Computer Automated Multi-Paradigm Modelling for Analysis and Design of Traffic Networks

“model everything”

Hans Vangheluwe

School of Computer Science, McGill University, Montréal, Canada

Juan de Lara

E.T.S. de Informática, Universidad Autónoma de Madrid, Madrid, Spain
Overview

1. Computer Automated Multi-Paradigm Modelling (CAMPaM)
   ⇒ Domain Specific (Visual) Modelling – DS(V)M
   • What/Why of DS(V)M (and DS(V)Ls)?

2. Building DS(V)M Tools Effectively
   (a) Specifying textual/visual syntax of DS(V)Ls: *meta-modelling*
   (b) Specifying DS(V)L *semantics*: *transformations*
   (c) Modelling (and executing) *transformations*: *graph rewriting*

3. Traffic, a domain specific modelling formalism
   • Modelling a Traffic-Specific Modelling Tool
   • Various Transformations

4. Conclusions and Future Work
Computer Automated Multi-Paradigm Modelling (CAMPaM)

1. Different *levels of abstraction*

2. Mixing *different modelling formalisms* (coupling, transformation)

3. Modelling classes of models (formalisms) by *meta-modelling*

4. Modelling *model transformations* explicitly

Pieter J. Mosterman and Hans Vangheluwe.

Computer Automated Multi-Paradigm Modeling: An Introduction.

A Simple Example to Demonstrate Concepts: Model/Analyze/Simulate Traffic Networks
Approach: Domain Specific (Visual) Modelling
Why DS(V)M?
(as opposed to General Purpose modelling)

- **match the user’s mental model** of the problem domain
- **maximally constrain** user (to the problem at hand)
  - ⇒ easier to learn
  - ⇒ avoid errors
- **separate** domain-expert’s work
  from analysis/transformation expert’s work
- **re-use** transformation knowledge
  (e.g., in variations of a domain specific formalism)
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**:
  - Domain Specific (Visual) **Languages** (DS(V)Ls)
  - (reactive) **behaviour** of DS(V)M environments/tools
  - **model transformations** (for analysis, optimization, simulation . . .)

⇒ **model** everything
How to Build DS(V)M Tools Effectively?

1. Specify textual/visual syntax of DS(V)Ls: meta-modelling
2. Specify DS(V)L semantics: transformation
3. Model (and analyze and execute) transformations: graph rewriting
A Tool for Multi-formalism and Meta-Modelling

atom3.cs.mcgill.ca
Specifying textual/visual syntax of DS(V)Ls

- **abstract** syntax:
  - syntax grammar (text grammar, AToM³ Graph Grammar) or
  - **meta-model** (~ type graph)

- **concrete** syntax:
  - textual (lexical specification) or
  - **visual** (AToM³ “icons” + connections)
Meta-modelling: model-instance morphism

Level M2: model
a model of the Petri Net formalism,
an INSTANCE of the Entity Relationship formalism

Level M1: data
a Petri Net,
an INSTANCE of the Petri Net formalism
Un-timed and timed Traffic Formalism

meta-model
(a model in the UML Class Diagram Formalism)
Traffic Concrete Syntax (the Capacity Entity)
The generated Traffic visual modelling environment
Caveat: Statechart model of 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* consistency between denotational and operational semantics if both are given!
Traffic, the Big Picture

Timed Transition Petri Nets

Traffic (timed)

Traffic (un-timed)

Place-Transition Petri Nets

Coverability Graph

describe semantics by mapping onto

simulate

analyze: reachability, coverability, ...

compute all possible behaviours

simulate

analyze

describe semantics by mapping onto

simulate
Traffic’s (un-timed) semantics in terms of Petri Nets

- 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 Transformation for Model Transformations


Tools:
AGG, PROGRES, GME, AToM³, Fujaba, …
A very simple Traffic model

![Traffic Model Diagram]

- Segment 1
- Segment 2
- Capacity

WSC 2004, 5 December, Washington, DC
CAMPaM for Analysis and Design of Traffic Networks

22/54
Traffic to Petri Net Graph Transformation rules

INITIAL ACTION:
for node in graph.listNodes("RoadSection"): 
    node.vehiclesPNPlaceGenerated=False
Traffic to Petri Net Graph Transformation rules

