Tutorial:
MARTE: Also a UML profile for AADL

SAE AADL meeting Seattle 2009

Madeleine FAUGERE

Research & Technology
Modeling and analysis of real-time and embedded systems, including their software and hardware aspects

**MARTE:**
- Provides support for non-functional property modeling
- Adds rich time and resource models to UML
- Defines concepts for software and hardware platform modeling
- Defines concepts for allocation of applications on platforms
- Provides support for quantitative analysis (e.g. scheduling, performance)
- Annexes: ARINC 653 API, OSEK APIs, AADL guidelines,…

**AADL:**
- A core language providing full support for
  - modeling the application tasks and communication architecture,
  - the hardware platform and the physical environment of embedded software-intensive systems,
  - predeclared properties to characterize task execution and communication timing, as well as deployment of the application on the hardware platform.
- Annexes: a collection of standardized property to meet specific embedded system analysis needs such as
  - security analysis,
  - dependability analysis,
  - behavioral analysis,
  - ARINC 653,
  - support for automated generation and integration of systems.
Similarities

- Cover the same objectives: languages for RTES design and analysis
- Provide software/hardware application design, binding, analysis capabilities,…
- Have semantic equivalencies

Differences

- UML/MARTE comes from the UML world,
  - UML/MARTE covers a wide modeling range, with multiple abstraction layers, modeling capabilities from requirements, high level design, detail design, to code generation
  - but in return …benefits from UML loose semantics, which becomes inconvenient for code generation and simulation aspects.

- AADL comes from the DSL world
  - AADL address only the needed concepts
  - Benefits from less concepts, addresses a specific abstraction layer (in terms of resources)
  - Based on well defined and formalized execution semantics (AADL MoC)
Stakes of standards convergency

Langages, methods and tools interoperability
- Standard convergence
- Tool interoperability

Interopérabilité

Tool based design

Standards

Capitalize execution platform knowhow in libraries
**OMG Acceptance:**

- MARTE V1.0: standard available at OMG
- MARTE has a dedicated Annex to AADL: AADL guidelines
  - Ensure language interoperability between AADL and UML/MARTE
  - Catch among UML/MARTE rich features the right AADL semantics and AADL abstraction level
  - Explicit AADL implicit semantics
  - Provides a good AADL features graphical abstraction
  - Provide User friendly design
  - User will be placed in AADL code generation viewpoint

**SAE acceptance:**

- MARTE is the UML profile for AADL language
  - SAE acceptance for AADL v2 specification (AS5506 01/09)
Components and packages
Software components
Execution platform components
Systems
Features and Shared access
Ports and Ports connections
Flows
Modes and mode transition

- Each AADL explicit/implicit concept has been mapped on a MARTE concepts
- For each AADL predeclared property a MARTE property representation is proposed
## Components and packages

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract component</td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td></td>
</tr>
</tbody>
</table>

## Software components

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>MARTE memoryPartition stereotype on UML Classifier</td>
</tr>
<tr>
<td>Thread</td>
<td>MARTE swSchedulableResource stereotype on UML Classifier</td>
</tr>
<tr>
<td>ThreadGroup</td>
<td></td>
</tr>
</tbody>
</table>
| Data         | • Uml DataType for a AADL primitive type,  
|              | • Uml DataType with attributes for a structured data type,  
|              | • swMutualExclusionResource for a concurrent resource meaning |
| Data access  | UML2 port typed by UML interface composed of ONE attribute |
| Subprogram access | UML2 port typed by UML interface composed of ONE operation |
| Feature group | UML2 port typed by UML interface composed of at least two attributes or operations |
| Subprogram group + access |          |
## Execution platform components

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>hwProcessor stereotype on UML Classifier</td>
</tr>
<tr>
<td>Virtual processor</td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>hwMemory stereotype on UML Classifier</td>
</tr>
<tr>
<td>Bus</td>
<td>HwBus stereotype on UML Classifier</td>
</tr>
<tr>
<td>Virtual Bus</td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>hwDevice stereotype on UML Classifier</td>
</tr>
</tbody>
</table>

