Modellbasierte Softwareentwicklung (MODSOFT)

Part II

Domain Specific Languages

Introduction

Prof. Joachim Fischer /
Dr. Markus Scheidgen / Dipl.-Inf. Andreas Blunk

{fischer,scheidge,blunk}@informatik.hu-berlin.de

LFE Systemanalyse, III.310

Model-based Software Engineering – Two Ways

Model-based Software Engineering – Two Ways

Use existing, standardized languages,
like the *Unified Modeling Language*
(UML)

- often subsumed under the term
  Model Driven Architecture (MDA)
- based on expensive CASE-Tools
- tailored for different domains via
  profiles and specialized tools
- often requires elaboration of
  generated code
- waterfally, document heavy, linear
  processes

Model-based Software Engineering – Two Ways

Use existing, standardized languages, like the *Unified Modeling Language* (UML)

- often subsumed under the term *Model Driven Architecture* (MDA)
- based on expensive CASE-Tools
- tailored for different domains via profiles and specialized tools
- often requires elaboration of generated code
- waterfally, document heavy, linear processes

Write and use *Domain Specific Languages* (DSL)

- requires you to develop your own modeling languages first
- often requires to adopt the languages while using them
- rather light weight and flexible, incremental development processes, agile

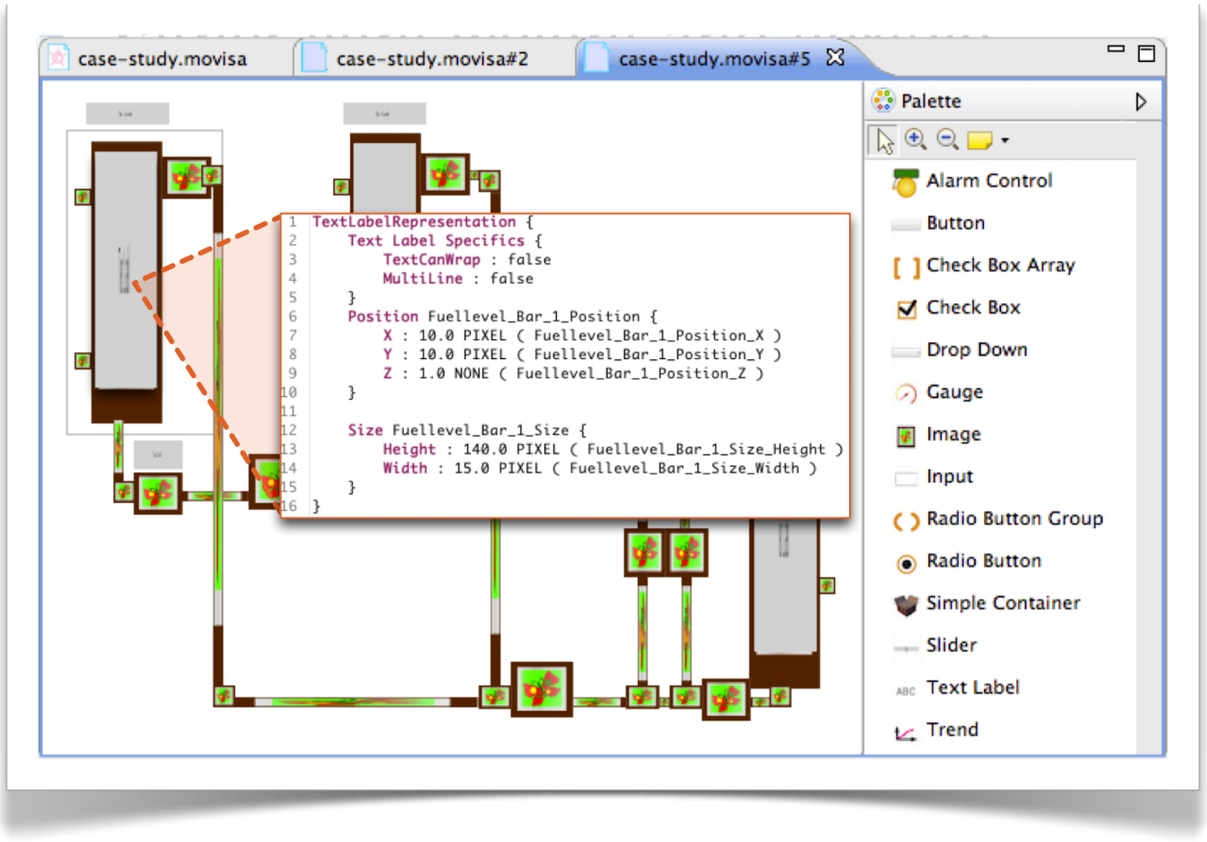
Model-based Software Engineering – Two Ways

Use existing, standardized languages, like the *Unified Modeling Language* (UML)

- often subsumed under the term *Model Driven Architecture* (MDA)
- based on expensive CASE-Tools
- tailored for different domains via profiles and specialized tools
- often requires elaboration of generated code
- waterfally, document heavy, linear processes

Write and use *Domain Specific Languages* (DSL)

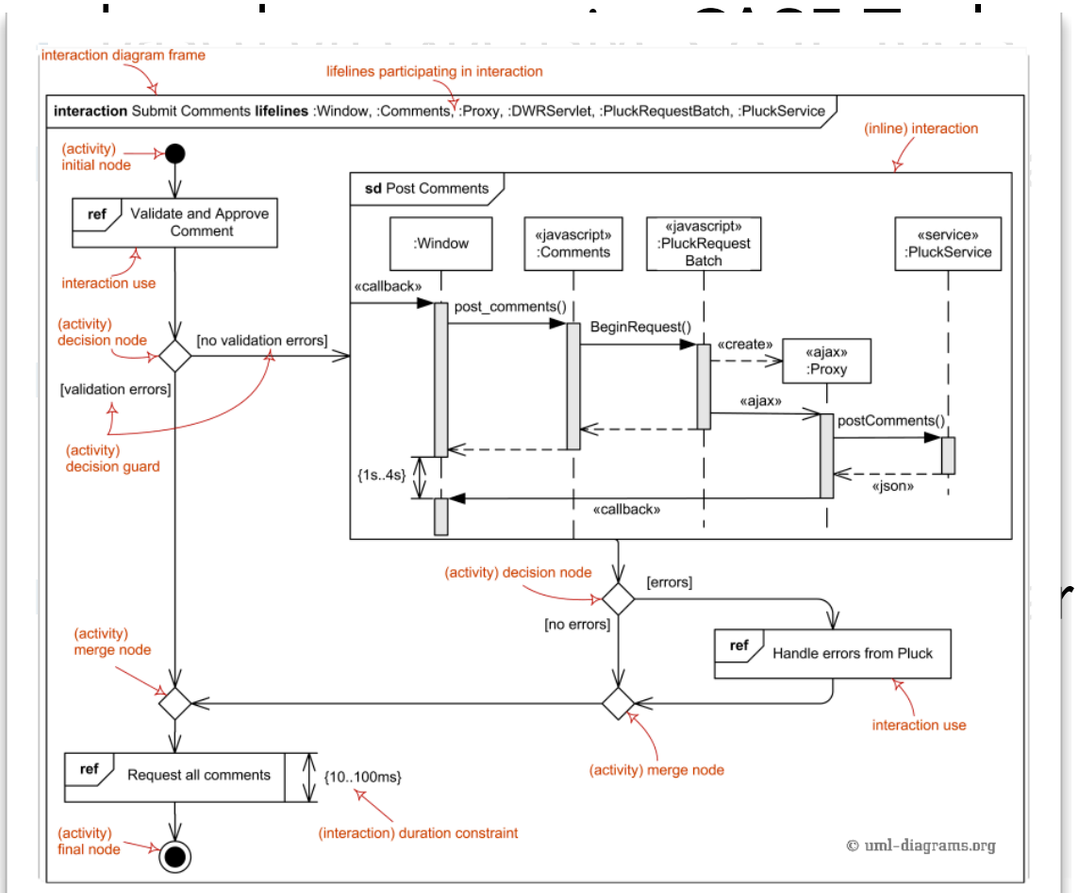
- requires you to develop your own modeling languages first
- often requires to adopt the languages while using them



Model-based Software Engineering – Two Ways

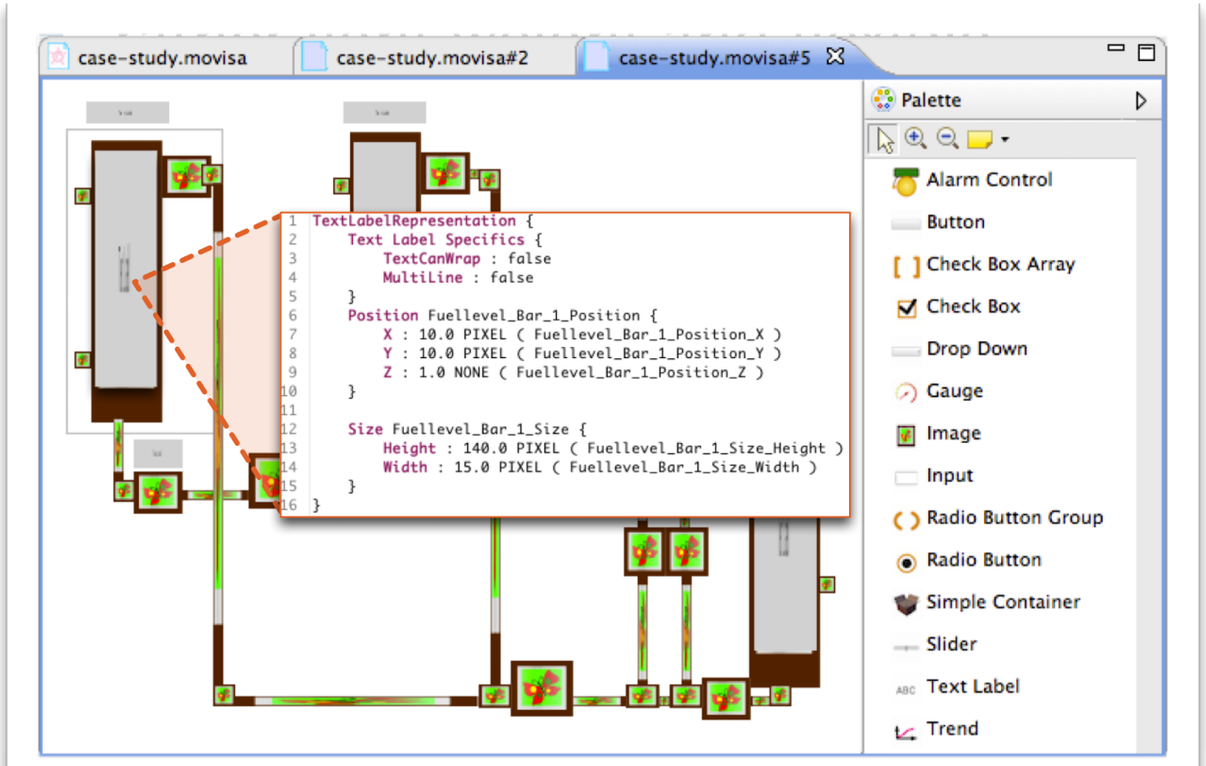
Use existing, standardized languages, like the *Unified Modeling Language* (UML)

