MARTE and Model transformation

Ansgar Radermacher / Asma Smaoui ansgar.radermacher@cea.fr

Acknowledgments – contains material from my CEA colleagues Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

Agenda

1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

2. Model transformation

- 1. Principles
- 2. Languages

MARTE

- Complex system modeling
 - UML as base modeling language (for OO software architectures)
 - SysML for system engineering concerns
 - Requirements, mathematical expressions, continuous behavior, etc...
 - Good industrial acceptance level
- What about concerns of real-time embedded systems (RTES)?
 - Embedded: autonomous system with limited resources & performance properties/constraints like energy consumption and resource contention
 - Real-time: data processing correctness also depends on time

MARTE

Need a language for Modeling and Analysis of Real-Time Embedded systems

MARTE

What is MARTE?

- A modeling language for the RTES domain, based on several industrial systems and standards
- Implemented as a UML profile
- Standardized by the Object Management Group (who also standardizes UML among other things...)

Why MARTE?

- Remember why UML was proposed:
 - Too many languages and tools for software design
 - ⇒ interoperability not guaranteed, engineering expertise difficult to manage
 - UML unifies all these languages around a standard language that shares many software concerns
- Same situation for RTES:
 - Too many languages and tools for RTES design
 - interoperability not guaranteed, engineering expertise difficult to manage
 - Same problems as software design, so same pattern to solve the problem!
 - A standard language for RTES is hard to propose since this domain is very heterogeneous

Real-time Embedded System

Embedded system

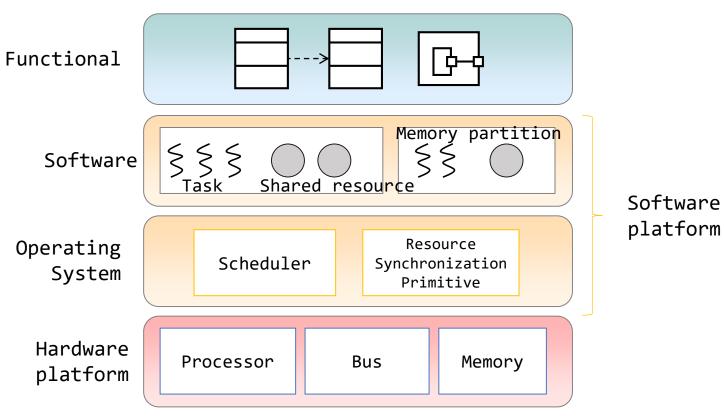
- Computing system composed of hardware and software
- ... with a specific mission while using limited resources (e.g. processing, memory, battery).
- Embedded within larger system. Interaction with execution environment

Real-time system

- Computing system composed of hardware and software,
- Computing is subject to time constraints take some input some data, then launch some behavior or produces some output within a time limit.
- Produced data must not only be correct (functional correctness) but also arrive in time (temporal correctness). Time constraints impose deadlines

Real-time Embedded System

A RTES can be characterized by its architecture



om the

This is a simplified view of the RTES architecture from the viewpoint of a real-time software engineer!

Real-time Embedded System – Terminology

 Function – a sequence of instructions to be executed. The system executes functions to accomplish its mission. Functions may be encapsulated.

 Functional architecture – specifies the functions, their eventual encapsulation, the data types they handle, and their data dependencies through interfaces.

Real-time Embedded System – Terminology

- Task a unit executing a set of instructions
 - task release = release a job of the task
 - Once released, instructions are executed on a processor and are either for computing or synchronization with other tasks.
- Priority indicates its order of importance for scheduling.
- Worst Case Execution Time, WCET longest possible time to execute a task on a specific processor.
- Worst Case Response Time, WCRT longest possible response time from release to completion of any of its jobs.
- Deadline the longest allowed response time.

Real-time Embedded System – Terminology

- Shared Resource accessed by several tasks in a mutual exclusive manner to enforce data consistency.
 - The shared resource is protected by some primitive.
 - A task that wants to access a shared resource may be blocked by another task that has access to the shared resource.
- Memory Partition a list of memory locations. A task can read from and write to the memory locations to which it is allocated.

Real-time Embedded System – RTOS

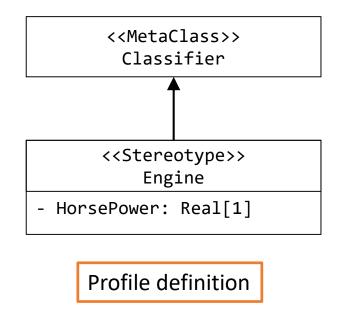
- Scheduler part of RTOS
 - Selects which task has access to the hardware processors at a given time.
 - **Scheduling** = method by which tasks are given access to resources, noticeably computing resources, with the goal of respecting time constraints
 - Implements a **scheduling policy**. A policy (or scheduling algorithm) describes how tasks are given access to resources, like computing resources
- Resource Synchronization Primitive
 - OS handles the protection of shared resources
 - Tasks call synchronization primitives when they want to access the shared resources
 - May have a consequence on task scheduling

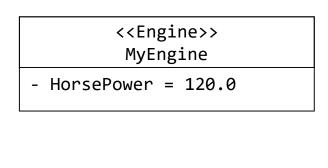
Real-time Embedded System – Hardware

- The hardware platform has processors for the execution of the software. The system may be characterized according to the number of processors and how they interact:
 - Uniprocessor only one processor core
 - Multiprocessor several processor cores that share a common memory. The cores are connected by buses that are optimized for this kind of architecture.
 - Distribute several processors that have their own memory. The processors are connected by buses forming networks.
 - Buses connecting processors may schedule messages with a scheduling policy.
 For example messages may have a fixed priority if transiting on some bus.
 Thus messages are scheduled on buses while tasks are scheduled on processors.

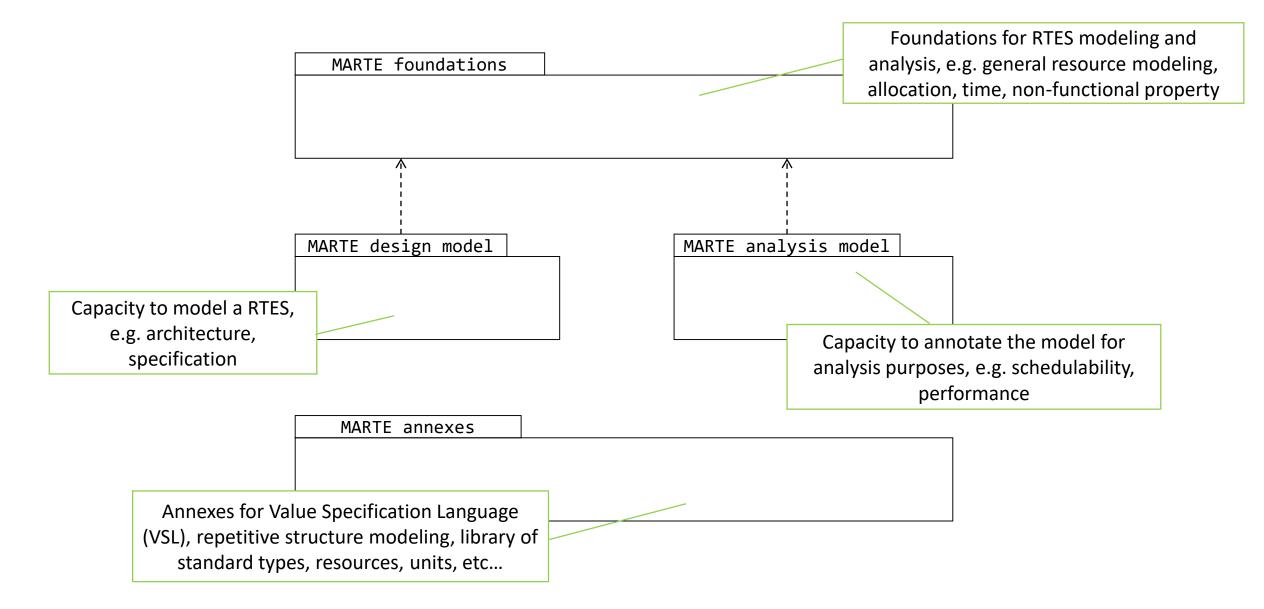
Modeling with MARTE

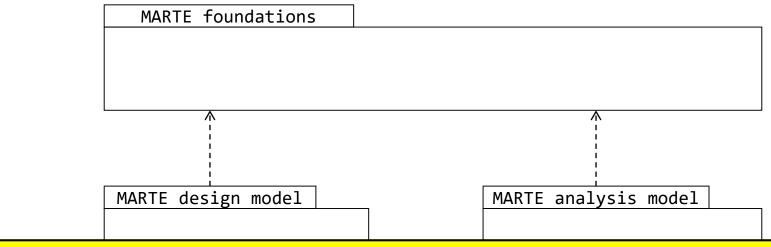
- MARTE is a UML profile for the RTES domain
- Modeling with MARTE depends on the development method, i.e. there is no unique way to model with MARTE, as long as the specification is respected





