Meta-Modelling and Graph Rewriting, Enablers for Domain-Specific (Visual) Modelling

“model everything”

Hans Vangheluwe

Modelling, Simulation and Design Lab (MSDL)
School of Computer Science, McGill University, Montréal, Canada
Overview

1. Domain-Specific (Visual) Modelling – DS(V)M
   - What/Why of DS(V)M (and DS(V)Ls) ?
   - Examples

2. Building DS(V)M Tools Effectively
   (a) Specifying textual/visual syntax of DS(V)Ls: *meta-modelling*
   (b) Specifying DS(V)L *semantics*
   (c) Modelling (and executing) *transformations*: *graph rewriting*
   (d) **Traffic**, a domain-specific modelling formalism
3. AToM$^3$, A Tool for Multi-formalism Meta-Modelling

- Different “aspects” of AToM$^3$
- A plethora of applications
- Current and future work
Domain-Specific (Visual) Modelling

DS(V)M “Domain-Specific (Visual) Modelling”
DS(V)Ls “Domain-Specific (Visual) Languages”

Some starting points:

- [www.dsmforum.org](http://www.dsmforum.org)
- OOPSLA Workshops on DSM
- The OMG’s Model Driven Architecture (MDA) [www.omg.org/mda](http://www.omg.org/mda)
Why DS(V)M?
DS(V)M Example 1: smart phones, the model

MetaEdit+ (www.metacase.com)
DS(V)M Example 1: smart phones, the application

MetaEdit+ (www.metacase.com)
Why DS(V)M?

- match the user’s mental model of the problem domain
- maximally constrains users
- separate domain-expert’s
  from transformation expert’s work,
  from target domain (e.g., code libraries) expert’s work
- re-use transformation knowledge
- model everything (nothing left implicit)
How to do DS(V)M?
DS(V)M Example 2: Waste Water Treatment Plant

NATO’s Sarajevo WWTP

http://www.nato.int/sfor/cimic/env-pro/waterpla.htm
WWTP Block Diagram (domain-specific) model
Model of the Transformation from WWTP to . . .
... its meaning (in the steady-state abstraction): Causal Block Diagram (CBD)
Multi-step Transformation: meaning of the CBD

\[
\begin{align*}
 f_{\text{influent}} &= C_{\text{influent}} \\
 f_{\text{bacteria}} &= C_{\text{bacteria}} \\
 f_{\text{mixed}} &= f_{\text{influent}} + f_{\text{bacteria}} \\
 \text{aeration\_fraction} &= C_{\text{aeration}} \\
 f_{\text{processed}} &= \text{aeration\_fraction} * f_{\text{mixed}} \\
 \text{settling\_fraction} &= C_{\text{settling}} \\
 \text{negated} &= \neg \text{settling\_fraction} \\
 \text{one} &= 1 \\
 \text{dump\_fraction} &= \text{one} + \text{negated} \\
 f_{\text{dump}} &= f_{\text{processed}} * \text{dump\_fraction} \\
 f_{\text{out}} &= \text{settling\_fraction} * f_{\text{processed}}
\end{align*}
\]
WWTP Simulation Environment

www.hemmis.com/products/west/

Building DS(V)M Tools Effectively

- development cost of DS(V)M Tools may be prohibitive!
- we want to effectively (rapidly, correctly, re-usably, ...) specify and generate/execute:
  - DS(V)Ls
  - (reactive) behaviour of DS(V)M environments
  - transformations

⇒ model everything
Building DS(V)M Tools Effectively

1. Specifying textual/visual syntax of DS(V)Ls: 
   *meta-modelling*

2. Specifying DS(V)L *semantics*

3. Modelling (and executing) *transformations*: 
   *graph rewriting*

4. **Traffic**, a domain-specific modelling formalism
Specifying textual/visual syntax of DS(V)Ls

- **abstract** syntax:
  - meta-model (∼ type graph)
  - syntax grammar (*SableCC* text grammar, *AToM*³ Graph Grammar)

- **concrete** syntax:
  - textual (*SableCC* lexing)
  - visual (*AToM*³ “icons” + connections)
Meta-modelling: model-instance morphism
Meta-modelling: model-instance morphism ctd.
How to break the meta-* chain?

