MDE Assignment 3: (Visual) Concrete Syntax

Joeri Exelmans
contact: joeri.exelmans@uantwerpen.be

October 22, 2024

1 Introduction

Up until this point, in the assignments, we haven't been very explicit about the distinction between (textual / visual) concrete syntax and abstract syntax. Nevertheless, we have already encountered many different concrete syntaxes, as summarized in Table 1.

When creating a (meta-)model, so far, we have always used some textual syntax. It doesn't have to be this way: in model-driven engineering, many tools, such as AToMPM, Itemis, MetaEdit+, StateMate, TAPAAL, ... let the user create models using a visual syntax. Meta-modeling tools, such as AToMPM, further allow the user to define their own visual syntax (choosing icons, colors, styles, etc.). In class, a demo will be given of AToMPM.

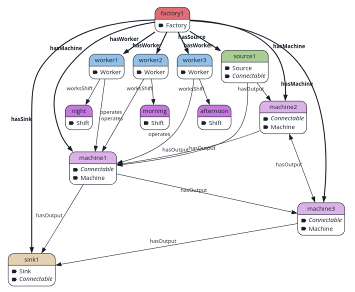
2 Assignment

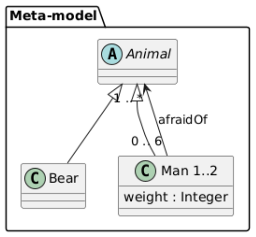
2.1 Exercises

Moody describes nine principles for designing cognitively effective notations [1]:

- **Semiotic Clarity** 1:1 correspondence between semantics constructs and graphical symbols
- **Perceptual Discriminability** visual symbols should be easily and accurately distinguishable from each other (e.g., circle and rectangle: easy vs. rectangle and square: hard)
- **Semantic Transparency** use visual symbols, colors, ... that suggest their meaning (e.g., green = good/safe, red = bad/danger)
- **Manageable Complexity** use mechanisms to keep complexity of diagrams manageable (e.g., use visual containment for hierarchical relationships, and edges for hierarchy-crossing or cyclical relationships)
- **(Cognitive Integration)** mechanisms to help the reader assemble information from separate diagrams into a coherent mental representation of the system
- **Visual Expressiveness** use the full range (extremes) and capacities of visual variables (e.g., black and white is better than different shades of gray) Concepts that have (sufficiently) different meaning, should

Formalism	Textual Concrete Syntax	Visual Concrete Syntax

Refinery	`class Factory {`
	`    contains Machine[1..*] hasMachine`
	`    contains Worker[1..*] hasWorker`
	
	`    contains Source[1] hasSource`
	`    contains Sink[1] hasSink`
	`}`
	
	`abstract class Connectable {`
	`    Connectable[0..2] hasOutput opposite hasInput`
	`    Connectable[0..2] hasInput opposite hasOutput`
	`}`
	
	`class Source extends Connectable.`
	`class Sink extends Connectable.`
		 factory1
		Factory
		hasMachine
		hasWorker
		hasSource
		hasSink
		worker1
		worker2
		worker3
		Worker
		source1
		Source
		Connectable
		worksShift
		operates
		night
		morning
		afternoon
		Shift
		machine1
		machine2
		machine3
		Machine
		hasOutput
		sink1
		Sink
		(non-editable)

PlantUML	`package "Meta-model" {`
	`class "Bear " as 00000000_0000_0000_0000_00000000048a {`
	`}`
	`abstract class "Animal " as 00000000_0000_0000_0000_00000000046d {`
	`}`
	`class "Man 1..2" as 00000000_0000_0000_0000_000000000498 {`
	`  weight : Integer`
	`}`
	
	`00000000_0000_0000_0000_00000000046d <|--`
	`00000000_0000_0000_0000_00000000048a`
	`00000000_0000_0000_0000_00000000046d <|--`
	`00000000_0000_0000_0000_000000000498`
	
