UNIVERSITETET
I OSLO

Meta-models and Grammars

Prof. Andreas Prinz

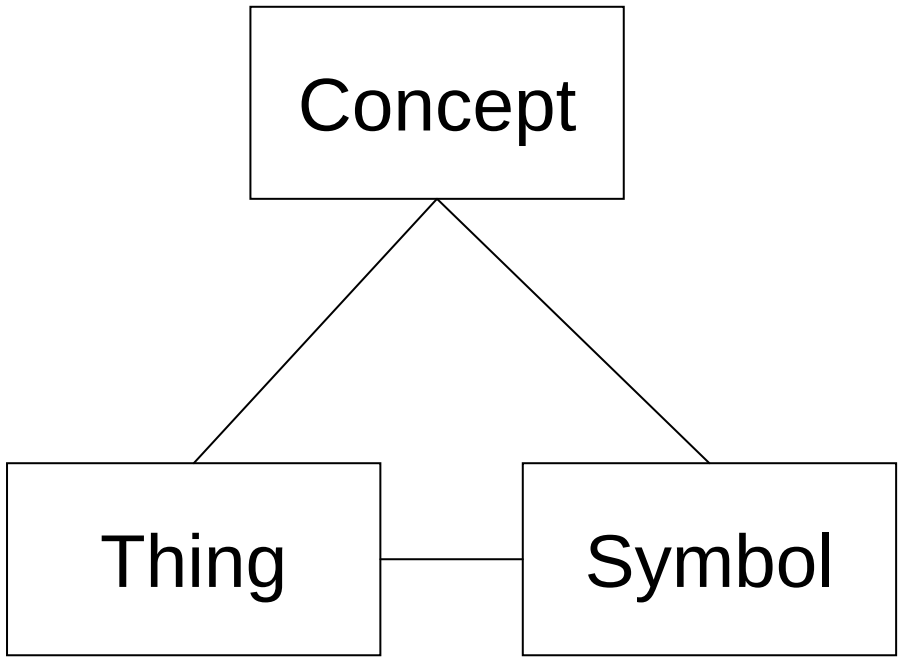
Introduction
Modelling
Meta-modelling
Compilers
Meta-models vs. Grammars
Summary

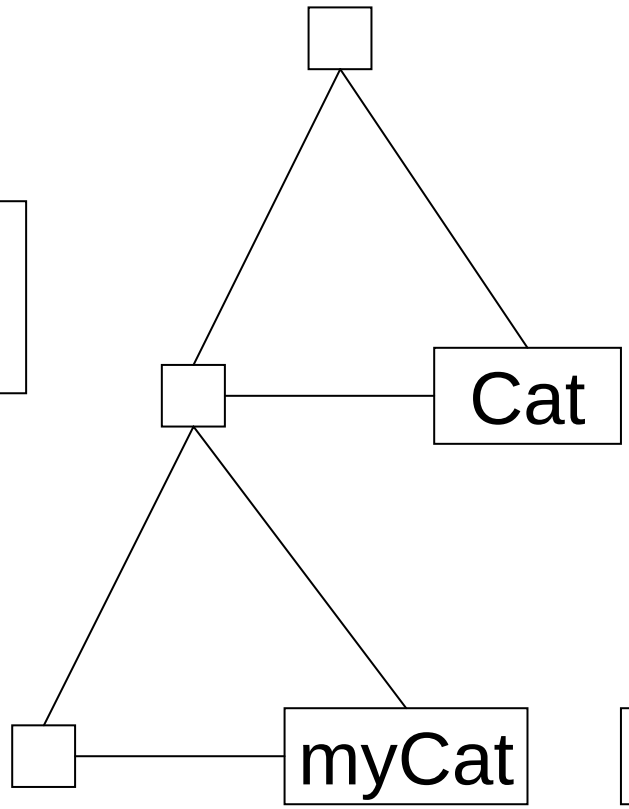
INF 5110 - 2005

What is a model?

- An abstraction of a system
- What is the best model of a cat?
  - It is a cat. But it has to be the same cat!
- A model needs a representation because it is abstract.
- A model describes several systems.
- A model is similar to a language.
  - It is applicable to some sort of systems.
  - It distinguishes between correct and wrong systems.
  - It has some (internal) structure.

