Design of Software for Embedded Systems
Programming Languages and Modeling Tools
MARTE for UML

Winter Term 2011/12
TU Chemnitz
Faculty of Computer Science
Operating Systems Group

Dr. Dirk Müller
Overview

- Introduction
- UML: Meta Modeling and Diagram Types
- Profile, Stereotype, Tagged Value
- Predecessors and Relatives
- Package Diagram
- Time
- Examples of Clocks
- Operations on Clocks
- Summary
- References
Introduction

- UML 2 profile for MARTE (Modeling and Analysis of Real-Time and Embedded systems)
- UML 2 profile: concrete extension of UML 2 meta model based on light-weight extension mechanism (UML meta model unchanged)
  - Stereotypes with tagged values
  - Constraints
- Analogy to language constraints on source code level (e.g. MISRA-C/C++): model constraints on model level (also: MISRA-AC-TL for TargetLink)
- Supports modeling, simulation and analysis
- For code generation not so well-suited [1]
- Version 1.0 was published in November 2009
**MOF Meta Modeling**

- "meta" as relative term
- M3 level has itself as meta model (cf. bootstrap)
- Only by such formally defined rules automatic code generation is enabled

Source: [http://upload.wikimedia.org/wikipedia/commons/9/93/M0-m3.png](http://upload.wikimedia.org/wikipedia/commons/9/93/M0-m3.png)

Dirk Müller: Design of SW for Embedded Sys., Winter term 2011/12
UML 2.3 Diagram Types

new in version 2.2

Source: [3], p. 704
UML Profile

“A profile defines limited extensions to a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain.” [3]

- Is a special form of a package
- Reference meta model is required (here: UML-MM)
- New diagram type profile diagram describing the application of profiles
- Attributes
  - Set of stereotypes
  - Set of constraints (restrictions)
    - As commentary (natural language, optionally also in OCL, Object Constraint Language, formally given)

Source: [3], p. 689
Example: EJB Profile

Enterprise Java Beans as components in Java EE

source: [3], p. 686
**Stereotype**

- **Concrete syntax:** name in angle quotes, «name»
- **Abstract syntax:** extension of other meta classes
- Application of a stereotype on a modeling element *tagged value*, typed (new in UML 2)
- Tagged values are specified in commentary
Predecessors and Relatives

- "UML Profile for Schedulability, Performance and Time (SPT)“, version 1.1 was published in 2005
  - For UML 1, thus no further development
- "OMG Systems Modeling Language (OMG SysML)“, current version 1.2 was published in June 2010
  - Modeling of HW, SW, information, processes, persons, facilities possible => no longer SW-centered
  - Additional diagram types
    - Requirement diagram
    - Parametric diagram for analyses
  - More compact language, supports viewpoints
- "UML Profile for System on a Chip (SoC)“, current version 1.0.1 was published in 2006
- "AUTOSAR UML Profile“, current version 1.0.1, 2006
  - Mapping of AUTOSAR meta model to UML/SysML
Package Diagram of MARTE

- NFP: non-functional prop.
- GRM: Generic Resource Modeling
- Alloc: Allocation of functionality to modules (time, space, WCET)
- GCM: Generic Component Model
- HLAM: High-Level Application Modeling
- SRM: SW Resource Model
- HRM: HW Resource Model
- GQAM: Generic Quantitative Analysis Modeling
- SAM: Schedulability Ana. M.
- PAM: Performance Ana. Mod.
- VSL: Value Specification Language
- RSM: Repetitive Structure Modeling

source: [2]
Working Cycle with MARTE

- Many steps automatable
- Sometimes even experts of the analysis domain are dispensable

Source: [2]
Time in MARTE

- **Chronometric**
  - Classical time concept, linked to a clock
- **Logical**
  - Linked to events, asynchronous
- **Synchronous**
  - Linked to events, synchronized execution
  - cf. synchronous MoC with *Lustre*, *Esterel*
  - Required for RT systems with highest requirements

Profile diagram

Source: [4], p. 73
Time Modeling: Clock

Source: [4], p. 74
**Example: 100-Mhz Clock in MARTE**

- **nature**
  - discrete
  - dense
- **isLogical**
  - False $\Rightarrow$ chronometric
  - true $\Rightarrow$ logical or synchronous
- **unitType**
  - Enumeration of possible time units