	`00000000_0000_0000_0000_000000000498 "0 .. 6" --> "1 .. *"`
	`00000000_0000_0000_0000_00000000046d : afraidOf`
	`}`
	(in our case: auto-generated)
		 Meta-model
		Animal
		Bear
		Man 1..2
		weight : Integer
		afraidOf
		1 ..*
		0 .. 6
		(non-editable)

muMLE: Class Diagrams (CD)	meta-model CD-syntax
	`abstract class Animal`
	
	`class Bear (Animal)`
	
	`class Man [1..2] (Animal)`
	
	meta-model OD-syntax
	`Animal:Class {`
	`    abstract = True;`
	`}`
	
	`Bear:Class`
	`:Inheritance (Bear -> Animal)`
	
	`Man:Class {`
	`    lower_cardinality = 1;`
	`    upper_cardinality = 2;`
	`}`
	`:Inheritance (Man -> Animal)`
	two concrete syntaxes, one model
		N/A

muMLE: Object Diagrams (OD)	meta-model OD-syntax
	`Animal:Class {`
	`    abstract = True;`
	`}`
	
	`Bear:Class`
	`:Inheritance (Bear -> Animal)`
	
	`Man:Class {`
	`    lower_cardinality = 1;`
	`    upper_cardinality = 2;`
	`}`
	`:Inheritance (Man -> Animal)`
	
	model OD-syntax
	`george:Man {`
	`    weight = 15;`
	`}`
	`billy:Man {`
	`    weight = 100;`
	`}`
	`bear1:Bear`
	`bear2:Bear`
	`:afraidOf (george -> bear1)`
	`:afraidOf (george -> bear2)`
	one concrete syntax, two models
		N/A

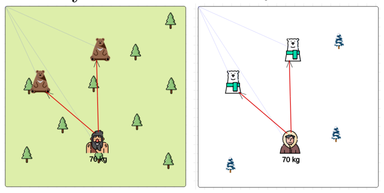
AToMPM: "woods"-formalism	N/A	summer syntax / winter syntax  *two concrete syntaxes, one model*

Table 1: Concrete syntaxes seen so far

look sufficiently different: use different line thickness, color, arrowhead, pattern, label font weight, . . . to make concepts easily distinguishable.

- **Dual Coding** use text to complement graphics, but not to distinguish graphical symbols
- **Graphic Economy** the number of different graphical symbols should be cognitively manageable
- **(Cognitive Fit)** use different visual dialects for different tasks and audiences (e.g., intuitive symbols for novice vs. more abstract symbols for expert)

Observe the 8 cases in Figure 1. For each of the diagram(-fragments), which principle(s) of Moody are being violated? Motivate your answer.

2.2 Create your own visual concrete syntax

Now open a diagramming tool of your choice (we recommend diagrams.net), and create a diagram containing the same information as case (5) (the Refinery model), applying Moody's principles as much as you can.

For an overview of 'visual expressiveness' permitted by diagrams.net, see Table 2.

