Important Characteristics of VHDL-AMS and Modelica with Respect to Model Exchange

1st International Workshop on Equation-Based Object-Oriented Languages and Tools, Berlin, July 30, 2007

Olaf Enge-Rosenblatt,
Joachim Haase,
Christoph Clauß

Fraunhofer Institute for Integrated Circuits
Design Automation Division
Dresden, Germany
Outline

1. Language characteristics
   - VHDL-AMS
   - Modelica

2. Examples
   - Conservative system
   - Structural description

3. Comparision of some aspects

4. Transformation of models
   - Modelica $\Rightarrow$ VHDL-AMS
   - VHDL-AMS $\Rightarrow$ Modelica

5. Conclusions
1. Language characteristics

VHDL-AMS

- VHDL 1076-1993
  - Time-discrete systems
  - Event-driven simulation algorithms

- VHDL-AMS
  - Extension to time-continuous (analog) systems
  - Analog DAE-solver for differential algebraic systems of equations
    \[ F(x, \dot{x}, t) = 0 \]
  - http://www.vhdl.org/analog

VHDL-AMS is a superset of VHDL. Concepts and language constructs of digital VHDL remain valid.
1. Language characteristics

Modelica

- Object-oriented modelling language
- Description of physical systems based on
  - differential equations
  - algebraic equations
  - discrete equations
- Modelica Association
  Modelica Language Specification
  Version 2.0, July 10, 2002
- http://www.modelica.org

Modelica is a unified language specification for systems characterized by differential-algebraic equations taking into account discrete events for definition of discontinuous behaviour or sampled-signal systems.
1. Language characteristics

Modelling approach

Mathematical model

- conservative systems (Kirchhoff’s network)
- non-conservative systems (signal flow graph)
- mixed description possible
  - analog-digital
  - electrical – non-electrical
  - conservative – non-conservative
  - ...

Implementation of models

- VHDL-AMS
- Modelica
Resistance model in VHDL-AMS (1)

```vhdl
library IEEE;
use IEEE.electrical_systems.all;

entity resistor is
  generic (res : resistance); -- Resistance [Ohm]
  port (terminal p1, p2 : electrical);
end entity resistor;

architecture ideal of resistor is
  quantity v across i through p1 to p2;
begin
  v == res * i;
end architecture ideal;
```

2. Examples – conservative system

Conservative System

Resistance model in VHDL-AMS (1)

- **Library Declaration**: `library IEEE;`
- **Use Declaration**: `use IEEE.electrical_systems.all;`  
- **Entity Definition**:
  - `entity resistor is`
  - `generic (res : resistance);` — Resistance [Ohm]
  - `port (terminal p1, p2 : electrical);`
- **Architecture Definition**:
  - `architecture ideal of resistor is`
  - `quantity v across i through p1 to p2;`
  - `v == res * i;`

**Diagram**:  
- A resistor with terminals `p1` and `p2`, representing the ideal resistance model.
- **Equation**: $v = res \cdot i$
2. Examples – conservative system

Resistance model in VHDL-AMS (2)

```vhdl
-- library IEEE_proposed
package ELECTRICAL_SYSTEMS is

-- subtype declarations
subtype VOLTAGE is REAL tolerance "DEFAULT_VOLTAGE";
subtype CURRENT is REAL tolerance "DEFAULT_CURRENT";

-- nature declarations
nature ELECTRICAL is
  VOLTAGE across CURRENT through ELECTRICAL_REF reference;

-- ...
end package ELECTRICAL_SYSTEMS;
```

**ELECTRICAL** – nature name to determine physical domain

**CURRENT** – type of the associated flow quantity

**VOLTAGE** – type of the associated non-flow quantity

**ELECTRICAL_REF** – identifier of the electrical reference node (potential = 0.0)
2. Examples – conservative system

Resistance model in Modelica (1)

model Resistor "Ideal linear electrical resistor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  parameter SIunits.Resistance R=1 "Resistance";

  equation
  v = R * i;
end Resistor;

model Resistor "Ideal linear electrical resistor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
  parameter SIunits.Resistance R=1 "Resistance";

  equation
  v = R * i;
end Resistor;
2. Examples – conservative system

Resistance model in Modelica (2)

```modelica
// Modelica.Electrical.Analog.Interfaces

connector PositivePin "Positive pin of an electric component"
   SIunits.Voltage  v "Potential at the pin";
   flow SIunits.Current i "Current flowing into the pin";
end PositivePin;

connector NegativePin "Negative pin of an electric component"
   SIunits.Voltage  v "Potential at the pin";
   flow SIunits.Current i "Current flowing into the pin";
end NegativePin;

// Modelica.SIunits

type ElectricCurrent = Real (final quantity="ElectricCurrent", final unit="A");
type Current = ElectricCurrent;

type ElectricPotential = Real (final quantity="ElectricPotential", final unit="V");
type Voltage = ElectricPotential;
```
2. Examples – structural description

