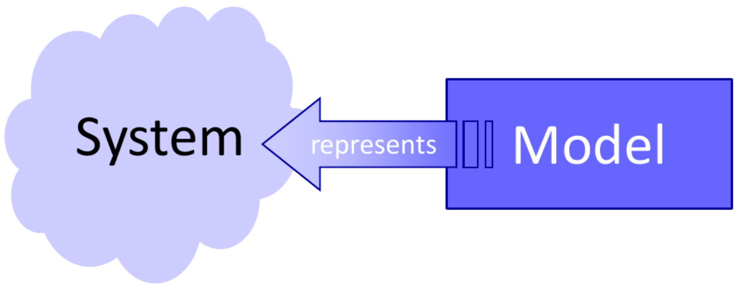
L1_M1 - Introduction

Model:
a simplified or partial representation of reality, defined in order to accomplish a task or to reach an agreement



Mapping Feature
A model is based on an original (=system)

Reduction Feature
A model only reflects a (relevant) selection/parts of the original's properties

Pragmatic Feature
A model needs to be usable in place of an original with respect to some purpose

Purpose can be classified into … prescriptive, descriptive, (predictive)

Never mistake the model for the reality
Attention: abstraction, abbreviation, approximation, visualization, …

Application area of modeling in Software Engineering
Models as drafts
Communication of ideas and alternatives
Objective: modeling per se

Models as guidelines
Design decisions are documented
Objective: instructions for implementation

Models as programs
Applications are generated automatically
Objective: models are source code and vice versa

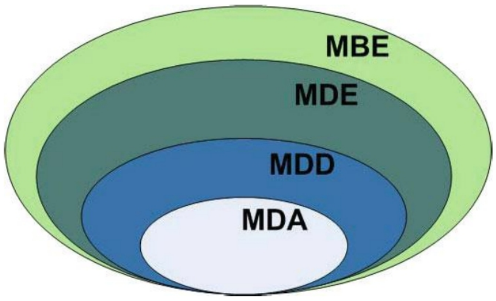
L1_M2 - MDSE Principles

- Model-Driven Software Engineering (MDSE) considers models as first-class citizens in software engineering

- Improved portability of software to new/changing technologies – model once, build everywhere
- Interoperability between different technologies can be automated

<u>**Models + Transformations = Software**</u>

Model-driven Architecture (MDA) is the particular vision of MDD proposed <u>by the Object Management Group</u> (OMG)



Model-Driven Development (MDD) is a development paradigm that uses <u>models as the primary artifact</u> of the development process

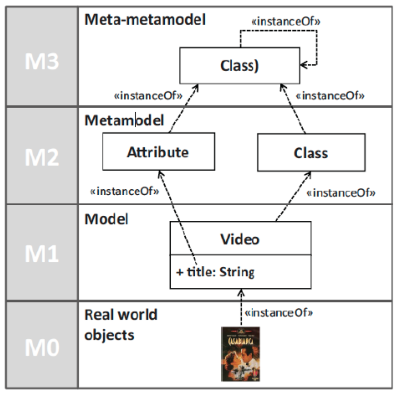
Model-Driven Engineering (MDE) is a superset of MDD
because it goes beyond of the pure development for example deployment artifacts

Model-Based Engineering (or “model-based development”) (MBE) is a softer version of MDE, where models do not “drive” the process

<u>**Modeling Languages:**</u>
<u>1) Domain-Specific (Modeling) Languages (DS(M)Ls)</u> are languages that are designed specifically for a certain domain
- DSLs have already an extensive use in in computer science
- Examples: HTML, SQL (SQL works only in the domain of relational databases)

<u>2) General Purpose (Modeling) Languages (GP(M)Ls)</u> are languages that can be applied to any domain for (software- related) modeling purposes
- Examples: UML, Petri nets, logic

<u>**4 Metamodeling Layers**</u>
M3 (Meta-Metamodel)
M2 (Metamodel)
M1 (Model)
M0 (Model instance)



model mapping, model correspondences, or model weaving ?
- transformations themselves can be produced automatically by <u>higher-level mapping rules between models</u>
- defining a mapping between elements of a model to elements of another one

<u>**Model Transformations**</u>
Transformations themselves can be seen **as models**! → Leads to higher-order transformations

