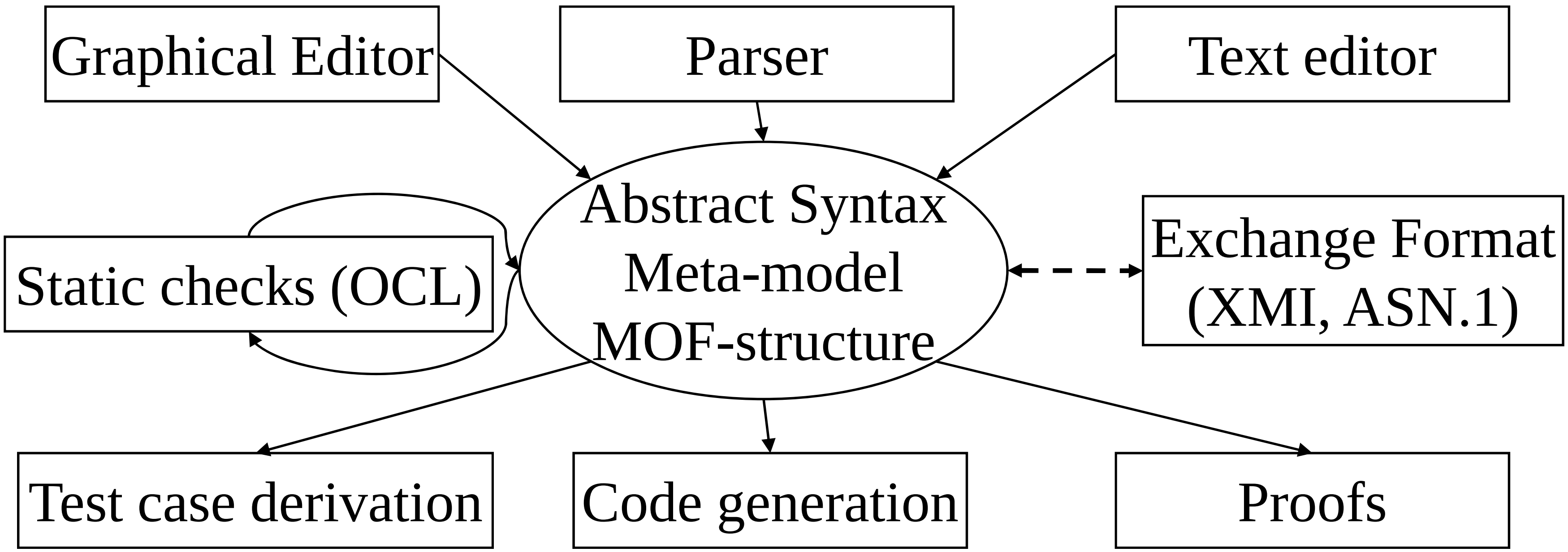
UNIVERSITY OF AGDER

# Meta-models and Grammars

Prof. Andreas Prinz

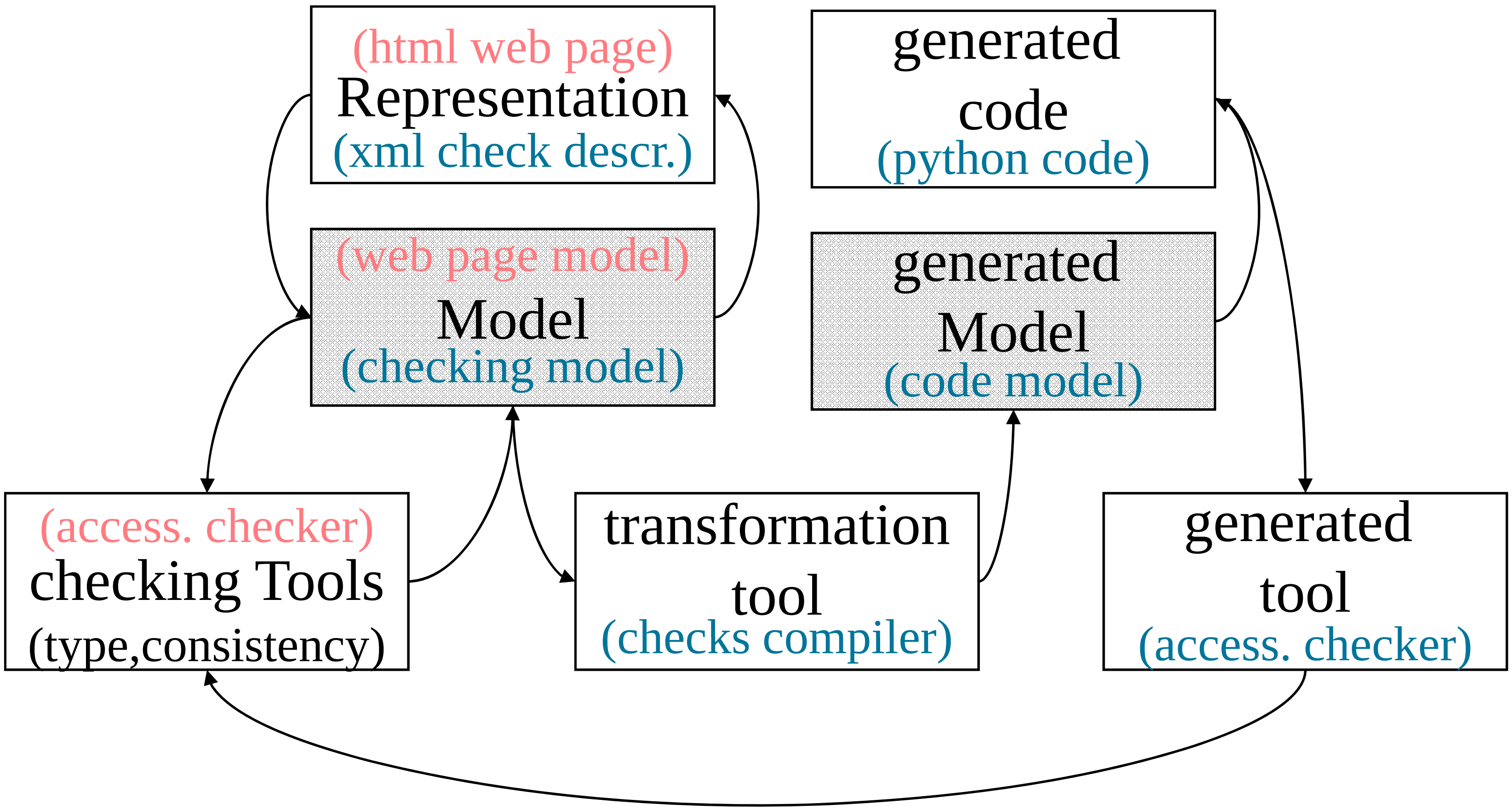
Introduction, Compilers
Modelling & Meta-modelling
Examples
Meta-models vs. Grammars
Summary

# Compilers




- Solved: many input/output formats
- Graphical / Domain specific languages, many transformations
- Internal representation: Meta-model vs. Abstract syntax

