(Domain-Specific) Modelling Language Engineering

Hans Vangheluwe

http://msdl.cs.mcgill.ca/
at the most appropriate level(s) of abstraction using the most appropriate formalism(s) explicitly modelling workflows

Enabler: (domain-specific) modelling language engineering including model transformation
Henk Vanhooren, Jurgen Meirlaen, Youri Amerlinck, Filip Claeys, Hans Vangheluwe and Peter A. Vanrolleghem.
Why DS(V)M?
(as opposed to General Purpose modelling)

- **match the user’s mental model** of the problem domain
- **maximally constrain** the user (to the problem at hand)
  ⇒ easier to learn
  ⇒ avoid errors
- **separate** domain-expert’s work from analysis/transformation expert’s work

**Anecdotal evidence of 5 to 10 times speedup**


DS(V)M Example in Software Domain
smart phones, the application

MetaEdit+ (www.metacase.com)
DS(V)M Example: smart phones, the Domain-Specific model
Model-Based Development:
Modify the Model
(e.g., based on feature model of product family)

model → transformation → app

small modification

model' → transformation → app'
Model-Based Development:
Modify the Model
(e.g., based on feature model of product family)

small modification

model → transformation → app

model' → transformation → app'

small modification in model may lead to large change in app
≈ choice of formalism (e.g., Statecharts)
Statecharts
Model-Based Development:
Modify the Model
(e.g., based on feature model of product family)

small modification

small modification in model may lead to large change in app

~ choice of formalism (e.g., Statecharts)
“fitness”

1. Daylight
2. Low Visual Distraction
3. Views to Outside
4. Adjacency Preference
5. Circulation
6. Work Styles
7. Low Acoustic Distraction
8. Low Density
A variable number of neighborhoods are seeded along spine, and given a parameterized range of motion.

One edge from each neighborhood is selected to generate zone for amenity clusters.

Automated “test fit” generates amenity rooms from space matrix and desk layout.

Teams are assigned by best-fit algorithm. Neighborhood amenities are assigned by team preferences.

Figure 1. Description of specification of geometric model
generated (evolutionary – e.g., Genetic Algorithms)
“fitness” evaluation for each of the generated designs

Figure 2. Design metrics (from left to right: adjacency preference, work style preference, buzz, productivity, daylight, and views to outside)
Model-Based Development:
Modify the Transformation
(e.g., target platform changes, or optimization)

- model
- transformation
- small modification
- transformation’
- app’

model
transformation
app
Can be Multi-Step/Multi-Formalism
Building DS(V)M Tools Effectively . . .

- development cost of DS(V)M Tools may be prohibitive!
- ⇒ need Modelling Language Engineering
Model Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapping feature</td>
<td>A model is based on an original.</td>
</tr>
<tr>
<td>reduction feature</td>
<td>A model only reflects a (relevant) selection of an original’s properties.</td>
</tr>
<tr>
<td>pragmatic feature</td>
<td>A model needs to be usable in place of an original with respect to some purpose.</td>
</tr>
</tbody>
</table>
Everything is a model!
Jean Bézivin

Everything is a model!

Jean-Marie Favre

Nothing is a model!

model

system

model

system

sus
Jean Bézivin

Everything is a model!

Jean-Marie Favre

Nothing is a model!

Hans Vangheluwe

Model everything!

model

system

model

system

sus

MPM
The image contains a graph showing the relationship between force and displacement, with the formula $F = -kx$ indicated. The graph is labeled "Model Validity Range." A table listing various spring parameters is also present, including columns for O.D., CENTURY STOCK NUMBER, FREE LENGTH, I.D., RATE, SUGG. MAX. DEFL., SUGG. MAX. LOAD, SOLID LENGTH, WIRE DIA., TOTAL COILS, MAT'L, and E. The website www.centuryspring.com is displayed at the bottom right corner.
Experimental (Validity) Frame

Abstraction Relationship

*foundation*: the *information* contained in a model $M$. Different *questions* (properties) $P = I(M)$ which can be asked concerning the model. These questions either result in true or false.