The meaning triangle

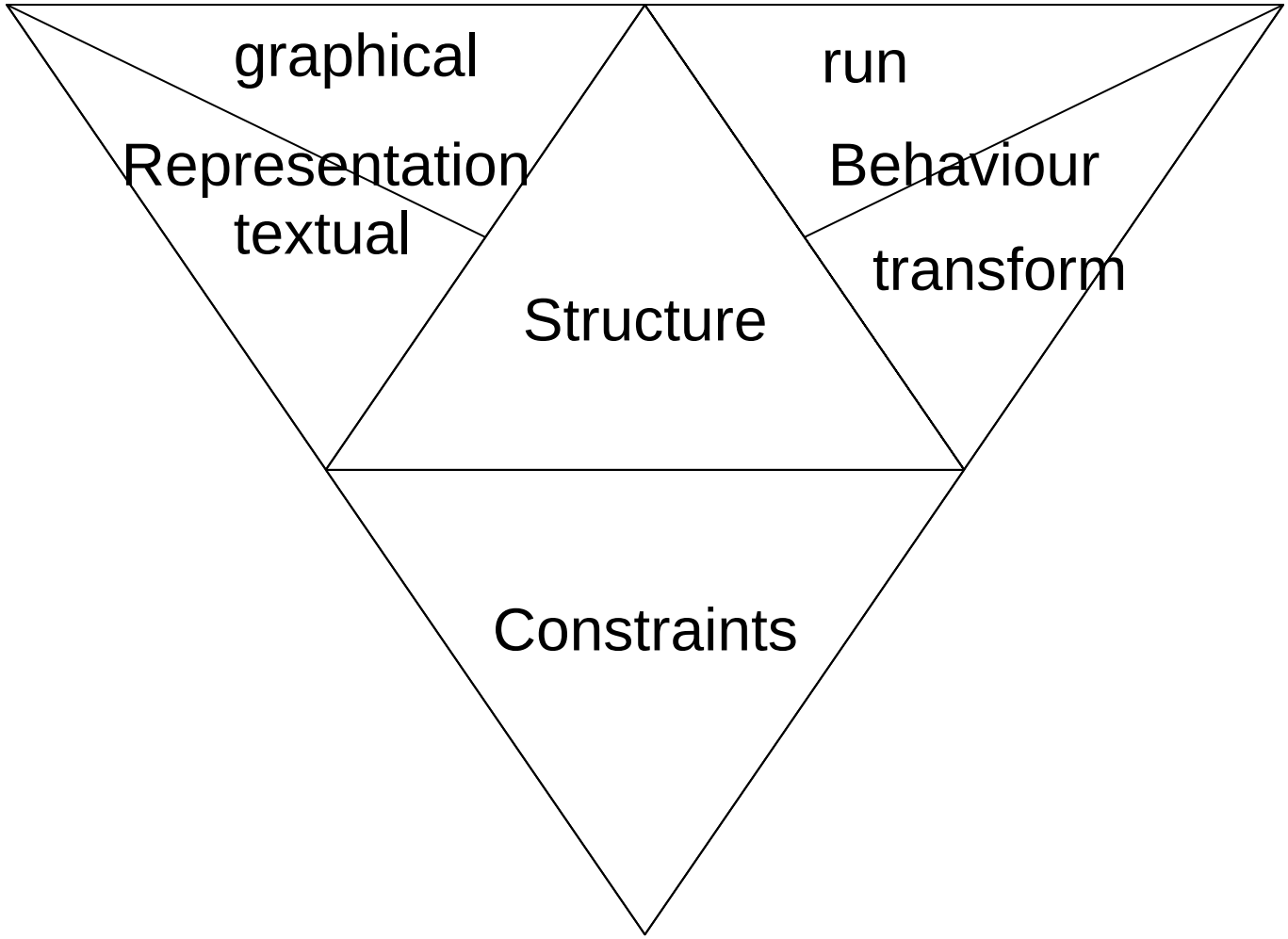




myCat:Cat

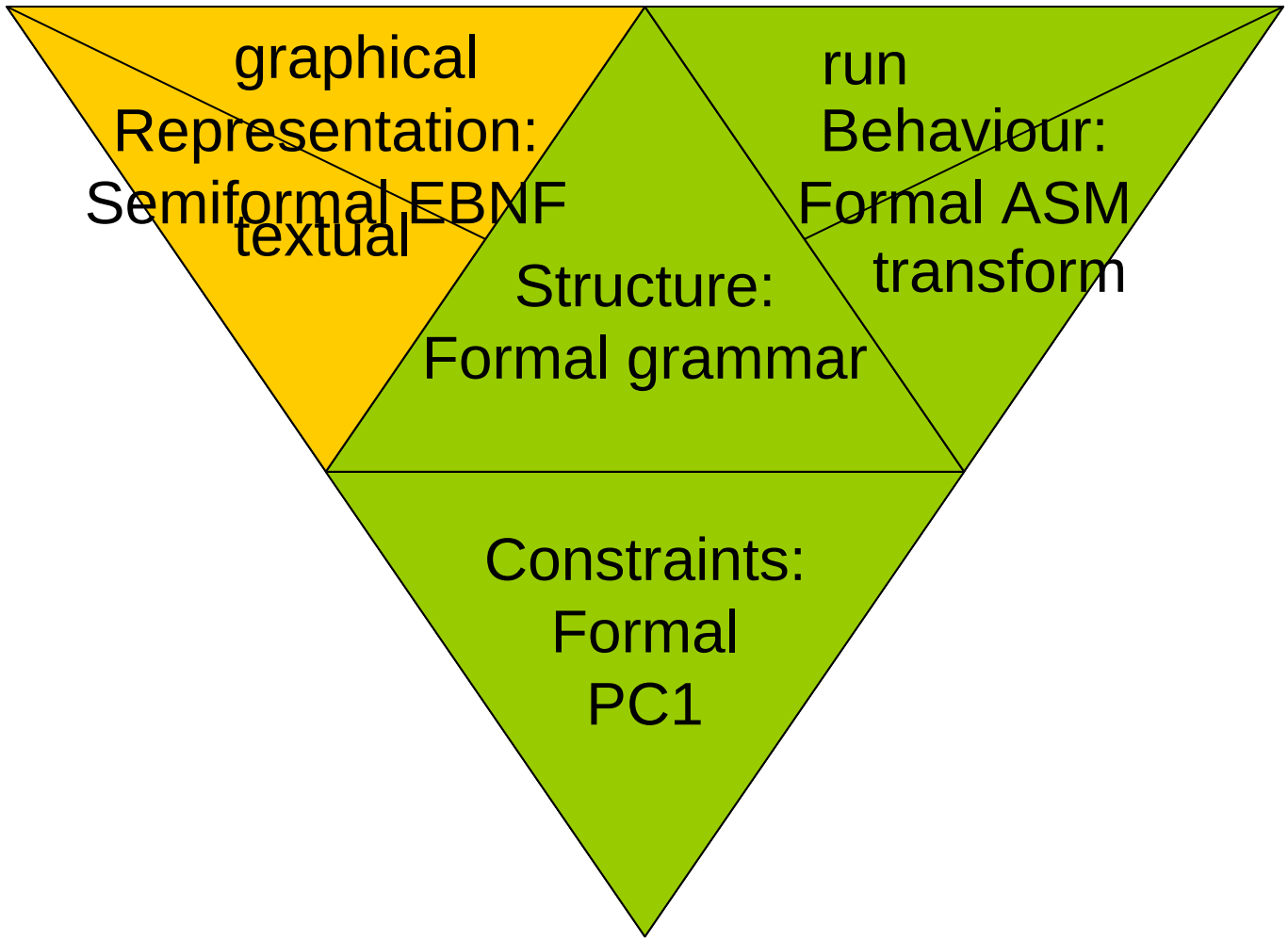
What is a meta-model?

- A description of a class of models
- Meta-models (languages) can have several aspects