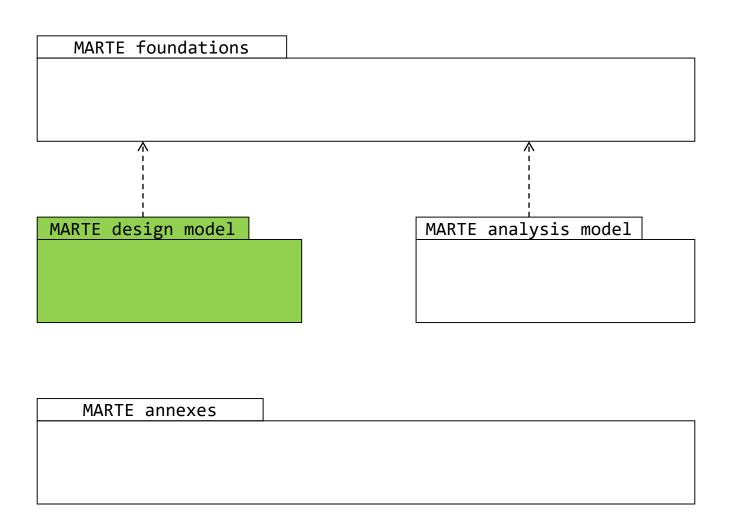
Profile usage





Now let us check some useful stereotypes in each package, with a sub-set of their attributes. If you are interested in the full metamodel, you may read the 754 pages specification. ©

MARTE annexes	



MARTE design model HLAM GCM Modeling of high-level Architecture modeling real-time entities with based on interacting qualitative and components quantitative concerns HRM SRM Hardware platform modeling Software platform modeling

Agenda

1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

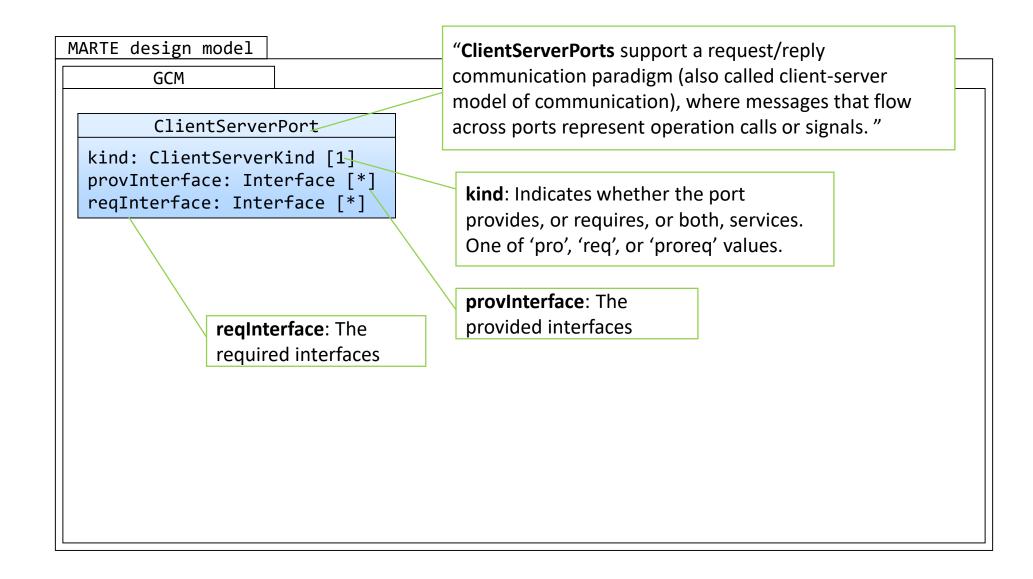
2. Model transformation

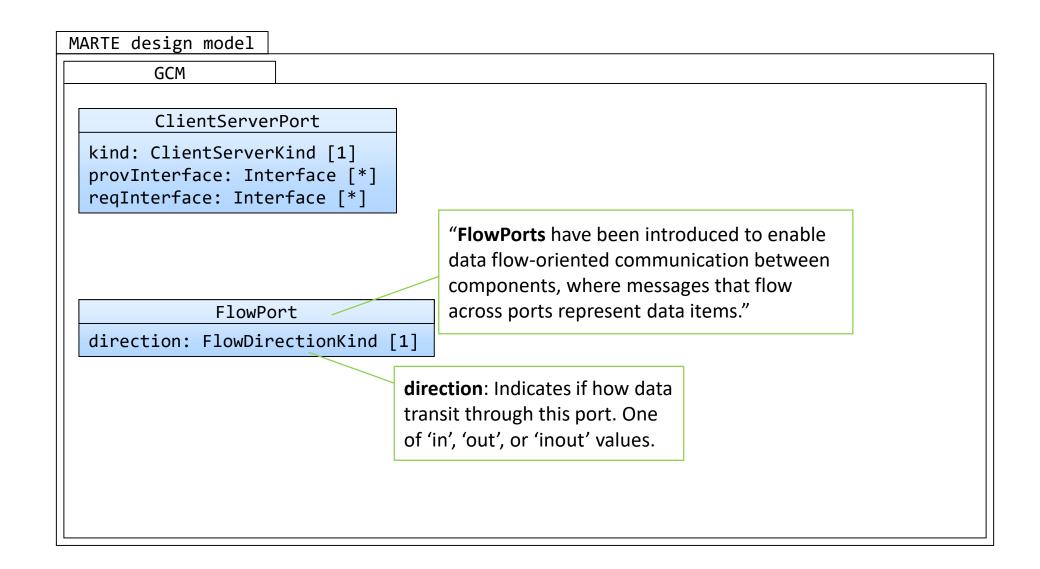
- 1. Principles
- 2. Languages

Modeling with MARTE – GCM

MARTE design model HLAM GCM Modeling of high-level Architecture modeling real-time entities with based on interacting qualitative and components quantitative concerns HRM SRM Hardware platform modeling Software platform modeling

Modeling with MARTE – ClientServerPort





MARTE design model HLAM GCM Modeling of high-level Architecture modeling real-time entities with based on interacting qualitative and components quantitative concerns HRM SRM Hardware platform modeling Software platform modeling

Modeling with MARTE – RtUnit

MARTE design model

HLAM

"RtUnit is similar to the active object of UML but with a more detailed semantics description. It owns one or several [tasks]" that will handle incoming messages at this RtUnit.

