeOperatingSystem**

| « schedulableResource, ep_allocated » OS_Task |
| « storageResource, ep_allocated » VirtualMemory |
Allocation example (3)

Application

mySpeedRegulator : SpeedRegulatorSystem [1]

« app_allocated »
SpeedController

« app_allocated »
CarSpeed

RealTimeOperatingSystem

« scheduleableResource, ep_allocated »
OS_Task

« storageResource, ep_allocated »
VirtualMemory

« allocate »
{kind=timeScheduling}

« allocate »
{spatialDistribution}
Allocation example (4)

**Application**

```
mySpeedRegulator : SpeedRegulatorSystem [1]
```

- « app_allocated » SpeedController
- « app_allocated » CarSpeed

**RealTimeOperatingSystem**

```
VirtualMemory
```

- « storage, ep_allocated » OS_Task
- « storageResource, app_allocated » OS_Memory
- « storageResource, app_allocated » Swap
- « storageResource, app_allocated » RootFs

**HardwareProcessingPlatform**

```
CPU
```

- « computingResource, ep_allocated » Memory
- « storageResource, ep_allocated » Bus
- « storageResource, ep_allocated » Disk
Platforms modeling concern within MARTE

MARTE foundations

« profile » GRM (Generic Resource Model)

« refine »

MARTE analysis model

GQAM
(Generic Quantitative Analysis Modeling)

GGAM_Resources

SAM
(Schedulability Analysis Modeling)

SAM_Resources

PAM
(Performance Analysis Modeling)

PAM_Resources

Detailed Resource Modeling

« profile » SRM (Software Resource Modeling)
« profile » HRM (Hardware Resource Modeling)

« refine »
HRM use cases

3 use cases = 3 levels of details
HRM structure

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

- Two modeling **views** to separate concerns
  - Logical / Physical
HRM structure -- Logical modeling

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

  - **HwComputing**: «HwProcessor», «HwPLD», «HwASIC»
  - **HwStorage**: «HwCache», «HwRAM», «HwROM», «HwDrive»
  - **HwDevice**: «HwDevice», «HwSupport», «HwI/O»
  - **HwCommunication**: «HwBridge», «HwMedia», «HwBus»
  - **HwTiming**: «HwClock», «HwTimer»
  - **HwSupport**: «HwDMA», «HwDMA»
  - **HwClock**: «HwMedia», «HwBus»
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

- **HwComponent**
  - kind: {Card, Channel, Chip, Port}

- **HwPower**
  - « HwPowerSupply »
  - « HwCoolingSupply »
HRM profile overview

« profile »
MARTE::GRM

« profile »
HRM

« profile »
HwLogical

HwGeneral

HwComputing

HwCommunication

HwTiming

HwStorage

HwMemory

HwStorageManager

HwDevice

« profile »
HwPhysical

HwGeneral

HwLayout

HwPower

« import »
MARTE::Library::BasicNFP_Types

« import »
MARTE::GRM
HRM profile -- HwMemory

« profile »
MARTE::GRM

« modelLibrary »
MARTE::Library::BasicNFP_Types

« import »

« profile »
HRM

« profile »
HwLogical

HwGeneral

HwComputing

HwCommunication

HwTiming

HwStorage

HwMemory

HwStorageManager

HwDevice

« profile »
HwPhysical

HwGeneral

HwLayout

HwPower

HwPower

HwGeneral
HRM profile -- HwMemory

**HwMemory**

- **« stereotype »** MARTE::GRM::Storage
- **« stereotype »** HwResource
- **« stereotype »** HwMemory
  - memorySize: NFP_DataSize
  - addressSize: NFP_DataSize
  - timings: Timing [*]

**HwMemory**

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

- **« stereotype »** HwRAM
  - organization: MemoryOrganization
  - isSynchronous: NFP_Boolean
  - isStatic: NFP_Boolean
  - isNonVolatile: NFP_Boolean
  - repl_Policy: Repl_Policy
  - writePolicy: WritePolicy

- **« stereotype »** HwROM
  - type: ROM_Type
  - organization: MemoryOrganization
  - sectorSize: NFP_DataSize

