The High-Level Variability Language: an ontological approach

Angela Villota, Raúl Mazo and Camille Salinesi

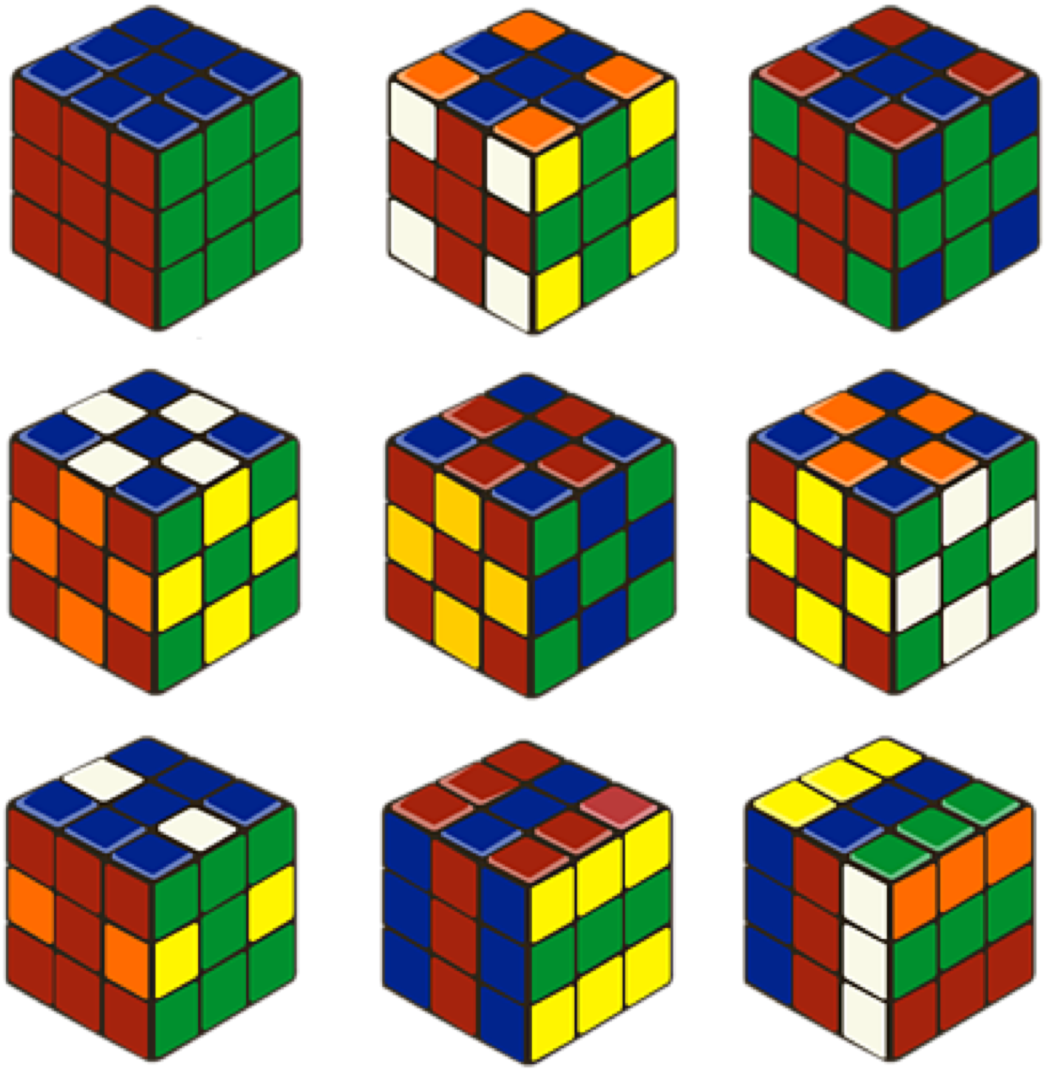
Context

Differences

- Style: orthogonal
- Targets
- Structure
- Way to be used

Similarities