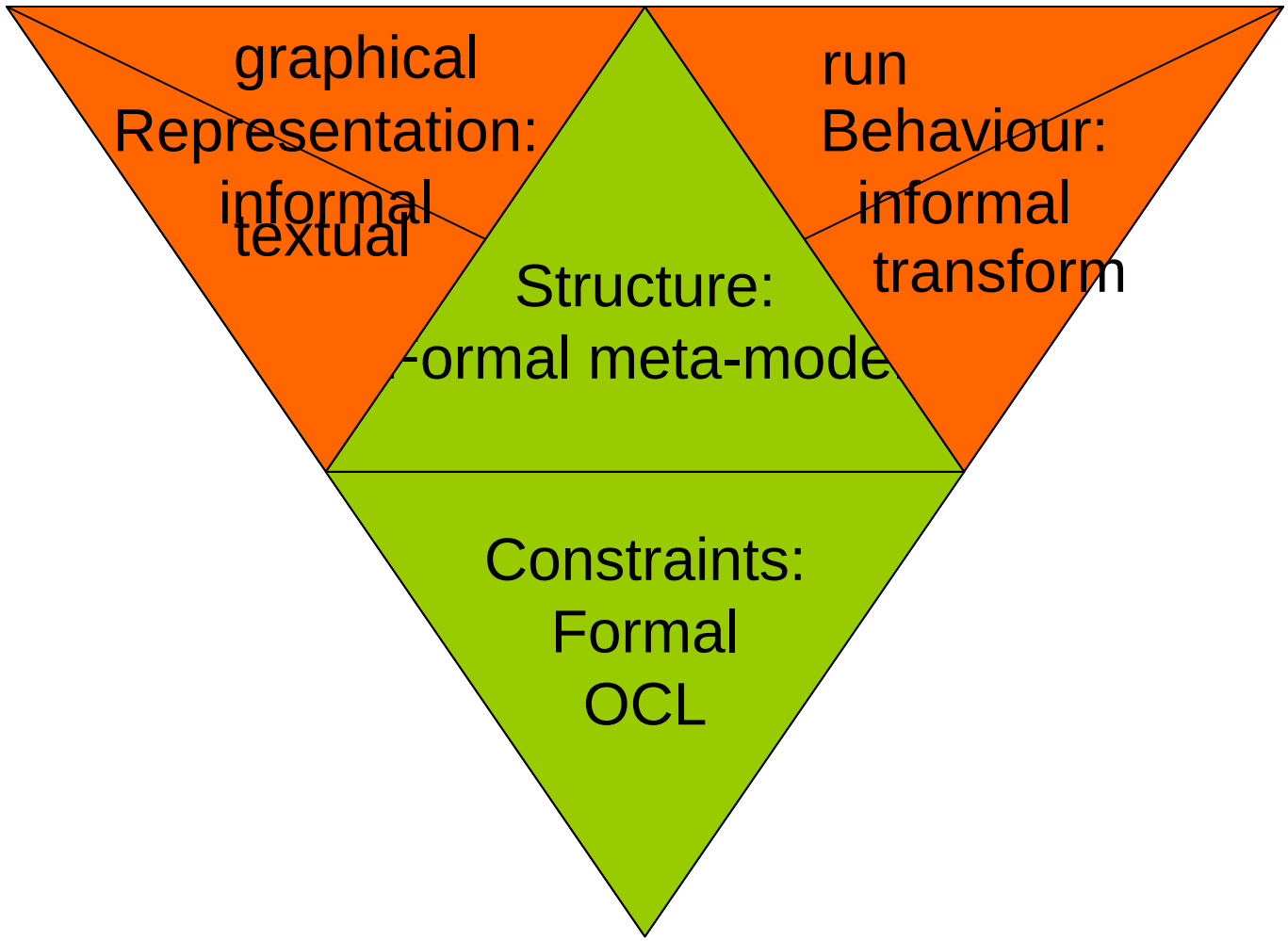
How is this done in SDL?

- SDL-2000 as standard of the SDL language.



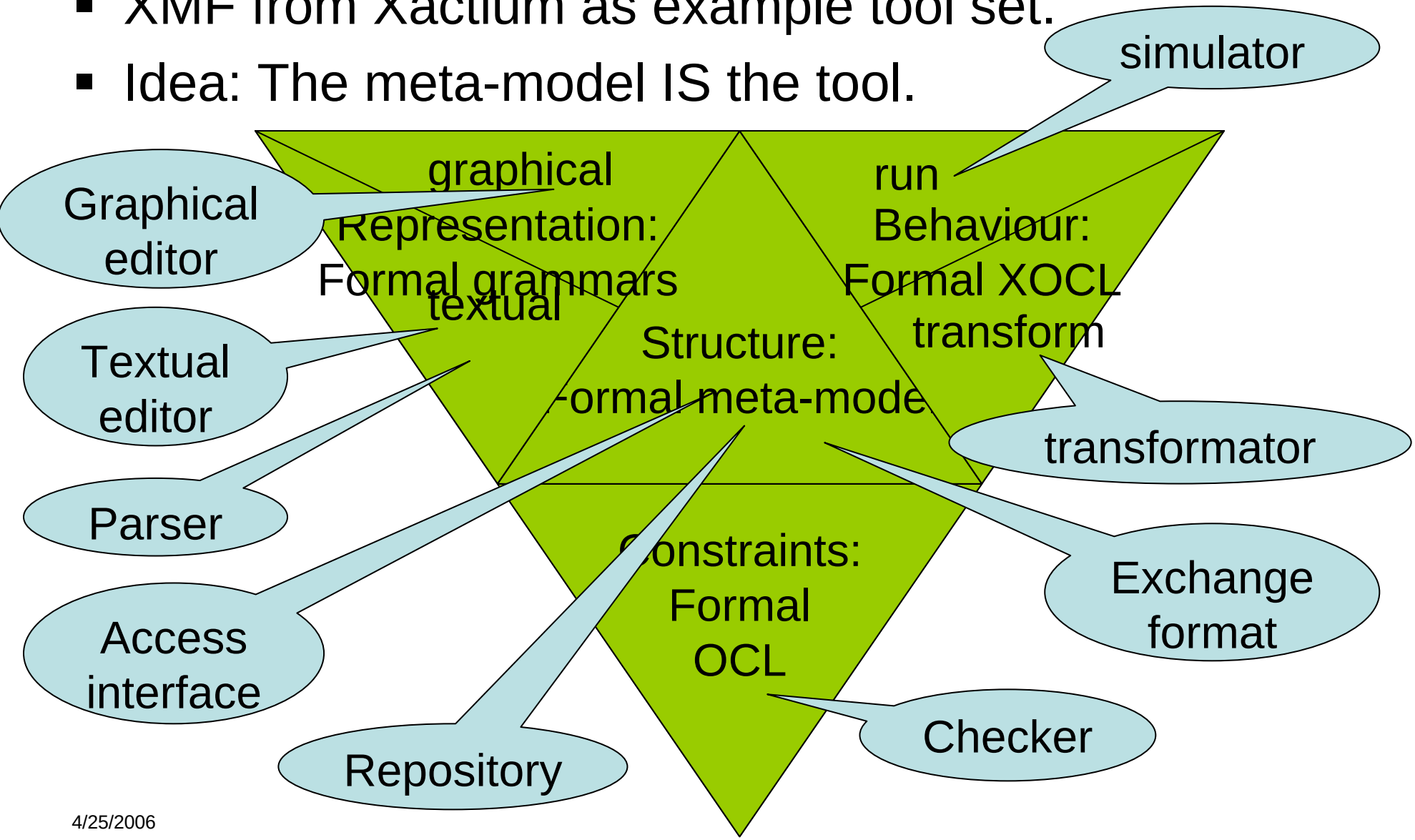
How is this done in UML?

- UML 2.0 as OMG standard.