*bootstrap* ER (Core == Entity Relationship)

note: interpret or compile meta-models
Example: the Traffic domain
a (domain-specific) model in the Traffic formalism
A Traffic meta-model (an Entity Relationship model)
Traffic meta-model notes

1. ER was extended with *constraints*
   (OCL soon, now Python code)

2. alternative meta-formalisms: could use UML Class Diagrams

3. the given meta-model only specifies the Abstract Syntax of Traffic,
   need to add Concrete (Visual) Syntax
Traffic Concrete Syntax (the Capacity Entity)
The generated Traffic visual modelling environment
Caveat: specifying the GUI’s Reactive Behaviour
The GUI’s reactive behaviour in action

current work: what is the *optimal* formalism to specify GUI reactive behaviour?
Modelling Traffic’s Semantics

- choices: timed, un-timed, ... (level of abstraction)
- “denotational”: map onto known formalism (TTPN, PN) good for analysis purposes
- “operational”: procedure to execute/simulate model may act as a reference implementation
- note: need to prove that denotational and operational semantics correspond if both are given!
Traffic, the Big Picture

Timed Transition Petri Nets

Place-Transition Petri Nets

Coverability Graph

Traffic (timed)

Traffic (un-timed)

simulate

describe semantics by mapping onto

analyze: reachability, coverability, ...

compute all possible behaviours

simulate

describe semantics by mapping onto

simulate

describe semantics by mapping onto

Hans Vangheluwe
hv@cs.mcgill.ca
MM, GG, DSVM

32/78
Traffic’s semantics in terms of Petri Nets (PN)

- need a meta-model of Traffic (shown before)
- need a meta-model of Petri Nets (shown before)
- need a model of the mapping: Traffic $\Rightarrow$ Petri Nets
Graph Grammars to Specify Model Transformations

References:

Tools:
AGG, PROGRES, GME, AToM$^3$, Fujaba, ...
Traffic to Petri Net Graph Grammar rules

INITIAL ACTION:
for node in graph.listNodes["RoadSection"]:  
node.vehiclesPNPlaceGenerated=False
A very simple Traffic model

segment1

capacity

segment2
Traffic to Petri Net Graph Grammar rules

LHS

1

<ANY>

<ANY>

RHS

1

<COPIED>

<COPIED>

rule1: RoadSection2PNPlace

CONDITION:
node = LHS.nodeWithLabel(1)
return not node.vehiclesPNPlaceGenerated

ACTION:
node = RHS.nodeWithLabel(1)
node.vehiclesPNPlaceGenerated = True
Road Sections converted to Petri Net Places
Traffic to Petri Net Graph Grammar rules

rule 2: Flow2PNTransition

CONDITION:
node = getMatched(LHS.nodeWithLabel(1))
return node.in_connections_ == []

ACTION:
node = RHS.nodeWithLabel(1)
node.capacityPNPlaceGenerated = True
Traffic Flow to Petri Net Transitions
Traffic to Petri Net Graph Grammar rules

LHS

rule 3: Capacity2PNPlace

RHS

LHS.nodeWithLabel(1)).name
LHS.nodeWithLabel(1)).capacity

Hans Vangheluwe
hv@cs.mcgill.ca
MM, GG, DSVM
Traffic Capacity to Petri Net Place
Traffic to Petri Net Graph Grammar rules

LHS

RHS

rule 4: Capacity2PNPlaceLinks
Traffic Capacity to Petri Net Place (links)
Traffic to Petri Net Graph Grammar rules

rule 5: Capacity2PNPlaceCleanup
Traffic Capacity to Petri Net Place cleanup
Traffic to Petri Net Graph Grammar rules