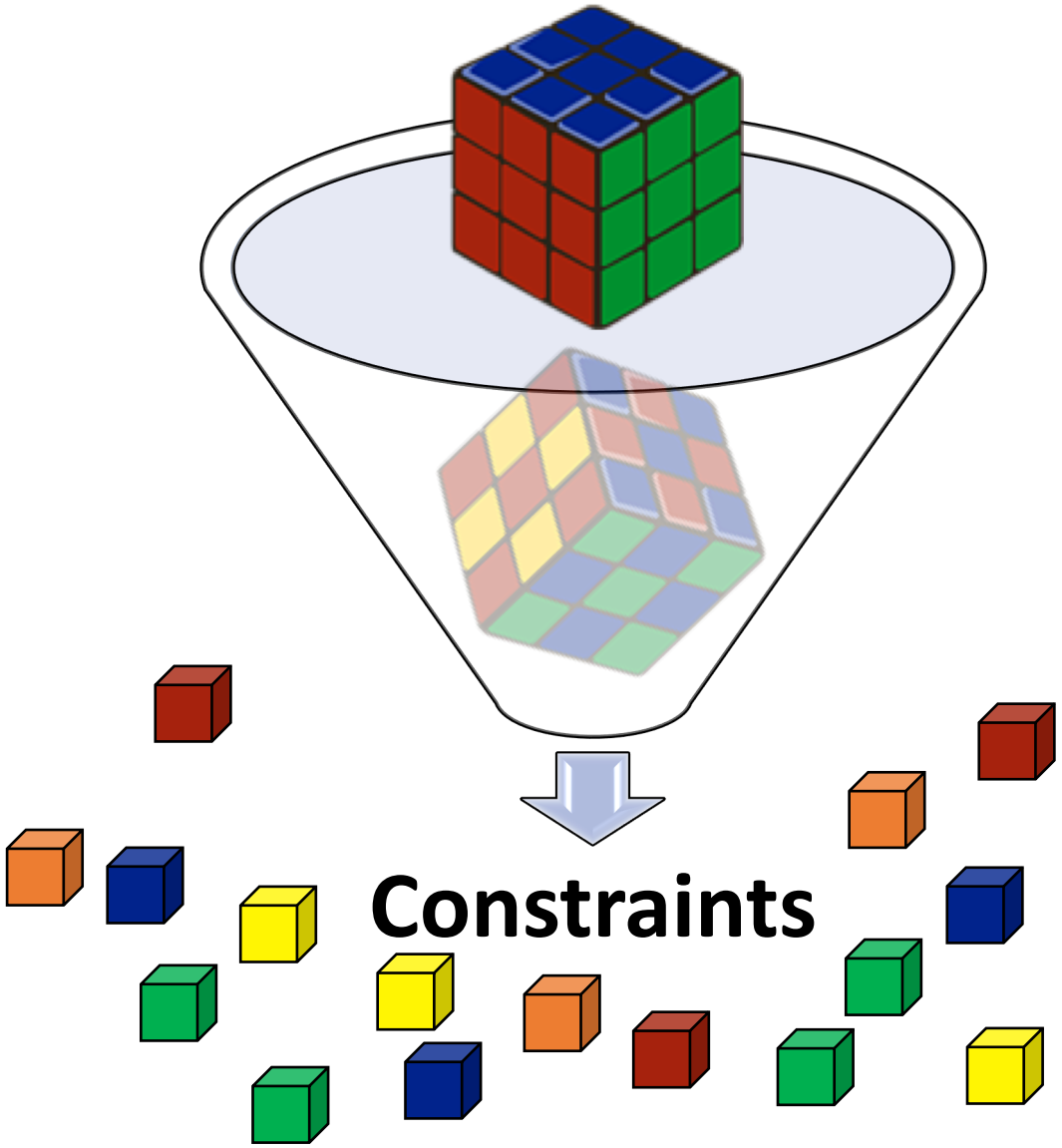
- Syntax
- Semantics



Context

Translational Semantics

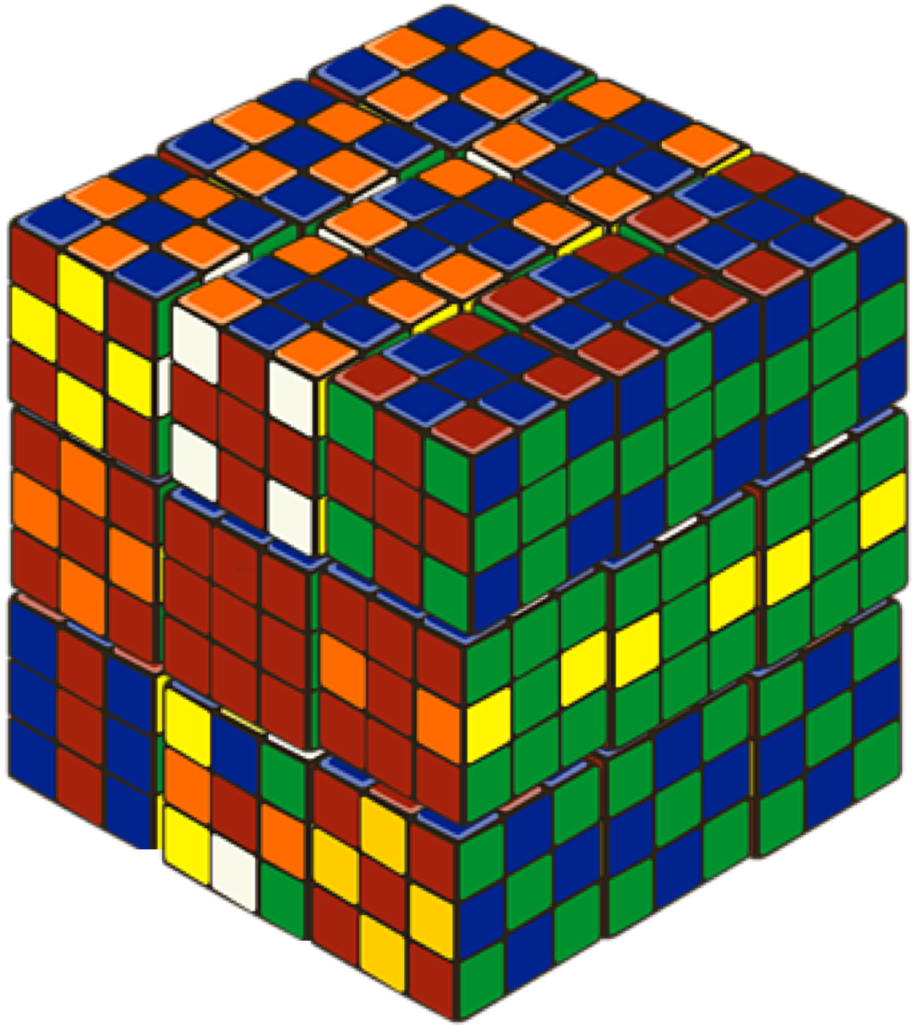
Map models in propositional formulas or constraint satisfaction problems to perform reasoning tasks.