*Abstraction* and its opposite, *refinement* are *relative to a non-empty set of questions* (properties) $P$.

- If $M_1$ is an *abstraction* of $M_2$ with respect to $P$, for all $p \in P$: $M_1 \models p \Rightarrow M_2 \models p$. This is written $M_1 \sqsupseteq_P M_2$.
- $M_1$ is said to be a *refinement* of $M_2$ iff $M_2$ is an *abstraction* of $M_1$. This is written $M_1 \sqsubseteq_P M_2$. 
Token Models
Rôles a Model may Play
\( \mathcal{S} \triangleleft \mathcal{M} \)

\[ \rho(\mathcal{S}, \mathcal{M}) \rightarrow \mathcal{S} \triangleleft \mathcal{M} \]

\[ \alpha = \tau \circ \alpha' \circ \pi \]

\[ h(a \oplus b) = h(a) \otimes h(b) \]
\[ \pi(e_1) = \pi(e_2) \implies e_1 = e_2 \]

\[ e_1 \sim_\pi e_2 \implies \pi(e_1) = \pi(e_2) \]

\[ \varepsilon(C) = \{ x \mid P(x) \}, \text{ where } P = \iota(C) \]

\[ \forall x : \iota(C_{\text{special}})(x) \implies \iota(C_{\text{general}_1})(x) \]

\[ \varepsilon(C_{\text{special}}) \subseteq \varepsilon(C_{\text{general}}) \]
\[ e_1 R^n e_2 = \begin{cases} 
\exists e : e_1 R^{n-1} e \land e R e_2, & n > 1 \\
 e_1 R e_2, & n = 1 
\end{cases} \]

**acyclic** \quad \forall e_1, e_2, n : e_1 R^n e_2 \rightarrow \neg e_2 R e_1, and

**anti-transitive** \quad \forall n \geq 2 : R^n \cap R = \emptyset.

**level respecting** \quad \forall n, m : 
\[ (\exists e_1, e_2 : e_1 R^n e_2 \land e_1 R^m e_2) \rightarrow n = m \]
<table>
<thead>
<tr>
<th>notation</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>abstraction</td>
<td>creates a model from a system using projection ($\pi$) and possibly classification ($\Lambda$) and generalization ($\Gamma$), hence $S \triangleleft \alpha(S)$.</td>
</tr>
<tr>
<td>$\Lambda$</td>
<td>classification</td>
<td>creates a type model, hence $\mathcal{M} \triangleleft_t \Lambda(\mathcal{M})$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>generalization</td>
<td>creates a supermodel, hence $S \triangleleft_t \mathcal{M} \rightarrow S \triangleleft_t \Gamma(\mathcal{M})$</td>
</tr>
<tr>
<td>$\pi$</td>
<td>projection</td>
<td>a homomorphic mapping creating a reduced system from a given system, using selection and reduction of information.</td>
</tr>
<tr>
<td>$\rho$</td>
<td>represents</td>
<td>records the intention of a model to represent a system.</td>
</tr>
<tr>
<td>$\mu$</td>
<td>meaning</td>
<td>assigns meaning to a model (element). If $\rho(S, \mathcal{M})$ then one may define $\mu(\mathcal{M}) = \pi(S)$.</td>
</tr>
<tr>
<td>$\triangleleft$</td>
<td>model-of</td>
<td>holds between a system and a model describing the former.</td>
</tr>
<tr>
<td>$\triangleleft_i$</td>
<td>token model-of</td>
<td>holds between a system and a model representing the former in a one-to-one fashion. Model elements may be regarded as designators for system elements.</td>
</tr>
<tr>
<td>$\triangleleft_t$</td>
<td>type model-of</td>
<td>holds between a system and a model classifying the former in a many-to-one fashion. Model elements are regarded as classifiers for system elements.</td>
</tr>
<tr>
<td>$\triangleleft^o$</td>
<td>ontological model-of</td>
<td>indicates that the model controls the <em>content</em> of its elements, hence $S \triangleleft^o \mathcal{M} \iff \mu(S) \in \varepsilon(\mu(\mathcal{M}))$ and $S \triangleleft_i \mathcal{M} \iff \mu(\mathcal{M}) = \pi(\mu(S))$. Assuming $\mu(S) = S$, for systems which do not model anything, we have $S \triangleleft^o \mathcal{M} \iff \mu(\mathcal{M}) = \pi(S)$ and thus $\rho(S, \mathcal{M}) \rightarrow S \triangleleft^o \mathcal{M}$ (see definition of $\mu$ above).</td>
</tr>
<tr>
<td>$\triangleleft^l$</td>
<td>linguistic model-of</td>
<td>indicates that the model controls the <em>form</em> of its elements. This automatically implies $\triangleleft_t^l$ and, hence $S \triangleleft_t^l \mathcal{M} \iff S \in \varepsilon(\mu(\mathcal{M}))$.</td>
</tr>
</tbody>
</table>
\( S \rightarrow S_{r \cdot \text{type}} \)
\( S_{r \cdot \text{type}} \rightarrow \mathcal{L} \)
\( \mathcal{L} \rightarrow \mathcal{M}_{\text{type}} \)
\( \mathcal{M}_{\text{type}} \rightarrow \mathcal{M}_L \)
\( \mathcal{M}_L \rightarrow \mathcal{M}_{\text{token}} \)
\( \mathcal{M}_{\text{token}} \rightarrow S_{r \cdot \text{token}} \)