RtUnit

```
isMain: Boolean [0..1]
main: Operation [0..1]
srPoolSize: Integer [0..*]
srPoolPolicy: PoolMgtPolicyKind [0..1]
srPoolWaitingTime: NFP_Duration [0..1]
queueSchedPolicy: SchedPolicyKind [0..1]
queueSize: Integer [0..1]
```

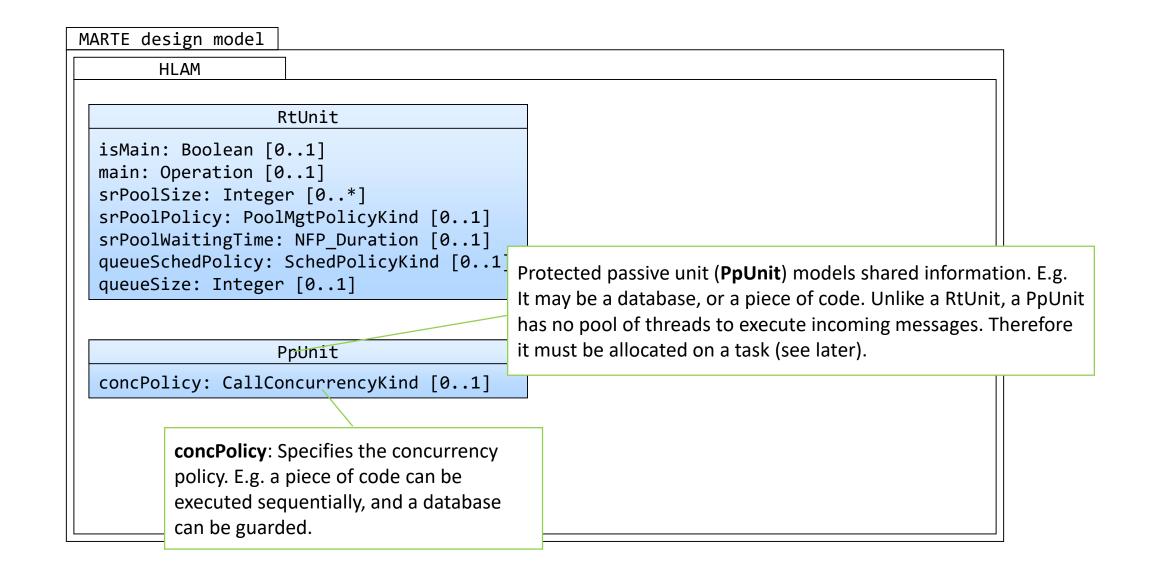
isMain: Indicates if this RtUnit is the main object in the system that will start the execution.

main: The main operation executed when the RtUnit is triggered.

queueSchedPolicy& queueSize: The RtUnit has a queue of messages that are handled in the order defined by the 'queueSchedPolicy'. The queue may also be bounded by 'queueSize'.

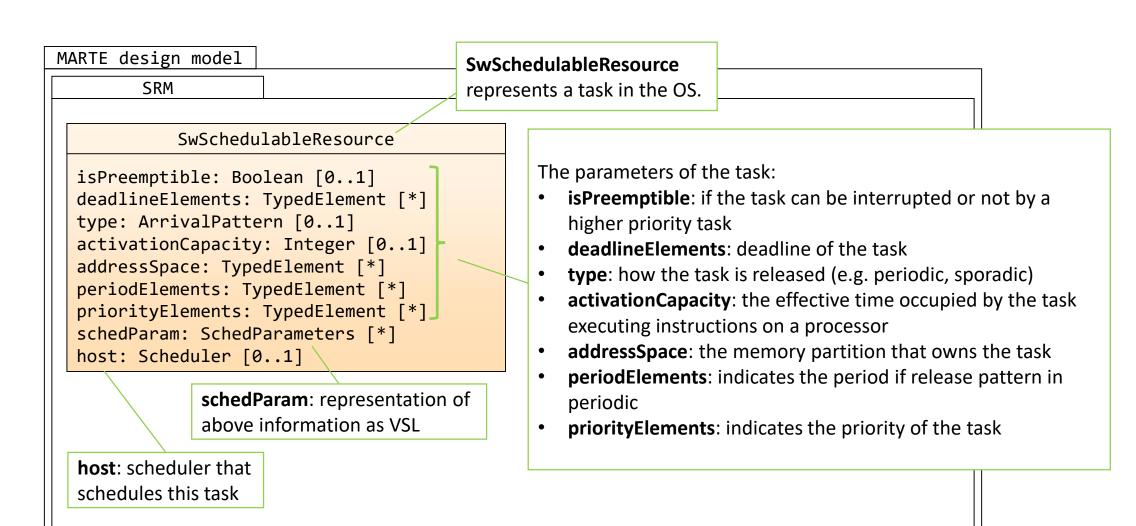
srPoolPolicy & srPoolWaitingTime: The RtUnit has a limited number of tasks indicated by 'srPoolSize'. When a request arrives, and there are no more tasks, we may define the 'srPoolPolicy', e.g. we wait indefinitely until one becomes available. We may also wait only during the 'srPoolWaitingTime'.

Modeling with MARTE – PpUnit

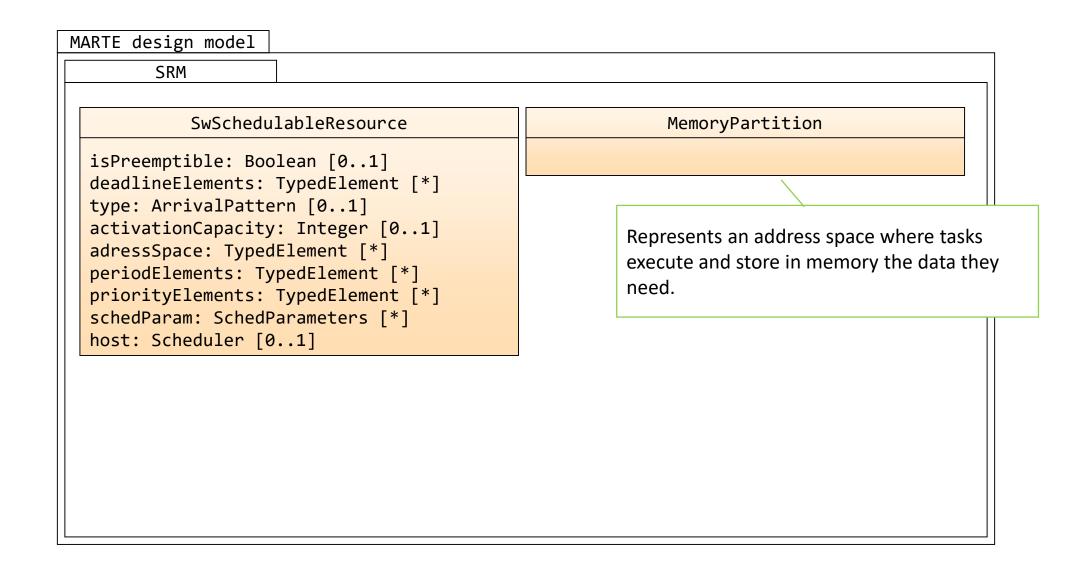


MARTE design model HLAM GCM Modeling of high-level Architecture modeling real-time entities with based on interacting qualitative and components quantitative concerns HRM SRM Hardware platform modeling Software platform modeling

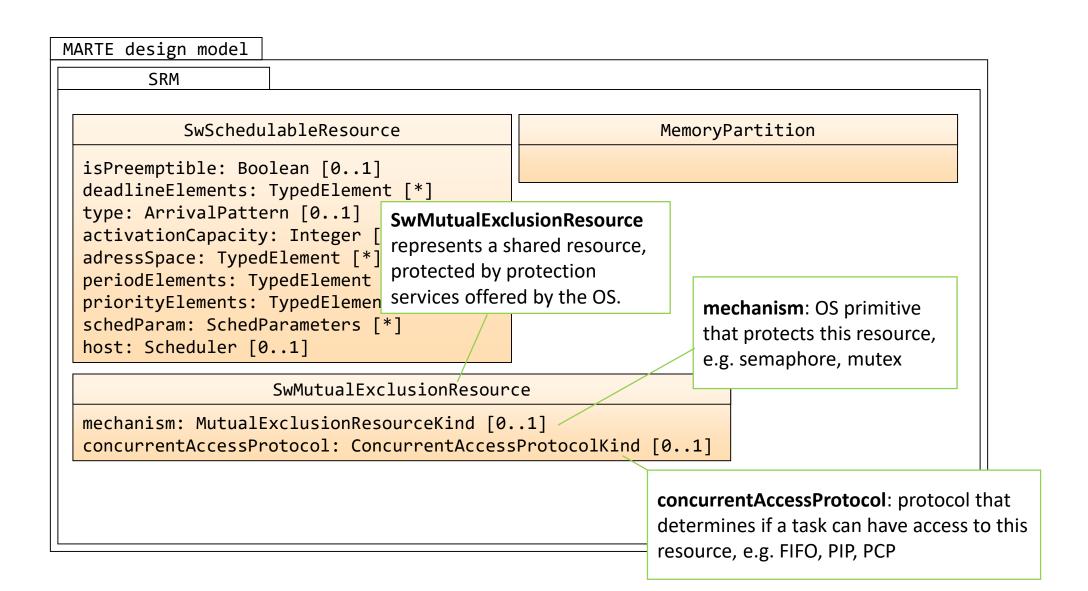
Modeling with MARTE – SwSchedulableResource



