Agenda

- Part 1
  - Introduction to MDD for RT/E systems & MARTE in a nutshell

- Part 2
  - Non-functional properties modeling
  - Outline of the Value Specification Language (VSL)

- Part 3
  - The timing model

- Part 4
  - A component model for RT/E

- Part 5
  - Platform modeling

- Part 6
  - Repetitive structure modeling

- Part 7
  - Model-based analysis for RT/E

- Part 8
  - MARTE and AADL

- Part 9
  - Conclusions
Outlines of the GRM package

- Provides basic concepts for modeling a general (high-level) platform for processing RTE applications

- Includes the features for modeling processing platforms at different level of details.
  - The level of granularity needed depends on the concern motivating the description of the platform
    - E.g., the type of the platform, the type of the application, or the type of analysis to be carried out on the model

- Build in a bottom-up process to abstract finer-level platforms
  - Processing platform for design concern
    - See HRM and SRM
  - Processing platform for analysis concern
    - See GQAM-related ptf and further refinements for performance and schedulability analysis
Essence of the GRM Package

Object concern

ResourceInstance

ResourceServiceExecution

Classifier concern

Resource

ResourceService

ownedElement

resMult: Integer [0..1]
Generic Resource Modeling

Resource offers Services and may have NFPs for its definition and usage

A rich categorization is provided: Storage, Synchronization, Concurrency, Communication, Timing, Computing, and Device Resources may be defined.

Shared resources, scheduling strategies and specific usages of resources (like memory consumption, computing time and energy) may be annotated.
Example of UML extensions for Generic Resources

```
Example of UML extensions for Generic Resources

- profile
  - GRM
    - metaclass
      - UML::Classes::Kernel::Property
    - metaclass
      - UML::Classes::Kernel::InstanceSpecification
    - metaclass
      - UML::Classes::Kernel::Classifier
    - metaclass
      - UML::Interaction::BasicInteractions::Lifeline
    - metaclass
      - UML::CompositeStructures::InternalStructures::ConnectableElement

- stereotype
  - Resource
    - resMult: Integer = 1
    - isProtected: Boolean
    - isActive: Boolean
  - ComputingResource
    - elementSize: Integer
  - CommunicationEndPoint
    - packetSize: Integer
    - CommunicationMedia
  - SynchronizationResource
    - SchedParams: SchedParameters [0..*]
  - ConcurrencyResource
    - isActive: Boolean = true (isReadOnly)
  - SchedulableResource
    - schedPolicy: SchedPolicyKind
    - schedule: OpaqueExpression
  - SecondaryScheduler
  - DeviceResource
    - elementSize: Integer
  - ProcessingResource
    - speedFactor: NFP_Real = (value = 1.0)
```
Generic resource modeling example

- **NT_Station**
  - **ComputingResource**
    - processingRate=1.0

- **Controller**
  - **ComputingResource**
    - processingRate=0.6

- **Robot Arm**
  - **Device**
    - processingRate=1.0

- **CAN_Bus**
- **VME_Bus**
- **Storage**
  - processingRate=8.5
  - elementSize=1024 x 1024 x 8, maxRI=256
Allocation & Refinement

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

- **Inspired by the SysML allocation**
  - Can only allocate application to execution platform
  - Can attach NFP constraints to the allocation
A two step process for allocation modeling

- Identify possible sources and targets of allocations

What can be allocated, the logical view:
\[\rightarrow\] structure or behavior

What can serve as a target of an allocation, the physical view:
\[\rightarrow\] a resource or a service.

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

Application

- mySpeedRegulator : SpeedRegulatorSystem [1]
  - SpeedController
  - CarSpeed

RealTimeOperatingSystem

- « schedulableResource »
  - OS_Task
- « storageResource »
  - VirtualMemory
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

« schedulableResource, ep_allocated »
OS_Task

« storageResource, ep_allocated »
VirtualMemory
Allocation example (4)

Application

mySpeedRegulator : SpeedRegulatorSystem [1]

- « app_allocated »
  - SpeedController
- « app_allocated »
  - CarSpeed

RealTimeOperatingSystem

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

HardwareProcessingPlatform

- « computingResource, ep_allocated »
  - CPU
- « storageResource, ep_allocated »
  - Memory
- « communicationMedia, ep_allocated »
  - Bus
- « storageResource, ep_allocated »
  - Disk
What is the Software Resource Modeling Profile (SRM)?

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

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

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

In which steps shall I use SRM?

[Diagram showing the integration of SRM in the software development lifecycle]
Why shall I use SRM for modeling RTOS APIs?

- **RTOS API modeling with UML is already possible**
  - But, generics UML is lacking RTE native artifacts!
    - No modeling artifacts to describe specific concepts
      - E.g. tasks, semaphores and mailboxes
    - Consequently, models rely only on naming conventions
      - **Not possible to define generic tools using these models**
        - E.g. code generator or model transformations for analysis.