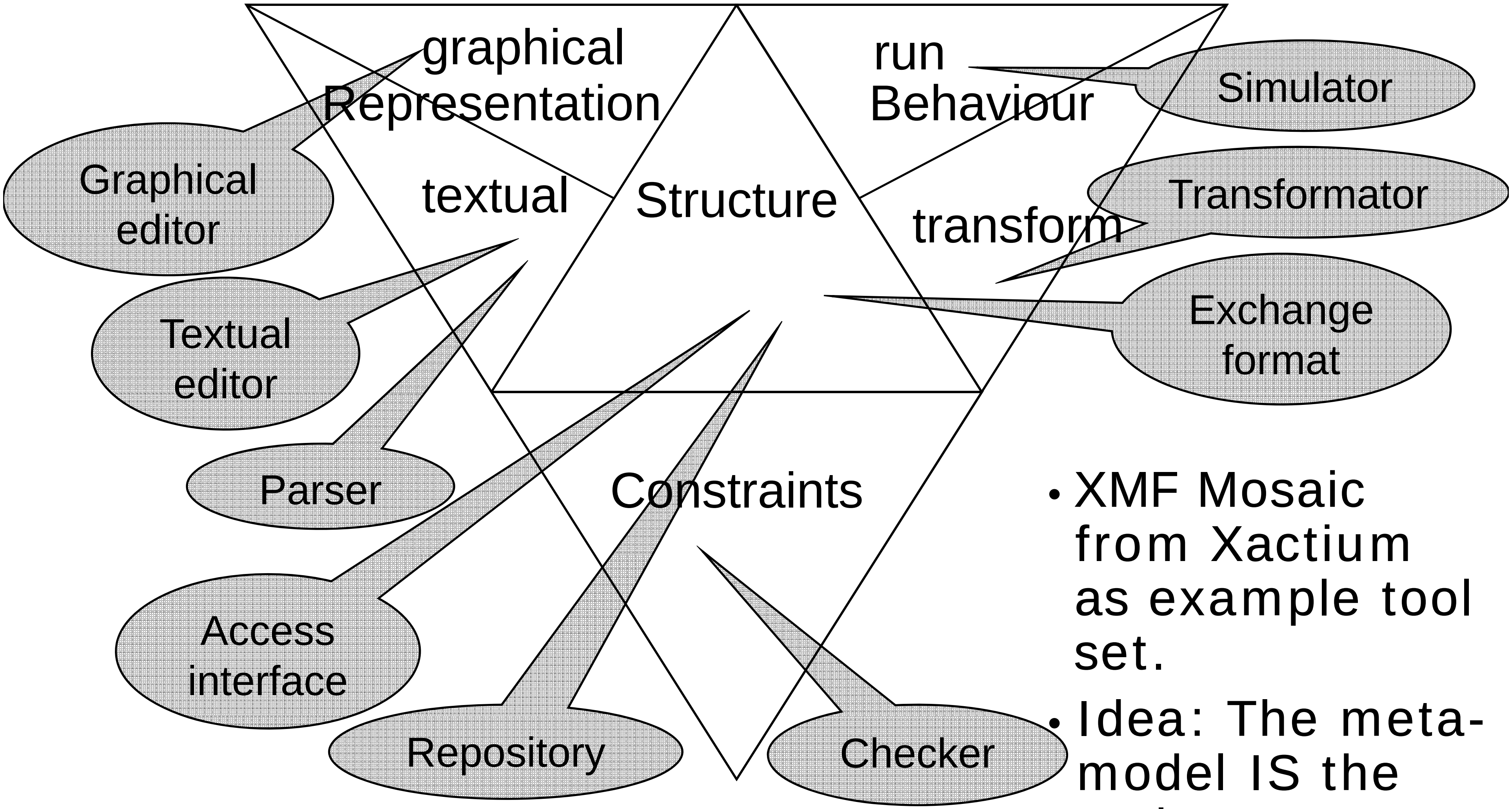
# Importance of internal structure



# Aspects of Compilers/Languages

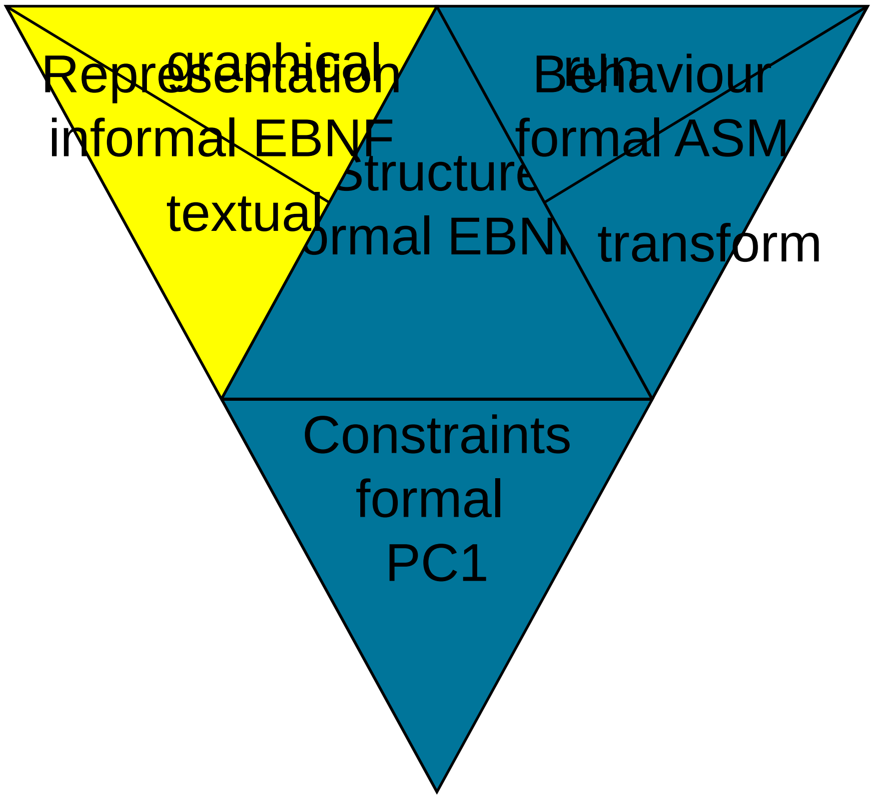
- Language structure: What are the concepts? How are they related?
- Static semantics: additional conditions, what is allowed?
- Representation: How are programs written? -> graphical vs. textual
- Dynamic semantics: What do the programs mean? How to generate code for them?