- often subsumed under the term *Model Driven Architecture* (MDA)



Write and use *Domain Specific Languages* (DSL)

- requires you to develop your own modeling languages first
- often requires to adopt the languages while using them



Agenda

➡ prolog (1 VL)	**Introduction:** languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model
0. (2 VL)	**Eclipse/Plug-ins:** eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCP*s
1. (2 VL)	**Structure:** *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*
2. (3 VL)	**Notation:** Customizing the tree-editor, textural with *XText*, graphical with *GEF* and *GMF*
3. (4 VL)	**Semantics:** interpreters with Java, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*
epilog (2 VL)	**Tools:** persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System* (MPS)

Languages

Languages Involved in Model-based Software Engineering

- Natural Languages
- Non-Computer comprehensible languages
  - e.g. ad-hoc sketches
  - informally defined
- Computer Languages
  - formally defined
  - written by humans or machines
  - understandable by machines (and sometimes by humans)
  - modeling languages, programming languages (and others)

Modeling != Programming

- Propose
  - abstract description with various purposes vs. concrete instructions
- Level-of-abstraction
  - models only contain the information necessary to fulfill a specific purpose
  - its quite common to have different models of the same thing to cover different purposes (validate specific properties, test, implementation, design, deployment, etc.)
  - some techniques (especially formal verification) only work on abstractions
- Complete vs. Incomplete
  - programs have to be complete to be useful
  - models can be incomplete and still fulfill their purpose
- Domain model vs. system model
  - a model can be the model of a problem or problem domain
  - a model can be a model of a (software) system that solves a problem
- Syntax-based differences
  - graphical vs. textual

Modeling ⊇ Programming

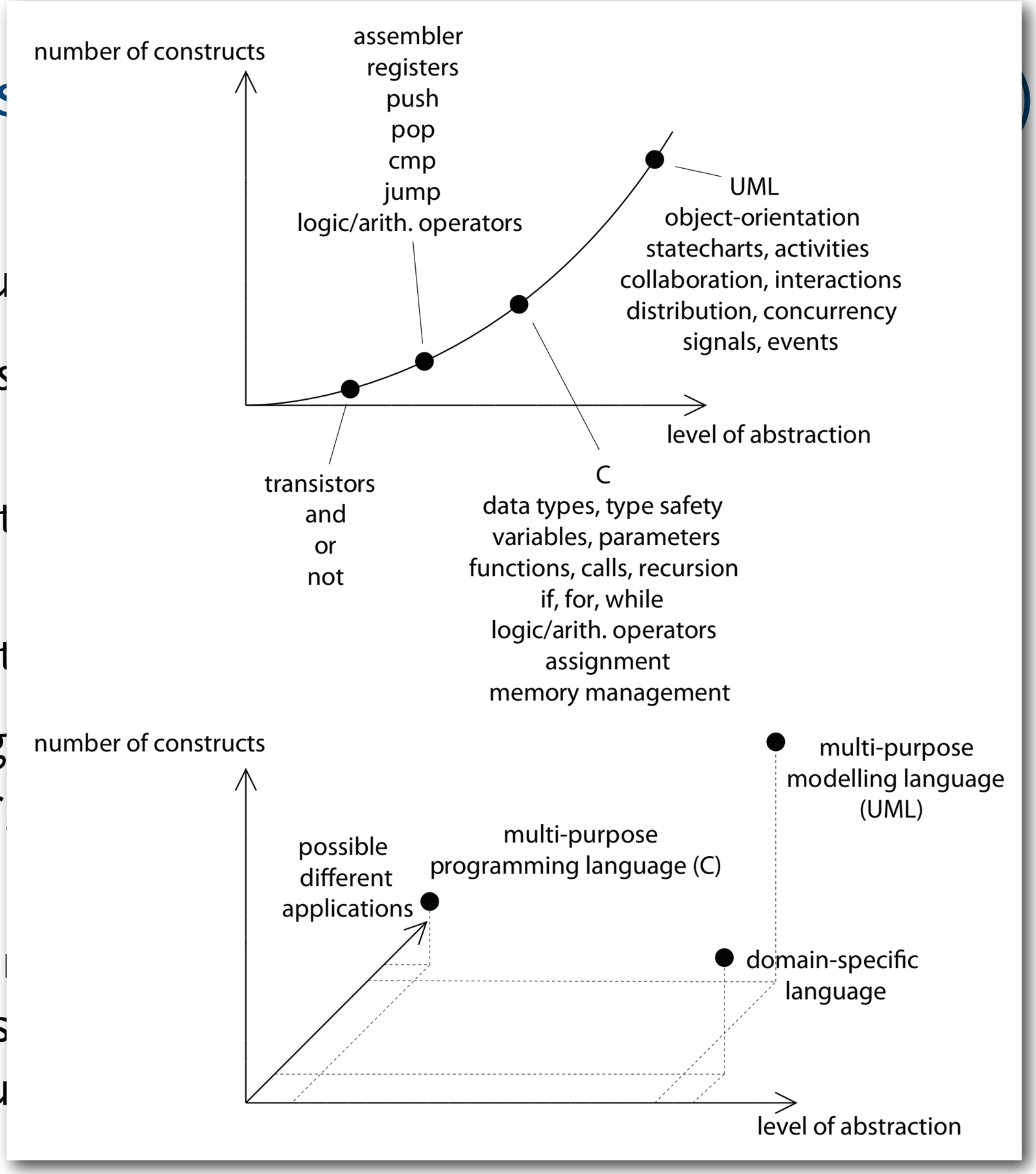
- Programs are models
  - of a software system
  - on a low level of abstraction
  - that are complete
  - with the purpose to fully describe a run-able system
- This view on modeling and programming is not shared by all people

General Purpose vs. Domain Specific Languages (DSL)

- This can be said about both, modeling and programming languages
- Types of *expressiveness* (and levels of abstraction)
  - *expressiveness* means that you can express something or not with the given language constructs (e.g. to be Turing-machine equivalent)
  - *expressiveness* means that you can express more with fewer uses of the given language constructs (i.e. can say the same with fewer words)
- General purpose languages have only few constructs that can be used to describe a large class of things, but generally require larger artifacts
- DSLs have a set of very specific constructs that can only be used to describe a small class of things (in a specific domain), and therefore usually require smaller artifacts

General Purpose vs

- This can be said abou
- Types of *expressiveness*
  - *expressiveness* means
    given language const
  - *expressiveness* means
- General purpose lang
  used to describe a lar
  artifacts
- DSLs have a set of ve
  describe a small class
  therefore usually requ



Examples

Deutsch

- lots of constructs (syntax rules + words)
- can express everything, but not very precise
- depends on interpretation

- natural language
- high expressiveness
- high expressiveness

Java

- only a few constructs
- Turing-machine equivalent

- general purpose programming language
- high expressiveness
- low expressiveness

UML

- many constructs
- Turing-machine equivalent (with the right semantics)

- general purpose modeling language
- high expressiveness
- high expressiveness

HTML

- only a few constructs
- can only do web-page markup

- domain specific language
- low expressiveness
- high expressiveness

Types and Examples of DSLs

- API/vocabulary
  - libraries in general purpose (programming) languages provide functionality in a reusable form
  - functions are vocabulary from a language perspective
- internal DSL
  - some general purpose languages have a very flexible set of language constructs that allows to emulate a specific domain specific syntax
  - a library that exploits syntactic flexibility of the *host language* can be seen as an *internal* DSL
  - you can use the host language tooling
- external DSL
  - languages with an own syntax, semantics, and vocabulary
  - require to build there own set of language tools

Syntax vs. Vocabulary

- Different things in natural languages, melt in many computer languages
- Defined functions and APIs can be seen as vocabulary
- Language constructs have to be seen as syntax
- In DSLs language constructs have a syntactic function and present pieces of vocabulary
- For computer languages better use syntax and libraries, not syntax and vocabulary

Summary

- Programing languages
- Modeling languages
- (Level of) abstraction
- Expressiveness (x2)
- General purpose language
- Domain specific language
- External and internal DSL
- Host language

Martin Fowler: *Domain Specific Languages* (Addison-Wesley 2010)

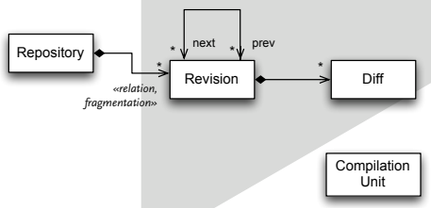
Meta-Languages

Languages

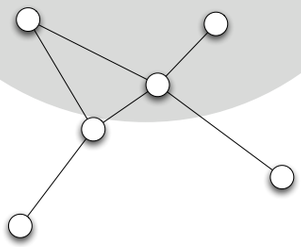
- A *computer language* (or simply language) is the set of all the language instances generated by a language description.
- A *language instance* is a well defined representation for a piece of information.
- A *language description* is a finite system of rules that describes what constitutes the valid instances of the described language. Therefore, a language description is a means to generate all the valid instances of the described language by accepting valid instances.