- **« stereotype »** HwDrive
  - buffer
    - subsets owned: HW

**« 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

**« data Type »** MemoryOrganization
- nbRows: NFP_Natural
- nbColumns: NFP_Natural
- nbBanks: NFP_Natural
- wordSize: NFP_DataSize

**« enumeration »** ROM_Type
- MaskedROM
- EPROM
- OTPEPROM
- EEPROM
- Flash
- Other
- Undefined
HRM profile -- HwMemory -- HwCache

- 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 : 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>

**enumeration**

- **Repl_Policy**
  - LRU
  - NFU
  - FIFO
  - Random
  - Other
  - Undefined

- **WritePolicy**
  - WriteBack
  - WriteThrough
  - Other
  - Undefined

- **CacheType**
  - Data
  - Instruction
  - Unified
  - Other
  - Undefined

- **CacheStructure**
  - dataType : NFP_Natural
  - blocSize : NFP_DataSize
  - associativity : NFP_Natural

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

#### Specifies the cache level.
- Default value is 1

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

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

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

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

<table>
<thead>
<tr>
<th>« dataType » CacheStructure</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbSets : NFP Natural</td>
</tr>
<tr>
<td>blocSize : NFP DataSize</td>
</tr>
<tr>
<td>associativity : NFP Natural</td>
</tr>
</tbody>
</table>

Specifies the cache level.
- Default value is 1
HRM profile -- HwMemory -- HwCache

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

```
« 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

« enumeration »
CacheStructure

nbSets : NFP_Natural
blocSize : NFP_DataSize
associativity : NFP_Natural

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

```

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

Only HwResources

```
« hwResource »
CPU

« hwResource »
FSB

« hwResource »
DMA

« hwResource »
SDRAM

« hwResource »
Battery

« hwResource »
UL2
```
HRM usage example: Logical view 1

```
« hwLogical::hwResource »
SMP

« hwProcessor »
CPU

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

« hwBus »
FSB
{isSynchronous = true}

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

« hwSupport »
Battery

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

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

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

« hwCache »
l2 : UL2
{memorySize = 512kB}

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

« hwSupport »
battery : Battery

« hwDMA »
dma : DMA
{managedMemories = sdam}

« 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}

« hwRAM »
sdram : SDRAM
{frequency = 266Mhz, memorySize = 256MB}
```
HRM usage example: Physical view 1

- **« hwComponent »**
  - SMP
    - {kind = Card}
  - CPU [4]
    - {kind = Chip}
    - UL2
      - {kind = Unit}
  - FSB
    - {kind = Channel}
  - DMA
    - {kind = Chip}
  - SDRAM
    - {kind = Card}
  - Battery
    - {kind = Other, capacity = 40Wh}
HRM usage example: Physical view 2

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

- **« hwCard »**
  - **smp : SMP**

- **« hwChip »**
  - **cpu1 : CPU**
    - Position = [1,1], [1,1]
    - StaticConsumption = 5W
  - **cpu3 : CPU**
    - Position = [2,2], [1,1]
    - StaticConsumption = 5W

- **« hwChip »**
  - **cpu2 : CPU**
    - Position = [1,1], [3,3]
    - StaticConsumption = 5W
  - **cpu4 : CPU**
    - Position = [2,2], [3,3]
    - StaticConsumption = 5W

- **« hwCard »**
  - **sdram : SDRAM**
    - Position = [3,4], [1,1]
    - NbPins = 144

- **« hwChip »**
  - **dma : DMA**
    - Position = [3,3], [3,3]

- **« hwChannel »**
  - **fsb : FSB**
    - Position = [1,4], [2,2]

- **« hwPowerSupply »**
  - **battery : Battery**
    - Position = [4,4], [3,3]
    - Capacity = 10Wh
    - Weight = 150g
Platforms modeling concern within MARTE
What is the Software Resource Modeling Profile (SRM)?

