A Software Architecture for Multi-paradigm Modelling

Hans Vangheluwe

School of Computer Science, McGill University, Montréal, Canada

Juan de Lara

E.T.S. de Informática, Universidad Autonóma de Madrid, Madrid, Spain

Ghislain Vansteenkiste

BIOMATH department, Ghent University, Ghent, Belgium
Presentation Overview

- Complex systems
- Multi-paradigm Modelling and Simulation
- 1. Levels of abstraction
- 2. Multi-formalism Modelling and Simulation
- 3. Meta-modelling formalism syntax and semantics
- The AToM³ environment
Complexity (system/model) due to . . .

1. Number of interacting (coupled, concurrent) components (+ feedback)
2. Variety of views at different levels of abstraction
3. Variety of components (software/hardware, continuous/discrete)
4. Uncertainty

→ **focus** on 1 and 3
Proposed solution: 

*multi-paradigm* modelling and simulation

1. Different *levels of abstraction*

2. Mixing different *formalisms*

3. Modelling syntax and semantics of classes of models (formalisms): *meta-modelling*

   All are closely related to *model transformation*
Multi-paradigm dimensions

- Meta Level
- Abstraction Level
- Formalism
System under study: $T, l$ controlled liquid
Detailed (continuous) view, ALG + ODE formalism

Inputs (discontinuous → hybrid model):
- Emptying, filling flow rate $\phi$
- Rate of adding/removing heat $W$

Parameters:
- Cross-section surface of vessel $A$
- Specific heat of liquid $c$
- Density of liquid $\rho$
- Temperature of influent $T_{in}$

State variables:
- Temperature $T$
- Level of liquid $l$

Outputs (sensors):
- $is_{low}, is_{high}, is_{cold}, is_{hot}$

\[
\begin{align*}
\frac{dT}{dt} &= \frac{1}{l} \left[ \frac{W}{c_p A} - \phi(T - T_{in}) \right] \\
\frac{dl}{dt} &= \phi \\
\text{is}_\text{low} &= (l < l_{low}) \\
\text{is}_\text{high} &= (l > l_{high}) \\
\text{is}_\text{cold} &= (T < T_{cold}) \\
\text{is}_\text{hot} &= (T > T_{hot})
\end{align*}
\]
High-level (discrete) view, FSA formalism
Levels of abstraction/views: trajectories

**Continuous State Trajectory**

**Discrete State Trajectory**

- **level**: full, l_in_between, empty
- **temperature**: cold, T_in_between, hot
- **sensor types**: is_full, is_empty, is_cold, is_hot
- **actions**: fill, heat, on, off
Multi-paradigm dimensions: abstraction/formalism
Multi-paradigm dimension: abstraction
Levels of abstraction/views: morphism

model \( \rightarrow \) M_t \( \rightarrow \) M_d

simulation

trajectory \( \rightarrow \) traj_t \( \rightarrow \) traj_d

detailed (technical) level \( \rightarrow \) abstraction \( \rightarrow \) abstract (decision) level
Multi-paradigm dimensions: formalisms
Forrester System Dynamics model of Predator-Prey

2-species predator-prey system
Causal Block Diagram model of Harmonic Oscillator
Petri Net model of Producer Consumer

[Diagram of Petri Net model showing places and transitions like P.Calculating, Wait4Cons, Produce, Buffer, Put in Buffer, Buffer-p, C.Calculating, Rem.from buffer, Consume, and Wait4Prod.]
Statechart model of Producer Consumer
Event Scheduling + DAE model of a Train
Multi-paradigm dimensions: meta
What is Meta-modelling?

- A meta-model is a model of a modelling formalism.

- A meta-model is itself a model. Its syntax and semantics are governed by the formalism it is described in. That formalism can be modelled in a meta-meta-model.

- As a meta-model is a model, we can reason about it, manipulate it, . . . In particular, properties of (all models in) a formalism can be formally proven.

- Formalism-specific modelling and simulation tools can automatically be generated from a meta-model (AToM³ A Tool for Multi-formalism Meta-Modelling).
- Formalisms can be tailored to specific needs by modifying the meta-model (possibly through inheritance if specializing).
  ⇒ Building domain/applicatin specific, possibly graphical modelling and simulation environments becomes affordable.

- Semantics of new formalisms through extension or transformation (multi-formalism).
FSA model of Even Binary Number recognizer
ER model of the FSA formalism syntax (meta-model)
ER formalism + constraints (OCL/Python)

# check for unique input labels (FSA)
for transition1 in state.out_connections:
    for transition2 in state.out_connections:
        if transition1 != transition2:
            if transition1.in == transition2.in:
                return("Non-determinism: input " + transition1.in)
ER model of the ER formalism (meta-meta-model)
Meta-meta-...

Meta-model processor

- create
- delete
- verify (local, global)

Meta-model

- create-delete-verify (local, global)

Meta-meta model

a model of a class of models (the formalism MF)
semantics within formalism MMF
describes: structure and constraints

Meta-model

a model in formalism MF

meta-model user

input

Model

a model in formalism F

meta-model user

input

Model

a model of a class of models (the formalism F)
semantics within formalism MF
describes: structure and constraints
Causal Block Diagram Semantics?
Causal Block Diagram Denotational Semantics

\[ \begin{align*}
\frac{dx}{dt} &= y & x(0) &= 0 \\
\frac{dy}{dt} &= -Kx & y(0) &= 1 \\
K &= 1
\end{align*} \]
FSA model Operational Semantics ?