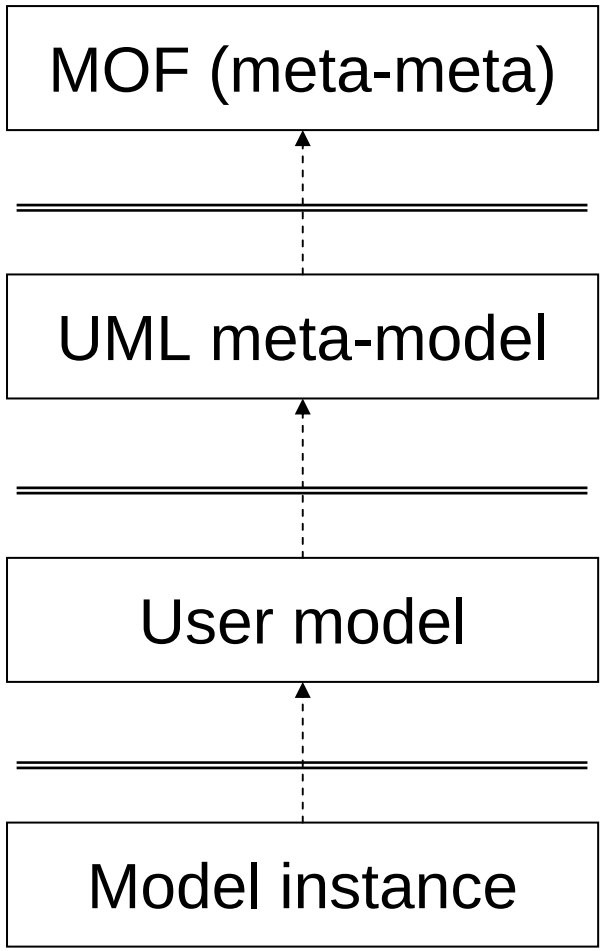


Meta-modelling and Tools

- XMF from Xactium as example tool set.
- Idea: The meta-model IS the tool.