Why don’t we use a constraint-based language to model variability?

High-Level Constraint Language

- Djebbi et al. (2009)
- Mazo et al. (2011)
- Dumitrescu et al. (2014)
- Muñoz et al. (2015)
- Villota et al. (2018)

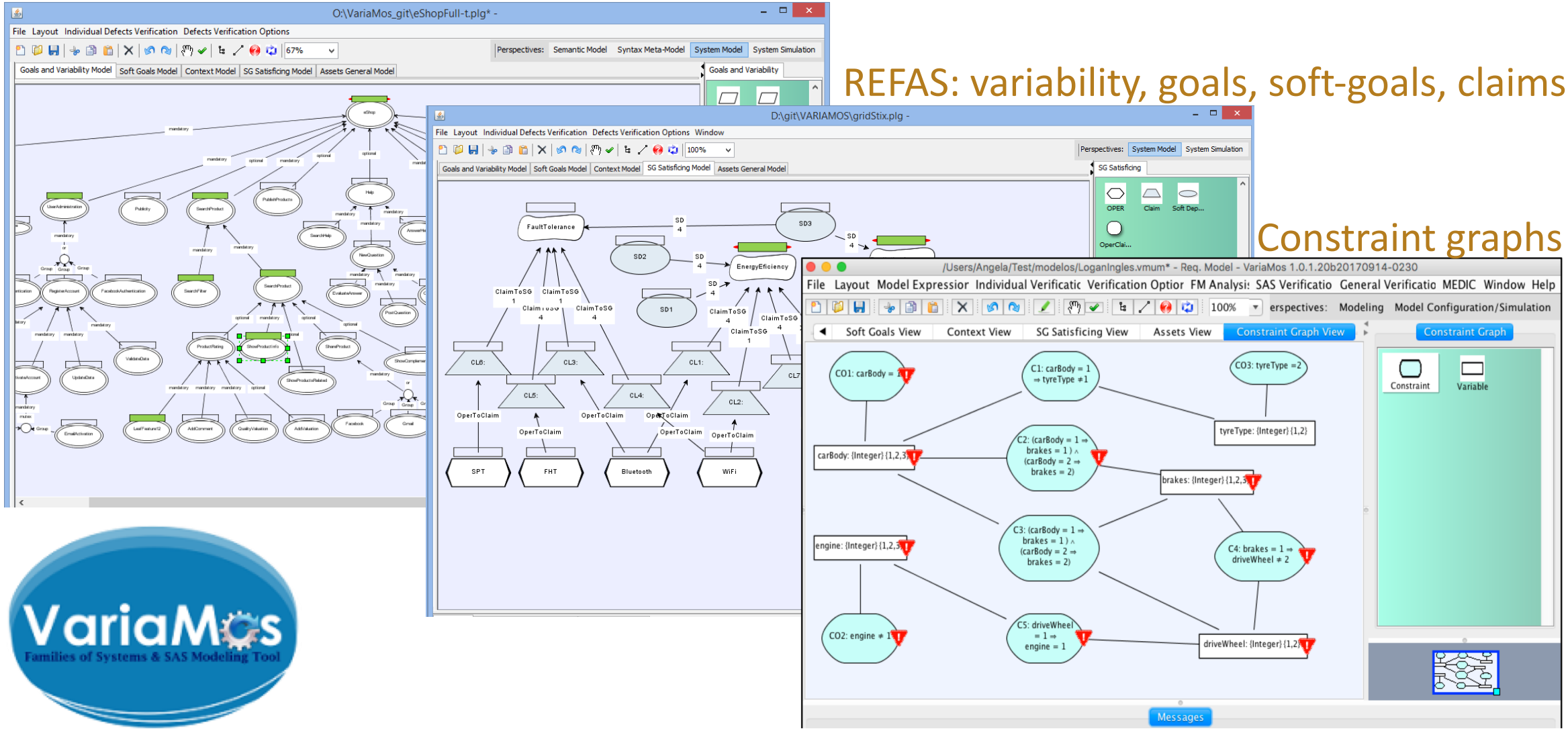


Context

Feature models

REFAS: variability, goals, soft-goals, claims

Constraint graphs

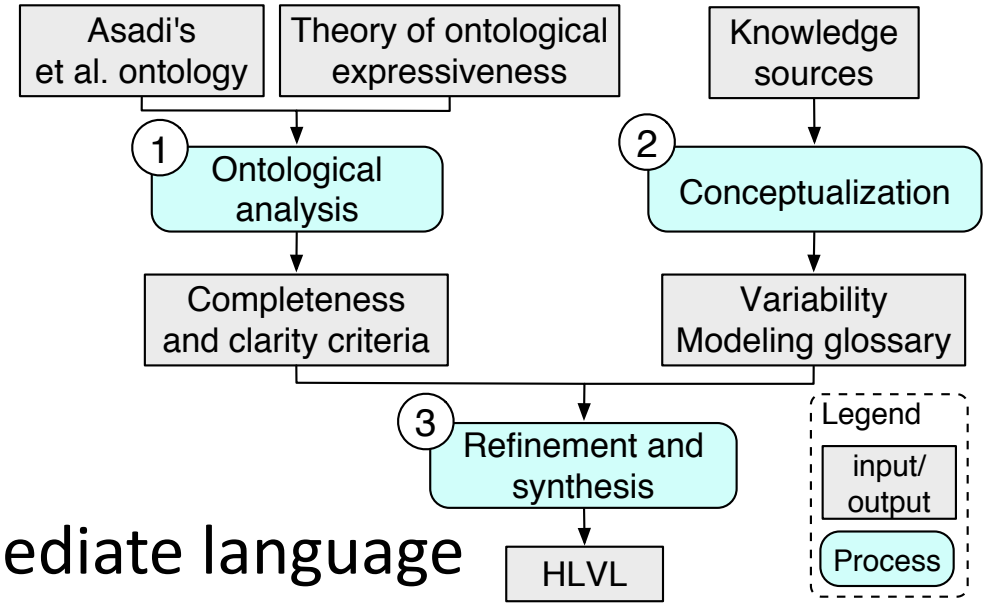


VariaMos
Families of Systems & SAS Modeling Tool