\( S \rightarrow \mathcal{M} \)
\( \mathcal{M} \rightarrow S_r \)

\( \alpha = \tau \circ \pi \)
\( \pi \)
\( \rho \)

\( \mu \)
\( \mu \)

\( \text{member of the extension of} \)
\( \text{member of the extension of} \)
Ontological vs. Linguistic Conformance/Instantiation

[Diagram showing concepts and relationships involving "Collie", "Lassie", and "Object" within two layers, L₀ and L₁, illustrating the distinctions between ontological and linguistic conformance/instatiation.]
tr432:
- number = 432
- startDate = 1/1/2008
- colour = "silver"

conductor1:
- name = "Joeri"

conductor2:
- name = "Bentley"
TrainType
+ number: int
+ startDate: Date

ConductorType
+ name: String

MS82
+ colour: String

MS83
+ colour: String
+ feature: int

tr432:
number = 432
startDate = 1/1/2008
colour = "silver"

conductor1:
name = "Andrei"

conductor2:
name = "Bentley"
TrainType
+ number: int
+ startDate: Date

ConductorType
+ name: String

OtherType
+ name: String

does not conform

MS82
+ colour: String

MS83
+ colour: String
+ feature: int

tr432:
number = 432
startDate = 1/1/2008
colour = "silver"

conductor1:
name = "Andrei"

conductor2:
name = "Bentley"
constraints:

- distinct names of ConductorType instances
- total # instances of MS82 < 10
**tr430:**
- number = 430
- startDate = 1/1/2008
- colour = "bluer"

```
Prototype
```

**tr432:**
- number = 432
- startDate = 1/1/2000
- colour = "silver"

**conductor1:**
- name = "Joeri"

**conductor2:**
- name = "Bentley"
Textual language for (multi-level) (meta-)modelling

Constraint and operations in the Epsilon Object Language (EOL)
https://www.eclipse.org/epsilon/

EOL is similar the UML’s Object Constraint Language (OCL)

EOL can be used in metaDepth for:

- Queries within the model (limited to navigation)
- Computation/assignment, for example to give operational semantics
Model TrainNet {

    Node Train{
        train_name : String {id};
        maxVelocity : int;
        maxVConstraint: $self.maxVelocity > 0 \text{ and } self.maxVelocity \leq 300$
        currSegment : Segment[1];
    }

    Node Segment{
        currTrain : Train[0..1];
        nextSegment : Segment[0..1];
    }

    Node FastTrain : Train {}
    Node IntercityExpress: Train {}
}
TrainNet tn {
    Train train1 {train_name = "T3456"; maxVelocity = 300; currSegment = seg1;}
    IntercityExpress train2 {train_name = "ICE1234"; maxVelocity = 250;}
    Segment seg1 {currTrain = train1;}
    Segment seg2 {}
}