Language Aspects

notation

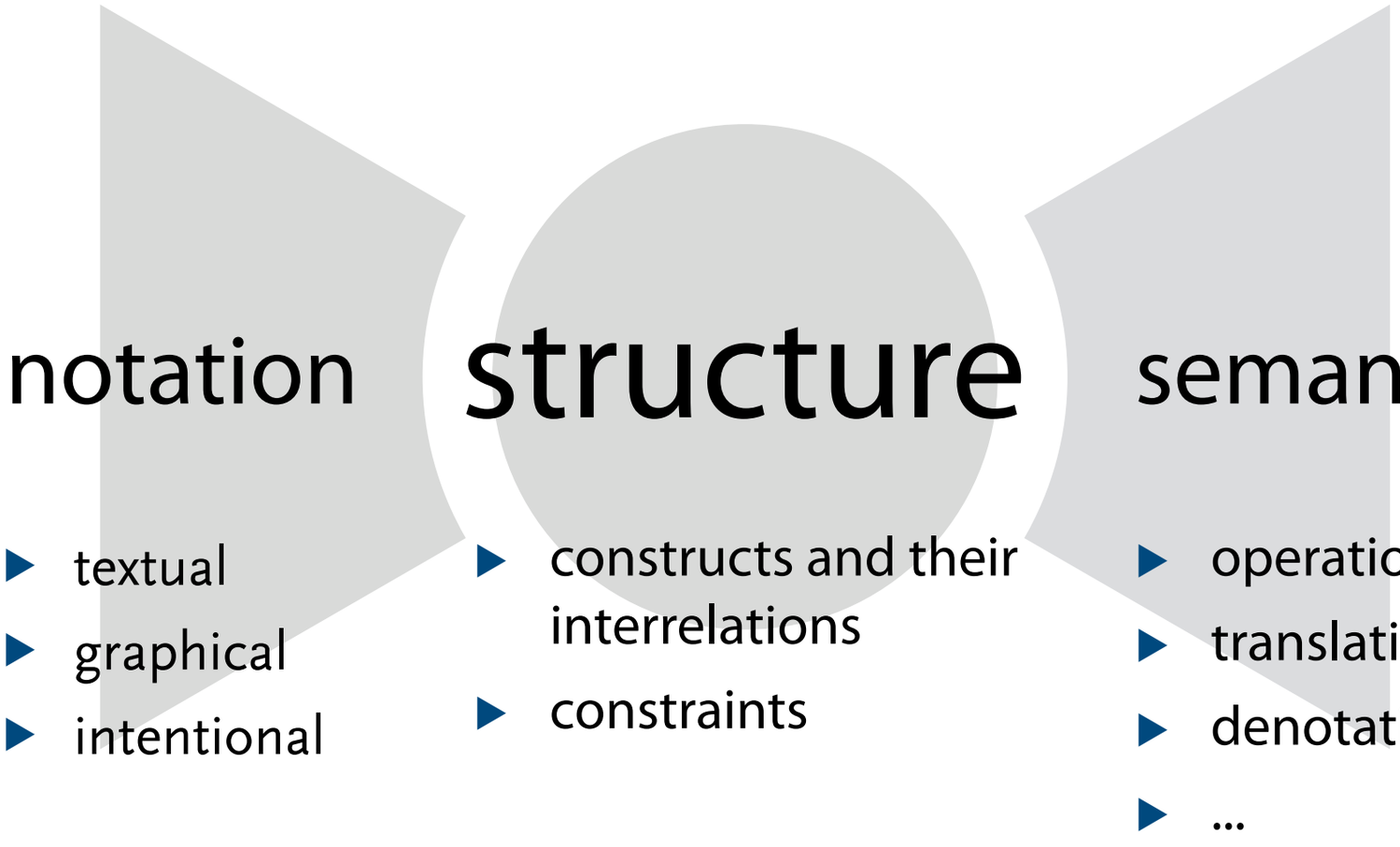


structure



semantics

Types of Language Aspects



notation

- textual
- graphical
- intentional

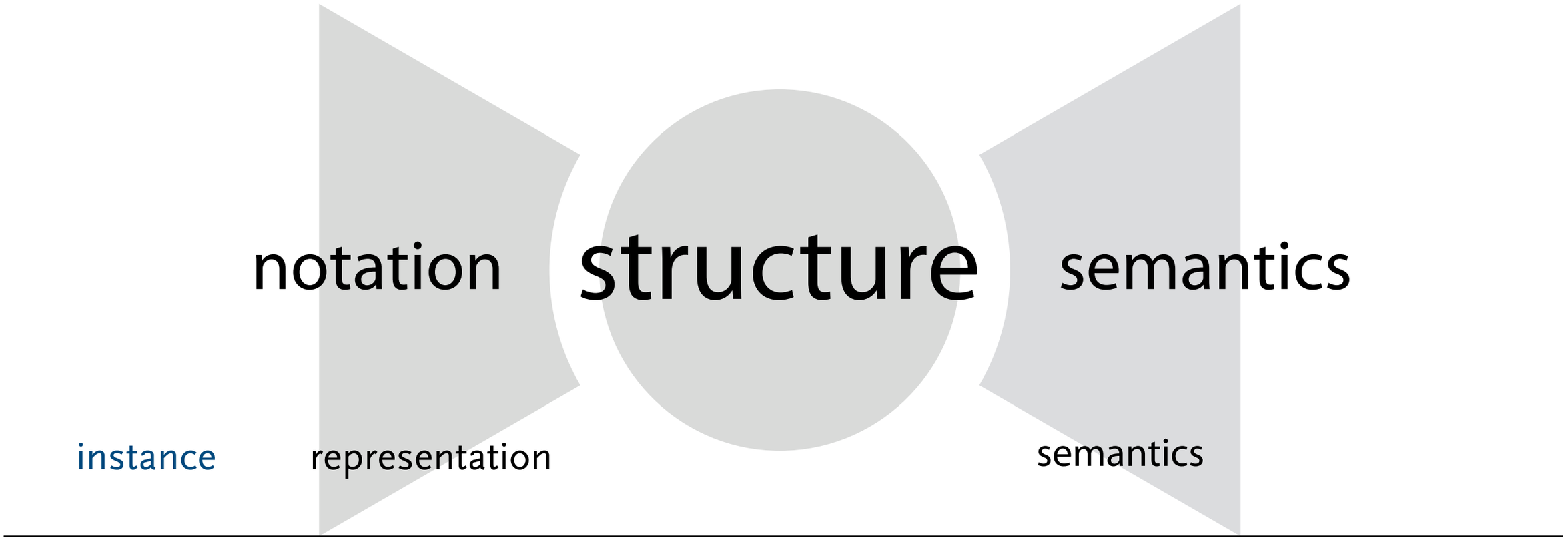
structure

- constructs and their interrelations
- constraints

semantics

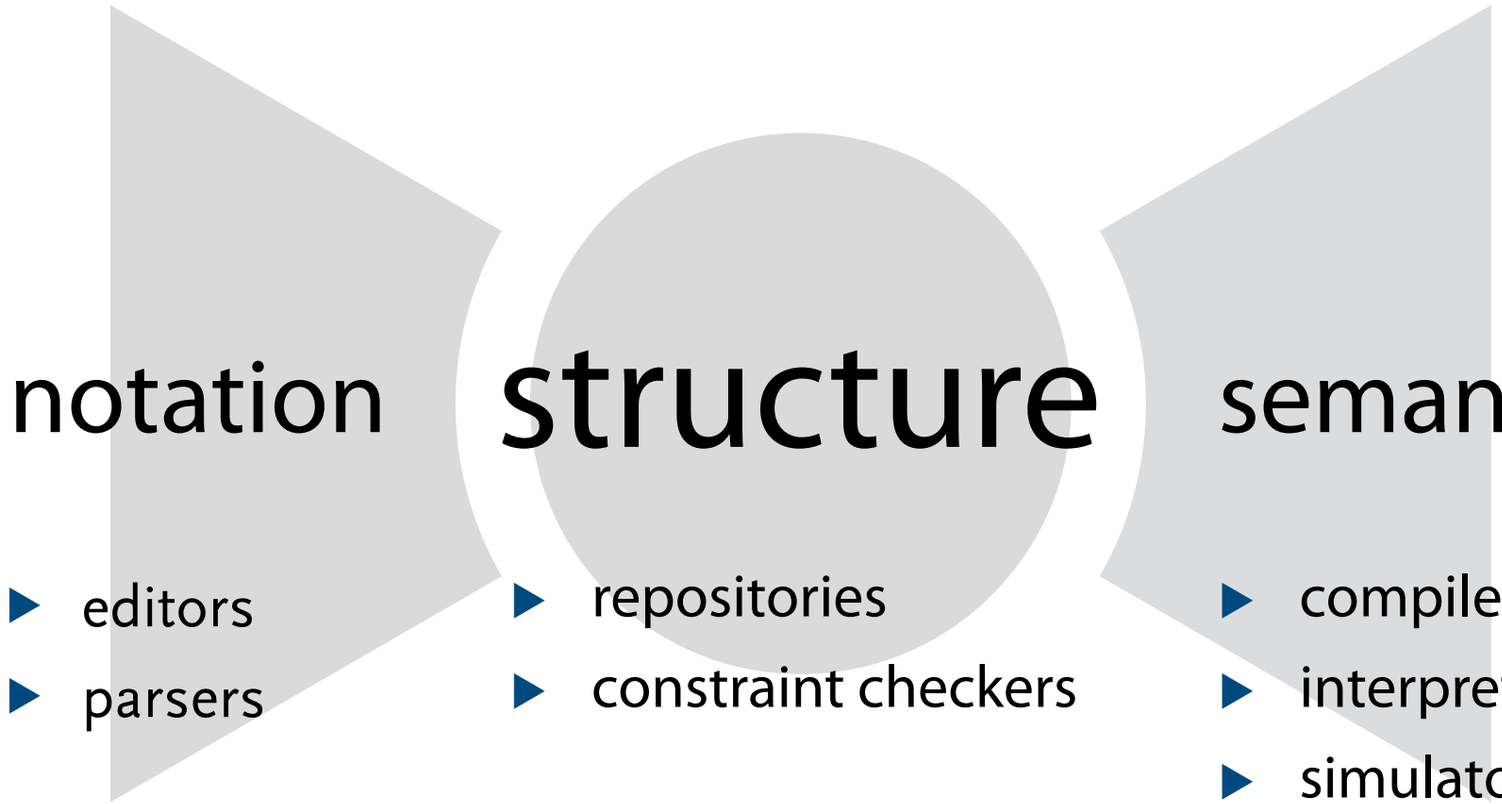
- operational
- translational
- denotational
- ...