# Aspects of a language & tools

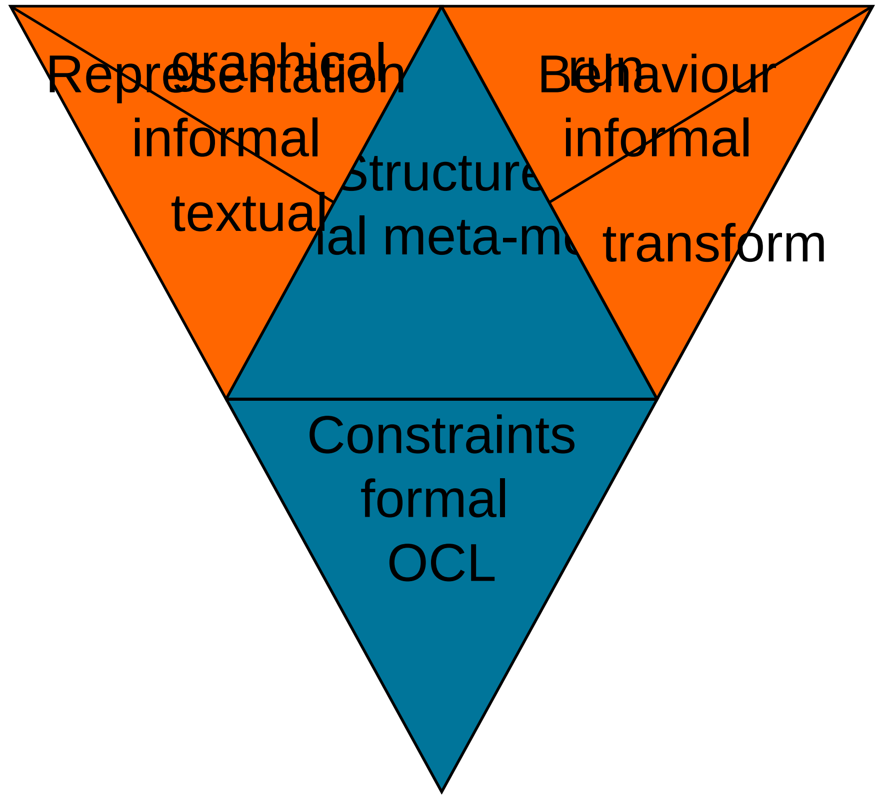



- XMF Mosaic from Xactium as example tool set.
- Idea: The meta-model IS the tool.

# Aspects for SDL and UML




SDL




UML

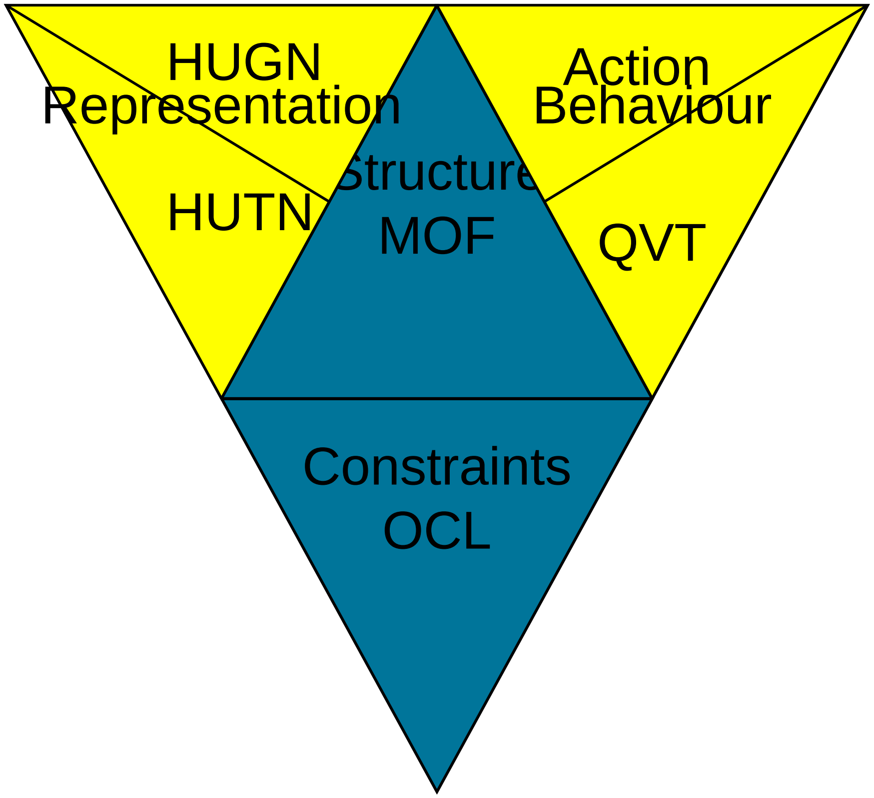
# What is a Model?

- A model is an abstraction of a (part of a) system.
  - one model describes several systems, one system can have several models
  - simplified view of a system with respect to criteria
  - can answer questions about the system if related to the view
  - needs a representation, e.g. using a modelling language
- Models on different abstraction levels
  - Models of the real Bits: Assembler
  - Models of the Control Flow: Prog. Lang.
  - Models of data storage: Database descriptions
  - Models of access: Interface languages
- What is the best model of a cat? → It is a cat. But it has to be the same cat!
- A model has aspects like a language.