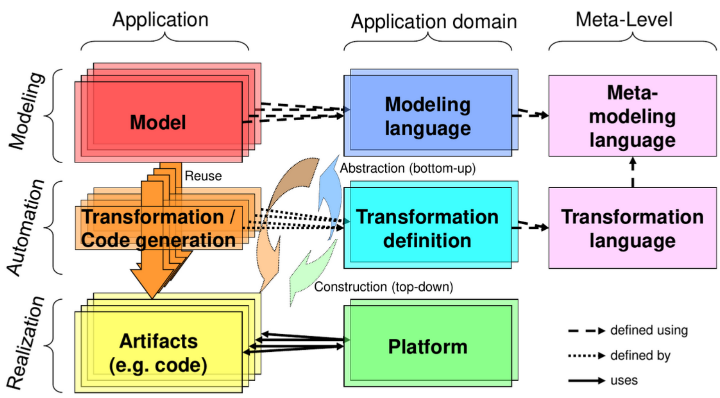


Figure 1: Model (Driven) Engineering - the basic architecture

<u>**Types of models**</u>

Models can be classified based on different dimensions

Concrete syntax: textual, graphical, hybrid
Content: requirements, design, deployment
Purpose: predictive, prescriptive, descriptive
...

In MDSE, the following **distinction** is useful

Static models: Focus on the static aspects of the system in terms of managed data and of structural shape and architecture of the system

Dynamic models: Emphasize the dynamic behavior of the system by showing the execution possibilities

Model Driven Architecture - MDA

Interoperability through platform independent models (PIM)

Standardization initiative of the Object Management Group (OMG), based on OMG standards, particularly UML, MOF

Counterpart to CORBA on the modeling level: interoperability between different platforms

Modifications to the basic architecture - Separation of the model level

Platform Independent Models (PIM): valid for a set of (similar) platforms

Platform Specific Models (PSM): special adjustments for one specific platform

Requires model-to-model transformation (PIM-PSM) and model-to-code transformation (PSM-Code)

CIM, PIM, PSM ?

Computation Independent Models (CIM)

Describe requirements and needs at a very abstract level, without any reference to implementation aspects (e.g., description of user requirements or business objectives)

Platform Independent Models (PIM)

Define the behavior of the systems in terms of stored data and performed algorithms, without any technical or technological details

Platform-Specific Models (PSM)

Define all the technological aspects in detail

Advantages/Disadvantages of Model Driven Architecture (MDA) ?

Advantages

Standardization of the Meta-Level
Separation of PIM and PSM (reuse)

Disadvantages

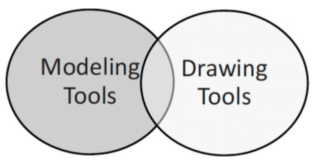
No special support for the development of the execution platform and the modeling language

Modeling language practically limited to UML with profiles

Therefore limited code generation (typically no method bodies, user interface)

Drawing vs. modeling

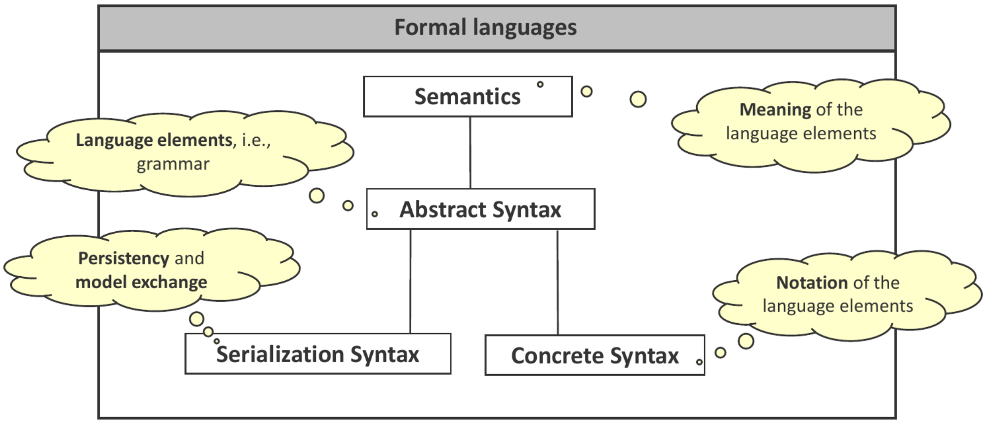
- Models are not "just" pictures!
- Drawing Tool can not tell you if the Model is valid



Difference between Drawing Tool and UML Editor/Modeling Tool:
Drawing Tool does not know syntax or semantic and the rules behind, if a class is drawn (rectangle with label)

L2_M1 – METAMODELING – METAMODELING INTRODUCTION

Semantic, Syntax ?