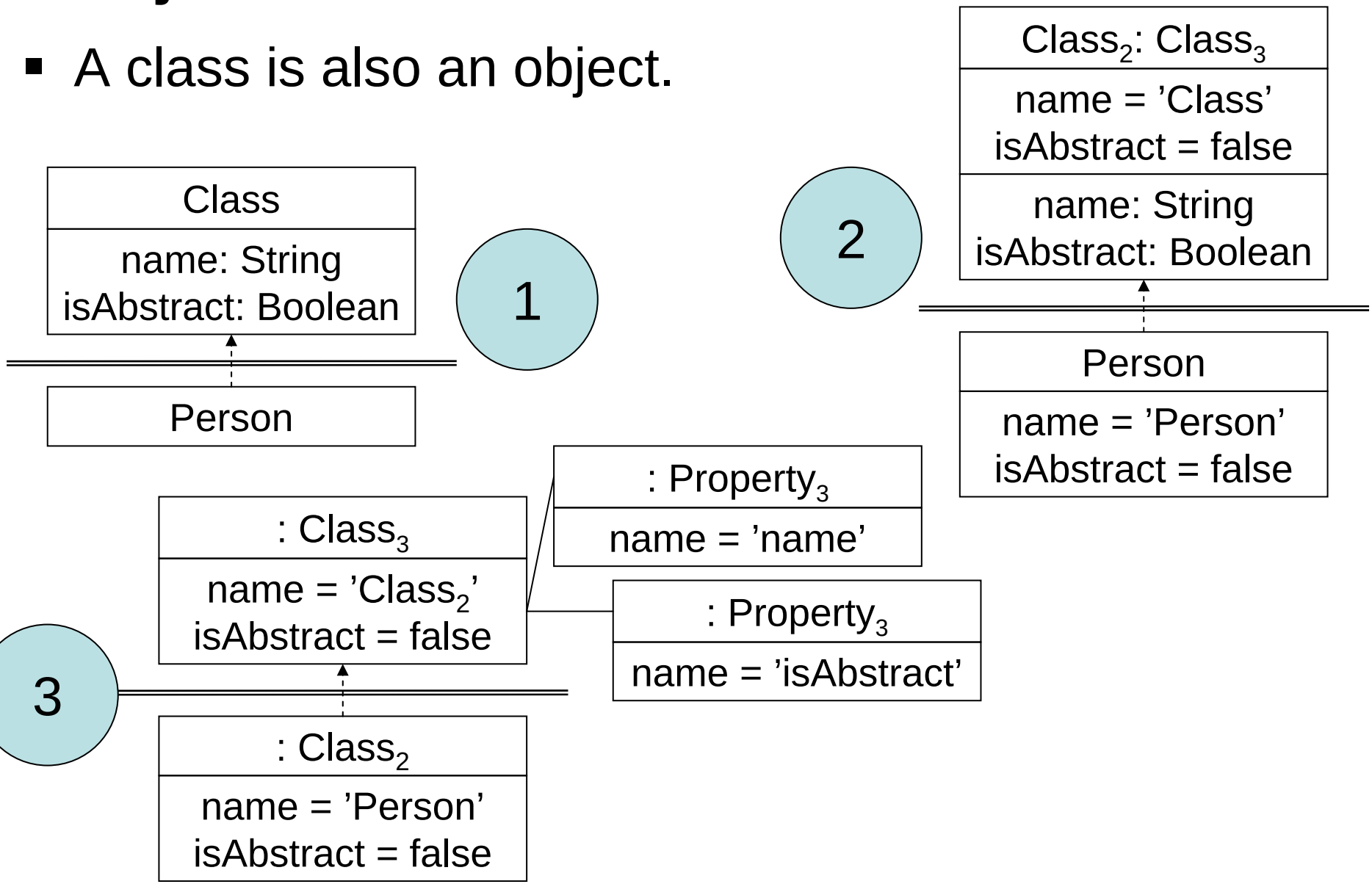
A meta-modelling architecture



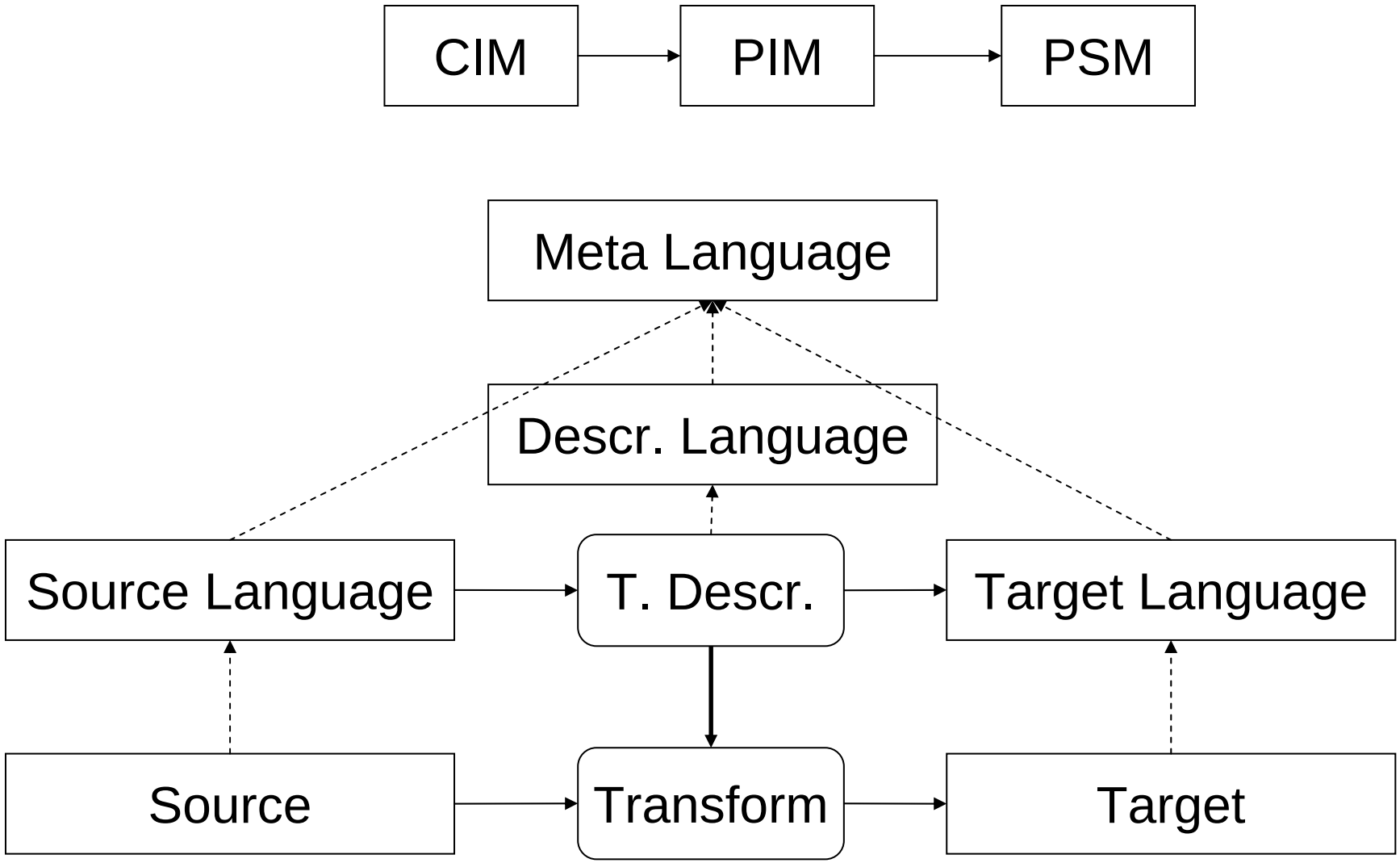
Level	Class	Property	Assoc.	OCL
2	class concept	property concept	Assoc. concept	OCL concept (meta-model)
1	specific class	specific property	specific assoc.	OCL formula
0	object of a class	slot with value	link between objects	value

Clabjects

- A class is also an object.

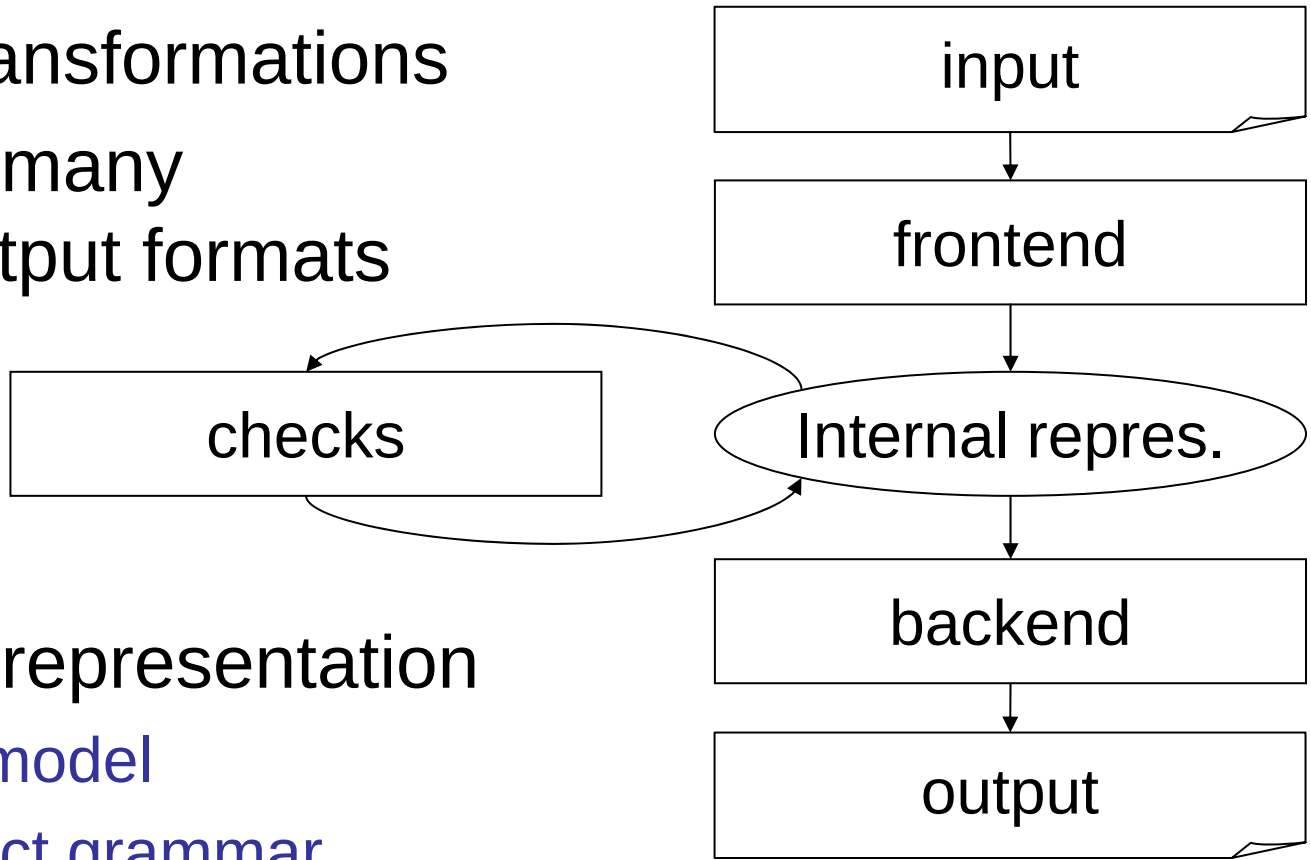


MDA in OMG context



Compilers & problems

- Graphical languages
- Domain specific languages
- Many transformations
- Solved: many input/output formats



- Internal representation
  - Meta-model
  - Abstract grammar

Meta-models versus grammars

- Advantages of grammars
  - Strong mathematical basis
  - Tree-based
  - Trees can be extended into general graphs
  - Several advanced tools available
  - Easily understandable
- Advantages of meta-models
  - Direct representation of graphs (graphics!)
  - Namespaces and relations between language elements (in particular for language transformations and combinations)
  - Object-oriented definition of oo languages
  - More problem-oriented
  - Reuse and inheritance
  - Tools allow direct handling of models (repositories)
  - Structuring possible (e.g. packages)