Modeling with MARTE – MemoryPartition

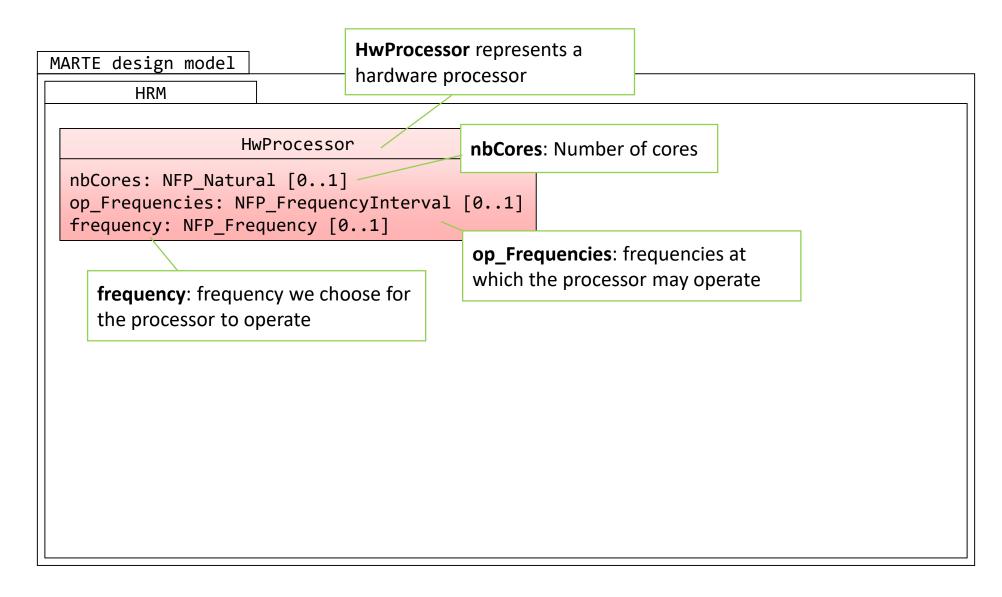


Modeling with MARTE - SwMutualExclusionResource

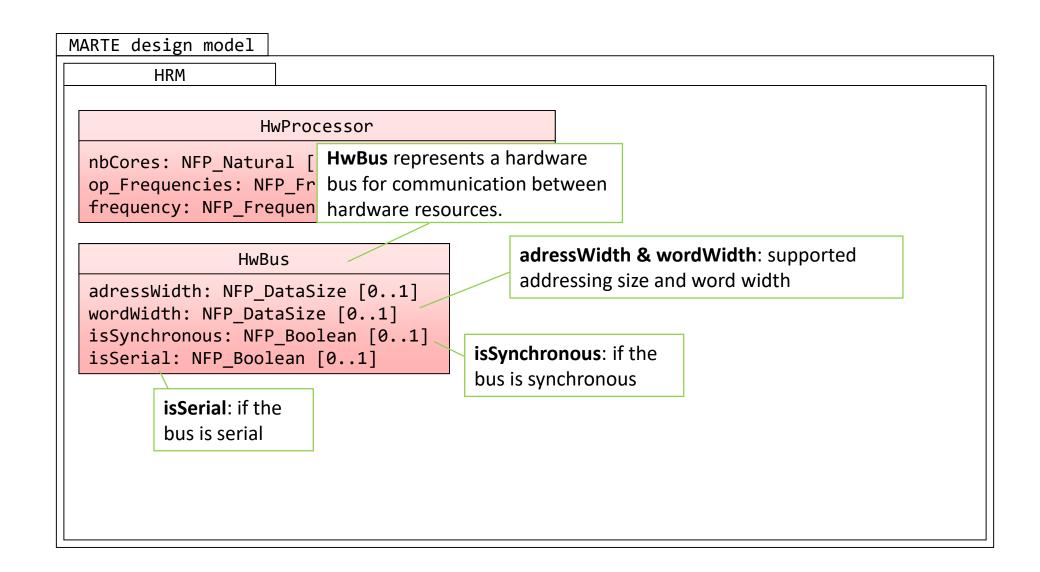


MARTE design model HLAM GCM Modeling of high-level Architecture modeling real-time entities with based on interacting qualitative and components quantitative concerns HRM SRM Hardware platform modeling Software platform modeling

Modeling with MARTE – HwProcessor



Modeling with MARTE – HwBus

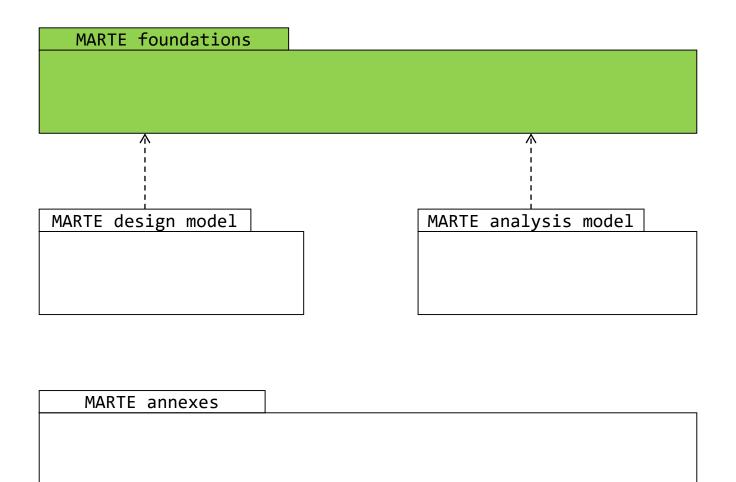


Modeling with MARTE – Further HRM stereotypes

MARTE design model HRM HwProcessor nbCores: NFP Natural [0..1] op_Frequencies: NFP_FrequencyInterval [0..1] frequency: NFP Frequency [0..1] HwBus adressWidth: NFP DataSize [0..1] wordWidth: NFP_DataSize [0..1] isSynchronous: NFP_Boolean [0..1] isSerial: NFP Boolean [0..1]

There are also many other stereotypes in HRM for memories (e.g. cache, RAM, drive), devices (sensors, actuators, IO), power units (power supply, cooling supply), etc...

MARTE packages – Foundations



Modeling with MARTE – foundations

MARTE foundations

Scheduler represents broker that arbitrates how a schedulable resources access other processing resources, e.g. how task access processor, how message access bus.