Source: [1]
Example: Chronometric Clocks

**Example:**

- **applicationTimeDomain:**
- **clock:**
  - { unit = s, standard = UTC }
  - cc1: Chronometric
  - resolution = 0.01

- **clock:**
  - { unit = s, standard = UTC }
  - cc2: Chronometric
  - resolution = 0.01

- **clock:**
  - { nature = dense, unitType = TimeUnitKind, resolAttr = resolution, getTime = currentTime }
  - IdealClock
  - currentTime(): Real

Source: [4], p. 85
**Clocks: NFPs und Operationen**

- Non-functional properties of chronometric clocks
  - Stability (variance measure)
  - Offset (difference)
  - Skew (1\textsuperscript{st} derivative)
  - Drift (2\textsuperscript{nd} derivative)
- Operations
  - Discretization
  - Filtering
  - Delay
  - Chaining

Source: [4], p. 475
Code Generation with MARTE

- Code generation as weak point of MARTE
- Approaches of combination with other languages, e.g. SysML and SystemC [1]

Example for model-driven SW development with several stages

- Specification and Implementation (by human)
- MARTE
- SysML
- Promela
- SPIN
- Model OK: yes/no
- Verification
- Generator
- SystemC
- RTL VHDL
- Verilog

Software: executable Code

Hardware: FPGAs or ASICs

Implementation (product)
SPARK Profile for UML

- Integration with code generation from the other side (compared to MARTE profile)
- UML quite C++ oriented => problems when specifying information flow
- Automating possible with Rhapsody in Ada (UML CASE tool)

Source: [7]
AADL

- Architecture analysis and design language
- Origin in aviation, today widely used in automotive
- Standardized by Society of Automotive Engineers (SAE), version 1.0 (November 2004), version 2 (January 2009)
- Support HW and SW => good for embedded systems (combination of application and execution platform)
- Supports scheduling analysis => good for RT systems
- Complicated constructs, not all orthogonal
- Better: mapping to UML MARTE, mostly possible

Source: [9]
EAST-ADL2

- **Modeling language** as DSL for **automotive industry**
- EU research projects since 2004, combines software and system development
- Combination of UML and natural language
- Fits well to **AUTOSAR** standard
- **Functional decomposition** by hierarchical modeling enabled
- Can be mapped to UML MARTE
- Also combination with MARTE suggested, in order to improve on schedulability analysis capabilities [11]
Summary

- **MARTE** is a profile for UML, a light-weight extension, in 2009 standardized by OMG
- Enables **modeling, simulation and analysis** of embedded and RT systems
- Extensive modeling of time as basis for real-time systems, also synchronous MoC supported
- Code generation more complicated
- Combination e.g. with SysML and SystemC as a good and viable solution
- Also combination of UML with **SPARK** possible: **SPARK** profile
# Simulink vs. UML MARTE

<table>
<thead>
<tr>
<th>Simulink</th>
<th>UML MARTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support of <strong>time-discrete</strong> and <strong>time-continuous</strong> Models of Computation</td>
<td><strong>OO concepts</strong> imply better modularity and re-usability, thus changeability</td>
</tr>
<tr>
<td><strong>Data flow</strong> is easier to specify</td>
<td>Better for system specification (NFPs)</td>
</tr>
<tr>
<td>Advance in <strong>automatic code generation</strong> (e.g., using RTW)</td>
<td><strong>Code blocks</strong> often have to be inserted manually, but sometimes already code generators</td>
</tr>
<tr>
<td><strong>Changeability</strong> restricted due to missing OO concepts</td>
<td></td>
</tr>
</tbody>
</table>

Source: [8]
Summary Languages and Modeling

- Discrimination concerning product
  - Software (executable program): classical compiler
  - Hardware (FPGAs/ASICs): HW description languages
  - HW-/SW-Codeign (both): e.g. SystemC

- Discrimination concerning abstraction level
  - Higher programming languages (C, C++, PEARL, Ada, Lustre, Esterel)
  - MDSD (Matlab/Simulink, MARTE/SysML)