Language Aspects Additional Nomenclature



instance	representation		semantics	
	notation	structure/construct	semantics	description

Language Tools



notation

- editors
- parsers

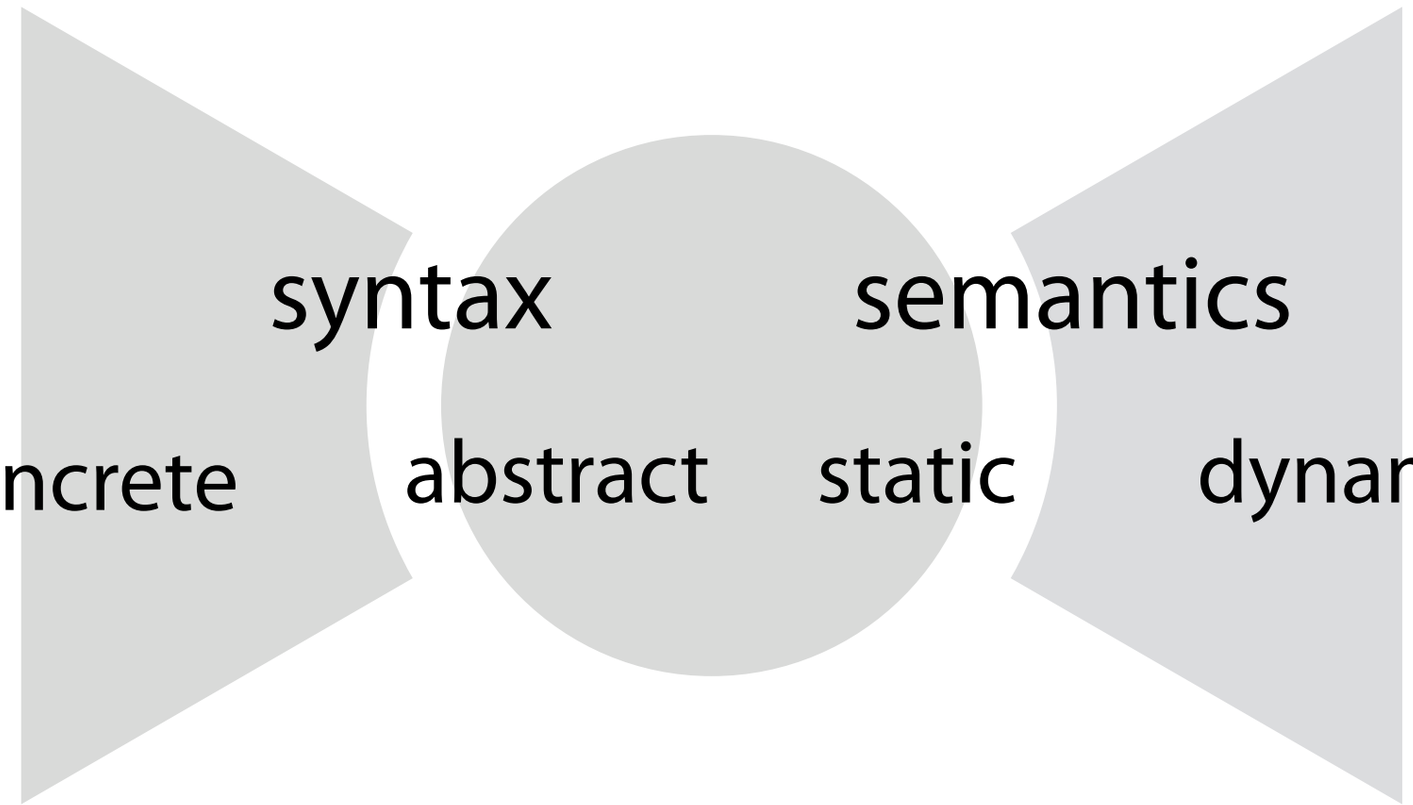
structure

- repositories
- constraint checkers

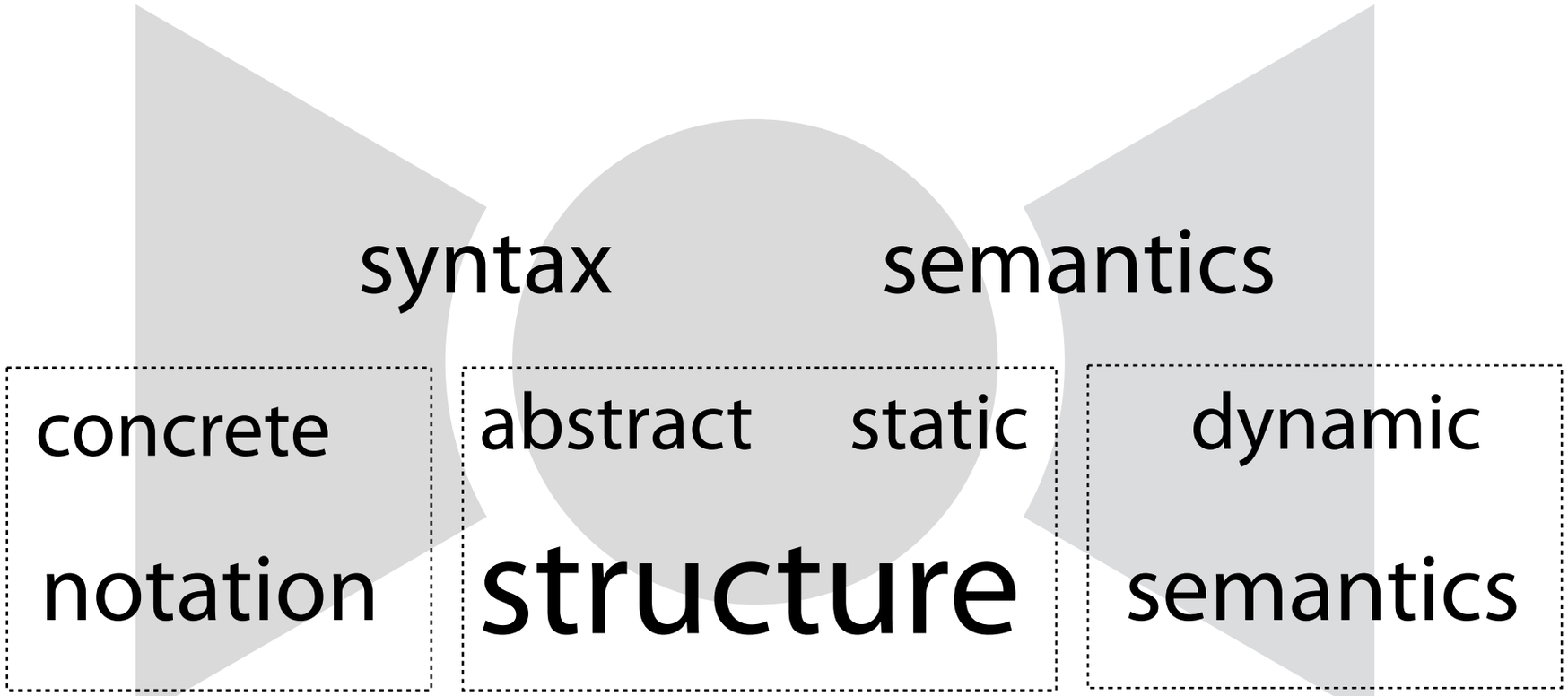
semantics

- compilers
- interpreters
- simulators
- model checkers
- ...