RLC Circuit in VHDL-AMS

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.electrical_systems.all;

entity RLC is end RLC;

architecture ex_rlc of RLC is
begin
  VIN : entity work.v_ramp(ideal)
    generic map(pulse => 5.0, duration => 10.0e-6)
    port map(pos => v_input, neg => ELECTRICAL_REF);
  R1 : entity work.resistor(ideal) generic map(res => 3.0)
    port map(p1 => v_input, p2 => v_middle);
  L1 : entity work.inductor(ideal) generic map(ind => 1.0e-3)
    port map(p1 => v_middle, p2 => v_load);
  C1 : entity work.capacitor(ideal) generic map(cap => 10.0e-6)
    port map(p1 => v_load, p2 => ELECTRICAL_REF);
end architecture ex_rlc;
```

Instantiation

- parameter assignment (generic map)
- terminal assignment (port map)
2. Examples – structural description

**RLC Circuit in Modelica (1)**

```modelica
model RLC

Modelica.Electrical.Analog.Sources.RampVoltage V1 (V=5, duration=10.0e-6);
Modelica.Electrical.Analog.Basic.Resistor R1 (R=3.0);
Modelica.Electrical.Analog.Basic.Capacitor C1 (C=10.0e-6, v(start=-5));
Modelica.Electrical.Analog.Basic.Inductor L1 (L=1.0e-3);

equation
  connect(V1.n, Gnd1.p);
  connect(V1.p, R1.p);
  connect(R1.n, L1.p);
  connect(L1.n, C1.p);
  connect(C1.n, Gnd2.p);

end RLC;
```

- **used libraries**
- **instances with their parameters**
- **super classes**
- **description of pin connection**
2. Examples – structural description

## RLC Circuit in Modelica (2)

```modelica
// Modelica.Electrical.Analog.Basic

model Ground "Ground node"
end Ground;
```

**Modelica Language Specification:**

```modelica
connect (a, b)
    a.across = b.across
    a.through + b.through = 0

connect (a, b)
connect (a, c)

    a.across = b.across = c.across
    a.through + b.through + c.through = 0
```
## 3. Comparison of some aspects

### Comparison (1)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>VHDL-AMS</th>
<th>Modelica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>IEEE Std. 1076.1 (revised 2007)</td>
<td>Modelica Specification 2.2 Modelica Association</td>
</tr>
<tr>
<td>Time-continuous modeling</td>
<td>conservative (networks)</td>
<td>physical modeling</td>
</tr>
<tr>
<td></td>
<td>non-conservative (signal-flow)</td>
<td>block-oriented modeling</td>
</tr>
<tr>
<td>Time-discrete modeling</td>
<td>event-driven</td>
<td>event-driven</td>
</tr>
<tr>
<td></td>
<td></td>
<td>no event queue</td>
</tr>
<tr>
<td>Model interface</td>
<td>entity</td>
<td>model block</td>
</tr>
<tr>
<td>Model parameter</td>
<td>generic parameter</td>
<td>parameter</td>
</tr>
<tr>
<td>Connection points</td>
<td>port (terminal, quantity, signal)</td>
<td>specified by connector classes</td>
</tr>
<tr>
<td>Connection point characterization</td>
<td>nature for terminals (through, across, reference)</td>
<td>connector (flow, non-flow)</td>
</tr>
</tbody>
</table>
### 3. Comparison of some aspects

#### Comparison (2)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>VHDL-AMS</th>
<th>Modelica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model behavior</td>
<td>architecture</td>
<td>equation part, algorithm</td>
</tr>
<tr>
<td>Organisation</td>
<td>one or more architectures corresponding to one entity</td>
<td>different models for different levels of abstraction</td>
</tr>
<tr>
<td>Analog behavior</td>
<td>equation oriented expression1 == expression2;</td>
<td>equation oriented expression1 = expression 2;</td>
</tr>
<tr>
<td>Analog waveform</td>
<td>quantity</td>
<td>dynamic variable</td>
</tr>
<tr>
<td>Initial conditions</td>
<td>break statement</td>
<td>initial equation fixed start values</td>
</tr>
<tr>
<td>Discontinuities</td>
<td>break statement</td>
<td>reinit();</td>
</tr>
<tr>
<td>Vector operations</td>
<td>overloading of operators</td>
<td>built-in functions</td>
</tr>
</tbody>
</table>
### 3. Comparison of some aspects

#### Comparison (3)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>VHDL-AMS</th>
<th>Modelica</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital waveform</td>
<td><em>signal</em></td>
<td><em>discrete</em> (Pin)</td>
</tr>
<tr>
<td>D/A conversion</td>
<td>‘RAMP, ‘SLEW</td>
<td><em>smooth</em> ()</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Not supported</td>
<td>Widely used</td>
</tr>
<tr>
<td>Netlists</td>
<td>Instance oriented</td>
<td>Connection point oriented</td>
</tr>
</tbody>
</table>
4. Transformation of models (Modelica $\rightarrow$ VHDL-AMS)

Model exchange

Mathematical description

Modelica $\rightarrow$ partially automated $\rightarrow$ VHDL-AMS
Underlying Branch Structure in Modelica

Basis for transformation of conservative model from Modelica to VHDL-AMS
4. Transformation of models (Modelica ➔ VHDL-AMS)

Modelica to VHDL-AMS (example Resistor)