Verifying conformance between tn and TrainNet:

Constraint fails: cardinality 0 nota allowed for currSegment in train2

Reason: train2 is not on a Segment
Example query: What are the trains in each segment?

operation main()
  for (s in Segment.allInstances()){
    if (Train.allInstances.excludes(s.currTrain)){
      ('Segment ' + s + " has no train.").println();
    } else{
      ("Segment " + s + " has train: " + s.currTrain.toString + ":").println();
    }
  }

Result:

Segment seg2 has no train.
Segment seg1 has train 'train1'

Caveat: explicit navigation (via . . . ), need to know types
(or be able to query: Types.allInstances())
Linguistic Conformance: morphism
Name-based conformance checking Algorithm 1

Input: model, type_model
1: type_to.elements ← populate_types(model, type_model)
2: for el in model do
3:   if not type(el) in type_to.elements then
4:     return False
5:   end if
6:   append el to type_to.elements[type(el)]
7: end for
8: for type in type_to.elements do
9:   if len(type_to.elements[type]) < get_minimum(type) then
10:  return False
11:  end if
12:  if len(type_to.elements[type]) > get_maximum(type) then
13:  return False
14:  end if
15:  for el in type_to.elements[type] do
16:    for attr in get_attributes(el) do
17:      if not type(attr) in get_attributes(type) then
18:        return False
19:      end if
20:      if not get_value(attr) = get_type(type) then
21:        return False
22:      end if
23:    end for
24:  end for
25: out.associations ← classify_by_type(get_out.associations(el))
26: for assoc_type in out.associations do
27:   if not get_in_type(assoc_type) in get_all_types(el) then
28:     return False
29:   end if
30:   if len(out.associations[assoc_type]) < get_minimum_out(assoc_type) then
31:     return False
32:   end if
33:   if len(out.associations[assoc_type]) > get_maximum_out(assoc_type) then
34:     return False
35:   end if
36: end for
37: in.associations ← classify_by_type(get_in.associations(el))
38: for assoc_type in in.associations do
39:   if not get_out_type(assoc_type) in get_all_types(el) then
40:     return False
41:   end if
42:   if len(in.associations[assoc_type]) < get_minimum_in(assoc_type) then
43:     return False
44:   end if
45:   if len(in.associations[assoc_type]) > get_maximum_in(assoc_type) then
46:     return False
47:   end if
48: end for
49: return True

In Algorithm 1, the linguistic conformance check of the MvK is shown. It checks whether a given model conforms to a given type model. It consists of four checks:

1. (Lines 1-7) Checks whether all elements in the model are typed by an element in the type model. Functions used are:
   (a) populate_types: returns a mapping between types of the type model, and instances of those type in the model.
   (b) type: returns the type of an element.

2. (Lines 8-14) Checks whether the minimum and maximum cardinality for each type in the type model is satisfied. Functions used are:
   (a) get_minimum: returns the minimum cardinality for an given type.
   (b) get_maximum: returns the maximum cardinality for an given type.

3. (Lines 15-24) Checks, for all attributes of all elements in the model, whether a type definition for the attribute can be found in the type model (lines 17-19), and whether the type of the attribute value corresponds to the type defined in the attribute type (lines 20-22). Functions used are:
   (a) get_attributes: returns all attributes of an element.
   (b) get_value: returns the value of an attribute.
   (c) get_type: returns the type of the values which can be assigned to an attribute.

