Multi-Aspect Modeling in Equation-Based Languages

EOOLT 2008, Cyprus

Author: Dirk Zimmer

ETH Zürich, Institute of Computational Science, Department of Computer Science
Overview

- Motivation
- Classification of aspects
- Multiple aspects in Modelica
- Current downfalls
- Improved handling in SOL
- Demonstration
- Conclusions
Motivation

• Contemporary equation-based modeling languages are embedded in modeling and simulation environments that feature various types of data-representation:
  – Icons for the graphical user interface (GUI)
  – 3D-Visualization
  – Sound-Module
  – Auto-Documentation
  – etc…

• Thus, the corresponding models contain more information than what is needed for the actual physical model.

• Nowadays, a modeler has to cope with many multiple aspects.
Motivation: Example

Physical modeling

```
equation
  connect(Torque1.flange_b, DiamondFrame1.flange_steering);
  connect(Torque2.flange_b, DiamondFrame1.flange_rw);
  connect(Constant1.y, Torque2.tau);
  connect(Sine1.y, Torque1.tau);
  connect(IdealRollingWheel2.frame_a, DiamondFrame1.frame_rw);
  connect(IdealRollingWheel1.frame_a, DiamondFrame1.frame_fw);
end IdealBike;
```

Visualization

Diagram

Documentation

Information

This example presents the model of a bicycle.

The joints of the bicycle are frictionless and the wheels are ideally rolling. The bicycle is uncontrolled, but due to its initial velocity it is self-stabilizing. Within a certain range of driving velocity a bicycle is stable.

A bicycle has 7 degrees of freedom on positional level and 3 degrees of freedom on velocity level.

System hints

[…] phi(stateSelect = if chooseStates then StateSelect.always else StateSelect.default ), […]
Classification of aspects

- **Following classification of aspects seems appropriate for Modelica**
  
  - **Physical modeling:** The modeling of the physical processes based on differential-algebraic equations (DAEs).
  
  - **System hints:** The supply of hints or information for the simulation-system.
  
  - **3D Visualization:** Description of corresponding 3D-entities that enable a visualization of the models.
  
  - **GUI-Representation:** Description of an iconographic representation for the GUI of the modeling environment.
  
  - **Documentation:** Additional documentation that addresses to potential users or developers.
Code - Analysis

• We analyzed the distribution of these aspects for three exemplary models.

• The examples originate from the Modelica-Standard-Library

• All formatting has been removed.

• The remaining characters have been manually categorized and then counted.

• Let us see the results…
Code - Analysis

A component model: Modelica.Electrical.Analog.Semiconductors.PMOS
The primary aspect cannot be stated to be predominant.

The discussion about Modelica and other EOO-languages is often constrained to its primary aspect.

The disregard of other modeling aspects cannot be justified.

The ability to cope with multiple aspects has become a definite prerequisite for many modern modeling languages.
Multiple Aspects in Modelica

- Certain modeling aspects are supported by keywords. For instance: `stateSelect`, `fixed`

- Modelica introduced the concept of annotations. These items are placed alongside the definition of models and the declaration of members.

- Example:

  ```
  Capacitor C1(C=c1) "Main Capacitor"
  annotation (extent=[50,-30; 70,-10],
              rotation=270);
  ```

- Since annotations tend to inflate the modeling code, they are mostly hidden by the editors
Situation in Modelica

- Overview on the current mixture of data-representation:
  - The **physics** of a model is naturally described by DAEs
  - **Hints** or information for the simulation-system are mostly also part of the main Modelica language but some of them have to be included in special annotations.
  - **Information that is used by the GUI** is included in annotations. But also information from textual descriptions is used.
  - The description of **3D-visualization** is done by dummy-models.
  - **Documentation** is extracted from the textual descriptions, but further documentation shall be provided by integrating HTML-code into a special annotation. Other annotations store information about the author and the library version.
Current Downfalls

- There is an evident lack of concept.
- Only pre-thought functionalities are applicable.
- The functionalities are mostly not customizable.
- The code-visibility is selected based on syntax not on semantics.
- The hiding of annotations hinders the editing.
Multiple aspects in Sol.

Sol is a language conceived for research purposes.

It aims to enable the future handling of variable-structure systems.