The Ontological Path

Can we model variability comprehensively using HLCL?

- Comparison with other PLMs languages. ☑
- User experience. ☑
- A more formal approach? ☐



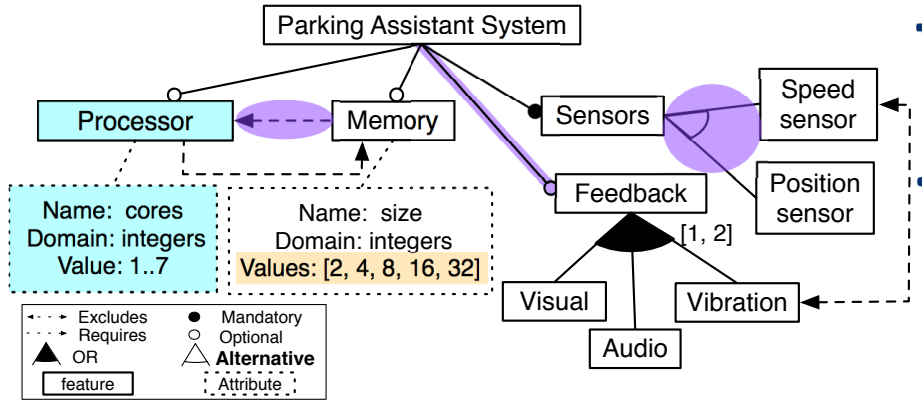
Ontological approach

- To evaluate the expressiveness of the intermediate language
- To conceptualize and structure knowledge about variability modeling
- To define variability constructs and language characteristics