Nomenclature for Traditional Languages



Nomenclature for Traditional Languages



Summary

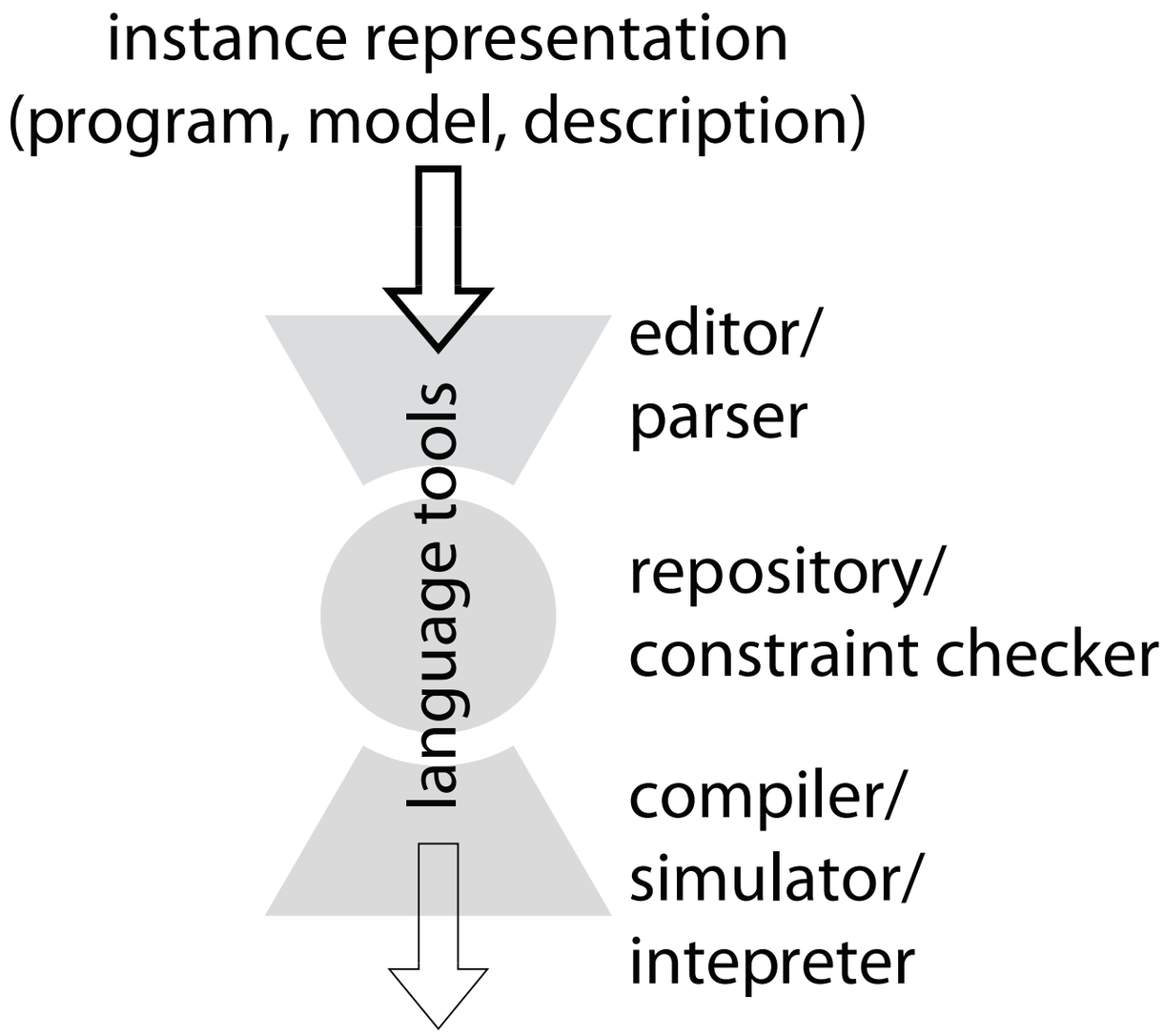
- Notation
- Structure
- Semantics
- Static, dynamic semantics
- Abstract, concrete syntax
- Representation
- Tool

Languages are Software too

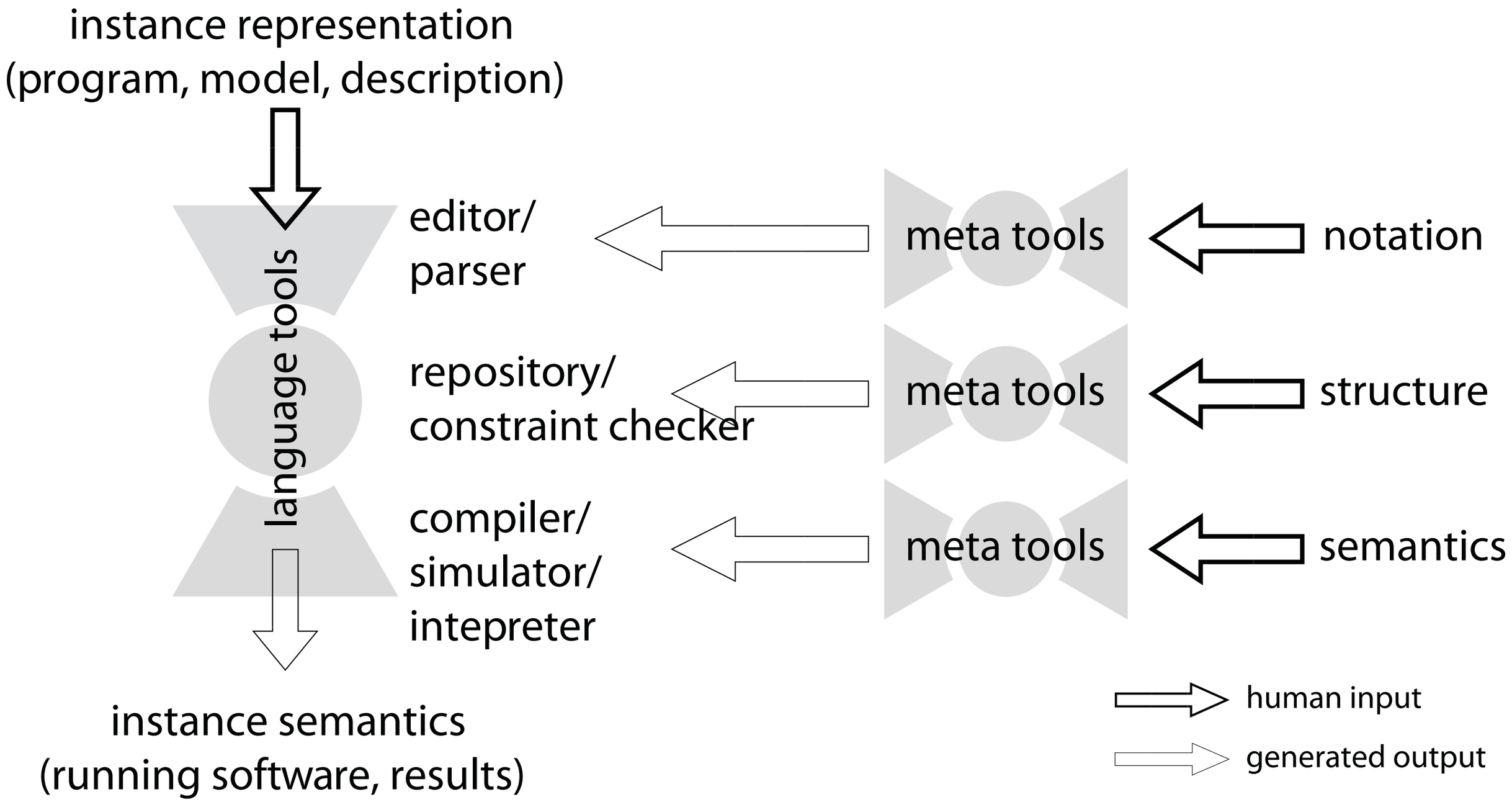
- Editors are pieces of software
- Repositories and constraint checkers are pieces of software
- Compilers, interpreters, simulators are pieces of software