4. (Lines 25-49) Checks, for each incoming and outgoing association of each element of the model, whether, respectively, the incoming and outgoing cardinalities are satisfied. Also checks whether the types of the connected elements correspond to those defined in the association. Functions used are:
   (a) classify_by_type: classifies the given elements by their type, and returns a mapping between types and instances.
   (b) get_out.associations: returns all outgoing associations of an element.
   (c) get_in.associations: returns all incoming associations of an element.
   (d) get_minimum_out: returns the minimum number of outgoing associations of a particular type.
Name-based conformance checking Algorithm 1

Input: model, type_model
1: type_to_elements ← populate_types(model, type_model)
2: for el in model do
3:   if not type(el) in type_to_elements then
4:     return False
5:   end if
6:   append el to type_to_elements[type(el)]
7: end for
8: for type in type_to_elements do
9:   if len(type_to_elements[type]) < get_minimum(type) then
10:      return False
11:    end if
12:   if len(type_to_elements[type]) > get_maximum(type) then
13:      return False
14:    end if
15:   for el in type_to_elements[type] do
16:     for attr in get_attributes(el) do
17:       if not type(attr) in get_attributes(type) then
18:         return False
19:       end if
20:     if not type(get_value(attr)) = get_type(type(attr)) then
21:        return False
22:     end if
23:   end for
24: end for
25: out_associations ← classify_by_type(get_out_associations(el))
26: for assoc_type in outAssociations do
27:   if not(get_in_type(assoc_type) in get_all_types(el)) then
28:     return False
29:   end if
30:   if len(out_associations[assoc_type]) < get_minimum_out(assoc_type) then
31:     return False
32:   end if
33:   if len(out_associations[assoc_type]) > get_maximum_out(assoc_type) then
34:     return False
35:   end if
36: end for
37: in_associations ← classify_by_type(get_in_associations(el))
38: for assoc_type in in_associations do
39:   if not(get_out_type(assoc_type) in get_all_types(el)) then
40:     return False
41:   end if
42:   if len(in_associations[assoc_type]) < get_minimum_in(assoc_type) then
43:     return False
44:   end if
45:   if len(in_associations[assoc_type]) > get_maximum_in(assoc_type) then
46:     return False
47:   end if
48: end for
49: end for
50: return True

- (e) get_maximum_out: returns the maximum number of outgoing associations of a particular type.
- (f) get_minimum_in: returns the minimum number of incoming associations of a particular type.
- (g) get_maximum_in: returns the maximum number of incoming associations of a particular type.
- (h) get_out_type: returns the type defined for the outgoing multiplicity of an association.
- (i) get_in_type: returns the type defined for the incoming multiplicity of an association.
- (j) get_all_types: returns the type of the element, as well as all its subtypes.
MetaDepth http://metadepth.org/

Meta-model ↔ model

strict Model PetriNets@1 {
  abstract Node NamedElement{
    name  : String {id};
  }
  Node Place : NamedElement {
    tokens   : int = 0;
    outTrans : Transition[*] {ordered,unique};
    inTrans  : Transition[*] {ordered,unique};
    minTokens: $self.tokens>=0$
  }
  Node Transition : NamedElement {
    inPlaces : Place[*] {ordered,unique};
    outPlaces: Place[*] {ordered,unique};
    minTokens: $self.tokens>=0$
  }
  Edge ArcPT (Place.outTrans, Transition.inPlaces) {
    weight: int = 1;
  }
  Edge ArcTP (Transition.outPlaces, Place.inTrans) {
    weight: int = 1;
  }
  minWeight(ArcTP, ArcPT): $self.weight>0$
  minPlaces:$Place.allInstances().size()>0$
}

load "PetriNets"
PetriNets Test{
  Place p0{name="p0"; tokens=2;}
  Place p1{name="p1"; tokens=0;}
  Place p2{name="p2"; tokens=2;}
  Transition t1{name="t1";}
  ArcPT pt0(p0,t1){weight=1;}
  ArcPT pt1(p2,t1){weight=1;}
  ArcTP tp(t1,p1){weight=2;}
}
MetaDepth

**operational semantics**

```java
operation main() {
    'Simulating the Petri Net'.println();
    while (Transition.allInstances()->exists(t|t.enabled() and t.fire()) ) {} 
}
operation Transition enabled() : Boolean {
    ('checking enabledness of '+self.name).println();
    return self.ArcPT->forAll(arc| arc.inPlaces.tokens>=arc.weight);
}
operation Transition fire() : Boolean {
    ('Firing '+self.name).println();
    for (arc in self.ArcPT)
        arc.inPlaces.tokens := arc.inPlaces.tokens-arc.weight;
    for (arc in self.ArcTP)
        arc.outPlaces.tokens := arc.outPlaces.tokens+arc.weight;
    return true;
}
```
Meta-hierarchy – OMG’s 4 Layer Architecture

Triangle of Meaning
aka triangle of reference or semiotic triangle
Ogden and Richards in “the meaning of meaning” (1923)

caveat: collaborative modelling only works if collaborators have a shared interpretation
David Harel, Bernhard Rumpe.