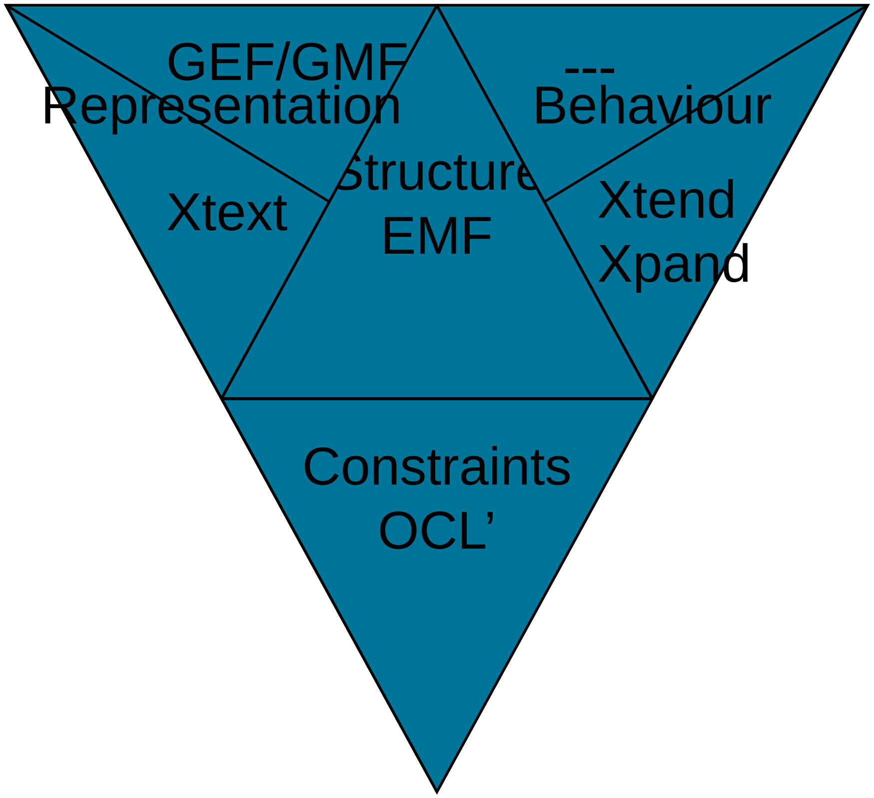
# What are Meta-Models?

- A description of a class of models
- Models / high-level descriptions of the modelling language
  - narrow view: structure of the modelling language
  - wider view: all important aspects of the language, i.e. structure, presentation, static and dynamic semantics
- Meta-models (languages) can have several aspects.