➡ Languages are pieces of software

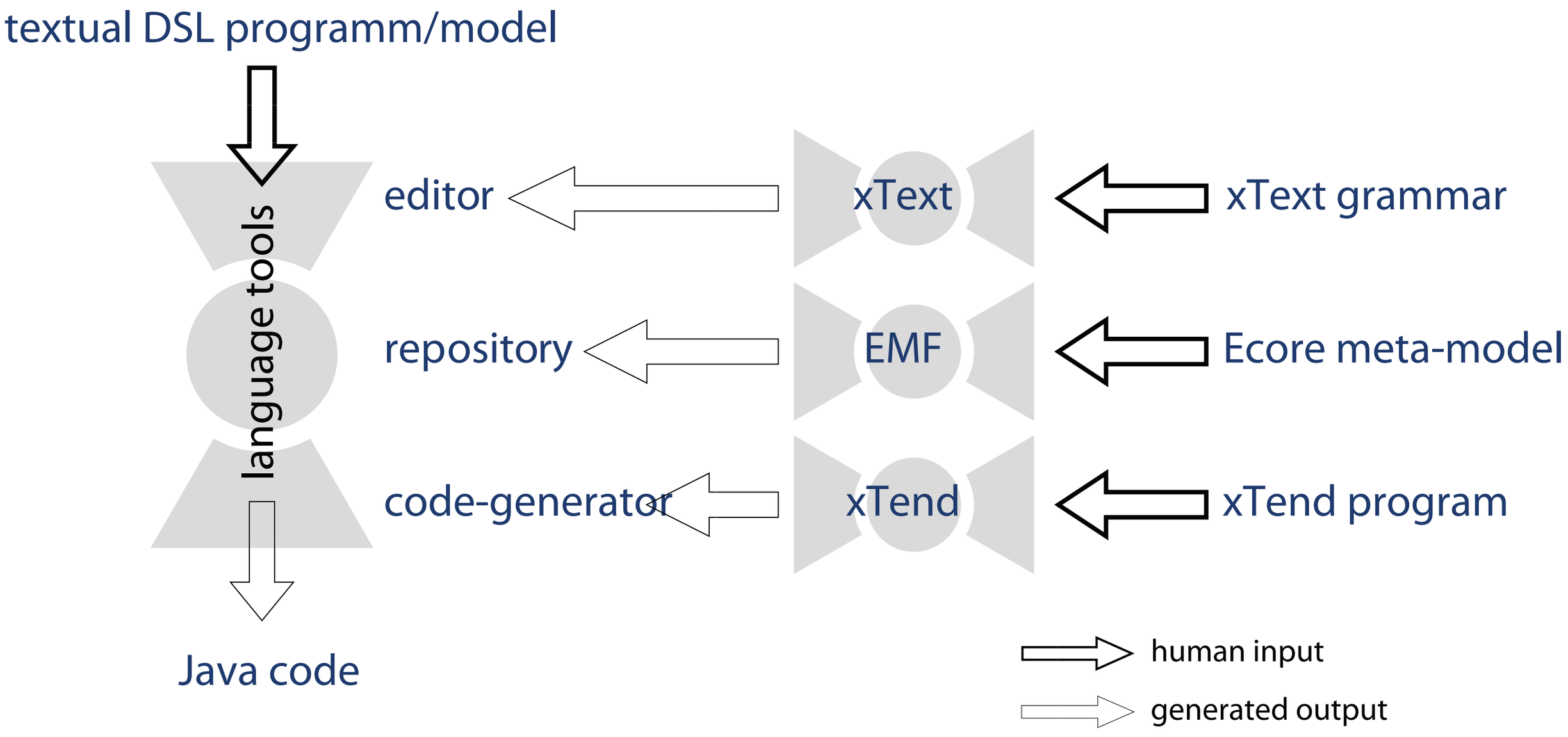
Meta-Languages



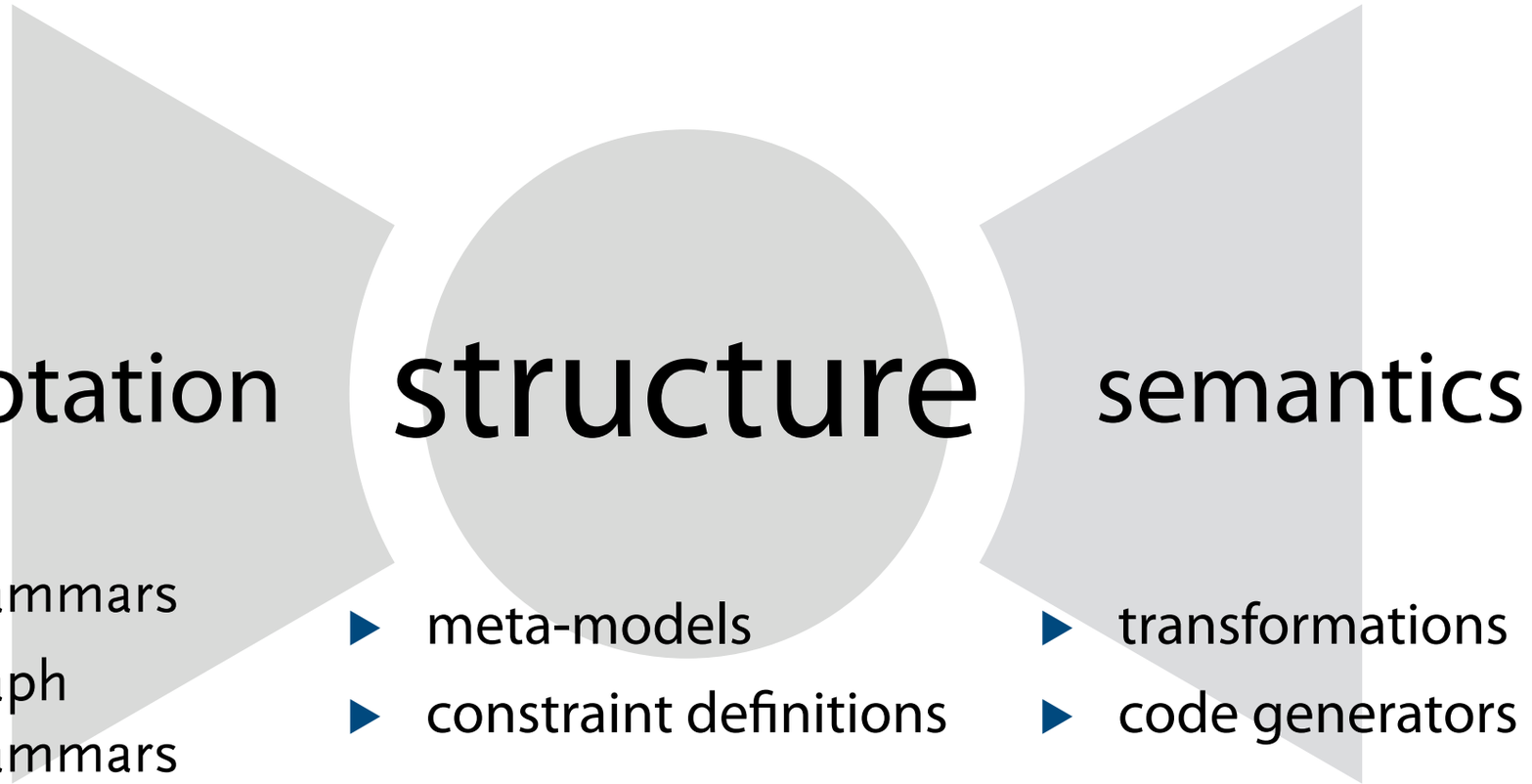
Meta-Languages



Concrete Meta-Languages



Other Meta-Languages/Tools



notation	structure	semantics
▶ grammars ▶ graph grammars	▶ meta-models ▶ constraint definitions	▶ transformations ▶ code generators