It owns a relatively simple grammar that is similar to Modelica.

Fundamentals have been reviewed in the language-design of Sol. New methods have been included in the language.

These methods aid also the modeling of multiple aspects.

The Sol project is supported by the Swiss National Science Foundation.
Multiple aspects in Sol.

- Starting from an example, I will present language constructs that meet the following requirements:

1. We shall have an open and transparent interface for each aspect. 
   \(\rightarrow\) Environment-packages

2. A convenient notation shall be provided. 
   \(\rightarrow\) Anonymous declarations

3. The modeler shall be enabled to form semantic entities. 
   \(\rightarrow\) Sections

4. The solution should well integrate into complex object-oriented model-structures. 
   \(\rightarrow\) Referencing mechanisms
Sol: The example

- The model consists of an engine that drives a flywheel. In the middle there is a simple gear box.

- The simulation yields to the plot on the right. It displays the angular velocity.

- The model contains a structural change: Reaching a threshold speed, causes the switch to a simpler engine model.
Sol: The example

model Machine

implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio<<1.8};
  dynamic Mechanics.Engine2 E {meanT<<10};

  connection c1(a << G.f2, b << F.f);
  connection c2(a << E.f, b << G.f1);

  when F.w > 40 then
      E <- Mechanics.Engine1{meanT << 10};
  end;

end Machine;
model Machine

implementation:

static Mechanics.FlyWheel F{inertia<<1};
static Mechanics.Gear G{ratio<<1.8};
dynamic Mechanics.Engine2 E {meanT<<10};

connection c1(a << G.f2, b << F.f);
connection c2(a << E.f, b << G.f1);

when F.w > 40 then
  E <- Mechanics.Engine1{meanT << 10};
end;

end Machine;
model Machine

implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio<<1.8};
  dynamic Mechanics.Engine2 E {meanT<<10};

  connection c1(a << G.f2, b << F.f);
  connection c2(a << E.f, b << G.f1);

when F.w > 40 then
  E <- Mechanics.Engine1{meanT << 10};
end;

end Machine;
model Machine

implementation:
  static Mechanics.FlyWheel F{inertia<<1};
  static Mechanics.Gear G{ratio<<1.8};
  dynamic Mechanics.Engine2 E {meanT<<10};

  connection c1(a << G.f2, b << F.f);
  connection c2(a << E.f, b << G.f1);

  when F.w > 40 then
    E <- Mechanics.Engine1{meanT << 10};
  end;

end Machine;
Sol: Environment-Packages

- Many modeling aspects refer to an external environment that is supposed to process the exposed information.
- The example presents a package of models that can be used to store information for the documentation of arbitrary models.
- The keyword environment does specify that the corresponding models address the environment and are therefore not self-contained.
- Environment-packages merely offer an interface.
- The concrete semantics is finally determined by the environment itself.
- Different environments may have different interpretations.
environment package Documentation

model Author
interface:
  parameter string name;
end Author;

model Version
interface:
  parameter string v;
end Version;

model ExternalDoc
interface:
  parameter string fname;
end ExternalDoc;

end Documentation
Sol: Environment-Packages

environment package Documentation

model Author
  interface:
    parameter string name;
  end Author;

model Version
  interface:
    parameter string v;
  end Version;

model ExternalDoc
  interface:
    parameter string fname;
  end ExternalDoc;

end Documentation
environment package Documentation

model Author
interface:
    parameter string name;
end Author;

model Version
interface:
    parameter string v;
end Version;

model ExternalDoc
interface:
    parameter string fname;
end ExternalDoc;

dend Documentation

„Dummy model“ that enables the specification of the author
Sol: Anonymous Declarations

- To take use of an environment package we have to declare instances of its models.

- In Sol, any model can be declared anonymously anywhere in the implementation.