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

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

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`
- `InterruptResource`
- `MemoryPartition`
- `Alarm`
- `EntryPoint`
- `SwTimerResource`

**SRM::SW_Interaction**

- `MessageComResource`
- `NotificationResource`
- `SharedDataResource`
- `SwMutualExclusionResource`

**SRM::SW_Brokering**

- `MemoryBroker`
- `DeviceBroker`
OSEK/VDX case study

- **OSEK/VDX standard** ([http://www.osek-vdx.org](http://www.osek-vdx.org))
  - Automotive industry 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 organizes the sequence of task executions.

- **Example of properties**
  - **Priority**: UINT32
    - Priority execution of the task
  - **StackSize**: UINT32
    - Stack size associated with 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
  a. Add model library applying the SRM profile
  b. Add a class and defines its features (properties and operations)

- Applying the «SwSchedulableResource» stereotype

- Fulfill 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
SRM modeling facilities

- **How to model multiple candidates for the same semantics?**
  - Answer: All stereotype tags have multiple multiplicities. Thus, it is possible to reference multiple candidates for the same tag.
  - Examples
    - Both *name* attributes and *taskId* parameter are task identifier
    - Both *activateTask* and *chainTask* operations are task activating services
SRM modeling facilities (seq.)

- How to model a feature which have multiple semantics?
  - Answer: Feature can be referenced by several different tags
    - Example
      - The *chainTask* service is both a terminate service and an activate service

- Is it possible to reference a feature even if the feature owner is not the stereotyped element?
  - Answer: Yes, there is no constraints on the feature owner

- SRM allows multiple usages
  - User can use constraints, such as OCL rules, to limit those possibilities
Focus on the OSEK/VDX Event definition

- **Semantics:**
  - The event mechanism is a means of synchronization that initiates state transitions of tasks to and from the *waiting* state.
  - Example of owned properties
    - **Mask : EventMaskType**
      - Define the mask associated with the event
  - Examples of provided services
    - **SetEvent (TaskID: TaskType, Mask: EventMaskType)**
      - The events of the task referenced by the TaskID parameter are set according to the event mask specified by the Mask parameter.
      - Calling the service SetEvent causes the task identified by the TaskID parameter to be transferred to the ready state, if it was waiting for at least one of the events specified in the Mask parameter.
    - **WaitEvent (Mask: EventMaskType)**
      - The state of the calling task is set to *waiting*, unless at least one of the events specified in the Mask parameter has already been set.
Which SRM concepts for OSEK Event?

Interactions between concurrent contexts:

- Communication
  - Shared data
  - Message (~Message queue)
- Synchronization
  - Mutual Exclusion (~Semaphore)
  - Notification Resource (Event mechanism)
Details of «NotificationResource»

- **Semantic**
  - *NotificationResource* supports control flow by notifying the occurrences of conditions to awaiting concurrent resources
  - ==> SRM artifacts for modeling OSEK-VDX Event!

- **Main features**
  - Examples of owned attribute
    - *maskElements* and *mechanism*
  - Examples of provided service
    - flushServices, signalServices, waitServices and clearServices

- **Extract from the SRM::SwInteraction meta model**
OSEK/VDX Event as a NotificationResource

- **Stereotype icon**

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

- **Stereotype shape**

```uml
+ « NotificationResource » maskElements = Event::mask
clearServices = EventService::ClearEvent
signalServices = EventService::SetEvent
waitServices = EventService::GetEvent
```
In which typical cases shall I use SRM?

Software Resource Modeling (SRM)

Describe execution support API

« include »

Execution Platform Provider

Use API model

« extend »

Methodology Provider

Model Transformation

« extend »

Code generation

Software Designer
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
Purpose and context of the example 1

- Provide a multitask design of the robot controller
  - Target of the design is an OSEK/VDX-based platform

- Design process
  - A platform provider supplies the OSEK/VDX model library
    - Model library is described with the SRM Profile (as previously shown)
  - A user designs a multitask model of the application
    - Step 1: Describe the application model (also called functional model)
    - Step 2: Propose a multitask design using the OSEK model library artifact
Application design

- **Application model at the functional level**
  - **One robot controller entity**
    - Aims at controlling the robot motions
    - **Main functions**
      - Acquire the sonar data
      - Compute the new speed of each 4 motions and send new orders
  - **A robot driver entity**
    - Aims at interfacing robot sensors and actuators with the control application

- **RobotController**
  - **MotionController**
    - + robot: RobotDriver[0..1]
    - speed_factor: Integer[1] = 1
    - speed_factor_turnrate: Integer[1] = 2
    - + acquire()
    - + trajectoryControl()
    - + terminate()
      - + trap_SIGUSR20
      - + trap_SIGUSR10
  - + terminate