## System

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>SysML Block on UML composite structure</td>
</tr>
</tbody>
</table>

## Features and shared access

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Port</td>
<td>FlowPort typed by an UML Primitive type or Data type</td>
</tr>
<tr>
<td>Event Port</td>
<td>ClientServerPort typed by UML signal without data attributes</td>
</tr>
<tr>
<td>EventDataPort</td>
<td>ClientServerPort typed by UML signal with only ONE data attributes</td>
</tr>
<tr>
<td>Inverse of</td>
<td>IsConjugated stereotyped attribute on “FlowPort” and “ClientServerPort” UML ports</td>
</tr>
</tbody>
</table>


## Connections and flows

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>UML delegation connectors between Ports and Parts on composite diagrams</td>
</tr>
<tr>
<td></td>
<td>UML assembly connectors between parts</td>
</tr>
<tr>
<td>Flows specifications</td>
<td></td>
</tr>
<tr>
<td>End-To-End Flows</td>
<td></td>
</tr>
</tbody>
</table>

## Mode

<table>
<thead>
<tr>
<th>AADL concept</th>
<th>MARTE/UML concept for design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td></td>
</tr>
</tbody>
</table>
Concepts mapping

- Component, packages
AADL language makes a distinction between declaration and implementation concepts

- This distinction will be kept in UML (Extension will be represented by an UML Generalization link, implementation by UML realization links)

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>package example</td>
<td></td>
</tr>
<tr>
<td>public</td>
<td></td>
</tr>
<tr>
<td>system sys1</td>
<td></td>
</tr>
<tr>
<td>features</td>
<td></td>
</tr>
<tr>
<td>e: in event port;</td>
<td></td>
</tr>
<tr>
<td>end sys1;</td>
<td></td>
</tr>
<tr>
<td>system sys2 extends sys1</td>
<td></td>
</tr>
<tr>
<td>features</td>
<td></td>
</tr>
<tr>
<td>e2: in event port;</td>
<td></td>
</tr>
<tr>
<td>end sys2;</td>
<td></td>
</tr>
<tr>
<td>system sub_sysA</td>
<td></td>
</tr>
<tr>
<td>end sub_sysA;</td>
<td></td>
</tr>
<tr>
<td>.... system sub_sysB</td>
<td></td>
</tr>
<tr>
<td>end sub_sysB;</td>
<td></td>
</tr>
<tr>
<td>system implementation sys1.impl</td>
<td></td>
</tr>
<tr>
<td>subcomponents</td>
<td></td>
</tr>
<tr>
<td>s1: system sub_sysA;</td>
<td></td>
</tr>
<tr>
<td>end sys1.impl;</td>
<td></td>
</tr>
<tr>
<td>system implementation sys2.impl extends sys1.impl subcomponents</td>
<td></td>
</tr>
<tr>
<td>s2: system sub_sysB;</td>
<td></td>
</tr>
<tr>
<td>end sys2.impl;</td>
<td></td>
</tr>
<tr>
<td>end example;</td>
<td></td>
</tr>
</tbody>
</table>
AADL package is a namespace with private and public sections.

Components, systems, subcomponents, properties and mode specifications can be made public/private in relation with other subcomponents or parents system.

- Public entities are visible to other components, while the private one are only accessible within the private section.