*Meaningful Modeling: What's the Semantics of "Semantics"?*

- “operational” semantics
- “denotational” (transformational) semantics
Operational vs. Denotational (Translational) semantics

NATO’s Sarajevo Waste Water Treatment Plant

www.nato.int/sfor/cimic/env-pro/waterpla.htm
What does this WWTP model mean?
...its meaning (steady-state abstraction): Causal Block Diagram (CBD)
Meaning of the CBD ... semantic mapping onto algEqns

\[
\begin{align*}
    f_{\text{influent}} &= C_{\text{influent}} \\
    f_{\text{bacteria}} &= C_{\text{bacteria}} \\
    f_{\text{mixed}} &= f_{\text{influent}} + f_{\text{bacteria}} \\
    aeration_{\text{fraction}} &= C_{\text{aeration}} \\
    f_{\text{processed}} &= aeration_{\text{fraction}} \ast f_{\text{mixed}} \\
    settling_{\text{fraction}} &= C_{\text{settling}} \\
    \negated &= -settling_{\text{fraction}} \\
    \text{one} &= 1 \\
    dump_{\text{fraction}} &= \text{one} + \negated \\
    f_{\text{dump}} &= f_{\text{processed}} \ast dump_{\text{fraction}} \\
    f_{\text{out}} &= settling_{\text{fraction}} \ast f_{\text{processed}}
\end{align*}
\]
“linguistic” view on Modelling Languages/Formalisms: Syntax and Semantics
Explicit “linguistic” Modelling of Modelling Languages/Formalisms
Meaning of the CBD... semantic mapping onto algEqns

\[
\begin{align*}
  f_{\text{influent}} &= C_{\text{influent}} \\
  f_{\text{bacteria}} &= C_{\text{bacteria}} \\
  f_{\text{mixed}} &= f_{\text{influent}} + f_{\text{bacteria}} \\
  aeration_{\text{fraction}} &= C_{\text{aeration}} \\
  f_{\text{processed}} &= aeration_{\text{fraction}} \times f_{\text{mixed}} \\
  settling_{\text{fraction}} &= C_{\text{settling}} \\
  \text{negated} &= -settling_{\text{fraction}} \\
  \text{one} &= 1 \\
  dump_{\text{fraction}} &= \text{one} + \text{negated} \\
  f_{\text{dump}} &= f_{\text{processed}} \times dump_{\text{fraction}} \\
  f_{\text{out}} &= settling_{\text{fraction}} \times f_{\text{processed}}
\end{align*}
\]
Causal Block Diagrams (syntax)
Causal Block Diagrams (semantics)
logicalTime ← 0
while not end_condition do
    schedule ← LOOPDETECT(DEPGRAPH(cbd))
    for gblock in schedule do
        COMPUTE(gblock)
    end for
    logicalTime ← logicalTime + Δt
end while
Causal Block Diagrams (semantics)
Formalism Transformation Graph (FTG)

- WWTP
- CBD
- (Diff)AlgEqns
- continuous-time signals
Formalism Transformation Graph (FTG)

Bran Selic: “fragmentation problem”

Multi-formalism coupled model: multi-formalism modelling

CouplingGraph

Msub_1

Msub_2

Msub_3

CoupledModel
Formalism Transformation Graph (FTG)


https://www.youtube.com/watch?feature=player_detailpage&v=RYtea2BiQ98
at the most appropriate level(s) of abstraction
using the most appropriate formalism(s)
explicitly modelling workflows

Enabler: (domain-specific) modelling language engineering,
including model transformation