DEPS™ Overview

Version 0.9

January 22, 2018

Laurent ZIMMER Pierre-Alain YVARS

Purpose

- Build a modeling and solving tool to address design problems:
  - sizing,
  - configuration,
  - allocation,
  - architectural design.

- Need of a modeling language supporting problem solving description:
  - complex system descriptions : Behavior, Functions, Structure,
  - local and global requirements,
  - partial descriptions : unknowns, degrees of freedom, alternatives, variability …

- Need of solving capabilities addressing :
  - under constrained problems,
  - mixed non linear mathematical problems,
  - both equations and inequalities.

➢ *Solving a design problem is completing a subdefined system !*

Background

- DEKLARE (Esprit project), KoMoD
  - P.A. Yvars stakeholder
  - A graphical design problem description language
  - A C++ constraint solving library

- CO2 (RNTL project)
  - L. Zimmer coordinator
  - The Constraint Explorer software tool
    - An integrated modelling and solving environment
    - A design problem description language
    - An interval constraint solver

DEPS Project

- DEPS = **DE**sign **P**roblem **S**pecification

  DEPS is a (Domain Specific) language and not a tool

- DEPS Studio is an Integrated Modelling and Solving Environment
  - Project manager, Editors, DEPS compiler, DEPS solver

- Application Fields:
  - Embedded Systems, Engineering Systems, Cyberphysical Systems…

- Design steps:
  - preliminary design, architecture generation, system integration…

Main Pillars

- DEPS is a declarative language
  - acausal modeling combining both equations and inequalities
- DEPS is an object-oriented language
  - a model is an object
- An Ontology for Engineers
  - constants and variables are physical (cardinal) or ordinal quantities
  - dimension, units, variation domain …
- A dual-purpose calculus
  - integer or real values
  - continuous or discrete quantities

DEPS

Object-Oriented Language + Constraint Solving Language = DEPS Language

Subdefined Modelling + Constraints = Properties Modelling

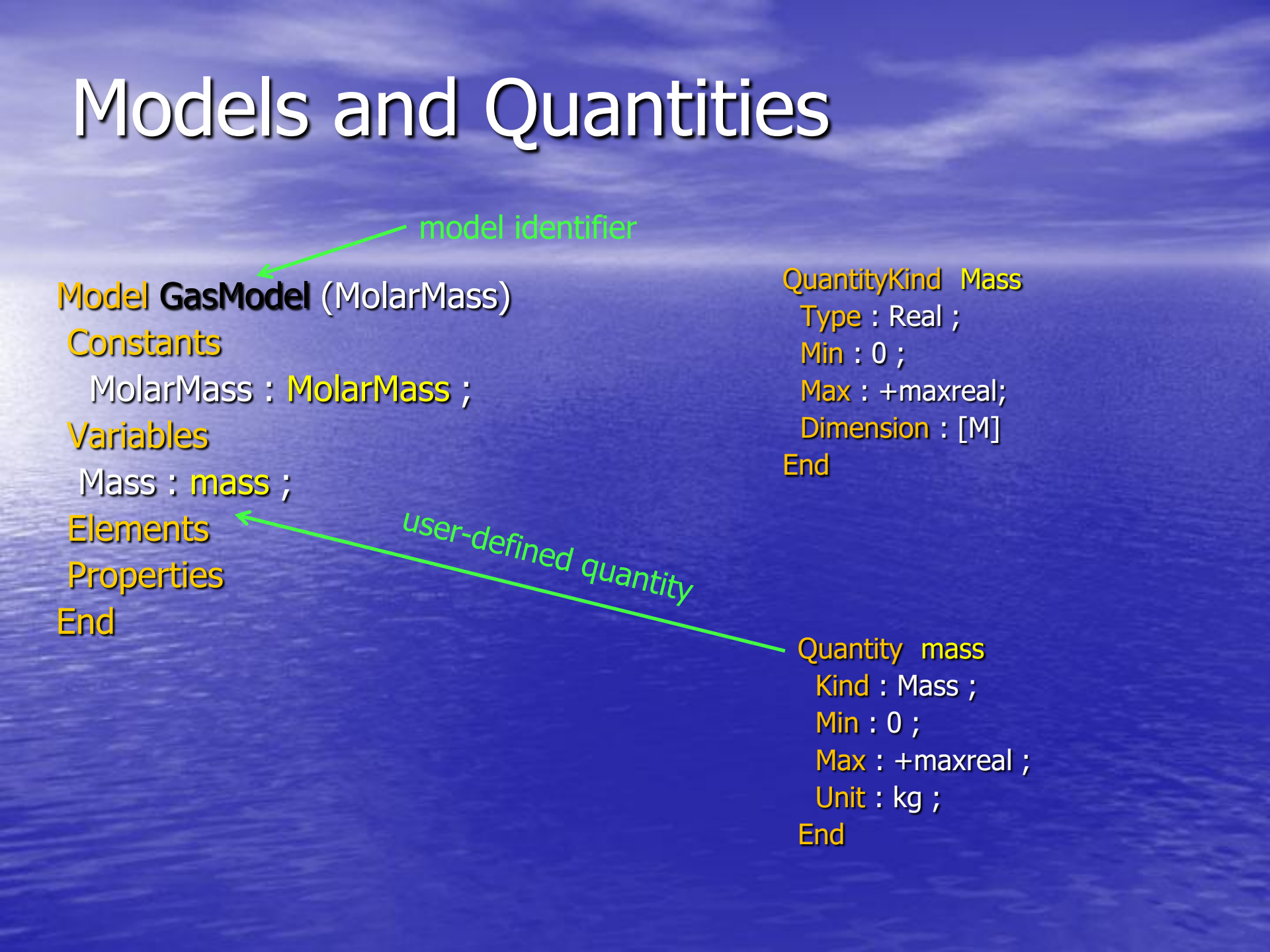
```
Model Partition ()
Constants
Variables
  icpu : CpuIndex ;
Elements
Properties
End
```

```
Quantity
CpuIndex
 Kind : Integer ;
 Min  : 1 ;
 Max : 4 ;
 Unit : u ;
End
```

```
P1.icpu = P2.icpu;
```

```
Model colocalisation (P1, P2)
Constants
Variables
Elements
 P1 : Partition ;
 P2 : Partition ;
Properties
  P1.icpu = P2.icpu;
End
```