---

▶ xText, TEF	▶ EMF, MOF	▶ QVT, ATL
▶ GMT	▶ OCL	▶ xTend, Jet

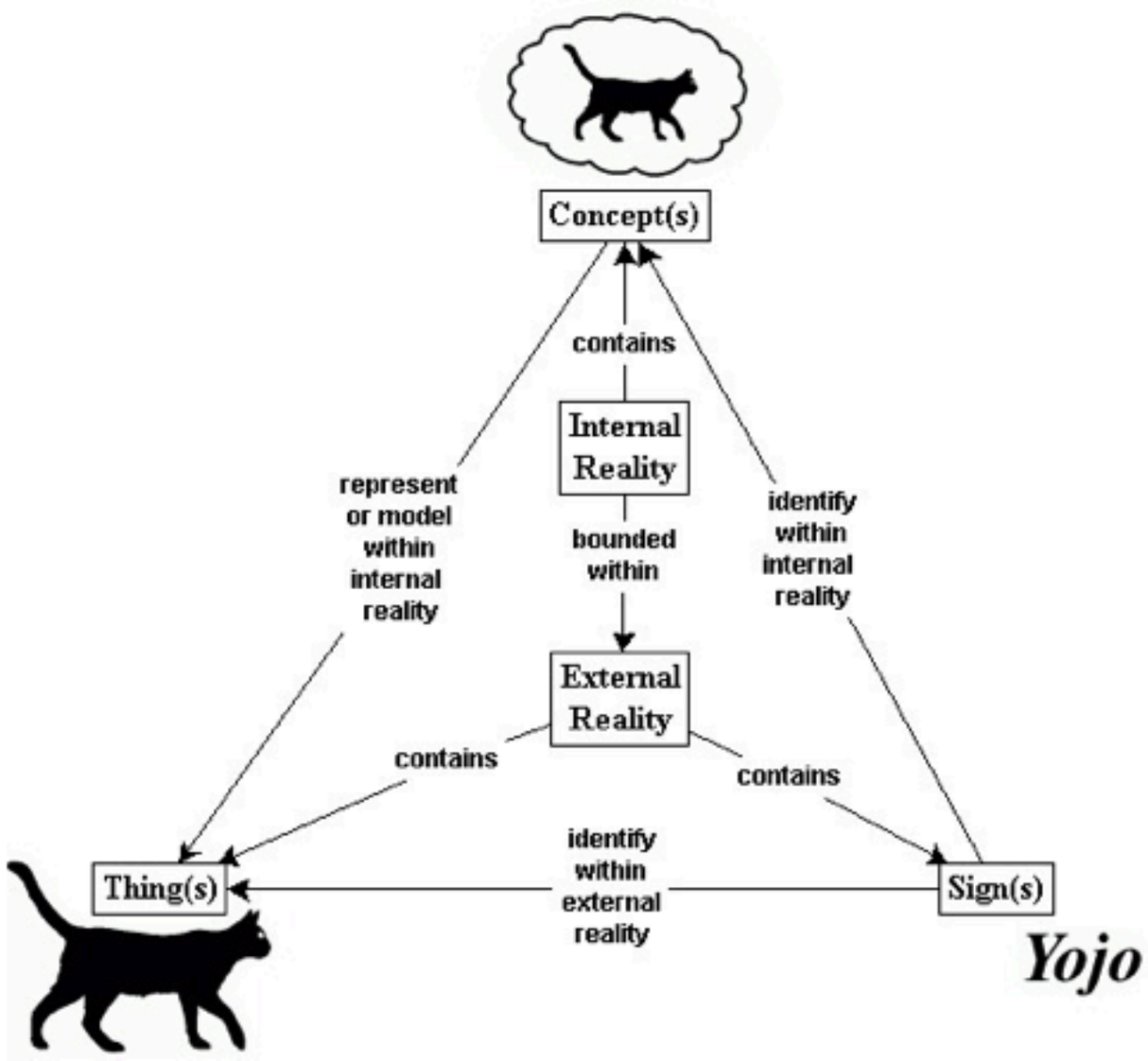
Summary

- Meta-Languages
- Meta-Tools

Meta-Modeling

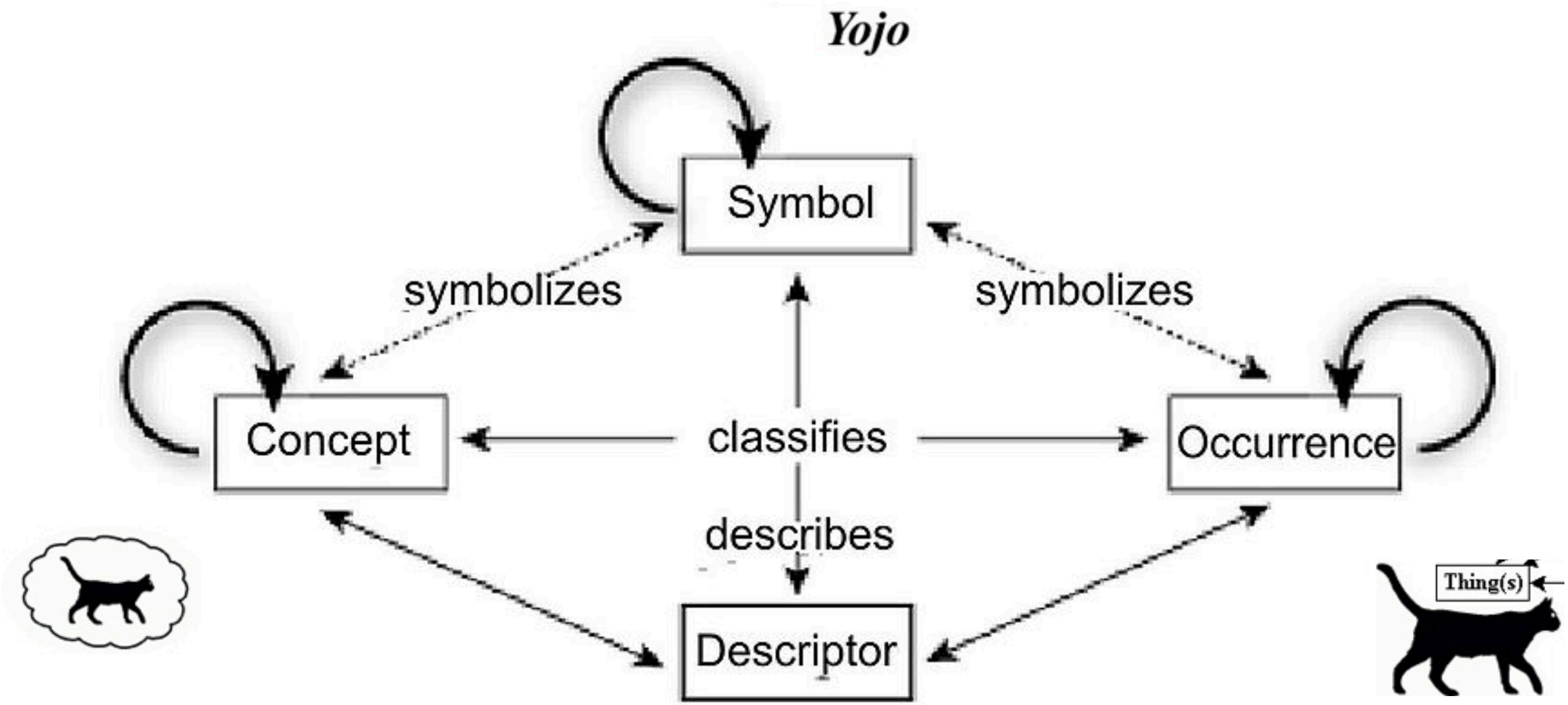
- The 4-layer model

Meta-Modeling – Semiotic Triangle (I)

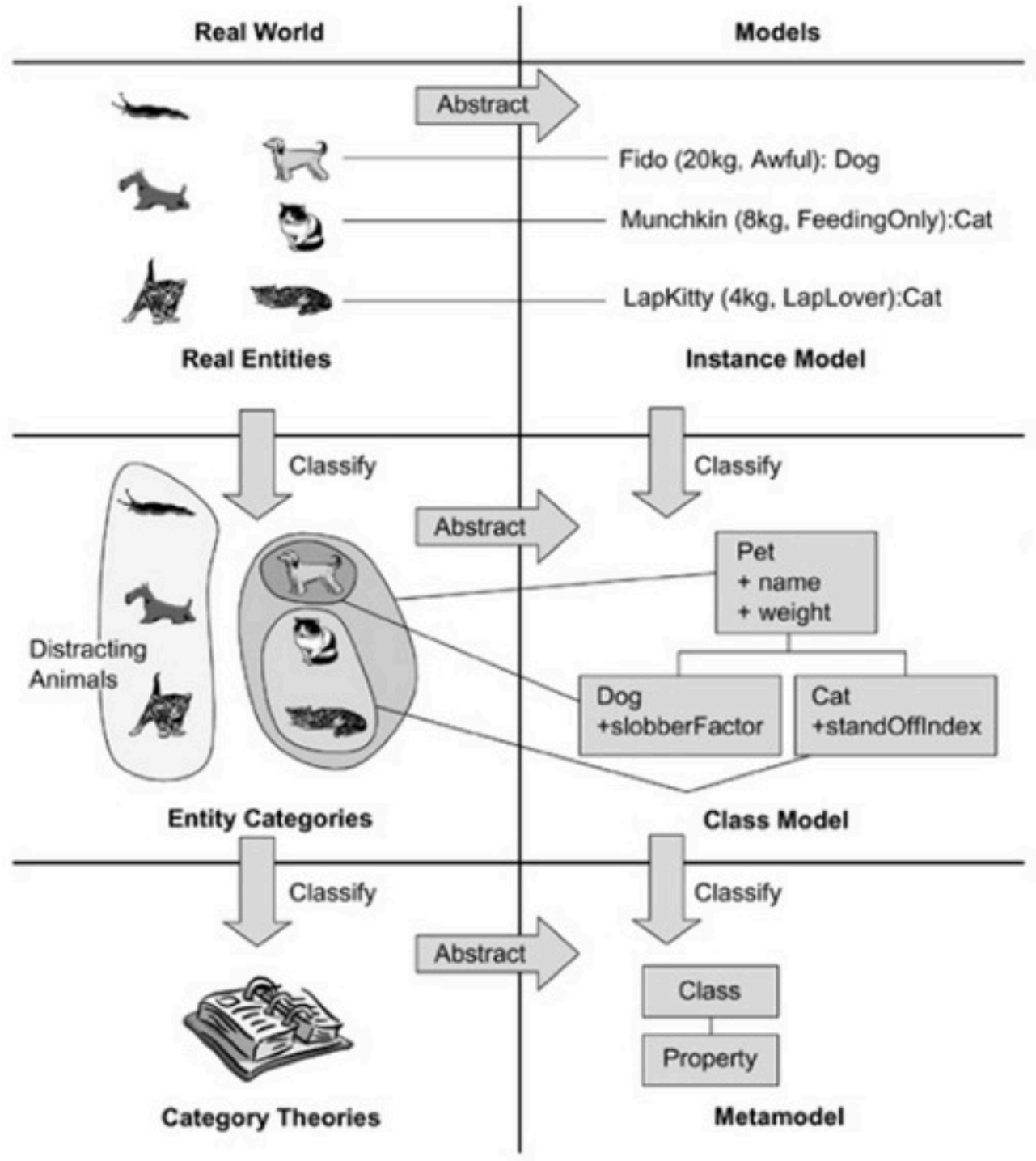


Ogdenand Richards: *The Meaning of Meaning* (1923)
Bernard Bolzano: *Beiträge zu einer begründeteren Darstellung der Mathematik* (1810)
Aristotle: *Peri Hermeneias*, 2nd book of his*Organon* (4th century BC)

Meta-Modeling – Semiotic Triangle (II)