- **RobotDriver**
  - + robot: RobotController[0..1]
  - + position2d: playerc_position2d_t[0..1]
  - + client: playerc_client_t[0..1]
  - + serverPort: String[1] = 132.166.135.110
  - + update()
    - + setSpeed(vx, vy, va, state): Integer
    - + create(): Integer
    - + delete(): Integer
    - + getSonarScan(sonarIndex): double

**Driver to interface sensors and actuators**

- Acquire sonar data from sensors
- Compute the 4 motion speed values
- Terminate a mission
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
A design pattern for implementing periodic task on OSEK/VDX-based platforms

- One OSEK/VDX Counter
  - Counter period = period of the required periodic task
- One OSEK/VDX Task
  - Entry point: periodic task Entry Point
- One OSEK/VDX Alarm
  - AutoStart: Triggered by the counter
  - Action: Activate the task

SRM Profile is used to describe the pattern
Basic Robot Controller task models

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**
  - For each CPU, there must be an OIL description
  - All OSEK system objects are described using OIL objects
  - OIL descriptions may be:
    - hand-written
    - or generated by a system configuration tool

```plaintext
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

- **Purpose**
  - Assist user to port the multitask design to an ARINC-653 RTOS
    - ARINC 653 standard provides avionics application software with the set of basic services to access the operating system and other system-specific resources.
HRM use cases

3 use cases = 3 levels of details
HRM use cases -- High level hardware modeling

- **How?**
  - High level of **abstraction**
  - **Architectural** view of the HW platform
  - With key properties:
    - E.g., instruction set and memory size.
  - A formal view of usual **block diagrams**

- **For**
  - High level description of existing and targeted HW platform
  - First steps of design of new HW architecture

- **By**
  - System architects
  - Software developers
HRM use cases -- Specialized hardware modeling

- **How?**
  - Specialized HW **description** model
  - Nature of details depends on the **point of view**
    - Ex1: autonomy analysis requires power consumption modeling
    - Ex2: WCET analysis need details on processor speed, communication bandwidth and memory organization…
- **For analysis purpose**
- **By analyzers**
HRM use cases -- Detailed hardware modeling

How?
- HRM is a detailed HW architecture design language
- Level of details depends on the description accuracy
  - Ex1: Functional simulator of a processor only requires its instruction set family
  - Ex2: Performance simulation need a fine description of processors micro-architecture.

For
- Model-based datasheets description
- Simulation
  - generation of configurations for simulation tools

By
- HW designers
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 », « HwDrive »
    - « HwMMU », « HwDMA »
  - **HwDevice**
    - « HwDevice », « HwSupport »
    - « HwI/O »
  - **HwCommunication**
    - « HwBridge »
    - « HwArbiter »
    - « HwMedia », « HwBus »
  - **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

```
.HwComponent
  kind : {Card, Channel, Chip, Port}
```

```
.HwLayout

.HwPower
```

« HwPowerSupply »  « HwCoolingSupply »
HRM profile -- HwMemory

« profile »
HRM

« profile »
HwLogical

HwGeneral

HwComputing

HwCommunication

HwTiming

HwStorage

HwMemory

HwStorageManager

HwDevice

« profile »
HwPhysical

HwGeneral

HwLayout

HwPower

« modelLibrary »
MARTE:Library:BasicNFP_Types

« import »
MARTE:GRM

« import »
MARTE:Library::BasicNFP_Types
HRM profile -- HwMemory

- **HwMemory**
  - **MARTE:GRM::Storage**
  - **HwResource**
    - **HwMemory**
      - `memorySize` : NFP_DataSize
      - `addressSize` : NFP_DataSize
      - `timings` : Timing

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

- **HwRAM**
  - `organization` : MemoryOrganization
  - `isSynchronous` : NFP_Boolean
  - `isStatic` : NFP_Boolean
  - `isNonVolatile` : NFP_Boolean
  - `repl_Policy` : Repl_Policy
  - `writePolicy` : WritePolicy

- **HwROM**
  - `type` : ROM_Type
  - `organization` : MemoryOrganization
  - `sectorSize` : NFP_DataSize

- **HwDrive**
  - `buffer` : {subsets ownedHW}
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**

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

**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
### HRM profile -- HwMemory -- HwCache

**« 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

**Specifications**

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

**« stereotype »** HwCache

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</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>

**« enumeration »** Repl_Policy

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

**« enumeration »** WritePolicy

- WriteBack
- WriteThrough
- Other
- Undefined

**« enumeration »** CacheType

- Data
- Instruction
- Unified
- Other
- Undefined

**« dataType »** CacheStructure

- nbSets : NFP_Natural
- blocSize : NFP_DataSize
- associativity : NFP_Natural

- 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`
Specifies the cache write policy

- WriteBack: Cache write is not immediately reflected to the backing memory.
- WriteThrough: Writes are immediately mirrored.
HRM stereotypes extends the main structural UML metaclasses
- Classifier, Class
- InstanceSpecification, Property
- Association (HwMedia, HwBus…), Port (HwEndPoint)

HRM can be used with all Structural UML diagrams:
- Class diagram
- Component diagram
- Composite Structure Diagram (well adapted for HW)

HRM profile application
- Definitions of the stereotype properties are optional
  - Specified if needed
  - Specified when needed (Refinement)
    - At class level for technology definition (e.g. type of HwCache)
    - At instance level for component definition (e.g. size of HwCache)
Very early Hw Architecture Description

- SMP (Symmetric MultiProcessing) hardware platform
  - 4 identical processors
    - Unified Level 2 cache for each
  - Shared main memory (SDRAM)
  - Central FSB (Front Side Bus)
  - DMA (Direct Memory Access)
  - Battery

![Diagram showing hardware resources]
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 = sdram}

« hwRAM »

sdram : SDRAM
{frequency = 266Mhz, memorySize = 256MB}
```