Models and Quantities



Part and shared models

reference binding

```
Model Tank(p, t, Gas)
 Constants
  R : Real = 8.314 ;
  p : Pressure ; instance declaration
  t :  Temperature ;
 Variables
  V : Volume ;
 Elements
 Gas : GasModel ;
 Properties
  p*V= (Gas.Mass/Gas.MolarMass)*R*t;
End
```

Instance construction
O2 part of Problem

reference binding
O2 shared by T1 and T2

```
Model Problem()
 Constants
 Variables
 Elements
  O2:GasModel(0.032);
  H2:GasModel(0.002);
  T1,T2:Tank(2.56e+7, 300, O2);
  T3:Tank(7.00e+7, 300, H2);
 Properties
  O2.Mass = 10 ;  (* or T1.Gas *)
  T1.V+T2.V< 500 ;
End
```

Aliases

```
Model Gas (ckilo, molarMass)
 Constants
  ckilo : DollarCostPerKilo;
  molarMass : MolarMass;
 Variables
  M : mass ;
  expr CoutStockage : DollarCost  ;  (* Alias declaration *)
 Elements
 Properties
  CoutStockage := ckilo*M ;  (* Alias definition *)
End
```

Aliases

```
Model Port
 Constants
 Variables
  V : Voltage ;
  expr I : Intensity ; (* Alias declaration *)
 Elements
 Properties
End
```

```
Model Dipole (index)
 Constants
  index : DipoleIndex ;
 Variables
  I : Intensity ;
 Elements
   P1, P2 : Port ;
 Properties
   P1.I :=  I;  (* Alias definition *)
   P2.I := -I;
End
```

Universal constants

```
Model circle ()
 Constants
 Variables
  Diameter, Circumference: length ;
 Elements
 Properties
  Circumference = uPi*D ;
End
```

Derived models (inheritance)

```
Model Resistor(R) extends Dipole[DipoleIndex]
 Constants
  R : Resistor;
 Variables
 Elements
 Properties
  P1.V-P2.V = R*I;
End
```

Signature

1 resistor model:

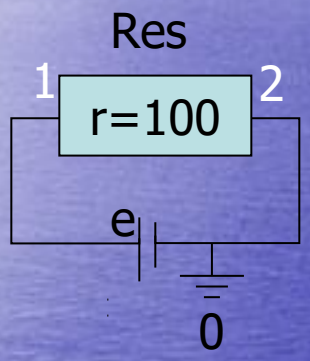
- 3 variables (unknowns)
- 1 equation
- 2 expressions

Subdefined Model

```
(* subdefined Resistor *)


Model SdResistor() extends Dipole [DipoleIndex]
 Constants
 Variables
  R : Resistor;
 Elements
 Properties
  P1.V-P2.V = R*I;
End
```

A very simple circuit



1 problem :
- 3 variables (unknowns)
- 1+2 = 3 equations

```
Problem One Resistor
 Constants
  e : Voltage = 10 ;
 Variables
 Elements
  Res : Resistor(1,100) ;
 Properties
  Res.P1.V  = e ;
  Res.P2.V  = 0;
End
```