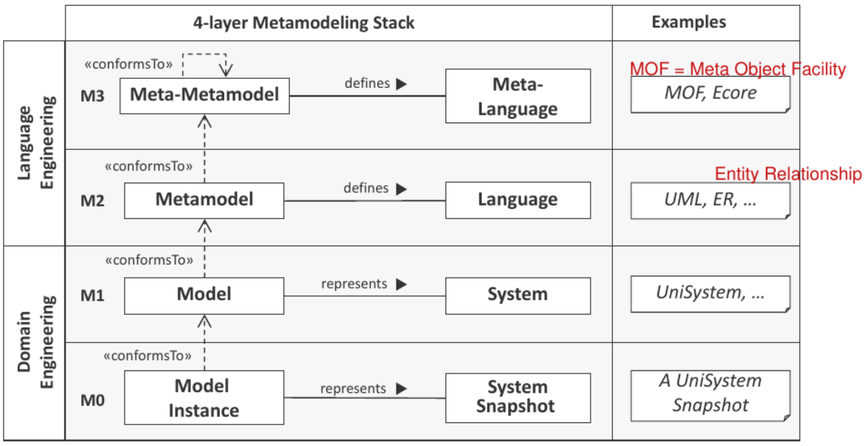


Abstract syntax: Defines the language concepts and how these concepts can be combined (~ grammar)
However, it **does not define** the **notation** or **meaning** of the concepts
Concrete syntax: **Notation** to illustrate the language concepts intuitively (2 ways: textual or graphical)
Semantics: Defines the **meaning** of the language concepts, how language concepts are interpreted
Serialization syntax: For persistent storage and model exchange between tools (XML)

- formal languages for the definition of languages – so-called **meta-languages**
- in grammarware (i.e., the technical space where languages are defined in terms of grammars)
- Examples for meta-languages: BNF, **EBNF**
- EBNF in M3 (Meta-Metamodel Layer): Definition of EBNF in EBNF – reflexive
- EBNF in M2 (Meta-Model Layer): Definition of Java in EBNF – grammar
- The abstract and the concrete syntax are defined

Metamodels define **language concepts** and their **grammar** for the specification of models
»meta« means »about« - hence a metamodel states something **»about«** other models

Generalization on a higher level of abstraction by means of the **meta-metamodel**
Language concepts for the definition of a metamodel
MOF (Meta Object Facility) is considered as a universally accepted **meta-metamodel**



Model/metamodel co-evolution problem

Metamodel is changed
Models already exist and may become invalid

Changes that may break conformance relationship

Deletion and renaming of metamodel elements

Conformance relationship covers different constraints

Example: The type of an object must exist in the metamodel, i.e., there has to be a class with the same name as given as type name of an object.

- meta-metamodel or metamodel language - a metamodel is also a model

---

L3_M1 - Object Constraint Language (OCL) – OCL Introduction

Graphical modeling languages are often not able to describe all facets of a problem description
MOF, UML, ER

Formal specification languages

Mostly based on set theory or predicate logic
Requires good mathematical understanding
Mostly used in the academic area, but hardly used in the industry
Challenging to learn and to apply
Problems when used for large systems

Object Constraint Language (OCL):
Combination of modeling languages and formal specification languages

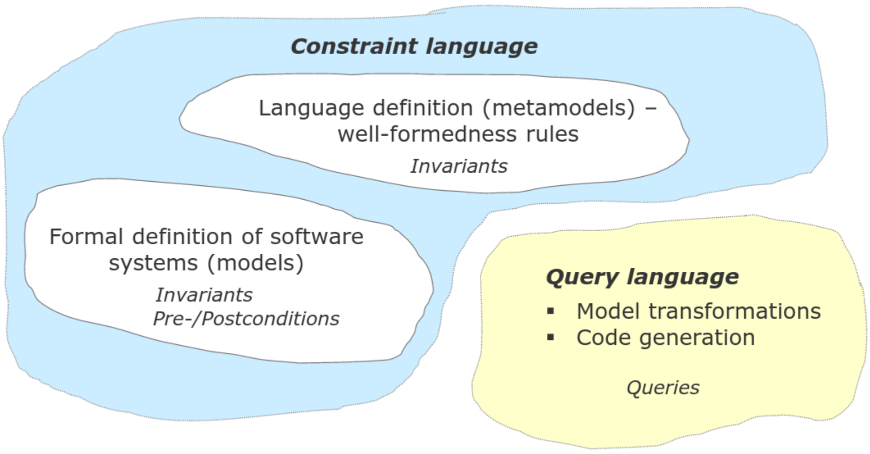
Not a programming language

OCL usage
1) Constraints in UML models
2) Constraints in metamodels

Invariants for metamodel classes
Derived attributes and references for metamodel classes
Definition of well-formedness rules attached to metamodels

3) Query language
similar to SQL for DBMS, XPath and XQuery for XML
Often used in transformation languages