Scheduler/

schedPolicy: SchedPolicyKind [0..1]
otherSchedPolicy: String [0..1]

otherSchedPolicy: user-defined

scheduling policy

schedPolicy: the scheduling policy, to choose from a list of usual policies

Modeling with MARTE – Allocate

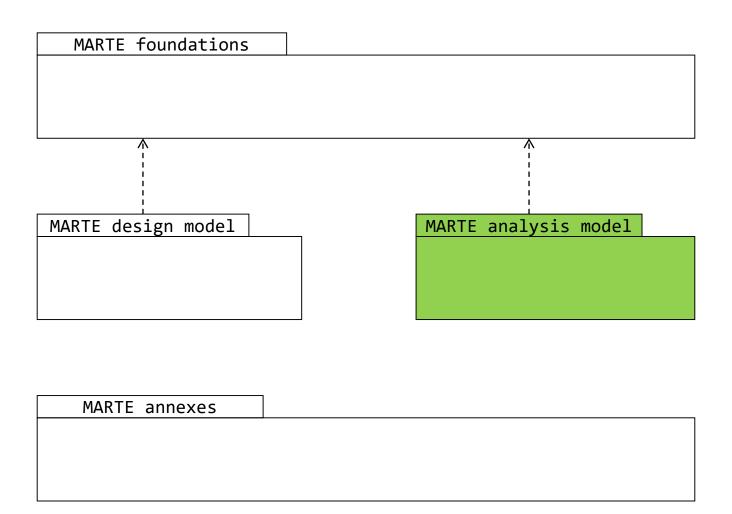
MARTE foundations

Scheduler

schedPolicy: SchedPolicyKind [0..1]
otherSchedPolicy: String [0..1]

Allocate

Allocate is an abstraction relationship, that indicates what abstract element is allocated onto what concrete element. E.g. a function is allocated onto a task if the task executes the function. A task is allocated onto a processor if the processor executes the task.



Modeling with MARTE – Analysis model

MARTE analysis model

GQAM

Annotations for general quantitative analysis

SAM

Annotations for schedulability analysis

PAM

Annotations for performance analysis

MARTE analysis model SAM **GQAM** Annotations for general Annotations for quantitative analysis schedulability analysis PAMAnnotations for performance analysis

MARTE design model

GQAM & SAM: Analysis context and container concepts

SaAnalysisContext

optCriterion: OptimalityCriterionKind [0..1]

optCriterion: optimization criteria. Can be for example to meet all hard time deadlines.

SaAnalysisContext is package that contains the elements necessary for a particular scheduling analysis. The elements are a GaResourcesPlatform and GaWorkloadBehaviors.

MARTE design model GQAM & SAM: Analysis context and container concepts SaAnalysisContext optCriterion: OptimalityCriterionKind [0..1] GaResourcesPlatform **GaResourcesPlatform** is a class that represents some instance of a platform resources: Resource [0..*] with platform resources modeled as properties (e.g. processors, tasks). **resources**: the resources owned by the platform instance.

MARTE design model GQAM & SAM: Analysis context and container concepts SaAnalysisContext optCriterion: OptimalityCriterionKind [0..1] **GaWorkloadBehavior** is a behavior (e.g. GaResourcesPlatform an activity) that acts as a container resources: Resource [0..*] other workload behaviors to analyze (e.g. activity partitions). GaWorkloadBehavior behavior: GaScenario [0..*] **behavior**: workload demand: GaWorkloadEvent [0..*] behaviors to analyze **demand**: events that trigger the workload behaviors

MARTE design model

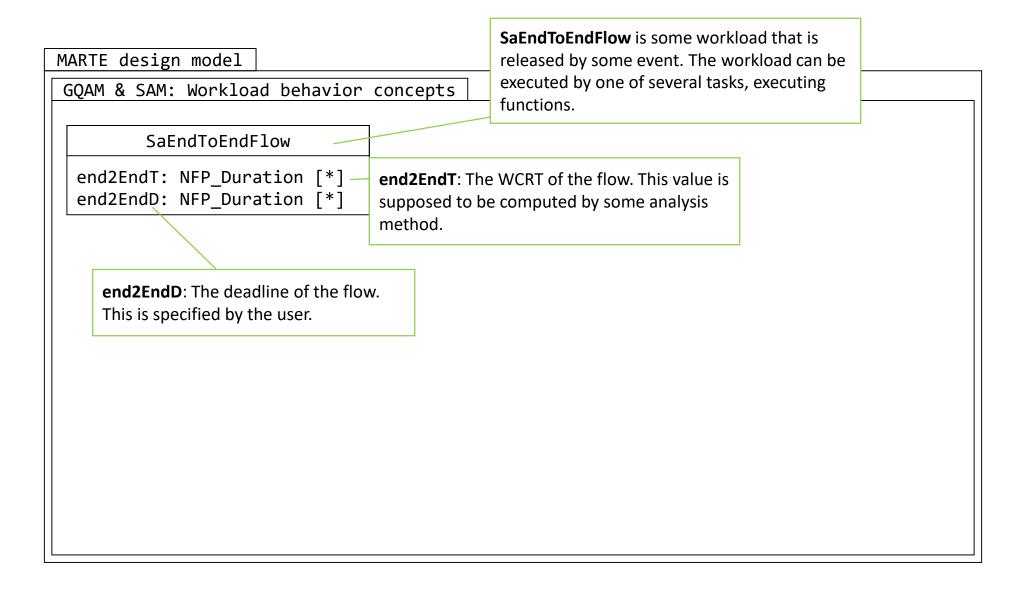
GQAM & SAM: Resource platform concepts

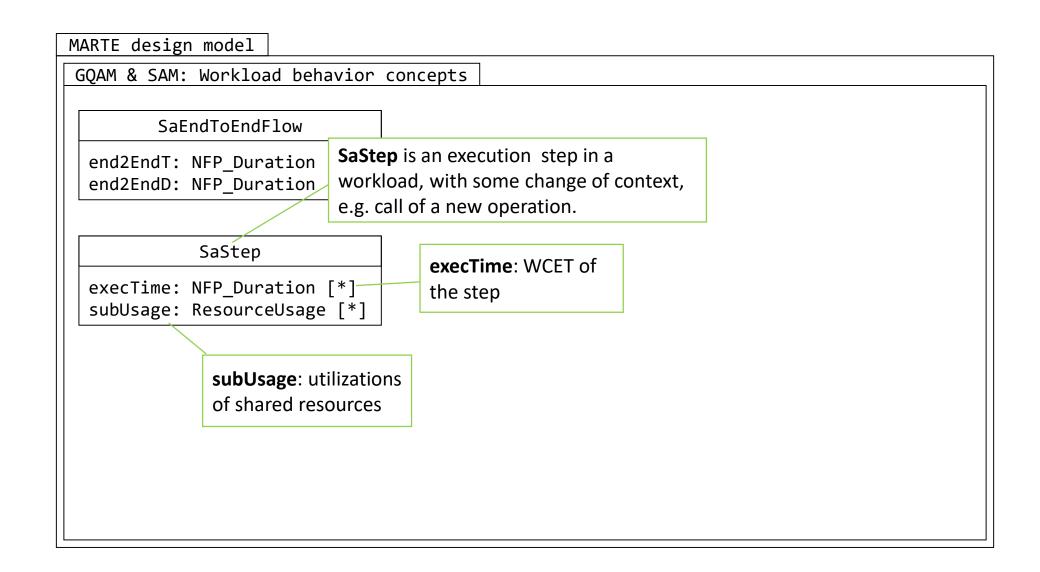
SaExecHost

schedPolicy: SchedPolicyKind [0..1]

schedPolicy: the scheduling policy, to choose from a list of usual policies

SaExecHost is some CPU with schedulability parameters (including analysis results). Redundant with HwProcessor and Scheduler stereotypes, but allows to separate analysis properties from architecture properties, and adds analysis results properties.





```
MARTE design model
GQAM & SAM: Workload behavior concepts
         SaEndToEndFlow
  end2EndT: NFP Duration [*]
 end2EndD: NFP_Duration [*]
             SaStep
  execTime: NFP_Duration [*]
  subUsage: ResourceUsage [*]
                                  GaWorkloadEvent is an event
                                  that initiates the workload
          GaWorkloadEvent
 pattern: ArrivalPattern [0..1]
                pattern: release
                pattern of the event
```