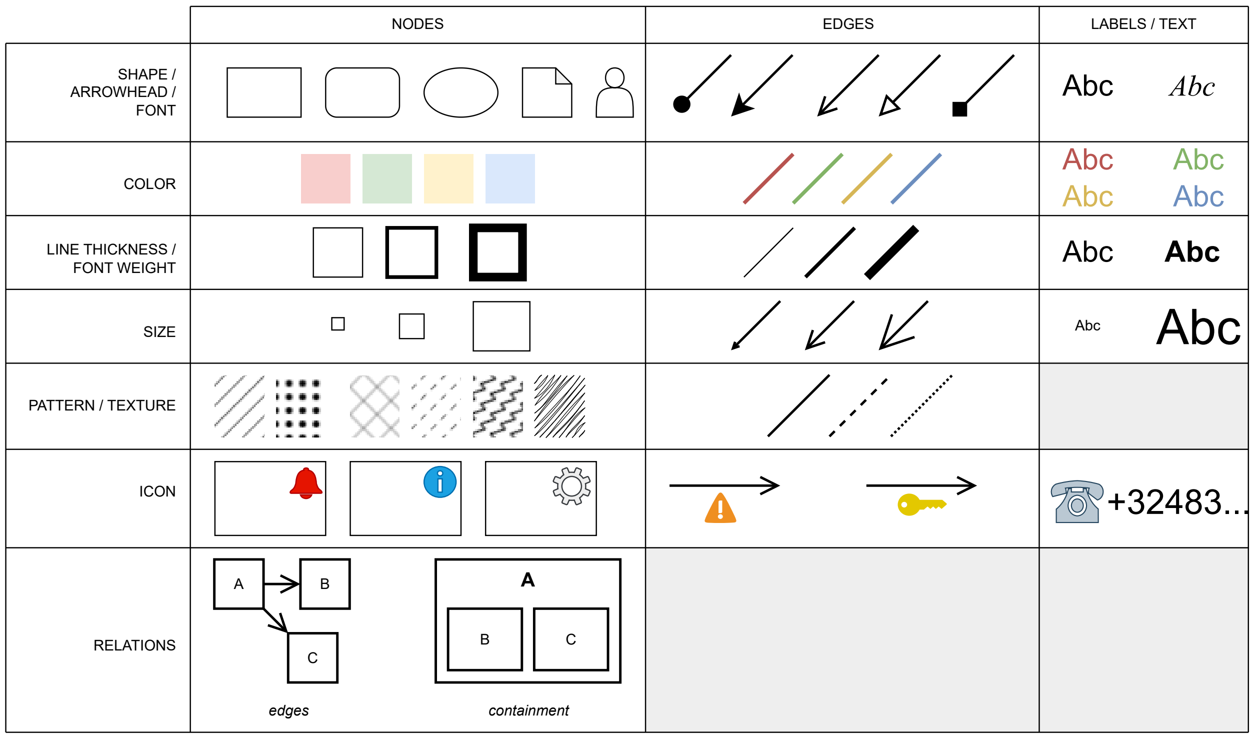

NODES
EDGES
LABELS / TEXT
SHAPE / ARROWHEAD / FONT
Abc
Abc
COLOR
Abc
Abc
Abc
Abc
LINE THICKNESS / FONT WEIGHT
Abc
Abc
SIZE
Abc
Abc
PATTERN / TEXTURE
ICON
+32483...
RELATIONS
A
B
C
edges
A
B
C
containment

Table 2: Dimensions of visual variability

3 Practical

- Students work individually.
- Submission via Blackboard.
- Deadline: Tuesday 29 October 2024, 23:59.

4 Extra material

- AToMPM “woods”-formalism (not needed to solve the assignment): `http://msdl.uantwerpen.be/people/hv/teaching/MSBDesign/assignments/demo/AToMPM-Woods-Formalism.zip` Put the ‘Woods’-directory in your ’Formalisms’-directory.

References

[1] Daniel L. Moody. The ”physics” of notations: a scientific approach to designing visual notations in software engineering. In Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel, editors, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 485–486. ACM, 2010.