Order of bindings

Direct problem : Res.R = 100 → Res.I = 0.1

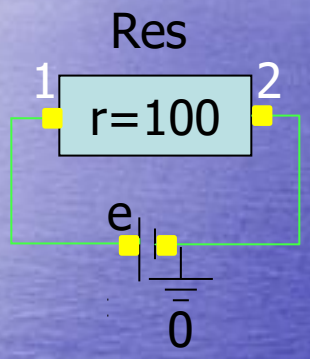
Other models

```
Model Node (P1,P2)
 Constants
 Variables
 Elements
  P1, P2 : Port ;
 Properties
  P1.V = P2.V ;
  P1.I+P2.I = 0 ;
End
```

```
Model Ground (P)
 Constants
 Variables
 Elements
  P : Port ;
 Properties
  P.V = 0 ;
End
```

```
Model VSource (e)
 Constants
    e : Voltage;
 Variables
    I1: Intensity;
    I2: Intensity;
 Elements
    P1, P2: Port ;
 Properties
    P1.I := I1;
    P2.I := I2;
    P2.V–P1.V =  e ;
End
```

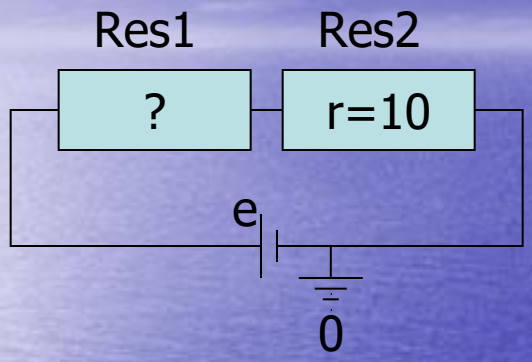
A very simple circuit (improved)



1 problem :
- 4+0+3+2*0 = 7 variables (unknowns)
- 1+1+1+2*2 = 7 equations

```
Problem One Resistor
 Constants
 Variables
 Elements
  VS: VSource(10) ;
  Gr: Ground(VS.P1) ;
  Res: Resistor(1,100) ;
  N1: Node(VS.P2, Res.P1);
  N2: Node(VS.P1, Res.P2);
 Properties
End
```

More complicated



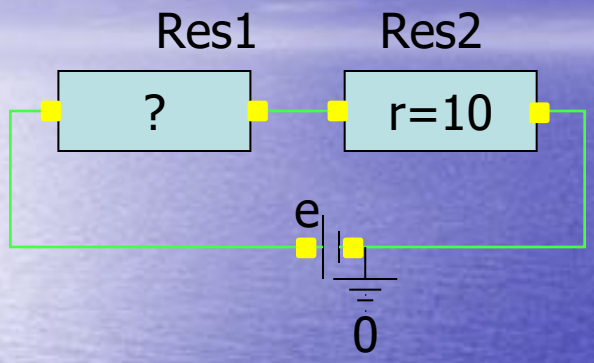
1 problem:
- 4+3 = 7 variables (unknowns)
- 2+5 = 7 equations

subdefined model:
Res1.R = ?

```
Problem SerialResistors
 Constants
  e: Voltage = 10;
 Variables
 Elements
  Res1: SdResistor(1) ;
  Res2: Resistor(2,10) ;
 Properties
  Res1.P1.V = e ;
  Res1.P2.V = Res2.P1.V ;
  Res1.P2.I + Res2.P1.I = 0 ;
  Res2.P2.V = 0 ;
  Res1.P1.I = 0.1;  (* requirement *)
End
```

Inverse problem : Res1.P1.I = 0.1 → Res1.R = 90

Improvement



1 problem :

- 4+0+4+3+3*0 = 11 variables (unknowns)
- 1+1+1+1+3*2+1 = 11 equations

```
Problem SerialResistors
 Constants
 Variables
 Elements
  VS: VSource(10) ;
  Gr: Ground(VS.P1) ;
  Res1: SdResistor(1) ;
  Res2: Resistor(2, 100) ;
  N1: Node(VS.P2, Res1.P1) ;
  N2: Node(Res1.P2, Res2.P1) ;
  N3: Node(Res2.P2, VS.P1) ;
 Properties
  Res1.P1.I = 0.1 (* requirement *)
End
```

Packages : declaration and use

```
Package A ;
Uses B , C  ;

Problem Pb
 Constants
 Variables
 a1 : Angle;
 a2 : Angle;
 a3 : Angle;
 Elements
 M1 : MC();
 Properties
End
```

```
Package B ;

Uses   ;

Quantity Angle
  Kind : Angle;
  Min : -uPi;
  Max : uPi;
  Unit : Rad;
End

Quantity Force
  Kind : Force;
  Min : 0;
  Max : 100;
  Unit : N;
End
```

```
Package C ;
Uses   B ;

Model MC ()
 Constants
 c1 : Force = 10;
 c2 : Force = 20;
 Variables
 a1 : Angle;
Elements
Properties
End
```