- A UML profile for modeling APIs of RT/E sw execution supports
  - Real Time Operating Systems (RTOS)
  - Dedicated Language Libraries (e.g. ADA)

- BUT it is NOT a new API standard dedicated to the RT/E domain!

\[ \text{SRM} = \text{a unified mean to describe such existing or proprietary APIs} \]

- In which steps shall I use SRM?
Main expected use cases of SRM

- **Software Designer**
  - Use API model
  - « extend »
  - « include »

- **Execution Platform Provider**
  - Describe execution support API
  - « extend »

- **Methodology Provider**
  - Model Transformation
  - « extend »
  - Code generation
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

- « SwScheduledResource »
- « EntryPoint »
- « InterruptResource »
- « MemoryPartition »
- « SwTimerResource »
- « Alarm »

SRM::SW_Interaction

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

SRM::SW_Brokering

- « MemoryBroker »
- « DeviceBroker »
The OSEK/VDX case study

OSEK/VDX standard (http://www.osek-vdx.org)
- Automotive industry a standard for an open-ended architecture for distributed control units in vehicles

- OSEK/VDX architecture consists of three layers:
  - OSEK-COM layer: Communication
    - Data exchange support within and between electronics control units (ECUs)
  - OSEK-NM layer: Network Management
    - Configuration determination and monitoring
  - OSEK-OS layer: Operating System
    - API specification of RTOS for automotive ECU
Overview of the OSEK/VDX-OS layer

Main characteristics
- A single processor operating system
- A static RTOS where all kernel objects are created at compile time

Main artifacts
- Support for concurrent computing
  - Task
    - A task provides the framework for the execution of functions
  - Interrupt
    - Mechanism for processing asynchronous events
  - Alarm & Counter
    - Mechanisms for processing recurring events
- Support for synchronizations of concurrent computing
  - Event
    - Mechanism for concurrent processing synchronization
  - Resource
    - Mechanism for mutual concurrent access exclusion
Focus on the OSEK/VDX Task definition

- **Semantic**
  - An OSEK-VDX task provides the framework for computing application functions. A scheduler 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
- Fullfill the tagged values of the applied stereotype

Models have been realized with the Papyrus Eclipse-based open-source tool for UML2: [http://www.papyrusuml.org](http://www.papyrusuml.org)
In which typical cases shall I use SRM?

Software Resource Modeling (SRM)

- Description execution support API
- Use API model
- Use model transformation
- Code generation

Software Designer

Execution Platform Provider

Methodology Provider
Use examples of one RTOS modeled with SRM

- Example 1: Model-based design of multitask applications
  - Illustrated on a robot controller application

- Example 2: OS configuration file generation
  - Generation of the OSEK OIL configuration files

- Example 3: Assistance to port applications
  - From OSEK to ARINC multitask design
Case study: A simple robot controller software

- **Goal**
  - A motion controller system for an exploration autonomous mobile robot.

- **Robot features**
  - Pioneer Robot (P3AT)
    - Four driving wheels
    - A camera
    - Eight sonar sensors, etc.

- **Design features of the robot controller**
  - OSEK/VDX execution support
    - Simulation on Trampoline
      (http://trampoline.rts-software.org/)
  - Two periodic tasks
    - Data acquisition task
      - Get position data from sonar sensors every 1 ms
    - trajectory computing task
      - Set new speed every 4 ms
Application design

**Application model at the functional level**

- **One robot controller entity**
  - Aims at controlling the robot motions
  - Main functions
    - Acquire the sonar data
    - Compute the new speed of each 4 motions and send new orders

- **A robot driver entity**
  - Aims at interfacing robot sensors and actuators with the control application

### RobotController

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquire()</td>
<td>Acquire sonar data from sensors</td>
</tr>
<tr>
<td>trajectoryControl()</td>
<td>Compute the 4 motion speed values</td>
</tr>
<tr>
<td>terminate()</td>
<td>Terminate a mission</td>
</tr>
<tr>
<td>trap_SIGUSR20</td>
<td></td>
</tr>
<tr>
<td>trap_SIGUSR10</td>
<td></td>
</tr>
</tbody>
</table>

### RobotDriver

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update()</td>
<td></td>
</tr>
<tr>
<td>setSpeed(vx, vy, vs, state)</td>
<td></td>
</tr>
<tr>
<td>create()</td>
<td></td>
</tr>
<tr>
<td>delete()</td>
<td></td>
</tr>
<tr>
<td>getSonarScan(sonarIndex)</td>
<td></td>
</tr>
</tbody>
</table>

Driver to interface sensors and actuators
Principles of the applied multitask design

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

RobotController

MotionController
- robot: RobotDriver [0..1]
- speed_factor: integer [1] = 1
- speed_factor_turnrate: integer ...