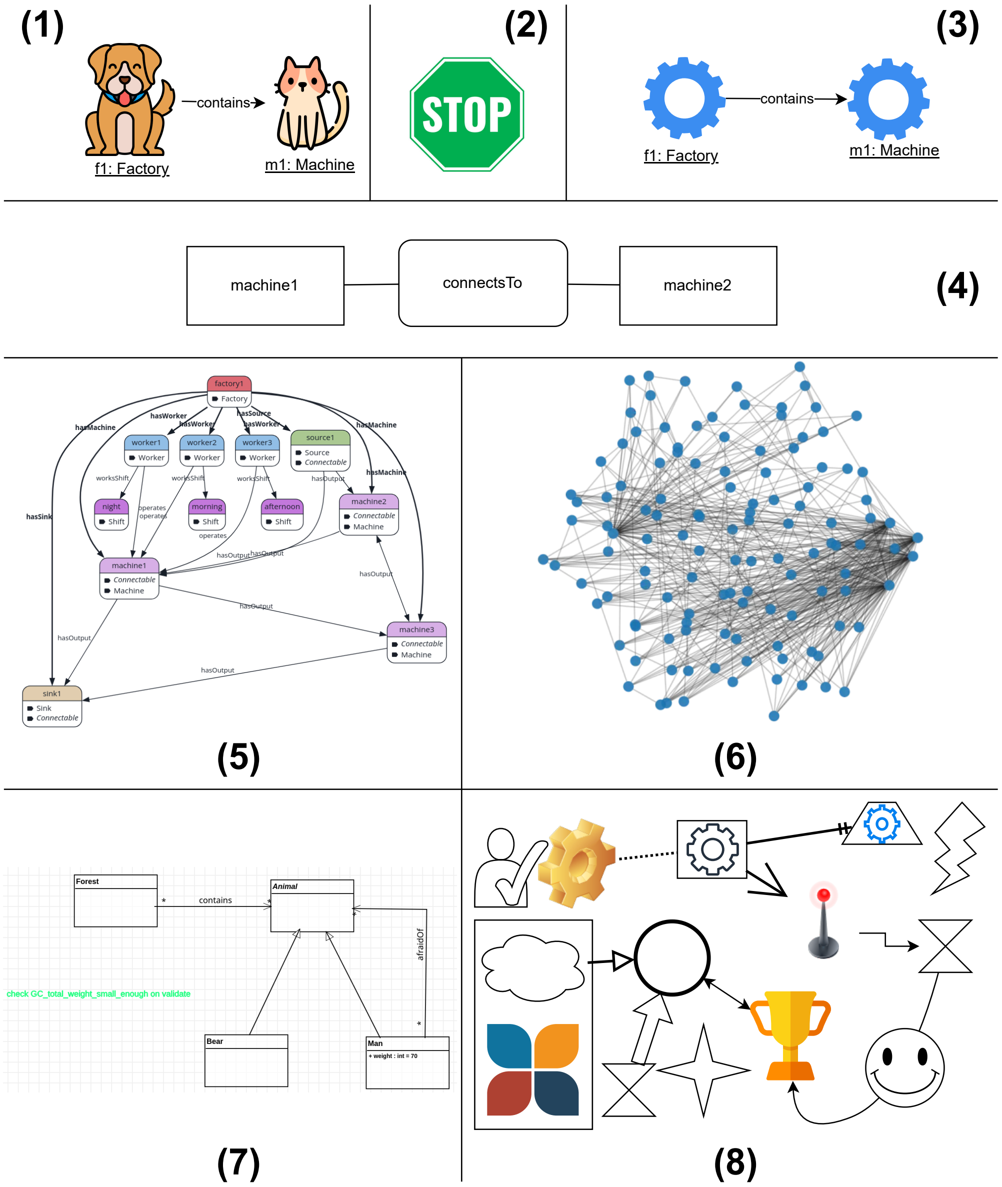


Figure 1: Bad concrete syntax design

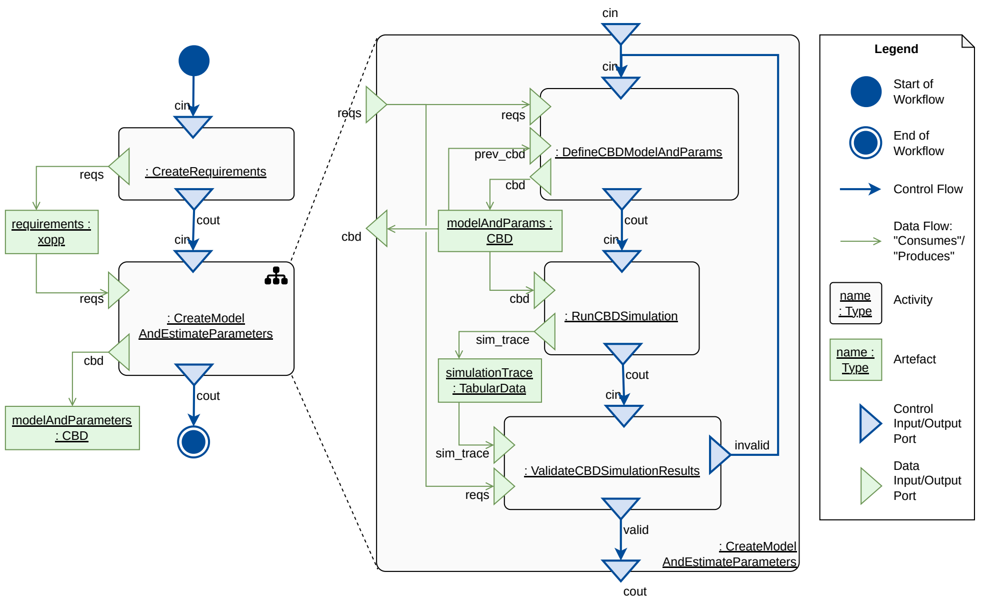


Figure 2: An example of good concrete syntax design: The FTG+PM language: control-flow elements are thick and dark-blue, data-flow elements are thin and light-green, making them easily distinguishable.