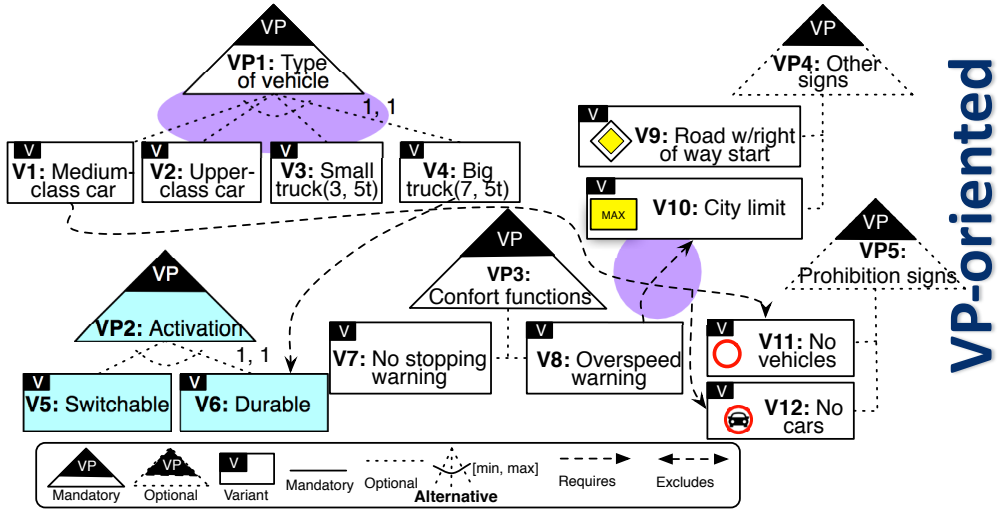
Variability modeling languages

not completely different, not completely the same

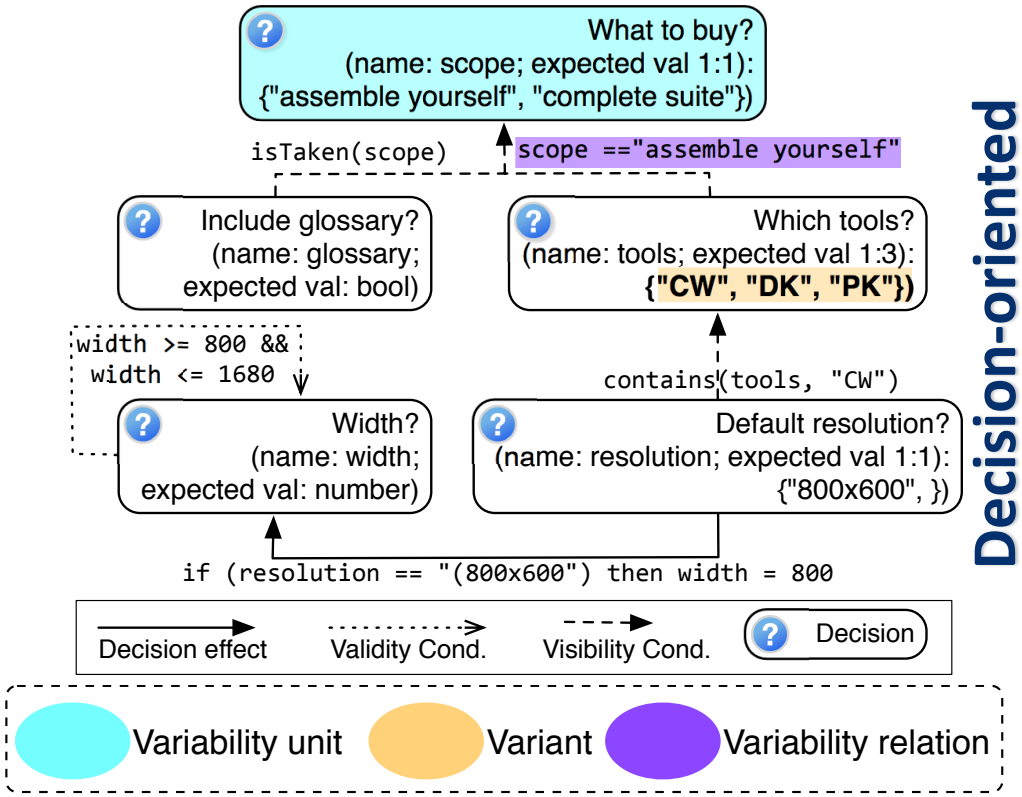
Feature-oriented



VP-oriented



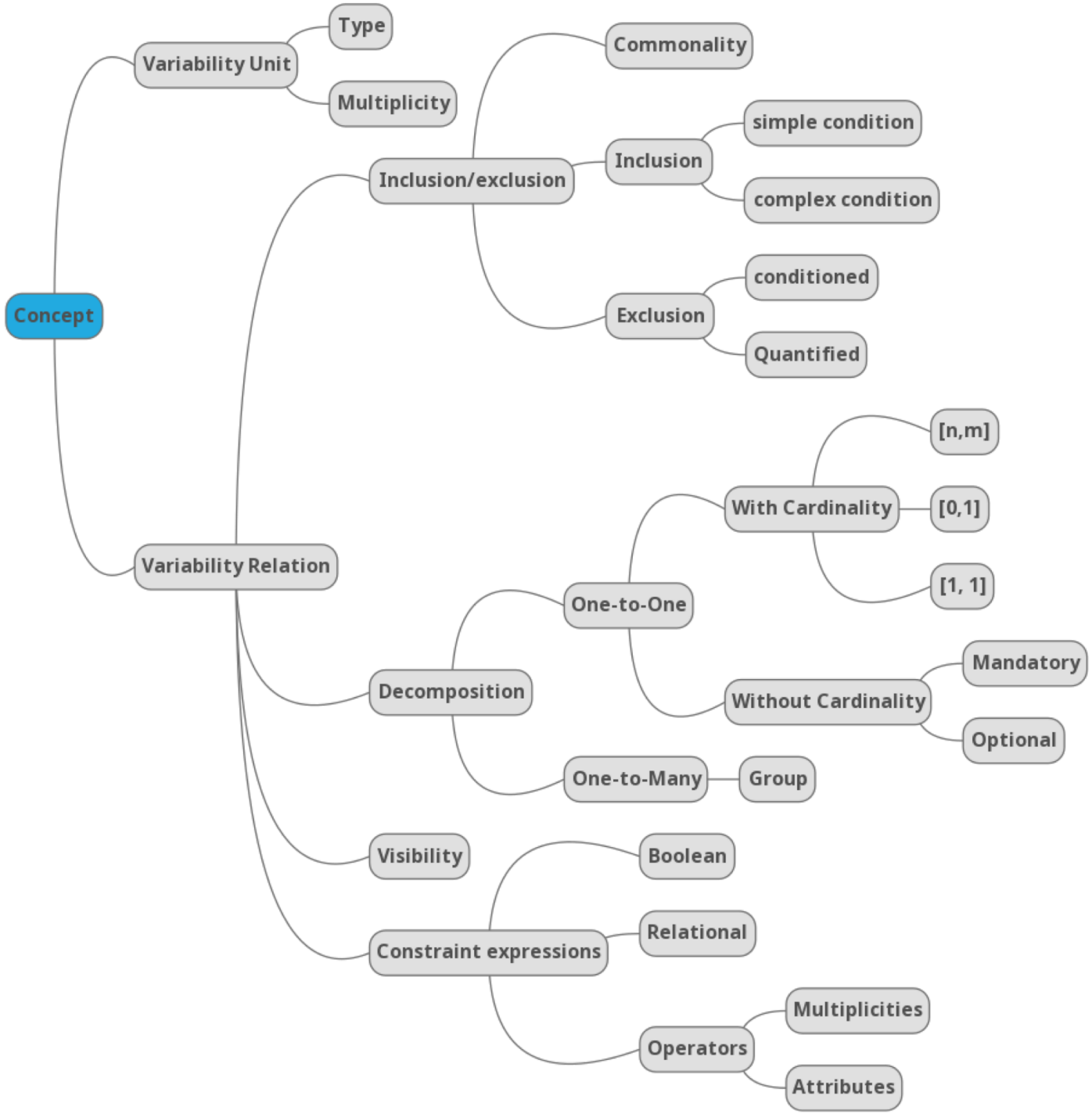
Decision-oriented



Constraint-oriented

```
var 0..1: PAS;
var 0..1: memory;
var {0, 2, 4, 8, 16, 32}: size;
var 0..7: cores;
var 0..1: processor;
var 0..1: sensors;
var 0..1: feedback;
var 0..1: positionSensor;
var 0..1: speedSensor;
var 0..1: visual;
var 0..1: audio;
var 0..1: vibration;
constraint PAS == 1;
constraint PAS == sensors;
constraint PAS >= processor;
constraint PAS >= memory;
constraint PAS >= feedback;