+ terminate()
+ trajectoryControl()
+ acquire()

RobotDriver
- robot [0..1]

+ update()
+ setSpeed(vx, vy, va, state): integer
+ create() integer
+ delete(): integer
+ getSonarScan(sonarIndex): double

OSEK/VDX Library: OSEK/VDX_Platform::Osak/VDXLibrary

OSEK/VDX Library: OSEK/VDX...

BasicTask

alarm
OSEK/VDX Library: OSEK/VDX...

Alarm

<alarm>

<alarmResource>

import

Counter

Period of the periodic task acquisition: 1 ms

SRM stereotype to bind application and platform
Example 2: OSEK Configuration File generation

**Purpose**
- Generation of the OSEK OIL configuration files from the multi-task design of the robot controller

**OIL: OSEK Implementation Language**
- [http://osek-vdx.org](http://osek-vdx.org)
- The goal of OIL is to provide a mechanism to configure an OSEK application for a particular CPU

**Principle**
- Each CPU has an OIL description
- All OSEK system objects are described using OIL objects
- OIL descriptions may be:
  - hand-written
  - or generated by a system configuration tool

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

IMPLEMENTATION OSEK {
    CPU cpu {
        APPMODE std {
        };
        COUNTER counter {
            MAXALLOWEDVALUE = 255 ;
            TICKSPERBASE = 1 ;
            MINCYCLE = 1 ;
        };
        ALARM alarmAcqu {
            COUNTER = counter ;
            ACTION = ACTIVATETASK {
                TASK = acquisition ;
            } ;
            AUTOSTART = TRUE {
                ALARMTIME = 1 ;
                CYCLETIME = 1 ;
                APPMODE = std ;
            };
        };
        TASK acquisition {
            PRIORITY = 2 ;
            SCHEDULE = FULL ;
            ACTIVATION = 10 ;
            AUTOSTART = FALSE ;
            STACKSIZE = 32768 ;
        };
    };
}```
Example 3: Assist user to port multitask designs

Goal

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

Pattern matching description
- Matching rules
- Design pattern description

OSEK/VDX Specific model using SRM

Model transformation toolkit
ATL
http://www.eclipse.org/m2m/atl/

ARINC-653 specific model using SRM
Matching pattern example (1/2)

For each UML InstanceSpecification {
    if its classifier has the SRM SwSchedulableResource stereotype then {
        • generate a new Instance Specification;
        • its target classifier is that which is stereotyped SwSchedulableResource in the target execution support;
    }endif
}endif
Matching pattern example (2/2)

For each UML InstanceSpecification {
    if its classifier has the SRM `SwSchedulableResource` stereotype then {
        • ...
        • each source `priorityElements` match one target `priorityElements`
    }
}

```plaintext
« SwSchedulableResource »
priorityElements = priority
```

```plaintext
« SwSchedulableResource »
Process
prio : Integer
```

```plaintext
« instanceOf »
acquisition : BasicTask
priority = 10
```

```plaintext
« instanceOf »
acquisition : Process
prio = 10
```

```plaintext
trajectoryController : BasicTask
priority = 1
```

```plaintext
trajectoryController : Process
prio = 1
```
Agenda of the Tutorial

- MDD for ES & MARTE in a nutshell
- Non-Functional Properties and Time
- CBSE & other high-level concepts for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
Goals in Quantitative (or Non-Functional) Analysis

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

Different Goals for Evaluate & Verify System Architectures

- Point evaluation of the output NFPs for a given operating point defined by input NFPs
- Search over the parameter space for feasible or optimal solutions
- Sensitivity analysis of some output results to some input parameters
- Scalability analysis: how the system performs when the problem size or the system size grow.
MARTE Features for Quantitative Analysis