- AADL “Private” and “Public” concepts have been mapped to the UML visibility concepts, represented as meta-attribute owned by every “NamedElement”.
package example
public
system sys1
features
  e: in event port;
end sys1;

system implementation sys1.impl
modes
  m1: initial mode;
  m2: mode;
  m1-[e]->m2;
  m2-[e]->m1;
end sys1.impl;

private
system sys2
end sys2;

system implementation sys1.impl
subcomponents
  s1: system sys2;
  s2: system sys2  in modes ( m2);
end sys1.impl;
end example;
AADL abstract component category represents a component that can be refined into any concrete component categories. It will be represented as an abstract UML classifier, refined using “refined” UML abstraction according AADL constraintes (features, access)

abstract car
end car;

abstract power_train
end power_train;

abstract exhaust_system
end exhaust_system;

abstract implementation car.generic
subcomponents
  PowerTrain: abstract power_train;
  ExhaustSystem: abstract exhaust_system;
end car.generic;

end carRT;

system implementation carRT.impl extends car.generic
subcomponents
  PowerTrain: refined to system power_train;
  ExhaustSystem: refined to system exhaust_system;
end carRT.impl;
Prototypes represent parameterization of component type, component implementation, and feature group type declarations.

- AADL provide also refinement capabilities (in/out, requires/provided, port kind, component category),
- UML’s template concept provides only “typing” capabilities
  - Only this AADL subset will be considered in a first time
Concepts mapping

- Component, packages
- Some software components concepts
The AADL data component represents different AADL concepts:
- primitive data types,
- structured data types
- mutual exclusion resources within a System
Each AADL primitive type from the data_types packages will have an UML/MARTE primitive type equivalent.

```plaintext
package data_types
public
  data integer
  properties
    data_model::data_representation => integer;
  end integer;
...
end data_types;
```
A structured data type will be represented by an UML Data Type with corresponding attributes:

```aaml
<<dataType>>
my_struct
  properties
  data_model::data_repr=>struct;
  data_model::enumeration=>("pA","pB");
  data_model::base_types=>(classifier integer, classifier float);
end my_struct;

data my_struct
  properties
    data_model::data_repr=>struct;
    data_model::enumeration=>("pA","pB");
    data_model::base_types=>(classifier integer, classifier float);
end my_struct;

data logs
end logs;
```
A MutualExclusion Resource on data declaration will be represented by a MARTE concurrency concept.

**UML Primitive type also stereotyped «MutualExclusionResource »**

```plaintext
data my_data
properties
    concurrency_control_protocol => (PIP, PCP, No, ...);
end my_data;
```

**UML Data Type also stereotyped «MutualExclusionResource »**

```plaintext
data my_struct
properties
    data_model::data_repr => struct;
    data_model::enumeration => ("pA", "pB");
    data_model::base_types => (classifier integer, classifier float);
    concurrency_control_protocol => (PIP, PCP, No, ...);
end my_struct;
```

**Prerequisite:**
An AADL property project with MARTE concurrency protocols (PIP, PCP, No, etc.) shall exist.
AADL Data instance considered as « mutual exclusion resource »

```
data logs
end logs;

system my_system
end my_system;

system implementation my_system.I
subcomponents
  l: data logs; {
    concurrency_control_protocol => PIP;}
end my_system.I;
```
Different types of features, different usages

- **AADL Data port**
  - Interfaces for typed state data transmission among components without queuing.
  - Connections between data ports are either immediate or delayed.

- **AADL Event port**
  - Interfaces for the communication of events raised by subprograms, threads, processors and devices (examples: trigger for the dispatch of aperiodic thread, initiator of mode switch, alarm communications,…).
  - Events may be queued. Event such alarms may be queued by the recipient, and the recipient may process the queue content.

- **AADL Event Data port**
  - Interfaces for message transmission with queuing. Enables the queuing of data associated with an event.
  - Message arrival may cause dispatch of the recipient and allow the recipient to process one or more messages.

- **Data access**
  - Modeling of shared access to a common data area or static data

- **Subprogram access**
  - Access to subprogram component in enclosing thread group, process, or system. Execution by calling thread.