Agenda

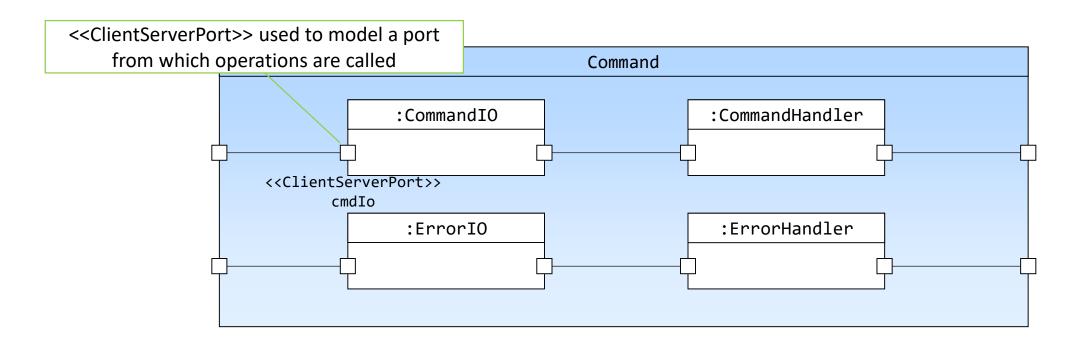
1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

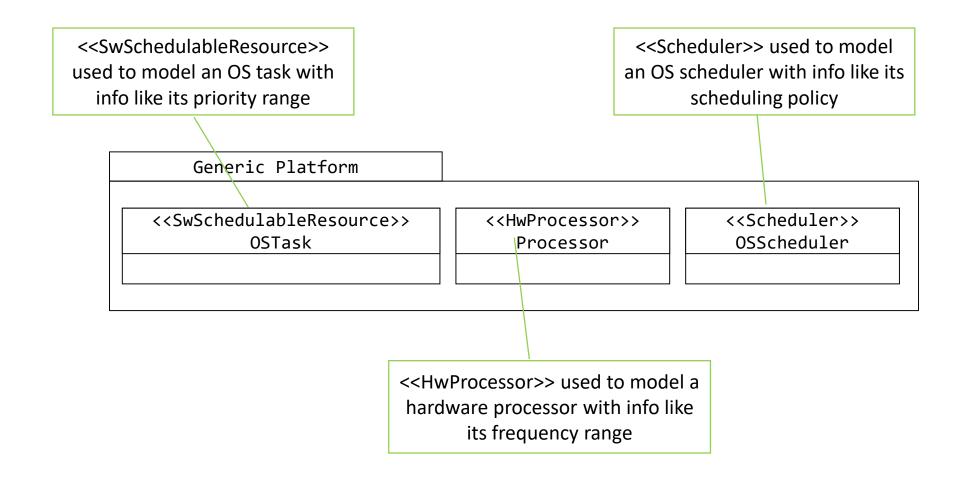
2. Model transformation

- 1. Principles
- 2. Languages

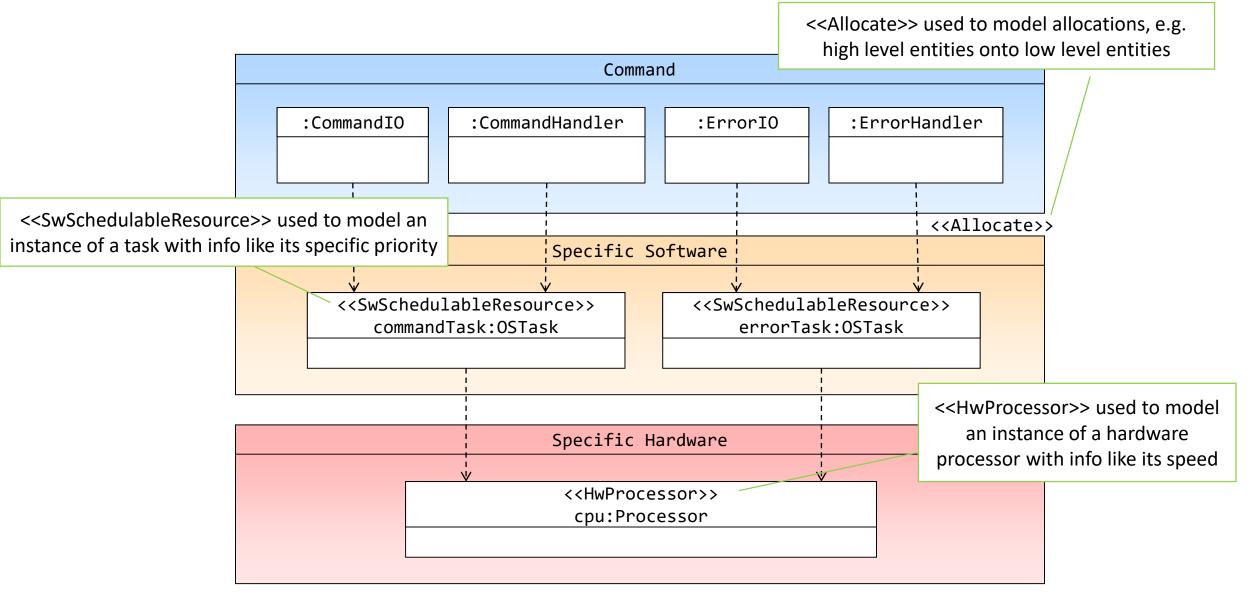
Modeling with MARTE – Functional Example



Modeling with MARTE – Platform model example



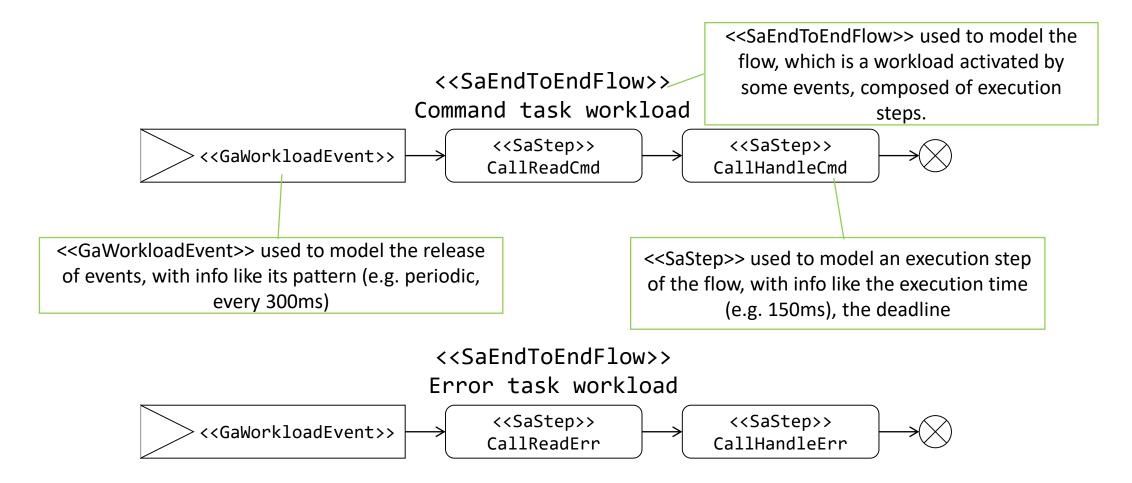
Modeling with MARTE – allocation model example