Example: EBNF in EBNF

BnfGrammar ::= Rule*

Rule :: NonTerminal (’::’ | ’=’ | ’::=’) Expression

Expression = Alternative | Composition | PExpression

PExpression =

Optional | AtLeastOne | Arbitrary | Symbol | ’(’ Expression ’)’

Alternative ::= PExpression ’|’ (PExpression | Alternative)

Composition ::= PExpression +

Optional ::= ’[’ Expression ’]’

AtLeastOne ::= PExpression ’+’

Arbitrary ::= PExpression ’*’

Symbol = Terminal | NonTerminal

Example: Abstract syntax of EBNF

BnfGrammar ::= Rule*

Rule :: NonTerminal Expression

Expression = Alternative | Composition | Optional |
AtLeastOne | Arbitrary | Symbol

Alternative ::= Expression +

Composition ::= Expression +

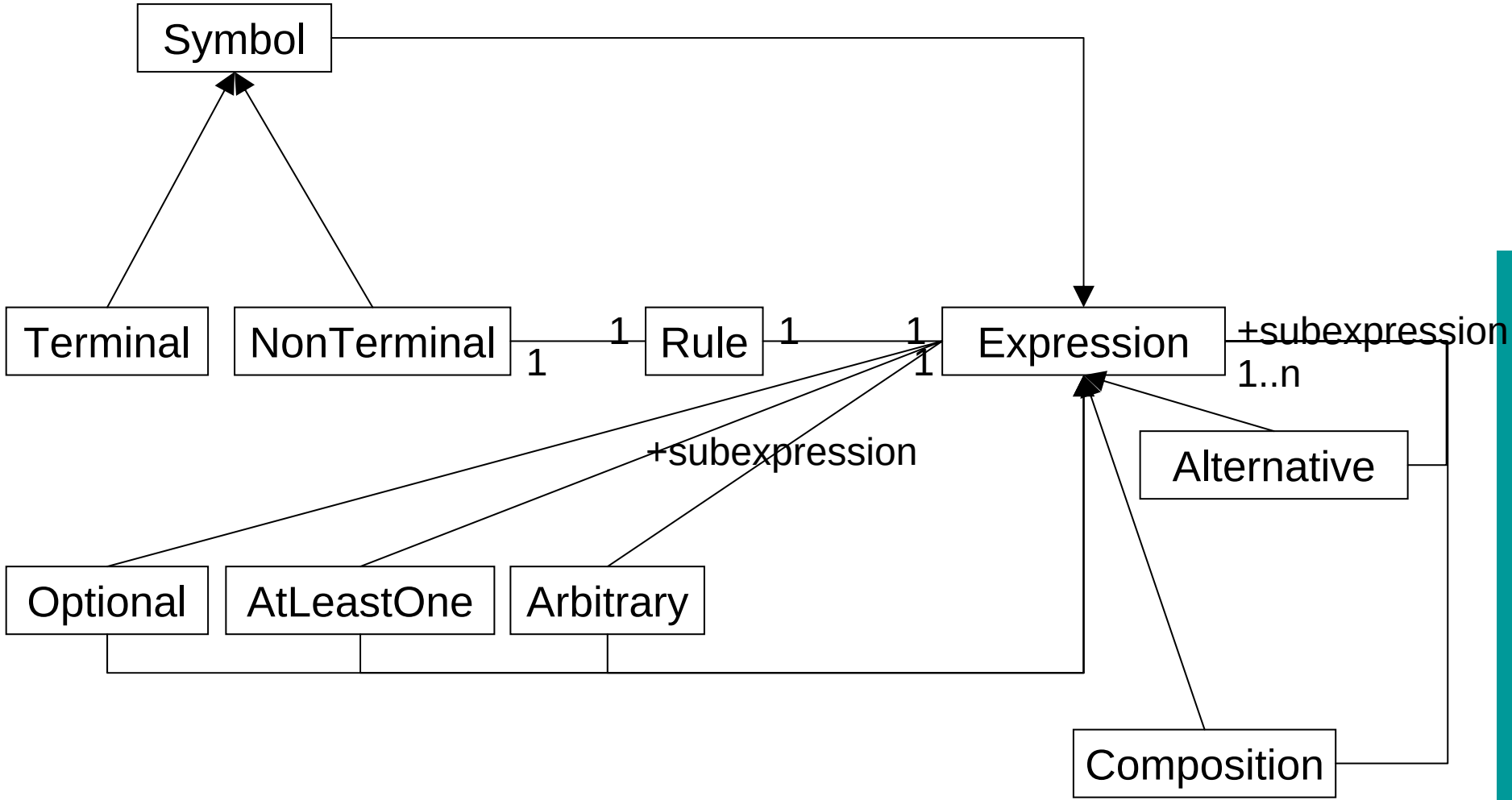
Optional ::= Expression

AtLeastOne ::= Expression

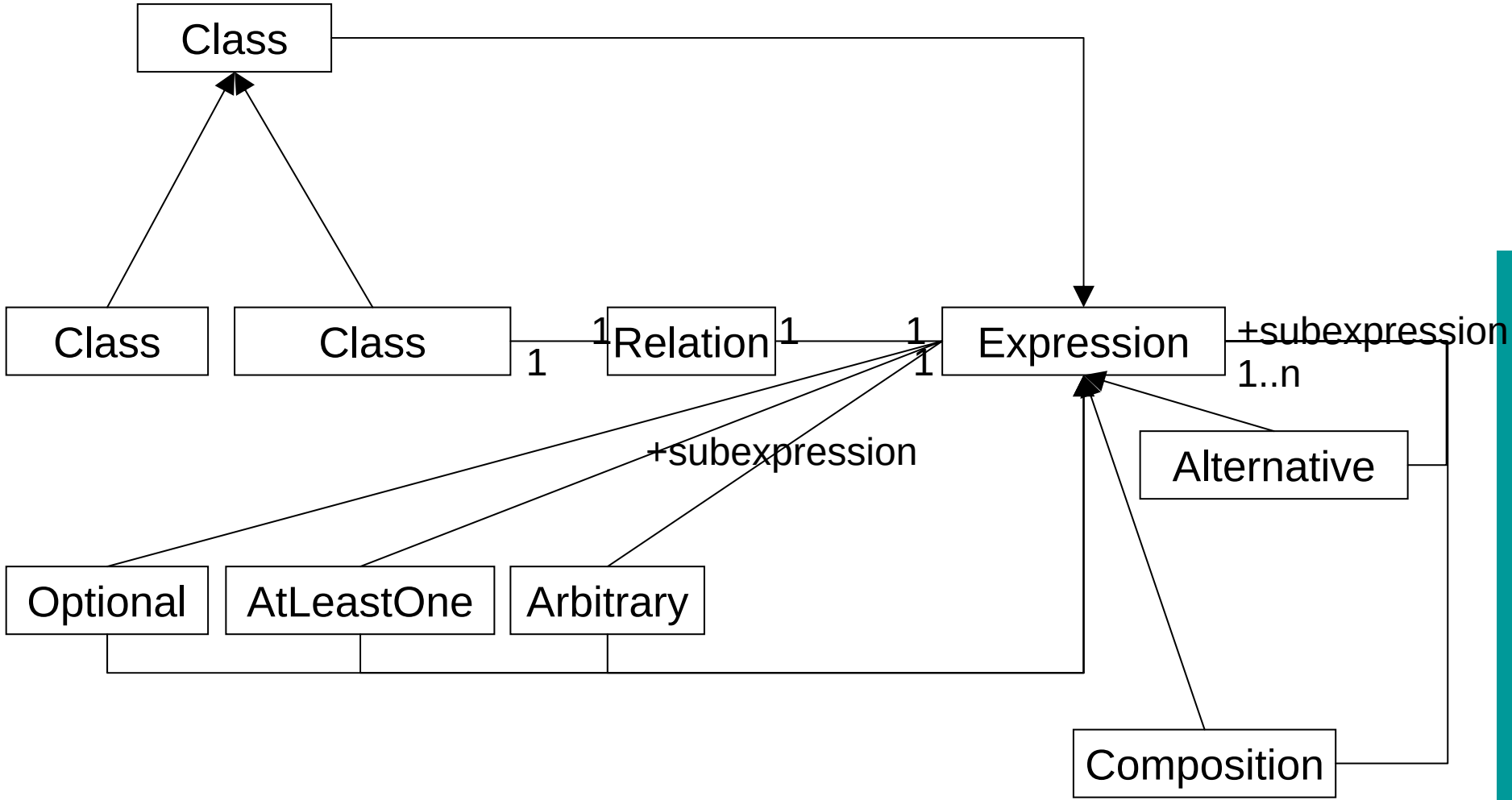
Arbitrary ::= Expression

Symbol = Terminal | NonTerminal

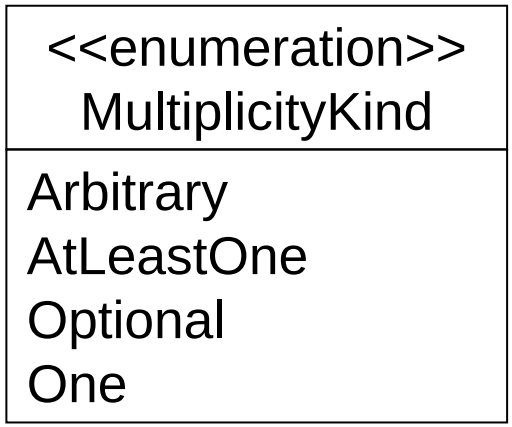