- **Bus access**
  - Connectivity of execution platform components through buses whose access they share.
AADL Data port

- AADL Data Ports will be represented as MARTE “Flow Ports” typed by an UML Primitive type or Data Type
- There will be no queuing information associated to the MARTE Flow Port

```plaintext
process control
features
  speed: in data port raw_speed;
  throttle_command: out data port Command_data;
end control;
```
AADL Event Data port:

AADL Event Data Ports will be represented as MARTE ClientServerPort typed by an UML signal with only ONE AADL data attribute

```aadl
<<ClientServerPort>>
isAtomic=true;
direction=in;
<<MessageComResource>>
MessageQueueCapacityElements=1.00;
Mechanism:MessageQueue;
messageQueuePolicy:FIFO;

<<signal>>
signal_raw_set_speed
  .data: raw_set_speed

<<dataType>>
raw_set_speed

<<memoryPartition>>
control

set_speed2: raw_set_speed;
end raw_set_speed;

process control features

set_speed: in event data port raw_set_speed;
set_speed2: in data port raw_set_speed;
end control;
```
AADL Event port

- AADL Event Ports will be represented as MARTE ClientServerPorts typed by an UML signal WITHOUT data attributes

```
<<signal>>
reset

<<memoryPartition>>
control
control : reset
```

process control
features
control: in event port;
end control;
AADL Data access will be represented by

- UML 2 ports typed by a UML interface
- UML delegation/assembly connection represents AADL data access connections and UML provided/required interface concept the AADL provides/requires data access
- UML Interface composed of an UML attribute representing the access to ONE AADL data

```
System global_system
end global_system;

System implementation global_system.I
subcomponents
  sys1 : system my_system1.i;
  sys2 : system my_system2.i;
connections
  data access sys1.dp -> sys2.dp;
end global_system.I;

System my_system1
features
  dp : provides data access D;
end my_system1;

System implementation my_system1.I
subcomponents
  d : data D;
connections
  data access d -> dp;
end my_system1.I;

System my_system2
features
  dp : requires data access D;
end my_system2;

System implementation my_system2.I
subcomponents
  proc1 : process my_process;
connections
  data access dp -> proc1.data_access;
end my_system2.I;

Process my_process
features
  data_access : requires data access D;
end my_process;

Data D
end D;
```
AADL Subprogram access will be represented by
- UML 2 ports typed by a UML interface
- UML Interface composed of an UML operations representing the access to ONE subprogram
- UML delegation/assembly connection represents AADL subprogram access connections and UML provided/required interface concept the AADL provides/requires data access

System implementation
global_system.I
subcomponents
sys1 : system my_system1.i;
sys2 : system my_system2.i;
connections
subprogram access sys1.sp ->
sys2.sp;
end global_system.I;

System my_system2
features
sp : requires subprogram access
sub_p1;
end my_system2;

Subprogram sub_p1
end sub_p1;

System my_system1
features
sp : provides subprogram access
sub_p1;
end my_system1;

System global_system
end global_system;
- AADL Bus access will be represented by
  - MARTE HwBus provides bus access services through a bus_access interface (without passing through ports)
  - UML delegation/assembly connection represents AADL bus access connections
An AADL feature group

- is represented by an UML interface composed by at least of two attributes or two subprogram access
- By default, the interface is provided
- Subprogram access and Data access are Feature Group particular cases
- UML delegation/assembly connection represents AADL feature group access

```
System implementation global_system.1
subcomponents
    sys1 : system my_system1.i;
    sys2 : system my_system2.i;
connections
    feature group sys1.share -> sys2.share;
end global_system.1;

System my_system2
features
    share : inverse of feature group multi_access;
end my_system2;

Subprogram sp1 end sp1;
Subprogram sp2 end sp2;
Data D end D;
Data E end E;

System my_system1
features
    share : feature group multi_access;
end my_system1;

Feature group multi_access
features
    d : provides data access D;
    e : provides data access E;
    sp1 : provides subprogram access sp1;
    sp2 : provides subprogram access sp2;
end multi_access;
```
Subprogram groups represent subprogram libraries accessible to other components through subprogram group access features and subprogram group access connections.
A thread group represents an organizational component to logically group threads contained in processes.

- Thread group types and implementations specifies the features, required subcomponent access, contained thread connectivity.