LHS

1

<ANY>

<ANY>

LHS.nodeWithLabel(1).name
LHS.nodeWithLabel(1).num_vehicles

RHS

1

<COPIED>

<COPIED>

rule1: RoadSection2PNPlace

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

ACTION:
node = RHS.nodeWithLabel(1)
nod.vehiclesPNPlaceGenerated = True

1

<SPECIFIED> LHS.nodeWithLabel(1).name

<SPECIFIED> LHS.nodeWithLabel(1).num_vehicles

2

3

<SPECIFIED> LHS.nodeWithLabel(1).name

<SPECIFIED> LHS.nodeWithLabel(1).num_vehicles
Road Sections converted to Petri Net Places
Traffic to Petri Net Graph Transformation rules

LHS

RHS

rule 2: Flow2PNTransition
Traffic Flow to Petri Net Transitions

WSC 2004, 5 December, Washington, DC
CAMPaM for Analysis and Design of Traffic Networks
Traffic to Petri Net Graph Transformation rules

**Rule 3: Capacity2PNPlace**

**Condition:**
```
return LHS.nodeWithLabel(1).in_connections_ == []
```

**Left-hand Side (LHS):**
- Node labeled 1
- Any
- Any

**Right-hand Side (RHS):**
- Node labeled 1
- 3
- Specified
- Specified
- Specified

- LHS\_nodeWithLabel(1).capacity
- LHS\_nodeWithLabel(1).name
Traffic Capacity to Petri Net Place
Traffic to Petri Net Graph Transformation rules

rule 4: Capacity2PNPlaceLinks

LHS

RHS
Traffic Capacity to Petri Net Place (links)
Traffic to Petri Net Graph Transformation rules

rule 5: Capacity2PNPlaceCleanup
Traffic to Petri Net Graph Transformation 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:
    if (in_link == out_trans) and 
        isinstance(in_link, tran2pl):
        capacity_transition_absent = False
        break
return capacity_transition_absent

rule 6: CapacityConstraintOnPl2Tr
Capacity Constraint on Place to Transition
Traffic to Petri Net Graph Transformation 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 Transformation rules

rule 8: InitialCapacity

LHS

RHS

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

rule 9: RemoveRoadSection

LHS

RHS
Removed Traffic Road Section, now only Petri Net
Static Analysis of the Transformation Model

The transformation specified by the Graph Transformation 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

These properties can often (but not always) be _statically_ checked/proved.
Constraints on Behaviour can be Guaranteed
Un-timed Analysis

Timed Transition Petri Nets

Traffic (timed)

Traffic (un-timed)

Place-Transition Petri Nets

Coverability Graph

analyze: reachability, coverability, ...

compute all possible behaviours

describe semantics by mapping onto

simulate

neglect time

describe semantics by mapping onto

simulate

simulate

analyze
An un-timed Traffic model
the Petri Net describing its behaviour obtained by Graph Rewriting
Conservation Analysis

\[1.0 \times \text{turn1\_CAP} + 1.0 \times \text{turn1} = 1.0\]

\[1.0 \times \text{cars} + 1.0 \times \text{bot\_W2E} + 1.0 \times \text{turn1} + 1.0 \times \text{to\_N\_or\_W} + 1.0 \times \text{turn2} + 1.0 \times \text{bot\_N2S} = 2.0\]

\[1.0 \times \text{top\_CAP} + 1.0 \times \text{to\_N\_or\_W} = 1.0\]

\[1.0 \times \text{turn2\_CAP} + 1.0 \times \text{turn2} = 1.0\]

\[1.0 \times \text{bot\_CAP} + 1.0 \times \text{bot\_W2E} + 1.0 \times \text{bot\_N2S} = 1.0\]
Timed Semantics by mapping onto TTPN

Juan de Lara, Hans Vangheluwe, and Pieter J. Mosterman.
Modelling and analysis of traffic networks based on graph transformation.

*Formal Methods for Automation and Safety in Railway and Automotive Systems.*
Traffic Network
Converted to TTPN
(ready for analysis/simulation)
Iterative Simulation (Executing GG)

Traffic (timed)
- Timed Transition Petri Nets
- describe semantics by mapping onto
- simulate
- analyze

Traffic (un-timed)
- Place-Transition Petri Nets
- compute all possible behaviours
- simulate
- analyze: reachability, coverability, ...

Coverability Graph
Conclusions

Demonstrated **feasibility** of rapidly and re-usably building Domain Specific Visual Modelling, Analysis, Simulation tools using **meta-modelling** and **graph rewriting**.

**model** everything
Future Work

... and add hierarchy