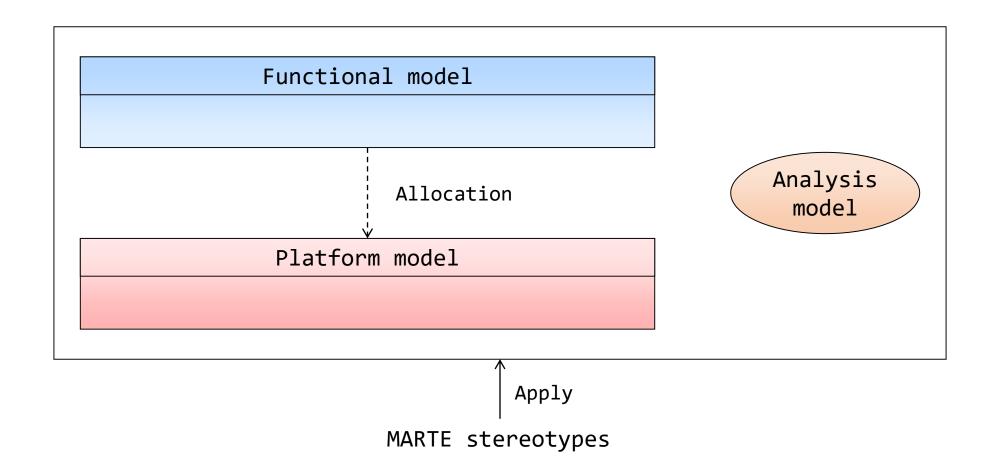
Small exercise

- Install MARTE via update site (Help->Install New Software)
 https://download.eclipse.org/modeling/mdt/papyrus/components/marte/
- Create a new Model, apply the registered MARTE HRM (sub-)profile
- Create a composite structure diagram
- Add a part "processor", apply «HwProcessor» stereotype, set some properties (frequency, architecture)

Performance analysis model example: end-to-end flow of tasks (behavior)



Modeling with MARTE – allocation summary



Agenda

1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

2. Model transformation

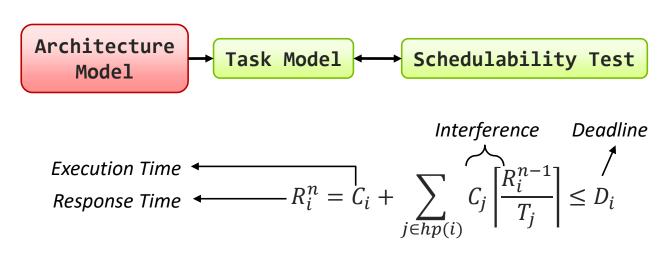
- 1. Principles
- 2. Languages

MARTE in Practice: Tasks scheduling in RTES

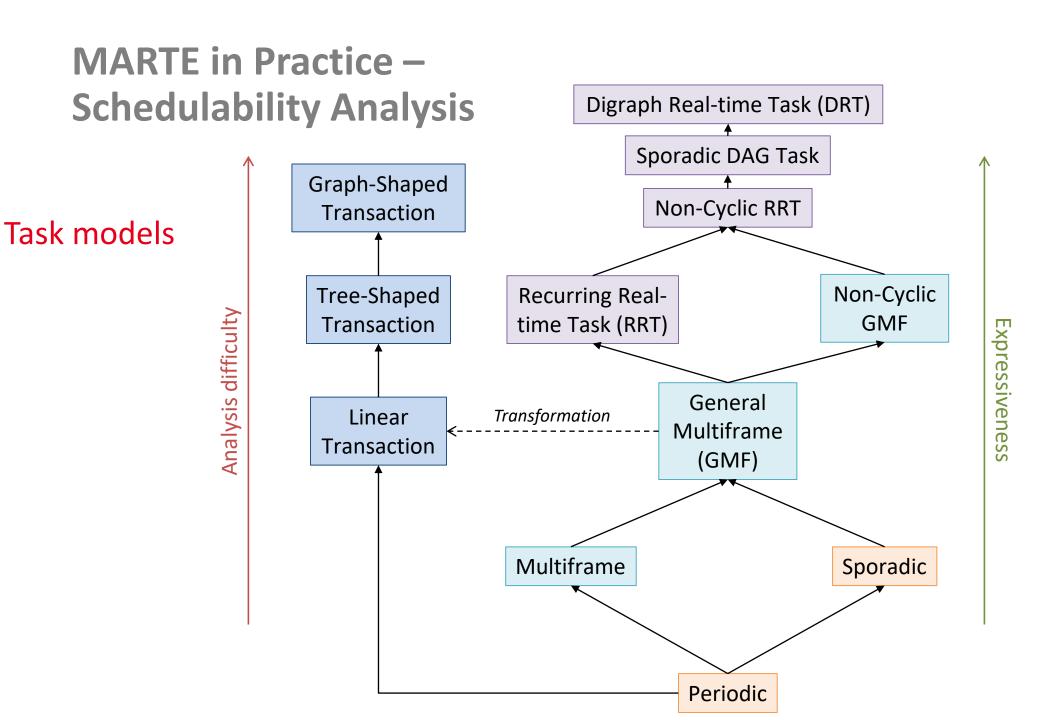
- RTES are (usually) implemented as a multi-tasking system; tasks are concurrent and must be scheduled
- Recap, scheduling method that gives tasks access to processors, i.e. according to a scheduling policy; performed by a scheduler in the OS

Schedulability analysis

 Verify that tasks meet their time constraints, when running on limited computing resources, according to a scheduling policy



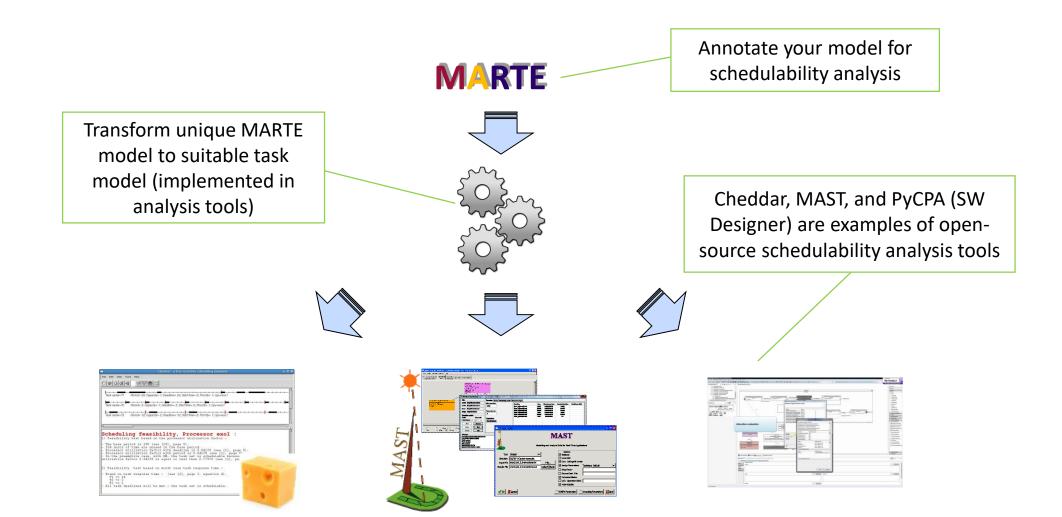
Joseph and Pandya (1986)



MARTE in Practice: Schedulability Analysis

- Periodic task model
 - Periodic task denoted by τ_i with parameters:
 - C_i is the WCET
 - D_i is the deadline
 - *T_i* is the period
 - $prio(\tau_i)$ is the priority
 - Suitable for uniprocessor systems
 - Suitable for tasks that share resources
 - Not suitable for task with precedence dependencies (i.e. task 2 waits for the end of task 1 before it can execute)

MARTE in Practice – Schedulability Analysis



MARTE (industrial) support

AUTOSAR is the standard software architecture in the automotive industry; EAST-ADL is its architecture description language

AADL

Architecture description language used in the avionics industry













MyCCM is a development framework used in the telecommunication industry (not recently any more)







MARTE support











Industrials

- Alcatel
- Lockheed Martin
- THALES
- Orange

Academics

- Carleton University
- CEA LIST
- ESEO
- ENSIETA
- INRIA
- INSA Lyon
- Software Engineering Institute (CMU)
- Universidad de Cantabria

Modeling tool vendors

- Atego
- IBM
- Mentor Graphical Corporation
- Softeam
- Tri-Pacific Software
- No Magic
- MathWorks

MARTE – Summary and conclusions

- MARTE proposed as a standard modeling language for RTES domain, based on several industrial systems and standards
- MARTE implemented as a UML profile
- MARTE allows to model RTES domain-specific information contrary to a generic software modeling language like UML
- MARTE models analyzed through model transformation to analysis tool models;
 MARTE models can also be transformed to several industrial standard models
- MARTE supported by modeling tool vendors, academics and industrials
- Further reading: B. Selic, S. Gérard, "Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems",
 1st edition, Burlington US-MA: Morgan Kaufmann, 2013

Agenda

1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

2. Model transformation

- 1. Principles
- 2. Languages

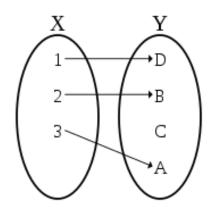
Model Transformations

- Goal transform one model into another
- Why?
 - Manual editing
 - Refactor your model (as IDEs do for code)
 - (automatically) obtain a model that is build for a specific purpose
 - Analysis
 - Model checking
 - Code generation
- Different kind of transformations
 - Do source and target comply with same MM?
 - Specific variant Model to text
 - Bijective? (import reverse transformation?)