A thread group will be represented as an abstract UML classifier stereotyped “swSchedulable Resource” (used to made de distinction with abstract component).
Concepts mapping

- Component, packages
- Some software components concepts
- Some hardware mapping concepts
A virtual bus component represents logical bus abstraction such as a virtual channel or communication protocol. It will be represented at resource level as a MARTE “CommunicationMedia” stereotyped UML connection or classifier allocated to the physical HWBus.

- If the communication media represents a bus, and the clock is the bus speed, "element size" would be the width of the bus, in bits.
- If the communication media represents a layering of protocols, "element size" would be the frame size of the uppermost protocol.
A virtual processor represents a logical resource that is capable of scheduling and executing threads and other virtual processors bound to them. It will be represented as a MARTE “swSchedulingResource” AND “ProcessingResource” stereotyped UML Classifier.
AADL Binding

MARTE "Allocated" concept
Concepts mapping

- Component, packages
- Some software components concepts
- Some hardware mapping concepts
- Modes representation
An operational mode can represent different things:

- A phase of a system operation e.g., starting, stopping, reconfiguring switchers, in a supervisory control system of an electric grid.
- An operational system (or subsystem) state that is managed by reconfiguration mechanisms (e.g., fault-tolerance management middleware) according to fault conditions.
- A state of system operation with a given level of QoS that can be handled by resource management infrastructures (e.g., middleware that assign resources at run time according to load demand, timing constraints, or resource usage).
MARTE has been enhanced with mode concepts

You are hereby notified that any review, dissemination, distribution, copying or otherwise use of this document is strictly prohibited without Thales prior written approval.

©THALES 2008. Template trtp version 7.0.5

Mode concepts

The information contained in this document and any attachments is the property of THALES. You are hereby notified that any review, dissemination, duplication, copyright or diversion use of this document is strictly prohibited without prior written approval. ©THALES 2008. Template trtp version 7.0.5
A UML Composite structure will represent the mode specific typology
A UML State Machine, State and Transition will be used to represent the reconfiguration mechanism.
Data access in modes
Concepts mapping

- Component, packages
- Some software components concepts
- Some hardware mapping concepts
- Modes representation
- Flow representation
### Flow Path

- A logical flow of data and control through a sequence of threads, processors, devices and port connections and data access connections.
- Flow path will be represented by “UML InformationItems”, represented by UML Dependencies stereotyped “flows”. Flow Sinks and Flow Source will be implicit, they are represented by UML ports staying for event, event data, data, and data access and feature group.

```
Flow declaration:
Flow path flow1 : ep → ep2
Flow path flow2 : dp → ep2;
```

```
Flow implementations:
Flow path flow1 : ep → C1 → B.F3 → C3 → C.F4 → C4 → ep2;
Flow path flow2 : ep → C2 → B.F3 → C3 → C.F4 → C4 → ep2;
ep → C1 → B.F3 → C3 → C.F4 → C4 → ep2
```
An end-to-end flow represents a logical flow of data and control from a source to a destination through a sequence of threads that process and possibly transform the data.

Two ways of representation:
- Sequence diagrams
- Activity diagrams
ETE1: end to end flow B.F3 → C3 → C.F4;
The use of activity diagrams for end-to-end flows representation makes explicit the different objects transmitted between the various actions (allocated one of the threads instances).

These objects properties intrinsically take into account timing aspects like queueing policies and dequeue protocols, impacting the final end-to-end latency.
Object Nodes represent queue between 2 tasks

- Event/ event-Data ports -> Central Buffer Nodes
  - Dequeued policy: Implicitly OneItem Protocol supported in UML
  - QueueSize: Lower/Upperbound property
- Data ports -> DataStore
  - UpperBound set to one, data never dequeud
Ongoing work and conclusion

- MARTE v1.0 has to be aligned with AADL v2 new features
  - New MARTE issues will be raised

- Mapping currently in validation process (Lambda project, Ellidiss)

- Objectives
  - Commit (OMG, SAE) the tutorial by the end of Q1 2010