- This way, we can conveniently create the documentation for our model.
model Machine
implementation:

[…] when F.w > 40 then
    E <- Mechanics.Engine1{meanT << 10 };
end;

Documentation.Author{name<<"DirkZimmer"};
Documentation.Version{v << "1.0"};
Documentation.ExternalDoc{fname<<"MachineDoc.html"};

end Machine;
Sol: Sections

- Sections can be defined using an arbitrary package name.
- Sections are a pure **grouping mechanism** and nothing more.
- Sections incorporate **three advantages**:
  1. Code can be structured into **semantic entities**.
  2. Sections add **convenience**, since the sub-models of the corresponding package can now be directly accessed.
  3. Sections enable an intuitive **control of visibility**.
model Machine

implementation:

[...]
when F.w > 40 then
    E <- Mechanics.Engine1{meanT << 10 };
end;

section Documentation:
    Author{name << "Dirk Zimmer"};
    Version{v << "1.0"};
    ExternalDoc{fname<<"MachineDoc.html"};
end;

section Simulator:
    IntegrationTime{t << 10.0};
    IntegrationMethod{method<<"euler",
        step << "fixed", value << 0.01};
end;

end Machine;
model Machine
implementation:
[...]
when F.w > 40 then
  E <- Mechanics.Engine1{meanT << 10 };
end;

section Documentation:
  Author{name << "Dirk Zimmer"};
  Version{v << "1.0"};
  ExternalDoc{fname<<"MachineDoc.html"};
end;

section Simulator:
  IntegrationTime{t << 10.0};
  IntegrationMethod{method<<"euler",
    step << "fixed", value << 0.01};
end;
end Machine;

The documentation is now grouped within a section.
The writing gets more convenient.

```plaintext
model Machine
implementation:
  [...] when F.w > 40 then
    E <- Mechanics.Engine1{meanT << 10};
  end;

section Documentation:
  Author{name << "Dirk Zimmer"};
  Version{v << "1.0"};
  ExternalDoc{fname<<"MachineDoc.html"};
end;

section Simulator:
  IntegrationTime{t << 10.0};
  IntegrationMethod{method<<"euler",
      step << "fixed", value << 0.01};
end;
end Machine;
```
model Machine
implementation:
  [...] when F.w > 40 then
    E <- Mechanics.Engine1{meanT << 10 };
  end;

section Documentation:
  Author{name << "Dirk Zimmer"};
  Version{v << "1.0"};
  ExternalDoc{fname<<"MachineDoc.html"};
end;

section Simulator:
  IntegrationTime{t << 10.0};
  IntegrationMethod{method<<"euler",
    step << "fixed", value << 0.01};
end;
end Machine;
model Machine
implementation:
   [...] when F.w > 40 then
       E <- Mechanics.Engine1{meanT << 10 };
   end;

section Documentation:
   Author{name << "Dirk Zimmer"};
   Version{v << "1.0"};
   ExternalDoc{fname<<"MachineDoc.html"};
end;

+ section Simulator:

end Machine;
Sol: Referencing

- This solution is feasible for simple applications.
- However, providing a GUI is more complex.
- The icons of a model-diagram relate to specific instances.
- Thus, we need to be able to refer on other model instances.
Sol: Referencing

• To refer on other model-instances Sol offers two solutions:

  1. **Member models:** These are models defined in the interface of a model and that are bounded to the corresponding instance of its top-model. Thus, they may address the top-model’s members.

  2. **First-class status** for any model instance: This means that instances of models can be treated as basic variables. Hence, they might be passed as parameters or they are dynamically transmitted.

• The demonstration example uses both techniques.
Demo
Summary

• Let us review the four language constructs:

1. **Environment-packages** that enable the aspect-specific declaration of interfaces.

2. **Anonymous declarations** of model instances.

3. **Sections** can be used to form semantic entities and control visibility.

4. **Referencing mechanisms** between model-instances. (In Sol, these mechanisms are provided by giving model-instances a first class status and enabling so-called member-models.)
Conclusions

- Environment packages provide a transparent interface.
- The interface is customizable
- Anonymous declarations enable a convenient usage
- User-defined sections help to organize the model.
- The text-filtering criteria are based on semantic entities.
- The embedment into an existing object-oriented framework enables a uniform approach for a wider range of modeling aspects.
Conclusions

Main conclusion:

- The ability of the language to help and to **extend itself** by its own means has been improved.

- Further development is now possible **within the language** and does not require a constant update and growth of the language definition.

- Important are not the precise grammar construct. Important is to **meet the four requirements** they have been built for. This way the proposed solution can be adopted for other languages.
Questions?