# Language support MDA and Eclipse

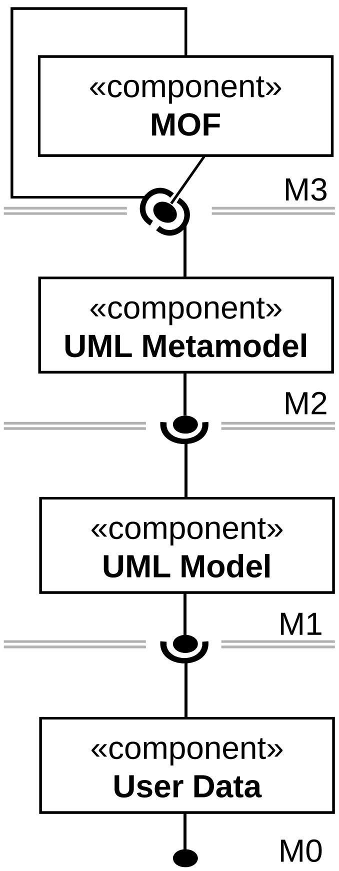



MDA



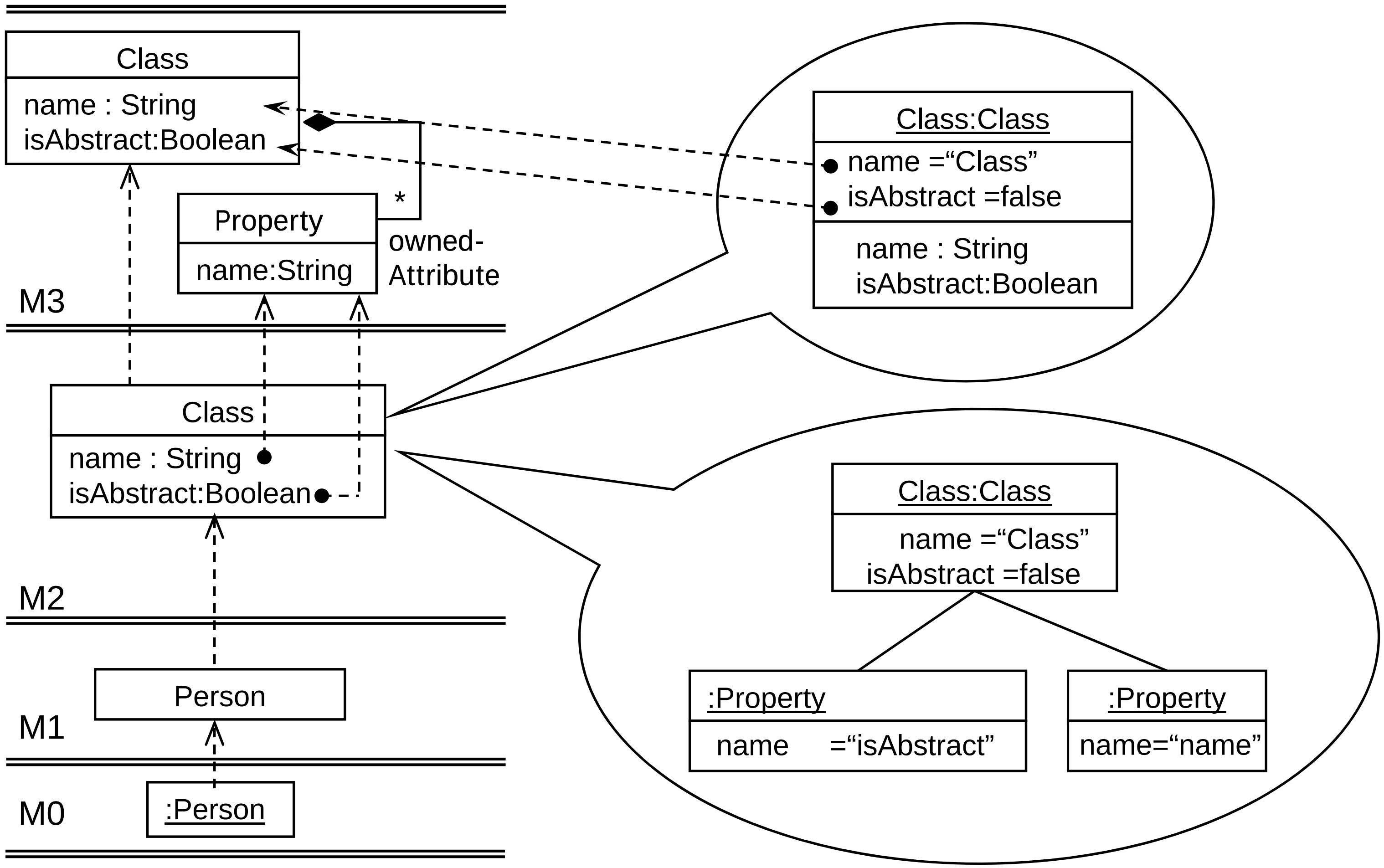

Eclipse (oaw)