Init

End_0

End_1

0

1

0

1

0
Simulation steps
Current State

Rule 1

input 0

Rule 2

input 1

Rule 2

end of input

Final Action

"Accept Input"
Graph Grammar model of FSA OpSem

Rule 1 (priority 3)

\[ \text{Rule 1 (priority 3)} \]

Locate Initial Current State

\[ \text{Locate Initial Current State} \]

Rule 2 (priority 1)

\[ \text{Rule 2 (priority 1)} \]

State Transition

\[ \text{State Transition} \]

Rule 3 (priority 2)

\[ \text{Rule 3 (priority 2)} \]

Local State Transition

\[ \text{Local State Transition} \]
Model Transformation meta-specification

- a model in formalism ER
  - meta-model
  - transformation meta-model
  - user input: create, delete, verify (local, global)
  - a model of a class of models (the formalism NFA)
  - semantics within formalism ER
  - model transformer
  - a model in formalism NFA
  - (multi-formalism) model transformer = meta-model processor
  - a model in formalism F
  - meta-model
  - a model in formalism MF
  - meta-model
  - a model in formalism MF
  - (multi-formalism) model transformer = meta-model processor
  - a model in formalism F
  - meta-model
  - a model in formalism MF
  - meta-model
  - a model in formalism MF
  - (multi-formalism) model transformer = meta-model processor
  - a model in formalism F

- create, delete, verify (local, global)
- a model of a class of models (the formalism F)
- semantics within formalism MF
- describes: structure and constraints
Timed Automata model of a Traffic Light + codegen
Generated Application
Model Transformation Uses (1)

- Code generation
- Operational Semantics (reference simulator)
- Denotational Semantics

May model transformation as Graph Grammar
FSD Denotational Semantics?

2-species predator–prey system
FSD denotational semantics in terms of DAE

- Semantics of “level” block:

\[ \frac{d \text{level}}{dt} = BR - DR. \]

- Semantics of “algebraic” block: algebraic relationship between block’s I/O signals

- Semantics of the full model: set of components’ semantics equations.
Formalism Transformation

Causal Block Diagram → DAE a-causal set → DAE causal sequence (sorted) → Difference Equations

System Dynamics → Transfer Function → DAE causal set → state trajectory data (observation frame)
Formalism transformation uses (2)

- Add new formalisms without much effort (only $\Delta$).
- Re-use lower level modelling/simulation environment.
- Answer questions at “optimal” level.
  2. DAE: algebraic dependency cycles.
  3. ALG + ODE: linear ?
  4. Trajectory, given initial conditions.
- Optimization possible at every level.
- Semantics of coupled multi-formalism models.
Compositional Modelling: Coupled Model (network)
Closure under Coupling/Composition: Block Diagram

Non-Causal:

\[ A.y = B.x \]

Causal:

\[ B.x := A.y \]
Closure under Coupling/Composition: non-causal Bond Graph

\[ A.p.\text{effort} = B.p.\text{effort} \]

\[ A.p.\text{flow} = B.p.\text{flow} \]
Closure under Coupling/Composition: Discrete Event

- *schedule* ARRivals
- *resolve* collisions
Closure under Coupling/Composition: Petri Net

- Transitions $A.ty, B.tx$ are used as ports
- Coupling between ports by means of a place $p$
Complex System: Coupling Different Formalisms

PaperPulp mill

Waste Water Treatment Plant

Fish Farm

Stormwater tank 1
Stormwater tank 2

System of WWTP and Stormwater tanks (DEVS)

Influent

Activated sludge unit (ASU)

Effluent

Recycle (return) flow

Input/Output function

Input function

Output function

algae

fish

MESM 2002, 29 September, Sharjah U.A.E  

hv@cs.mcgill.ca  

A Software Architecture for Multi-Paradigm Modelling
Semantics of Coupled Models

1. Super-formalism subsumes all formalisms
2. Co-simulation (coupling resolved at trajectory level)
3. Transform to common formalism
Multi-formalism coupled model: co-simulation
Co-simulation of multi-formalism coupled models

- Sub-models simulated with formalism-specific simulators.
- Interaction due to coupling is resolved at trajectory level.
  - Loss of information.
  - Questions can only be answered at trajectory level.
  - Speed and numerical accuracy problems for continuous formalisms.
  - Meaningful for discrete-event formalisms (but beware of legitimacy!).
    Basis of the DoD High Level Architecture (HLA) for simulator interoperability.
Multi-formalism coupled model: multi-formalism modelling

CoupledModel

M_{sub\_1} \rightarrow CouplingGraph \rightarrow M_{sub\_2} \rightarrow M_{sub\_3}
Formalism Transformation Graph
Multi-formalism modelling \neq co-simulation

1. Start from a coupled multi-formalism model. Check consistency of this model (e.g., whether causalites and types of connected ports match).

2. Cluster all formalisms described in the same formalism.

3. For each cluster, implement closure under coupling.

4. Look for the best common formalism in the Formalism Transformation Graph all the remaining different formalisms can be transformed to. Worst case: trajectory level (fallback to co-simulation).

5. Transform all the sub-models to the common formalism.

6. Implement closure under coupling of the common formalism.
The Future . . .

- Formalism Transformation (FTG)
- Graph Grammars *models* for all Transformations
- Simulator Meta-specification (reference implementation)
- Model exchange (DTD from meta-model, XML from model)
- Variations (flavours) of formalisms (syntax and semantics)
- Automatic equivalence proofs (bi-simulation)
- Meta-modelling Environment (ATOM$^3$)