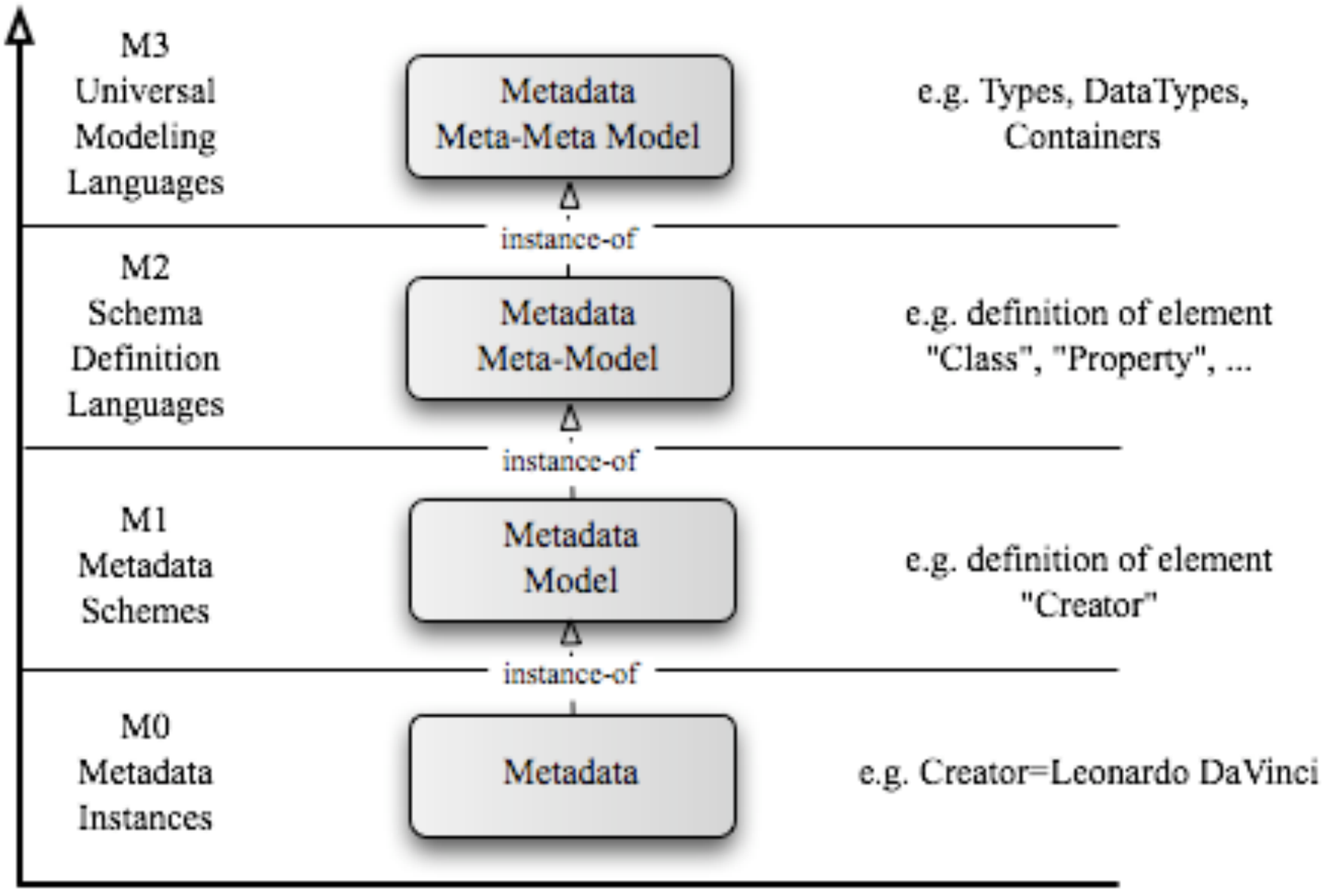


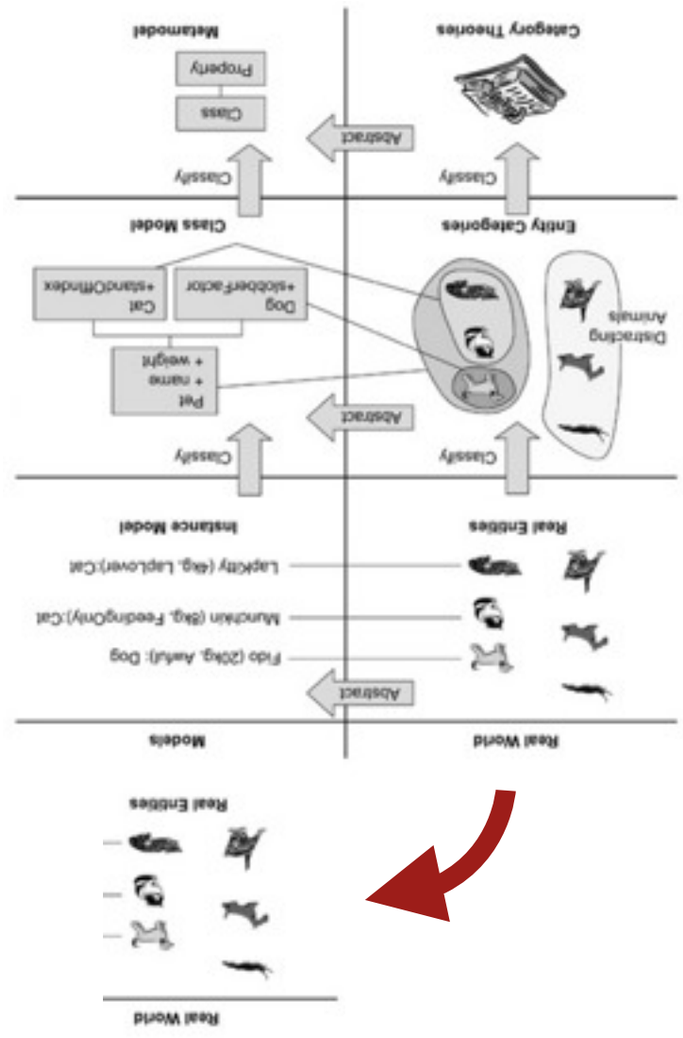
Meta-Modeling – Meta-Modeling-Hierarchy



Mellor, Scott, Uhl, Weise: *MDA Distilled* (Addison-Wesley, 2004)

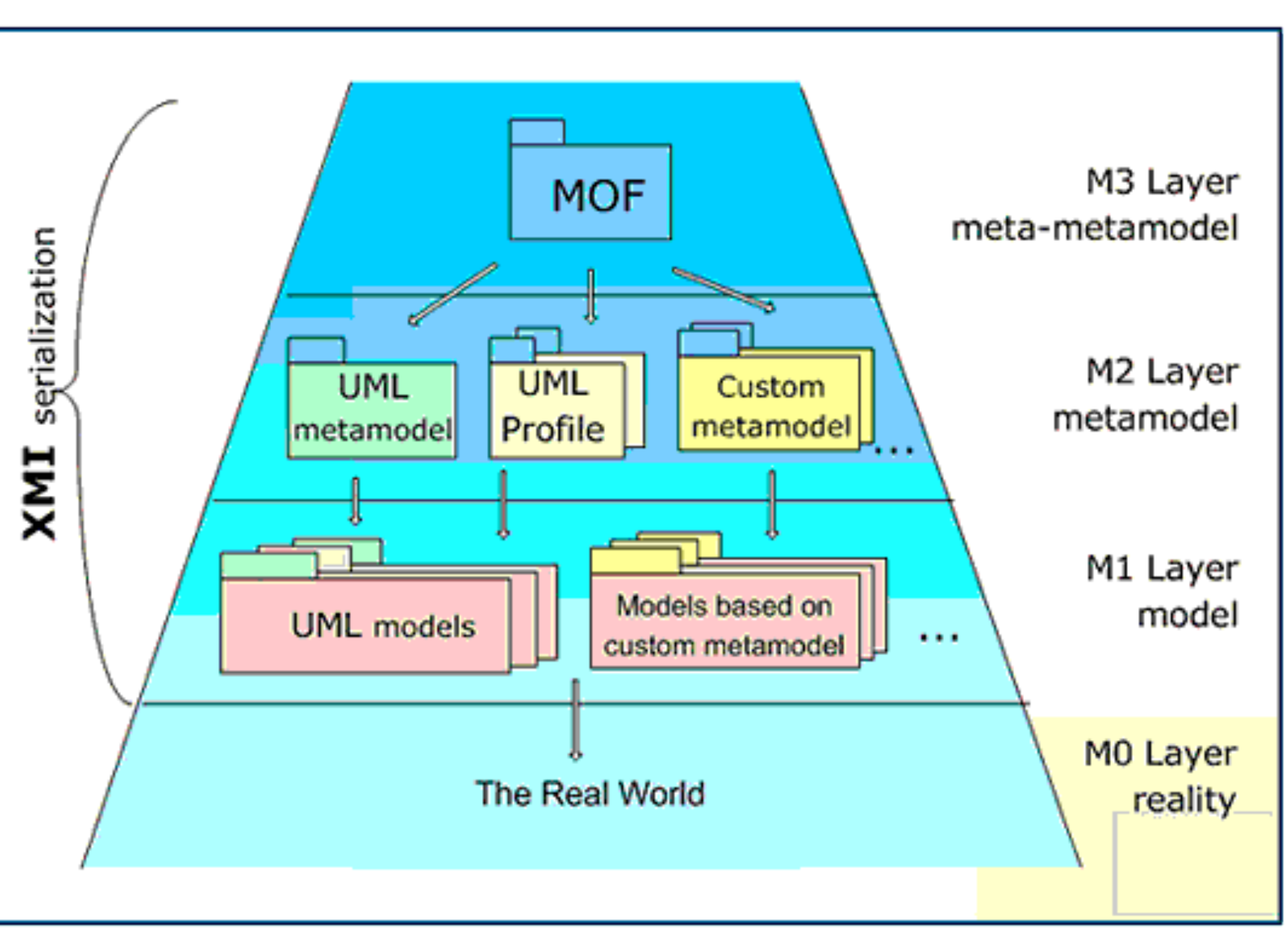
Meta-Modeling – M4 Model

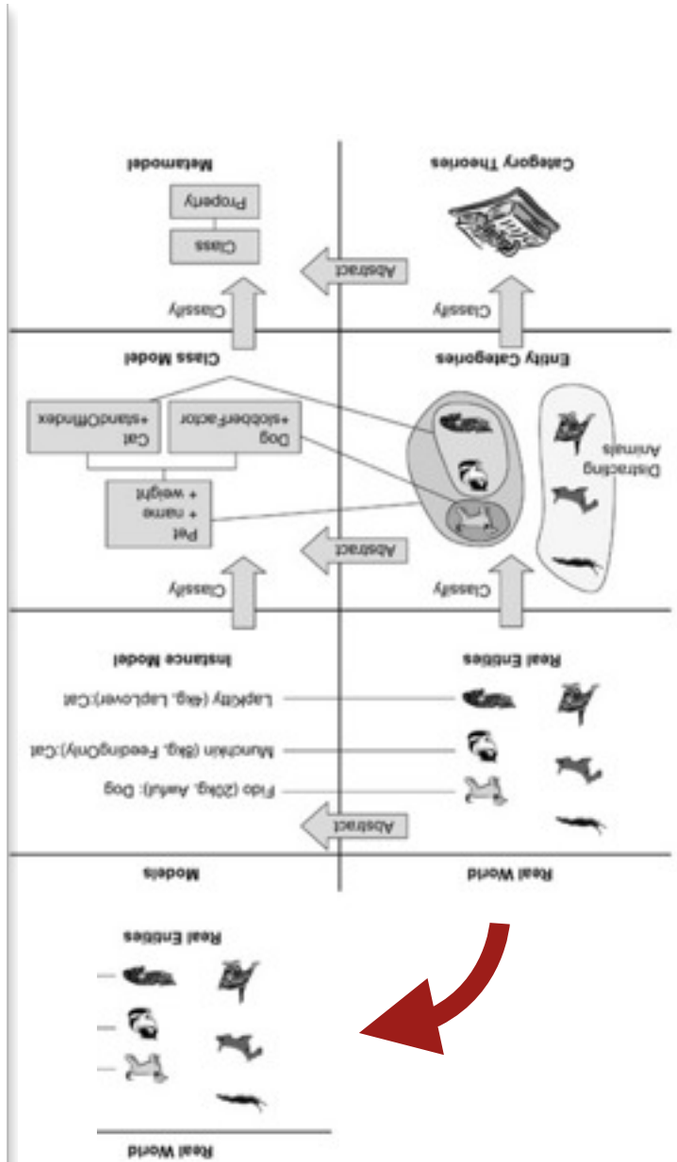




Object Management Group (OMG): *Meta Object Facility (MOF)*

Meta-Modeling – M4 Model





Object Management Group (OMG): *Meta Object Facility (MOF)*

Summary

- Real world things, objects
- Models
- Meta-Models
- Meta-Meta-Models
- M0-M3, M4-Model
- Instance-of relationship

Agenda

➡ prolog (1 VL) **Introduction:** languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model

0. (2 VL) **Eclipse/Plug-ins:** eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCP*s

1. (2 VL) **Structure:** *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*

2. (3 VL) **Notation:** Customizing the tree-editor, textural with *XText*, graphical with *GEF* and *GMF*

3. (4 VL) **Semantics:** interpreters with Java, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*

epilog (2 VL) **Tools:** persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System* (MPS)