**Improvements w.r.t. SPT**
- Extend implementation and scheduling models
  - e.g. distributed systems, hierarchical scheduling
- Extend the set of analysis techniques supported
  - e.g. offset-based techniques
- Extend timing annotations expressiveness
  - Overheads (e.g. messages passing)
  - Response times (e.g. BCET & ACET)
  - Timing requirements (e.g. miss ratios and max. jitters)

**New features w.r.t. SPT**
- Support for sensitivity analysis
- Improve modeling reuse and component-based design.
- Support of the MDA approach
Context of the SAM Sub-profile

MARTE Foundations

MARTE Design

MoCCs

SW Resources

HW Resources

MARTE Analysis

General Analysis

Schedulability (end-to-end)

Performance
Organization of Analysis Annotations

Analysis Context

Workload

Behavior

Resources

Platform

evaluate situation

uses

evaluate capacity

uses

resource allocation

load

scenarios

protected

resources

exec.host

broker

digiteo labs
Specialized extensions for analysis purposes:

- Stereotypes defining specific concepts for “analysis”
  - workload events, scenarios,…
  - schedulable entities, shared resources, processing nodes, schedulers,…

- Stereotype attributes defining pre-defined NFPs
  - e.g. event arrival patterns, end-to-end deadlines, wcet-bcet-acet,…

The MARTE analysis sub-profiles provide standard constructs to map UML models on well-established analysis techniques.
Example: A Teleoperated Robot

- Industrial Robot Controllers project (University of Cantabria)
  - 3 end-to-end flows
- Refresh Report, Control Arm, Send Command
  - Periodic activation (5 ms, 100 ms, 1 s)
- Evaluate end-to-end deadlines
  - Deadlines = activation period
Robot Arm: UML Structural View
Annotating Design Models for Analysis

**Stereotype attributes:**
- Stimuli models
- Execution times
- Transmission Delays
- Constraints

**Behavioral Models**

**Platform Models**

**Stereotype attributes:**
- Capacity
- Overheads
- Scheduling schemes
- Resources access
Building Basic Analysis Contexts

```
«saAnalysisContext»
{ isSched= (true, $sched1) }

SchedContext01_RobotArm

«var» $sched1: NFP_Boolean = (true, calc)

« gaWorkloadBehavior »
: NormalMode

« GaResourcesPlatform »
: Platform_RobotArm
```

Analysis Context:
- Several scenarios can be evaluated on the same (application & platform) models

Schedulability Analysis Tool
(e.g., SymTAS, MAST, RapidRMA, Cheddar)
Robot Arm: SymTAS Model

- Equivalent application view of our example in SymTAS

- Equivalent Hw architecture view of our example in SymTAS
Agenda of the Tutorial

- MDD for ES & MARTE in a nutshell
- Non-Functional Properties and Time
- CBSE & other high-level concepts for RT concerns
- Platforms modelling
- Model-based analysis of RTE properties
- Conclusions
MARTE Frontiers and Challenges

- MARTE define only language constructs!
  - Base building blocks related to RTE concerns
    - E.g., its standard framework for NFP annotations
  - Generic constraints that do not force specific execution models, analysis techniques or implementation technologies

- It does not cover methodologies aspects:
  - Agnostic w.r.t. how to use UML2
    - E.g. CBSE, OO, functional modeling
  - Means to manage refinement of NFP measurement models
  - No transformation rules link design and analysis models

MARTE is to the RTES domain as UML/SysML to the System & Software domain:
a family of rich and open specification formalisms for modeling and analyzing RT/E systems in model-driven process!
Questions?

- UML Profile for MARTE, Beta1
  - ptc/07-08-04 - August 6, 2007

- www.omgmar.te.org
  - The OMG official web site for MARTE
    - Specifications, related documents and references, tool support...

- www.papyrusuml.org
  - On open source Eclipse plug-in for UML2
  - Available plug-ins for MARTE
  - Other plug-ins
    - SysML
    - Code generation (JAVA, C, C++)
    - ...