```
library IEEE;
use IEEE.ELECTRICAL_SYSTEMS.all;

entity RESISTOR is
  generic (R : RESISTANCE := 1.0);
  port (terminal P: ELECTRICAL;
    terminal N: ELECTRICAL);
end entity RESISTOR;

architecture MODELICA of RESISTOR is
quantity P_V across P_I through P;
quantity N_V across N_I through N;
quantity V   : REAL;
quantity I   : REAL;
begn
  V   == P_V – N_V;
  0.0 == P_I + N_I;
  I == P_I;
  V   == R*I;
end architecture MODELICA;
```

```
partial model OnePort
  "Component with two pins
  ...Voltage v  "p.v - n.v";
  ...Current i  "from p to n"
  ....PositivePin p;
  ....NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;

model Resistor
  "Ideal resistor"
extends
  ...OnePort;
parameter ...Resistance R=1;
equation
  v = R*i;
end Resistor;
```

Model correct, but …
Establishing Small (Smart) Models

- group terminals
- connect terminals with norators
- establish (and reduce) network equations
4. Transformation of models (Modelica $\rightarrow$ VHDL-AMS)

**Example - Resistor (1)**

\[
\begin{pmatrix}
1 & -1 & \ldots & -1 \\
\ldots & 1 & 1 & \ldots \\
\ldots & 1 & \ldots & -1 \\
\ldots & \ldots & 1 & -R \\
1 & -1 & \ldots & \ldots & -1 \\
\ldots & 1 & \ldots & \ldots & 1 \\
\ldots & \ldots & 1 & \ldots & -1 \\
\end{pmatrix}
\begin{pmatrix}
p.v \\
n.v \\
p.i \\
n.i \\
v \\
i \\
V_{\text{NOR}} \\
I_{\text{NOR}} \\
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{pmatrix}
\]

**Reduction step**

\[
\begin{pmatrix}
1 & -1 & \ldots & \ldots & -R \\
\ldots & 1 & \ldots & 1 \\
\ldots & 1 & \ldots & -1 \\
\ldots & \ldots & 1 & R \\
\ldots & \ldots & 1 & 1 \\
\ldots & \ldots & 1 & -R \\
\ldots & \ldots & \ldots & \ldots \\
\end{pmatrix}
\begin{pmatrix}
p.v \\
n.v \\
p.i \\
n.i \\
v \\
i \\
V_{\text{NOR}} \\
I_{\text{NOR}} \\
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{pmatrix}
\]

**smart model**
- structure eq. norator structure
- simultaneous statements from (reduced) equations
Example - Resistor (2)

```vhdl
architecture IDEAL of RESISTOR is
  quantity V_NOR across I_NOR through P to N;
begin
  V_NOR == R*I_NOR;
end architecture IDEAL;
```

Only one branch and one simultaneous statement
Example - Resistor

architecture R1 of RESISTOR is
    quantity V across I through node1 to node2;
begin
    V == R*I;
end architecture R1;

model Resistor
    extends ....OnePort;
    equation
        v = R*i;
    end Resistor;
    Resistor R1;

    equation
        connect(R1.p, node1);
        connect(R1.n, node2);
    ...

connector vhdl_pin
    SIunits.Voltage v;
    flow SIunits.Current i;
end vhdl_pin;
vhdl_pin node1, node2;
Conclusions

• Same mathematical modelling approaches are supported by VHDL-AMS and Modelica

• Differences result from the semantic approach:
  - VHDL-AMS comes from electronics
  - Modelica deals mainly with multi-physics problems

• Potential to transform models from one language into the other
  - Special underlying modelling approaches should be considered

• Transformation of conservative models from Modelica to VHDL-AMS would be easier if access to the (reduced) model equations would be possible!
Thank You!