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
Non-Functional Properties (NFPs)

Non-functional properties describe the “fitness” of systems behavior (E.g., performance, memory usage, power consumption)

- **Nature of NFPs**
  - Quantitative: magnitude + unit (E.g., energy, data size, 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?
- Annotations mechanism?
- NFP Libraries?

- And UML expression syntax is also not sufficient!!

- Variables?
- Structured Values?
- Data Type System?
- Complex time expressions?
The MARTE’s NFP sub-profile

- Three mechanisms to annotate UML models:
  - Values of stereotype properties
    - «hwProcessor»
      - MyProcessor
      - speedFactor = 1
  - Slot values of classifier instances
    - MyProcessor
      - speedFactor: Integer [0..1]
      - proc1 : MyProcessor
        - speedFactor = 1
  - Constraints
    - MyProcessor
      - «nfpConstraint»
        - speedFactor = 1
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 tag 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 in user models
   - Define classifiers and their attributes using NFP types
   - Such attributes are tagged as «nfp»

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

Model-specific NFPs
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
- «nfpConstraint» is a required, offered, or contract constraint of NFPs
The MARTE’s NFP Modeling Framework

- Three main language extensions to UML syntax
  - Grammar for extended expressions
  - Stereotypes for extended data types
  - Complex time expressions
## Basic Textual Expressions in VSL

### Scope of the proposed extensions

- Extended Primitive Values
- Extended Composite Values
- Extended Expressions

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

- BoundedSubtype
- IntervalType
- CollectionType
- TupleType
- ChoiceType

Examples::DataTypesUse

MyClass

cl: MyClass

<table>
<thead>
<tr>
<th>length</th>
<th>212333</th>
</tr>
</thead>
<tbody>
<tr>
<td>priorityRange</td>
<td>[0..2]</td>
</tr>
<tr>
<td>position</td>
<td>(2,3)</td>
</tr>
<tr>
<td>shape</td>
<td>(2,3), (1,5)</td>
</tr>
<tr>
<td>consumption</td>
<td>(exp = e^x * v1, unit = mW, source = calc)</td>
</tr>
<tr>
<td>arrival</td>
<td>periodic (period = 10, jitter = 0.1)</td>
</tr>
</tbody>
</table>

MyClass

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

Declaration example...

Power

value: Real
expr: VSL_Expression
unit: PowerUnitKind
source: SourceKind

PeriodicPattern

period: Real
jitter: Real

SporadicPattern

minInterarrival: Real
maxInterarrival: Real

vectorElement: Integer [0..*]
matrixElement: IntegerVector [0..*]

«boundedSubtype»
{baseType = Integer, minValue = -480000, maxValue = +480000}
Long

«dataType»
«intervalType»
{intervalAttrib = bound}
IntegerInterval

bound: Integer [2]

«collectionType»
{collectionAttrib = vectorElement}
IntegerVector

vectorElement: Integer [0..*]

«collectionType»
{collectionAttrib = matrixElement}
IntegerMatrix

matrixElement: IntegerVector [0..*]

«choiceType»
Power

«tupleType»
ArrivalPattern

periodic: PeriodicPattern
sporadic: SporadicPattern

«tupleType»
PeriodicPattern

period: Real

«tupleType»
SporadicPattern

minInterarrival: Real
maxInterarrival: Real

VSL reuses UML DataType constructs, but adds...
Examples of Time Expressions with VSL

- **Specification example in Sequence diagrams…**
  - **Jitter constraint**
  - **Extended duration intervals with bound « [ ] » specification**
  - **Instant Interval Constraint**
  - **Duration expression between two successive occurrences**
  - **Constraint in an observation with condition expression**

```
Sd DataAcquisition

:Controller
  start() { jitter(t0)<(5, us) }
  acquire() { d1<=(1, ms) }
  @t0
  @t1
  @t2
  &d1

:Sensor
  sendData(data) { [(0, ms)..(10, ms)] }
  @t3

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

Duration Observation
Conclusions on MARTE::NFPs

Synthesis of best modeling practices...

- OCL: full constraint language, but hard to use and not real-time oriented
- SPT Profile: built-in TVL language is simpler, but not flexible
- QoS&FT Profile: annotation mechanism is flexible, but complex

⇒ NFP & VSL reuse selected modeling features, while still providing simplicity and flexibility

Foundations...

- Reuse OCL constructs: grammar for values and expressions
- Generic data type system: (based on ISO’s General-Purpose Datatypes)
- VSL extends UML Simple Time model (e.g. occurrence index, jitters)
- Formally defined by abstract and concrete syntaxes (grammar)