constraint feedback <= audio + vibration + visual;
constraint audio + vibration + visual <= 2 * feedback;
constraint processor == cores;
constraint memory == size;
constraint sensors >= speedSensor;
constraint sensors >= positionSensor;
constraint processor <= memory;
constraint vibration + speedSensor <= 1;
```

Variability concepts



The High-Level Variability Language

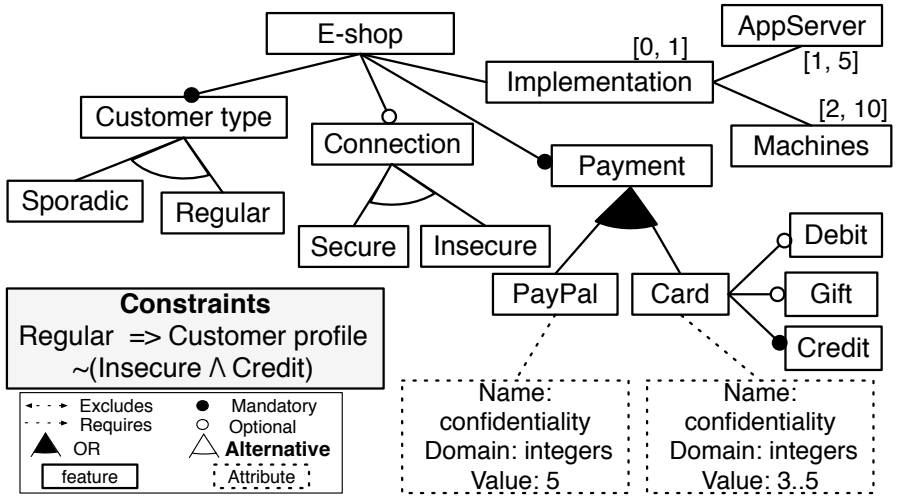
- Rich textual language to model variability.
- Syntax that resembles programming languages, to make it human readable
- HLVL is a language capable of supporting concepts of many variability languages. Why just features?

Would you use it?

- Would you leave your modeling tools for an HLVL-tool?