LHS

RHS

CONDITION:
cap_place = LHS.nodeWithLabel(6)
out_trans = LHS.nodeWithLabel(4)
capacity_transition_absent = True
for in_link in cap_place.in_connections:
    for out_link in out_trans.out_connections:
        if (in_link == out_link) and
           isinstance(in_link, tran2pl):
            capacity_transition_absent = False
            break
return capacity_transition_absent

rule 6: CapacityConstraintOnPl2Tr
Capacity Constraint on Place to Transition

---

Hans Vangheluwe  hv@cs.mcgill.ca  MM, GG, DSVM
Traffic to Petri Net Graph Grammar rules

**LHS**

**RHS**

**CONDITION:**
- `cap_place = LHS.nodeWithLabel(6)`
- `in_trans = LHS.nodeWithLabel(4)`
- `capacity_transition_absent = True`

for `out_link in cap_place.out_connections_`:
  for `in_link in in_trans.in_connections_`:
    if `(in_link == out_link)` and
      `isinstance(in_link, pl2tran)`:
      `capacity_transition_absent = False`
break
return `capacity_transition_absent`

rule 7: `CapacityConstraintOnTr2Pl`
Capacity Constraint on Transition to Place
Traffic to Petri Net Graph Grammar rules

rule 8: InitialCapacity

\[
\text{initial\_num\_vehicles} = \text{LHS}\.\text{nodeWithLabel(1)}\.\text{num\_vehicles} \\
\text{capacity\_tokens} = \text{LHS}\.\text{nodeWithLabel(2)}\.\text{tokens} \\
\text{return capacity\_tokens} - \text{initial\_num\_vehicles}
\]
Model Initial Capacity (applied rule twice)
Traffic to Petri Net Graph Grammar rules

LHS

1

ANY

2

RHS

rule 9: RemoveRoadSection

Hans Vangheluwe

hv@cs.mcgill.ca

MM, GG, DSVM

53/78
Removed Traffic Road Section, now only Petri Net
Static Analysis of the Transformation Model

The transformation specified by the Graph Grammar model must satisfy the following requirements:

- **Convergence**: the transformation process is finite
- **Uniqueness**: the transformation results in a single target model
- **Syntactic Consistency**: the target model must be exclusively in the target formalism
Traffic, the Big Picture

Timed Transition Petri Nets

Traffic (timed)

Timed Petri Nets

describe semantics by mapping onto

Traffic (un-timed)

Place-Transition Petri Nets

compute all possible behaviours

analyze: reachability, coverability, ...

Coverability Graph

simulate

analyze

simulate
A less trivial Traffic model
the Petri Net describing its behaviour obtained by Graph Rewriting
Analysis: a Coverability Graph for the Petri Net
Conservation Analysis

\[\begin{align*}
1.0 \ x[\text{turn1\_CAP}] + 1.0 \ x[\text{turn1}] &= 1.0 \\
1.0 \ x[\text{cars}] + 1.0 \ x[\text{bot\_W2E}] + 1.0 \ x[\text{turn1}] + \\
1.0 \ x[\text{to\_N\_or\_W}] + 1.0 \ x[\text{turn2}] + 1.0 \ x[\text{bot\_N2S}] &= 2.0 \\
1.0 \ x[\text{top\_CAP}] + 1.0 \ x[\text{to\_N\_or\_W}] &= 1.0 \\
1.0 \ x[\text{turn2\_CAP}] + 1.0 \ x[\text{turn2}] &= 1.0 \\
1.0 \ x[\text{bot\_CAP}] + 1.0 \ x[\text{bot\_W2E}] + 1.0 \ x[\text{bot\_N2S}] &= 1.0
\end{align*}\]
Model-Based Development, Modify the Model

model \xrightarrow{\text{transformation}} \text{app}

\text{small modification} \quad \text{model'} \xrightarrow{\text{transformation}} \text{app'}
Model-Based Development, Modify the Transformation