« 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}
### HRM usage example: Physical view 1

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL2</td>
<td>Unit</td>
</tr>
<tr>
<td>FSB</td>
<td>Channel</td>
</tr>
<tr>
<td>DMA</td>
<td>Chip</td>
</tr>
<tr>
<td>Battery</td>
<td>Other, capacity = 40Wh</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Card</td>
</tr>
</tbody>
</table>

**Example:**

- **Physical view 1**
  - **SMP**
    - **CPU [4]** (kind = Chip)
    - **UL2** (kind = Unit)
    - **FSB** (kind = Channel)
    - **DMA** (kind = Chip)
    - **Battery** (kind = Other, capacity = 40Wh)
    - **SDRAM** (kind = Card)
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
```

```
« hwChip »
cpu3 : CPU
position = [2,2], [1,1]
staticConsumption = 5W
```

```
« hwChip »
cpu2 : CPU
position = [1,1], [3,3]
staticConsumption = 5W
```

```
« hwChip »
cpu4 : CPU
position = [2,2], [3,3]
staticConsumption = 5W
```

```
« hwCard »
sdram : SDRAM
```

```
« hwCard »
dma : DMA
```

```
« hwCard »
battery : Battery
position = [4,4], [3,3]
capacity = 10Wh
weight = 150g
```
HRM case study -- TC1796 (μController)

- **Advanced 32-bit TriCore™-based Next Generation Microcontroller for Real-Time Embedded systems**
  - Automotive control systems
  - Industrial robotic control

- **Features**
  - Super-scalar TriCore CPU
    - Superior real-time performance
      - Efficient interrupt handling
    - 4 stage pipeline
    - DSP capabilities
    - 150 MHz operational frequency
HRM case study -- TC1796

- **Complex memory architecture**
  - Embedded Program Memory (>2MByte): PMI (ICACHE, SPRAM), PMU (BROM, PFLASH, DFLASH)
  - Data Memory: DMI(LDRAM, DPRAM), DMU(SRAM, SBRAM)…
  - Extendable memory using an external bus

- **High performance triple bus structure**
  - Two Local memory busses (64-bit) to program and data memories
  - 32-bit system peripheral bus to on-chip peripherals
  - 32-bit remote peripheral bus to external peripherals
  - Independent bus control units

- **16-channel DMA controller…**
Block diagram of the TC1796 CPU-Subsystem
HRM case study -- TC1796 CPU-Subsystem

See models examples

on www.papyrusuml.org
HRM application -- HW emulation

- UML models have now a precise standard XML representation (using the XMI definition).
- Then, all model manipulations and transformations can be easily done using widely known XML technologies.
  - Eclipse plugins (EMF, UML2…), Acceleo…

The steps are:
- **Describe** the HW models in UML using HRM
- **Parse** and Capture all the required HW properties
- **Verify** coherency and completion
- **Generate** the configuration file for the target emulation tool
- **Simulate** the application software on the emulated HW
Examples of Possible Hw Emulators

- **Simics (Virtutech, www.virtutech.com/)**
  - Support for most HW components
  - Functional and Performance simulation
  - Enable to run heavy software applications (e.g., linux)
  - Free for academics

- **Skyeye (www.skyeye.org/**)
  - Support for ARM-like processors, most of memories and peripherals
  - Functional simulation
  - Enable to run only light sw applications (E.g., µLinux and ARMLinux)
  - GPL

- **SimpleScalar (www.simplescalar.com/**)
  - Academic tool easy to extend
  - Performance simulation
  - Run C code