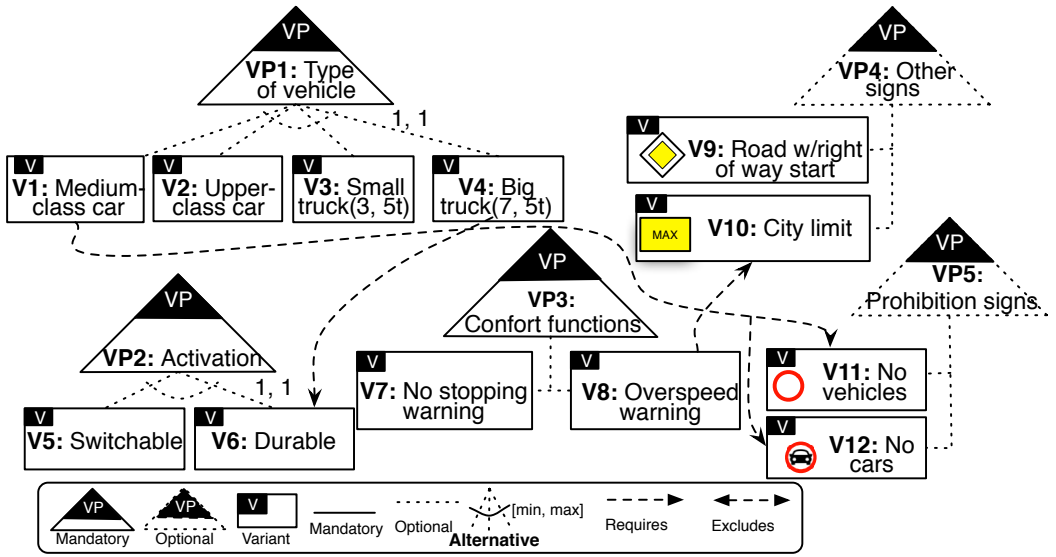
The High-Level Variability Language

```
model eShop
elements:
  boolean connectionType
  boolean secureConnection
  boolean insecureConnection
  boolean payment
  boolean payPal
  symbolic customerType variants: [ 'sporadic' , 'regular' ]
  comment: { "This element represents the customer type" }
  att integer confidentiality variants: 1 .. 5
  att integer confBounded is 2
  boolean customerProfile
  boolean SMS
  boolean card
  boolean creditCard
  boolean giftCard boolean debitCard
  boolean implementation
  boolean appServer
  boolean machines
  att integer certificateType variants: 1 .. 5
  symbolic productType variants: [ 'services' , 'products' ]
  relations:
  com1 : common ( customerType , payment )
  exp1 : expression ( 3 <= card . confidentiality AND card . confidentiality <= 5 )
  m1 : mutex (creditCard , insecureConnection)
  m2 : mutex ((customerType = 'sporadic') ,[giftCard , creditCard])
  imp1 : implies (payPal , secureConnection)
  imp2 : implies ((customerType = 'regular'), [secureConnection, customerProfile])
  dc1 : decomposition (card, [giftCard ,debitCard] ,[0 , 1])
  dc2 : decomposition (card, [creditCard] ,[1 , 1])
  dc3 : decomposition (implementation ,[appServer] ,[1 , 5])
  dc4 : decomposition (implementation ,[machines] ,[2 , 10])
  att1 : decomposition (payPal , [confidentiality ,certificateType] ,[ 1 , 1])
  att3 : decomposition (card , [confidentiality ,certificateType] ,[ 1 , 1 ])
  g1 : group (payment ,[payPal ,card] ,[1 ,*])
  v1 : visibility (productType = 'services' ,[implementation ,appServer ,machines])
```



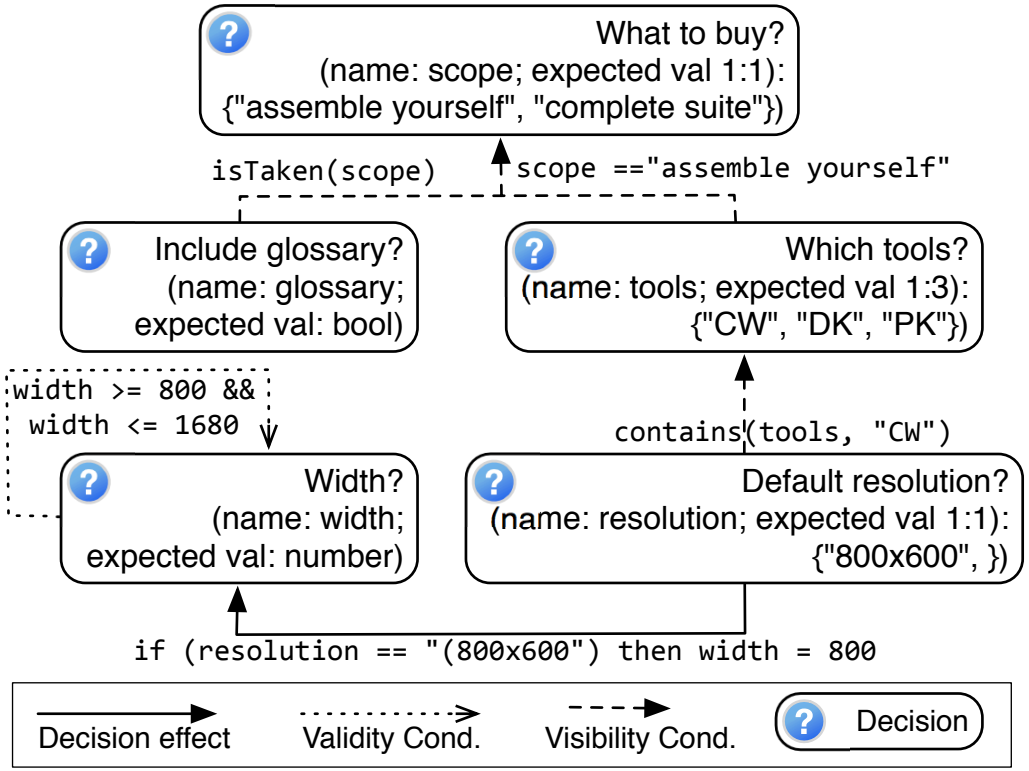
The High-Level Variability Language

```
model RFW
elements:
    symbolic VP1
    variants: ['medium-class car', 'upper-class car',
               'small truck', 'big truck']
    comment:{"Type of vehicle"}
    symbolic VP2
    variants: ['switchable', 'durable']
    comment:{"Activation"}
    boolean  VP3 comment:{"Comfort functions"}
    boolean  V7 comment:{"Non stopping warning"}
    boolean  V8 comment:{"Overspeed warning"}
    boolean  VP4 comment:{"Other signs"}
    boolean  V41 comment:{"Road w/right of way start"}
    boolean  V42 comment:{"city limit"}
    boolean  VP5 comment:{"Prohibition signs"}
    boolean  V51 comment:{"No vehicles"}
    boolean  V52 comment:{"No cars"}
relations:
    r1: common(VP1, VP2, VP3)
    d1: decomposition(VP5, [V51, V52], [0,1])
    d2: decomposition(VP4, [V41, V42], [0,1])
    d3: decomposition(VP3, [V7, V8], [0,1])
    exp1: expression(VP1 ='big truck' => VP2 ='durable')
    imp1: implies((VP1 ='medium-class car'), [V51,V52])
    imp2: implies(V8, V42)
```