- Model
- Transformation
- App
- Small Modification
- Model
- Transformation’
- App’
AToM$^3$, A Tool for Multi-formalism Meta-Modelling

atom3.cs.mcgill.ca

Hans Vangheluwe hv@cs.mcgill.ca MM, GG, DSVM
Forrester System Dynamics model of Predator-Prey behaviour
Causal Block Diagram model of a Harmonic Oscillator
Petri Net model of a Producer-Consumer system
Statechart model of a Producer-Consumer system
GPSS model of a Telephone Exchange
1 FUNCTION RN1,C24
0.0,0.0/0.1,0.104/0.2,0.222/0.3,0.355/0.4,0.509/0.5,0.69
0.6,0.915/0.7,1.2/0.75,1.38/0.8,1.6/0.84,1.83/0.88,2.12
0.9,2.13/0.92,2.52/0.94,2.81/0.95,2.99/0.96,3.2/0.97,3.5
0.98,3.9/0.99,4.6/0.995,5.3/0.998,6.2/0.999,7.0/0.9997,8.0
*
L1 GENERATE 12,FN1
L3 TEST G V2,2,OUT
L4 ASSIGN 1,V1,H
L5 GATE LR PH1,L4
L6 ASSIGN 2,V1,H
L7 TEST NE P1,P2,L6
L0 LOGIC R PH1
L8 TRANSFER BOTH,L9,L11
L9 LOGIC R PH1
L10 TERMINATE 1
OUT TERMINATE 0
L11 ENTER LNKS
L12 GATE LR PH2,L13
L16 LOGIC S PH2

1 VARIABLE XH1*RN1/1000+1
2 VARIABLE XH1-2*S$LNKS
HS=ES+ODE, a Hybrid Formalism

- To demonstrate the ease with which an HS-specific modelling environment can be built
- HS combines
  - Event Scheduling (ES) to describe discrete-event behaviour
  - Ordinary Differential Equations (ODEs) to describe continuous-time behaviour
  - Constructs to link ES and ODE
Example: Personalized Rapid Transit (PRT)

- Discrete-event behaviour: loading and unloading passengers
- Continuous-time behaviour: accelerating, decelerating
- Control: keep velocity between boundaries by switching between continuous “modes”
Passenger Arrival

```
START EVENT

Initialize_Model

\[ x = x_0 \]
\[ v = v_0 \]
\[ \text{passengers} = 0 \]

Passenger_arrive

\[ \text{passengers} = \text{passengers} \]

IF \[ \text{passengers} < 10 \]
AFTER \[ \text{random.uniform}(1, 10) \]

IF \[ 0 \]

IF \[ 1 \]

IF \[ 1 \]

IF \[ \text{passengers} \geq 10 \]
AFTER \[ 0 \]

Train is full
print "Train is leaving i"

Train_at_rest
```

Hans Vangheluwe
hv@cs.mcgill.ca

MM, GG, DSVM
Train Motion

TrainMotion

AcceleratingODE
\[ x_{vvk} \cdot (v - v_{init} + 5) \]

FrictionODE
\[ x_{vvk} \cdot (v - 20) \]

BrakingODE
\[ x_{vvk} \cdot (v + 3) \]

print "Train is leaving a"

Train_starts

Stop_Accelerating

Start_Accelerating

monitoring fct.:
\[ v_{max} - v \]

Start_Braking

FrictionODE
\[ x_{vvk} \cdot (v - 20) \]

monitoring fct.:
\[ v - v_{min} \]

AFTER

5

Hans Vangheluwe
hv@cs.mcgill.ca
MM, GG, DSVM
Train Arrival and Unloading

Train at rest

DepartureStart
print "Train arrived at t"

Departure_Event
passengers = passengers -

IF
passengers > 0

AFTER
0

IF
1

AFTER
5

Test_zerospeed

BrakingODE

\[ \dot{x} = y \]
\[ \dot{y} = -k \cdot (y + 3) \]
The Complete Model

...
Simulating the (compiled) Model
Train Passengers

[TrainSchedule] time vs passengers in red dots
Train Velocity

[TrainSchedule] time vs v in blue lines