- Goals of MDSD
  - Shorter development times (time to market)
  - Early testing enabled => dependability increases
  - Rapid Prototyping for customers
  - Decoupling of domain and platform knowledge
  - Portability (simplified change to new platforms)

- Integration of development cycle for Simulink and ASCET already succeeded; for UML MARTE work in progress
- SPARK with static analysis as replacement for tests; integrated development environments (GNAT)
Approaches to Correctness of SW

- **Selection** of suitable formalisms (languages, models)
- **Restriction** of language constructs, Establishing rules and guidelines ("Less is more."")
  - On code level
    - *MISRA-C:2004, MISRA-C++:2008*
    - *SPARK* based on *Ada*
  - On model level
    - *MISRA-AC-TL* for *TargetLink*
- **Design/Program by Contract** ("Trust is good, control is better.!")
  - Pre- and postconditions, dynamical, e.g. *Eiffel*
  - Annotations in source code (redundancy), e.g. *SPARK*
- **Model Checking**, theorem proofs ("The devil is in the details.")
  - Translation in *Promela*, check with *SPIN; UPPAAL*
  - Static check (theorem proofs) e.g. with *SPARK*
- Use of **specific** analysis tools
  - Use of tools for *Lustre/Esterel* [5]
  - Use of analysis tools for *MARTE*
### Comparison of Programming Languages/Modeling Methods

#### Table C.1 — Recommendations for specific programming languages

<table>
<thead>
<tr>
<th>Programming language</th>
<th>SIL1</th>
<th>SIL2</th>
<th>SIL3</th>
<th>SIL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Ada</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>2 Ada with subset</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3 MODULA-2</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>4 MODULA-2 with subset</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>5 PASCAL</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>6 PASCAL with subset</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>7 FORTRAN 77</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>8 FORTRAN 77 with subset</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>9 C</td>
<td>R</td>
<td>---</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>10 C with subset and coding standard, and use of static analysis tools</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>11 PL/M</td>
<td>R</td>
<td>---</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>12 PLM with subset and coding standard</td>
<td>R</td>
<td>R</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>13 Assembler</td>
<td>R</td>
<td>R</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>14 Assembler with subset and coding standard</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>15 Ladder Diagrams</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>16 Ladder Diagram with defined subset of language</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>17 Functional Block Diagram</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>18 Function Block Diagram with defined subset of language</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>19 Structured Text</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>20 Structured Text with defined subset of language</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>21 Sequential Function Chart</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>22 Sequential Function Chart with defined subset of language</td>
<td>HR</td>
<td>HR</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>23 Instruction List</td>
<td>R</td>
<td>---</td>
<td>NR</td>
<td>NR</td>
</tr>
<tr>
<td>24 Instruction List with defined subset of language</td>
<td>HR</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

**Source:** “IEC 61508-7: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 7: Overview of techniques and measures”
# Programming Languages and Safety

<table>
<thead>
<tr>
<th>Programming language</th>
<th>Safety characteristics (advantages)</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ada 83</strong></td>
<td>• Runtime checks mandatory&lt;br&gt;• Pointers are initialized&lt;br&gt;• Type-safe, also across package borders&lt;br&gt;• Recovery from failures</td>
<td>• Access to non-initialized variables (scalars) with 'Valid orpragma Initialize_Scalars;' in Ada 95 cured</td>
</tr>
<tr>
<td><strong>Modula-2</strong></td>
<td>• Type-safe, also across package borders&lt;br&gt;• (Limited) recovery from failures</td>
<td>• Unset variables (scalars) and pointers</td>
</tr>
<tr>
<td><strong>Pascal</strong></td>
<td>• Strongly typed</td>
<td>• Runtime checks only optional&lt;br&gt;• Unset variables (scalars) and pointers</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>• (only by additional tools: <em>make</em> and <em>lint</em>)</td>
<td>• 150 not defined „Features“&lt;br&gt;• Often no runtime checks</td>
</tr>
<tr>
<td><strong>Fortran 77</strong></td>
<td>• Type checking&lt;br&gt;• No pointers</td>
<td>• Default assignments&lt;br&gt;• No checks across package borders</td>
</tr>
</tbody>
</table>

Source: [6], p. 412
References