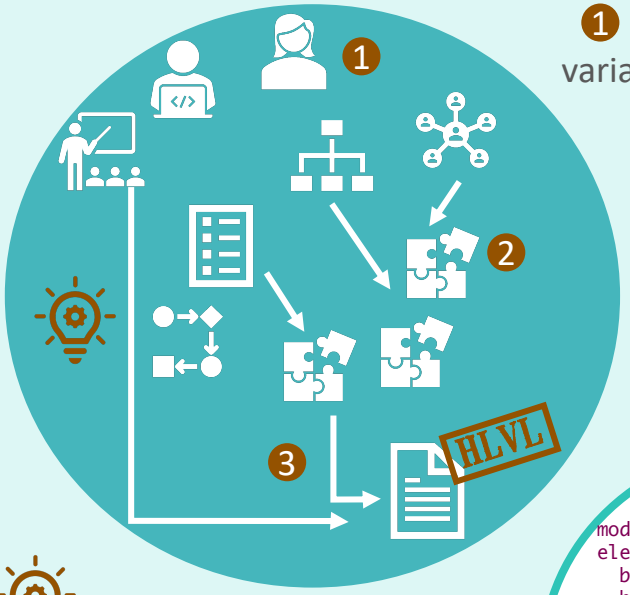
The High-Level Variability Language

```
model Dopler
elements:
    symbolic scope variants: ['assemble yourself', 'complete suite']
                    comment: {"What to buy?"}
    boolean glossary comment:{"Include glossary?"}
    symbolic resolution variants:['800x600', '']
                        comment:{"Default resolution?"}
    integer width comment:{"Width?"}
    boolean tools
    boolean configurationWizard
    boolean decisionKing
    boolean projectKing
relations:
    c1: common(tools)
    g1: group(tools, [configurationWizard, decisionKing, projectKing], [1,3])
    val1: expression(width >= 800 AND width <= 1680)
    e1: expression(resolution='800x600' => width = 800)
    vis1: visibility(configurationWizard=true, [resolution])
    vis2: visibility(entailed(scope), [glossary])
    vis3: visibility(scope='assemble yourself', [glossary])
```



What's next?

Coffee



1 Domain engineers model variability using their preferred language/tool

2 Then, models are transformed to produce their **HLVL** representation

3 Variability models represented in **HLVL** are ready for the reasoning process

HLVL

```
model mobilePhone
elements:
  boolean calls
  boolean GPS
  boolean screen
  boolean basicScreen
  boolean hrScreen
  boolean camera
relations:
 R0: coreElements(calls,
screen)
 R1: mutex(GPS, basicScreen)
 R2: implies(camera, hrScreen)
```

The transformation process is shorter using our **HLVL** variability modelling editor.

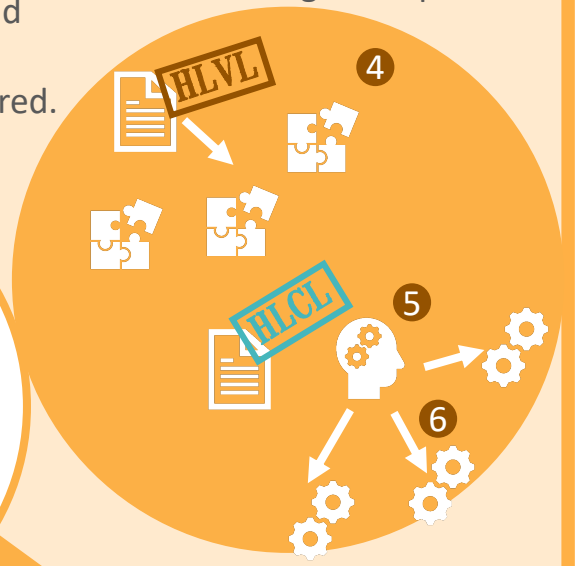
4 **HLVL** models are transformed into constraint programs in **HLCL**

5 Models and reasoning questions are processed to choose the best solving tool and algorithm for answering each question

6 The solving tool is executed, the output is processed, and the questions about the variability model are answered.

HLCL

```
% Autogenerated code from
% the Coffee framework
% Variables from elements defs.
var 0..1: calls;
var 0..1: GPS;
var 0..1: screen;
var 0..1: basicScreen;
var 0..1: hrScreen;
var 0..1: camera;
% Variables and constraints
constraint calls = 1;
constraint screen = 1;
constraint GPS + basicScreen <1;
constraint hrScreen < camera;
% Solving parameters
   solve satisfy;
```



Thank you!

Angela Villota

Contact: angievig@gmail.com