Example: simple meta-model for EBNF

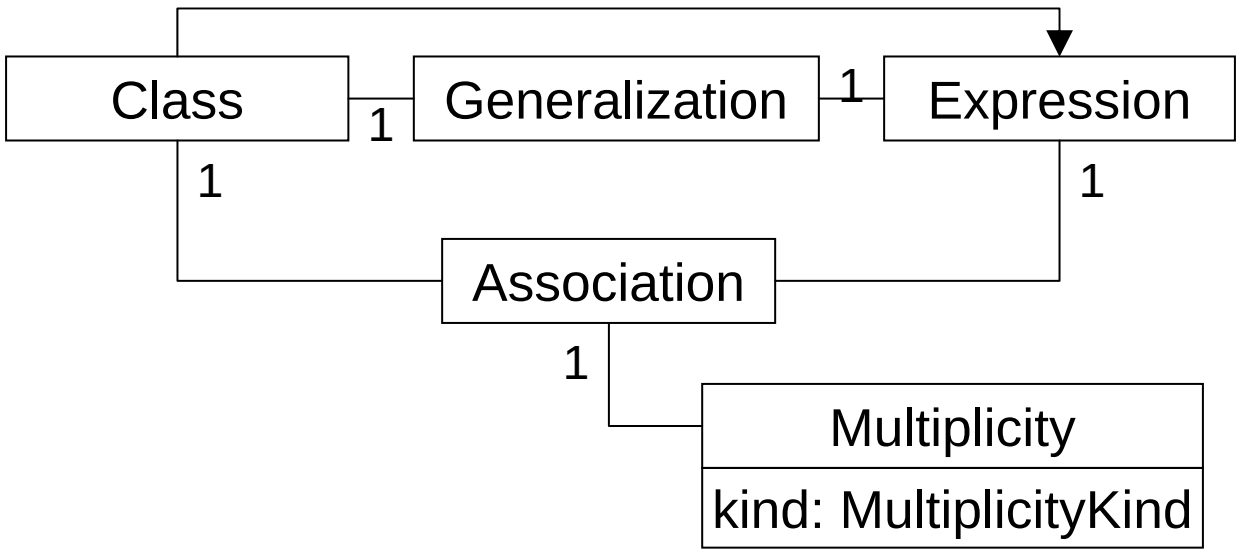


Example: reworked meta-model for EBNF

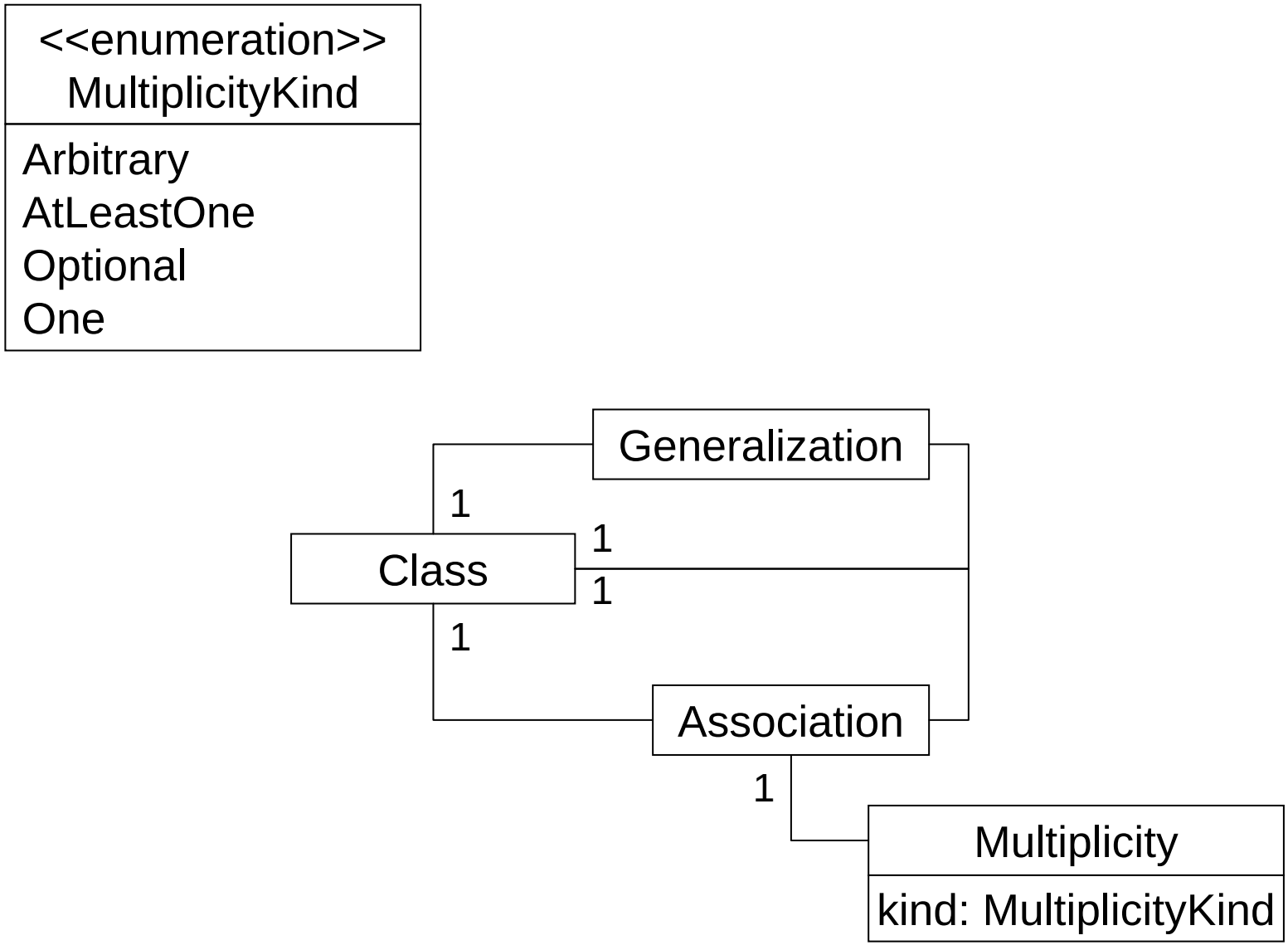


Example: reworked meta-model for EBNF





Example: final meta-model for EBNF

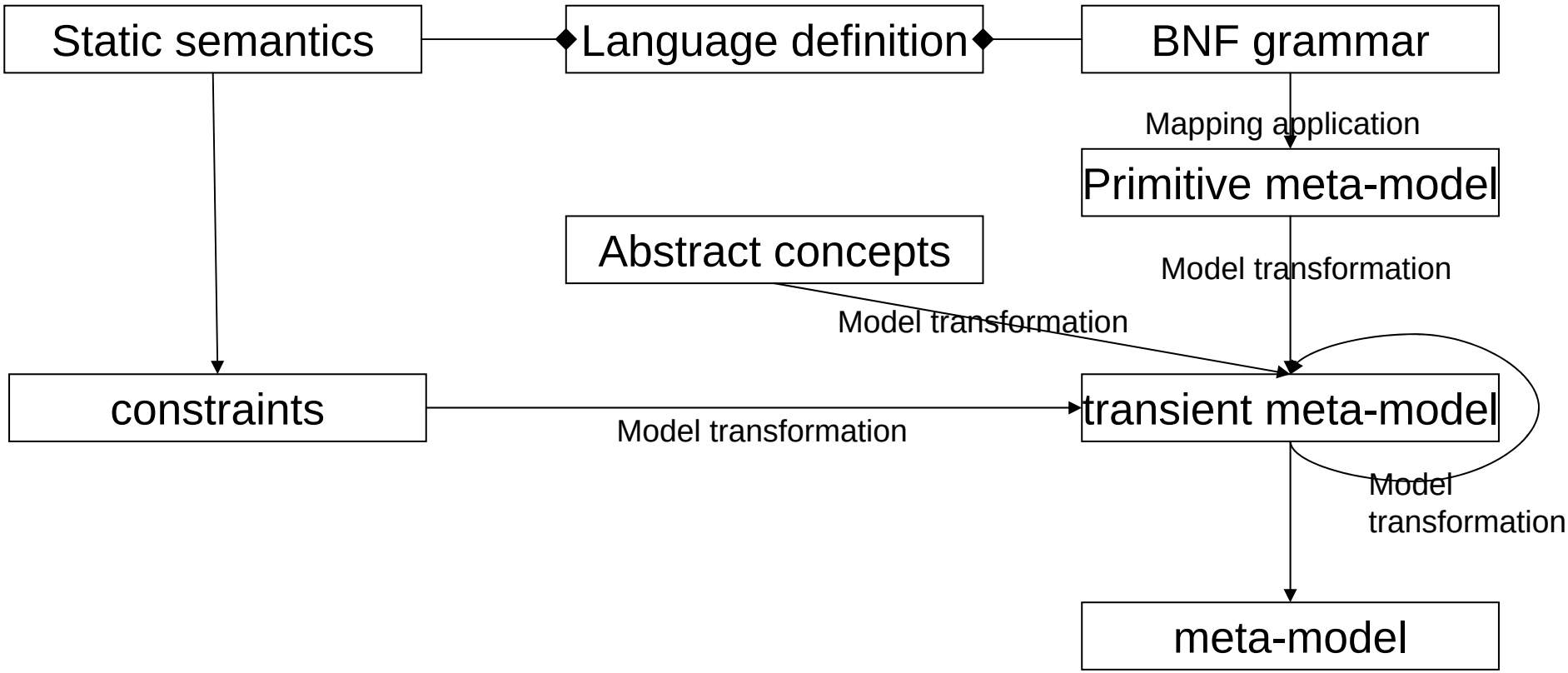


Transformation to the meta-model

1. Every symbol is represented with a class.
2. A rule with a single symbol on the rhs is represented with an association between the class representing the lhs and the rhs.
3. A rule with a composition on the rhs is represented with an association for every sub-expression.
4. A rule with an alternative on the rhs is represented with a generalization for every sub-expression.
5. A sub-expression consisting of just one symbol is represented with the symbol's class.
6. A sub-expression being a composition or an alternative is represented with a new class with new name. The composition is then handled like a rule.

From grammars to metamodels

- Nowadays languages are usually grammar-based
- How can we come to a metamodel?



Using the transformation for SDL

- Introduction of abstract concepts
  - General: namespace, namedElement, typedElement
  - Specific: parametrizedElement, bodiedElement
- Introduction of relations
  - Procedure name versus procedure definition
- Deletion of grammar artefacts
  - Referencing: identifier, qualifier
  - Names in general
  - Superfluous structuring

Summary

- Languages of the future will be defined using meta-models
- Meta-model language definitions allows
  - Direct access to the models
  - Easy exchange of representation or several of them
  - Combination of tools handling the language
  - Description of relations between languages
- An important future work is the identification of joint concepts.
  - MOF is not enough here, describe more in-depth relations
  - Have communities discuss their concepts.