# A meta-modelling architecture

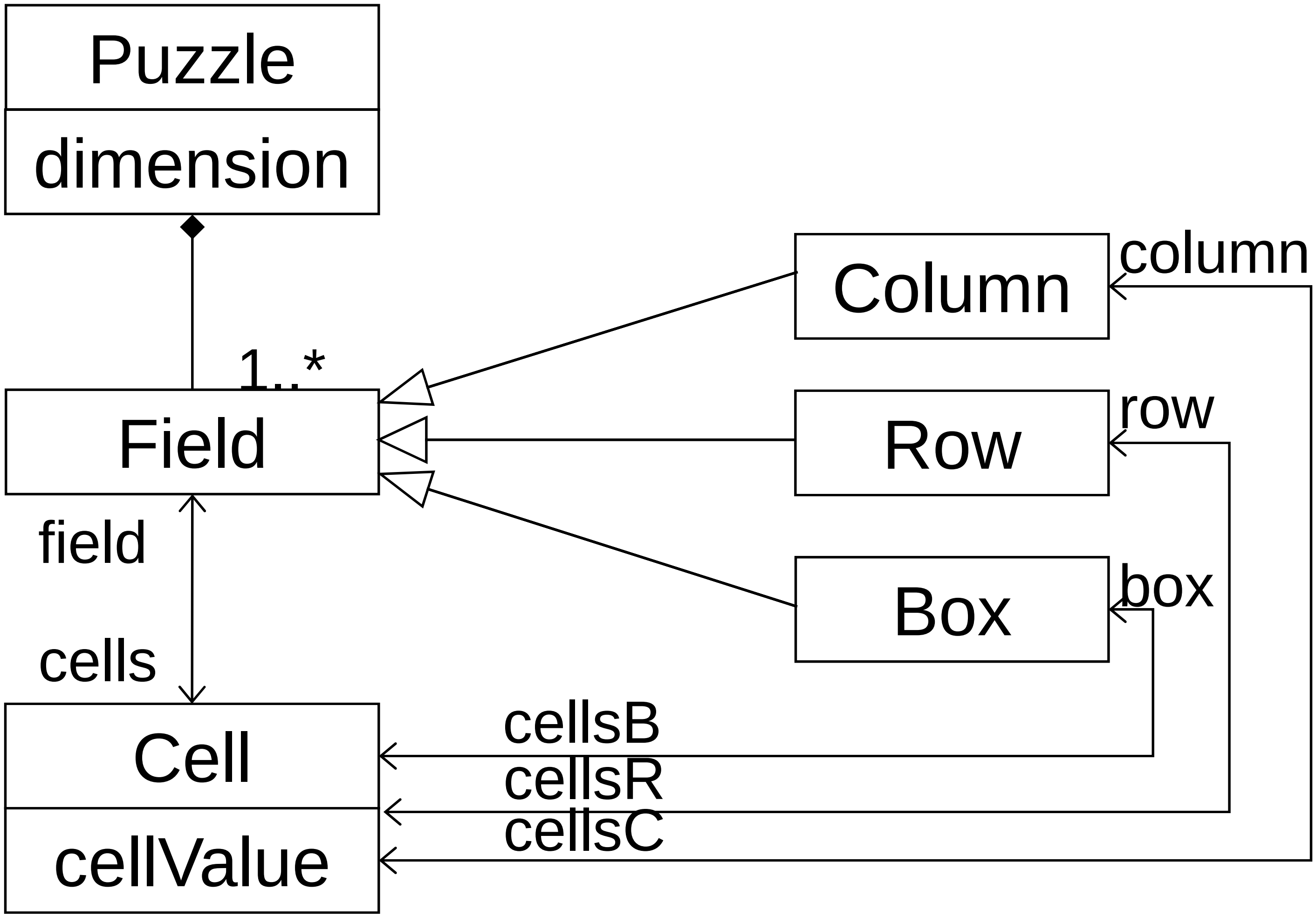



| OMG Level | Examples | Grammar example | OCL example |
|---|---|---|---|
| 3 = meta meta model | MOF | EBNF | MOF |
| 2 = meta model | UML MM | Java grammar | OCL language |
| 1 = model | UML Model | a program | a formula |
| 0 = instances | real objects | A run | a truth value |

# Instances on several levels



# Simple sample structure (EMF)



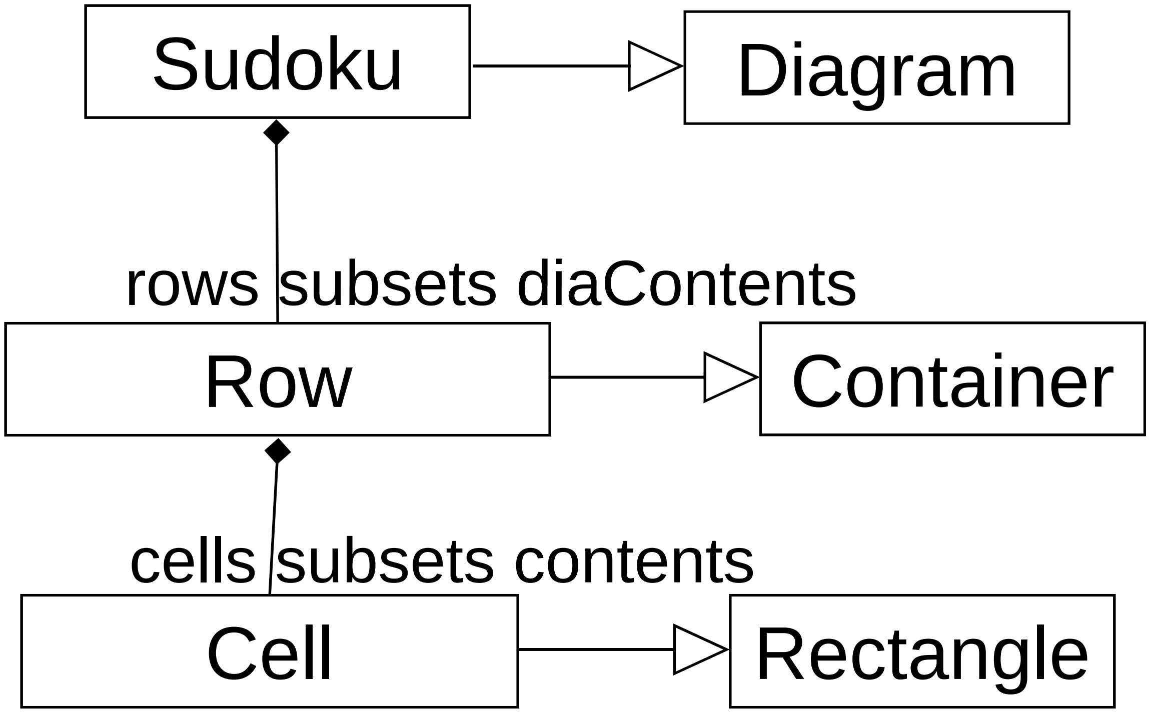
# Simple sample constraints (OCL)

```
context Field inv uniqueICellValues:
self.cells->forAll(c1,c2 : Cell | c1<>c2 implies
        c1.iCellValue <> c2.iCellValue)

context Cell inv rowFromCell:
self.row -> size()=1

context Puzzle inv numberOfBoxes:
self.Elements->select(f : Field |  f.oclIsTypeOf(Box))
-> size()=9
```

# Simple sample text syntax (TEF)

```
syntax toplevel PuzzleTpl, ecorepath “..." {
  element CellTpl for Cell{ single for iCellValue, with INTEGER; }
  element RowTpl for Row{
    "Row"; "(";
    sequence for cellsInRow, with @CellTpl, seperator ",", last false;
    ")";
  }
  element PuzzleTpl for Puzzle{
    "Puzzle"; "("; single for iDimension, with INTEGER; ")"; "=";
    sequence for Elements, with @FieldTpl, seperator ",", last false;
  }
  choice FieldTpl for Field{ @RowTpl }
}
```