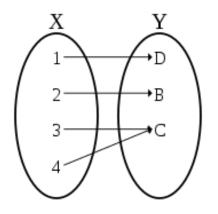
Model Transformations

- Different kind of transformations
 - Do source and target comply with same MM?
 - Specific variant Model to text
- Mathematical properties
 - Injective? map distinct elements of source to distinct elements of target model
 - Surjective? for each element in the target model, there is an associated element in the source model
 - Bijective? injective and surjective

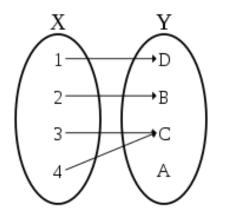
Mathematical properties



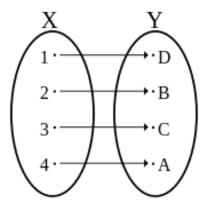
Injective, but not surjective



Non injective, but surjective



Non injective, non surjective



Bijective

Source: wikipedia

Practical implications?

- For a bijective transformation, there is an associated reverse transformation without loss of information
- Would enable round-trip engineering, e.g.
 model => code => modified code => model

- Might need additional annotations (e.g. special comments)
- Not useful, if differences in abstraction level are too high

Agenda

1. UML MARTE

- 1. Motivation, history, packages
- 2. Selected packages
- 3. Example
- 4. Schedulability analysis

2. Model transformation

- 1. Principles
- 2. Languages

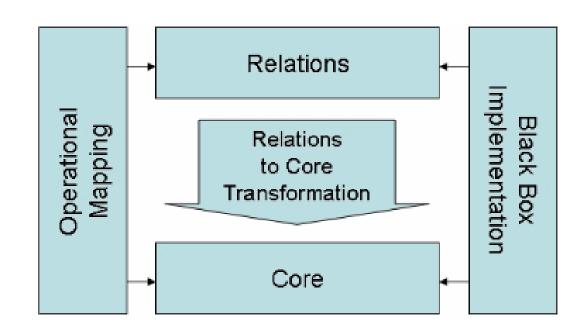
Model transformation implementations

- General purpose programming language, e.g. Java
 - Use UML2 (modeling language) API to access model, remove, add or change model elements programmatically
 - Papyrus SW designer follows this approach, supported by a lazy model copier
 - Advantage: known language, debugging environment
 - Disadvantage: less adapted to task

- Use a dedicated model transformation language
 - QVT (Query/View/Transformation), OMG
 - ATL (ATLAS Transformation Language), INRIA, maintained by OBEO
 - For M2T: Acceleo and xtend (we will see these languages later)

QVT (Query/View/Transformation)

- OMG standard
- Set of three languages for model transformation
 - Operational imperative language for unidirectional transformations.
 - Relations declarative language for bidirectional model transformations
 - Core infrastructure language for the first two, but currently not implemented like this



History

- 2002: OMG issued a Request for proposal (RFP) on MOF Query/View/Transformation
 - Seek a standard compatible with the Model Driven Architecture (MDA) recommendation suite (UML, MOF, OCL, etc.).
 - Several replies, evolution towards common proposal,
- 2005: first version was submitted and approved
- 2011: QVT Version 1.1
- 2016: QVT Version 1.3

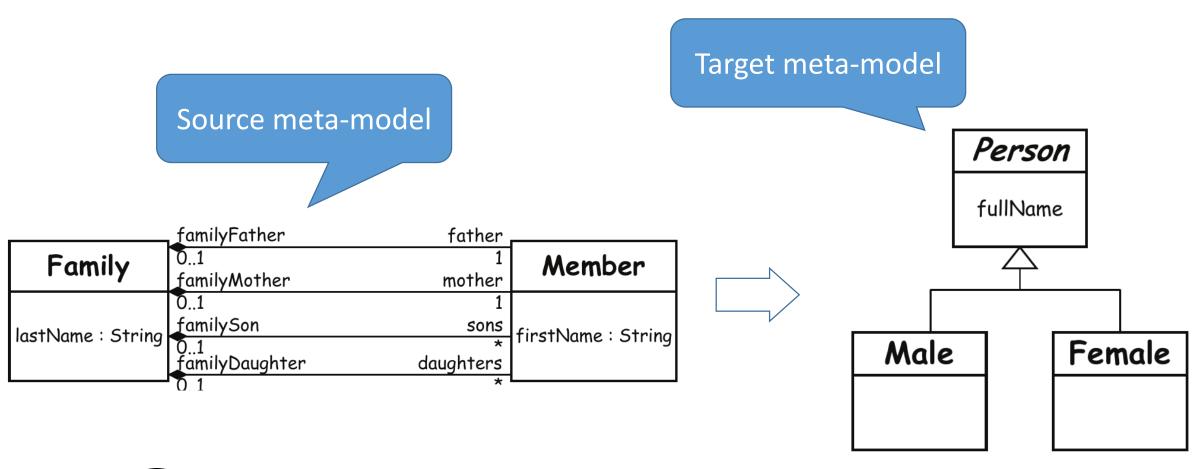
ATL

- Originally developed by INRIA, maintained by OBEO, less used today
- Example (courtesy of Freddy Allilaire, Frédéric Jouault INRIA)

Family March
Father: Jim
Mother: Cindy
Son: Brandon
Daughter: Brenda
... other Families

Not text, but models

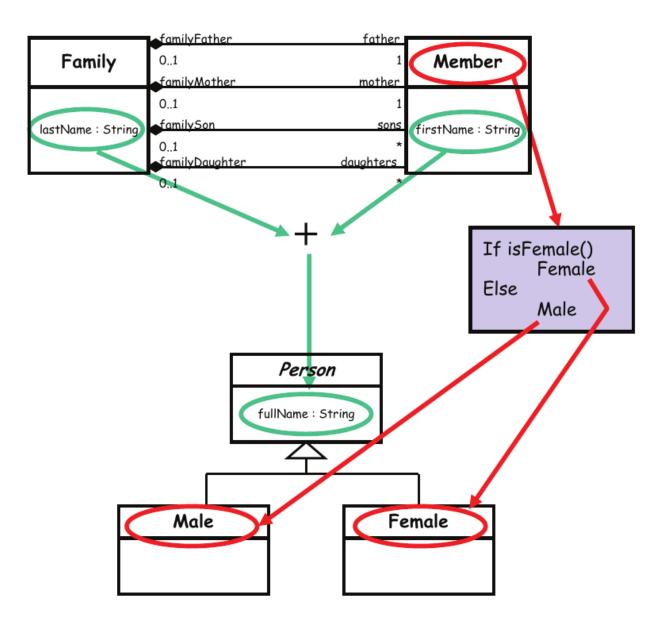
Source and target meta-models



Bijective transformation?

Transformation rules

```
rule Member2Male {
   from
       s : Families!Member (not s.isFemale())
   to
       t: Persons!Male (
           fullName <- s.firstName + ' ' + s.familyName
rule Member2Female {
   from
       s : Families!Member (s.isFemale())
   to
       t: Persons!Female (
           fullName <- s.firstName + ' ' + s.familyName
```



Merci!

Questions?