# Simple sample graphics



# Simple sample transformation (QVT)

```
transformation theOne (source : sudoku, target: sudoku){
 top relation change1to16 {
  checkonly domain source sudoku:Cell { iCellValue = 1 };
  enforce domain target sudoku:Cell { iCellValue = 16 };
 }
 top relation change6to11 {
  checkonly domain source newstructure:Cell { iCellValue = 6 };
  enforce domain target newstructure:Cell { iCellValue = 11 };
 }
 top relation nochange { value: Integer;
  checkonly domain source newstructure:Cell { iCellValue = value };
  enforce domain target newstructure:Cell { iCellValue = value };
  when{ iCellValue <> 1 or iCellValue <> 6; }
} }
```

# Simple sample execution

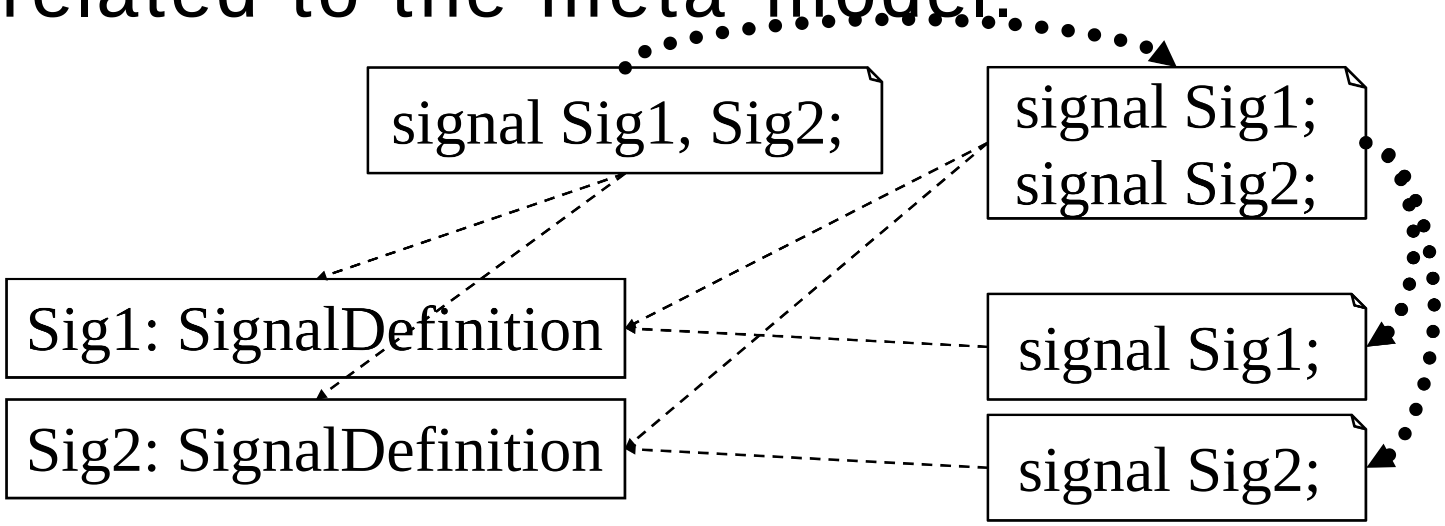
```
Run(s:Sudoku) =def
  forall f in self.field do RunF(f)
Runf(f:Field) =def
  choose c in self.cell with c.value=null
      and c.possible.size = 1
    choose v in c.possible do c.value:= v
  choose c in self.cell with c.value<>null
    forall cc in self.cell do
      delete c.value from cc.possible
```

# Problem area execution

| | Syntax | Runtime |
|---|---|---|
| Meta-model | Cell | RTCell |
| Model | X:Cell | A: RTCell<br>e.g. history<br>B: RTCell |

# Problem area “representation”

- There are usually several representations for the same meta-model instances.
- Tools and theory exist only for the case 1:1.
- A representation is a separate model that is related to the meta-model.



# 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)

# Grammars → meta-models

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.

# Using the transformation for SDL

- Joachim Fischer, Michael Piefel, Markus Scheidgen: A Metamodel for SDL-2000 in the Context of Metamodelling ULF in Proceedings of SAM2006
- 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

# Conclusions / Summary

- Future languages will be defined using meta-models.
  - definition of good meta-models is difficult
  - need also agreement (standard)
  - patterns for good models needed, maybe joint concepts
- Meta-models / Languages have several aspects: structure, syntax, static and dynamic semantics
- Meta-model language definitions allow tool generation
  - Direct access to the models
  - Easy exchange of representation or several of them
  - Combination of tools handling the language
  - Description of relations between languages
- Meta-models are the models to be used in model-driven compiler technology.