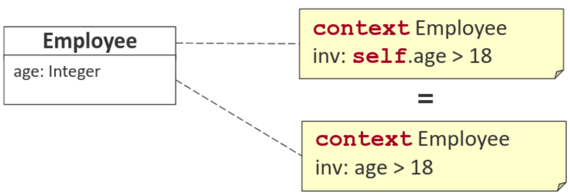
<u>OCL: fields of application</u>
Invariants
Pre-/Postconditions
Query operations
Initial values
Attribute/operation definition
<u>Side effects are not allowed in OCL!</u> Only get methods are allowed, no set methods



<u>Invariant ?</u>
- OCL invariants are OCL expressions that **return a Boolean value** indicating whether a model element fulfills the invariant
- Invariant means that this is always true, must be always true
- Invariant of TU Wien -> we are all humans, this is true for all in the TU

<u>On which level is a OCL constraint defined and evaluated?</u>
On model but also on meta model level
OCL is defined on the model level and evaluated/executed on the instance level

A **context** has to be assigned to each OCL-statement
Starting address – for which model element is the OCL statement defined
Specifies which model elements can be reached using path expressions



<u>OCL can be shown in two different ways</u>
As a comment
Separate document file

<u>OCL is a typed language</u>
<u>Predefined types</u> (Integer, Boolean, Set, Bag) and <u>User-defined Types</u> (Instances of Class in MOF)

OCL Expressions:

Each OCL expression has a typed return value
OCLConstraint is an OCLExpression with Boolean return value

Abstract syntax of OCL is described as a metamodel

OCL Standard Library – Types:

oclIsTypeOf(type:OclType): Boolean

- True, if type is the type of obj (true only if **same direct type**)

context Student
self.oclIsTypeOf(Person) : false
self.oclIsTypeOf(Student) : true

oclIsKindOf(type:OclType): Boolean

- True, if type is a **direct or indirect supertyp** or the type of obj

context Student
self.oclIsKindOf(Person) : true
self.oclIsKindOf(Student) : true

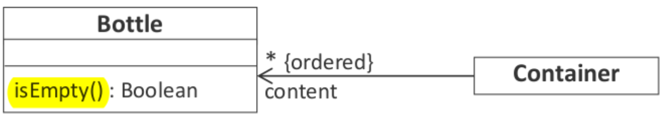
OCL is based on a **three-valued (trivalent) logic**
Expressions are mapped to the three values **{true, false, undefined}**

Undefined: Return value if an expression fails
1. Access on the first element of an empty set
2. Error during Type Casting

Collection is an abstract supertype for all set types
Caution: Operations with a return value of a set-valued type create a new collection (no side effects)

Set – no duplicates
Bag – with duplicates

Model operations vs. OCL operations



OCL-Constraint	*Semantik*
context Container inv: self.content -> first().isEmpty()	First bottle in each container instance has to be empty
context Container inv: self.content -> isEmpty()	Container instances must not contain bottles

Iterator-based operations:

select(exp) : Collection → return subset of collection, iterate over complete collection and collect elements
reject(exp) : Collection → return subset of collection, iterate over complete collection and collect elements
collect(exp) : Collection → returns a new collection from an existing one. It collects the Properties of the objects and not the objects itself. **Result of collect always Bag<T>**

iterate(...) – Iterate over all elements of a Collection (Generic operation)

L3_M2 - Object Constraint Language (OCL) – OCL Tools Examples

---

L4_M1 – Textual Modeling Languages

- One abstract syntax may have multiple concrete syntaxes (one to many relation)

Textual Languages:

- **Generic Syntax**
  Generic serialization of models
  Advantage: Metamodel is sufficient, i.e., no concrete syntax definition is needed
  Disadvantage: Syntax is generic and not domain-specific; no syntactic sugar
  Example: XMI (OMG Standards)

- **Language-specific Syntax**
  Example framework: Xtext (Eclipse plug-in)

Metamodel First
(this is done in the lecture - assignments - first assignment metamodel was specified and now in second assignment Step 2)
Step 1: Specify metamodel
Step 2: Specify textual syntax
Grammar First
Step 1: Syntax is specified by a grammar (concrete syntax & abstract syntax)
Step 2: Metamodel is derived from grammar

L4_M2 – Textual Modeling Languages - Xtext

Scoping

```
IScope scope_[EClassName]_[EReferenceName](MyType context, EReference ref)
```

- [EClassName]: Name of the class defining the reference
- [EReferenceName]: Name of the reference
- MyType context: Parse context (Type of element in which the reference is to be set)
- EReference ref: Targeted reference

Example:

```
public IScope scope_Entity_extends(Entity entity, EReference eReference)
```

- Scoping enables the definition of the references visibility in a Xtext grammer

---

L5 M1 - Graphical Modeling Languages - Introduction

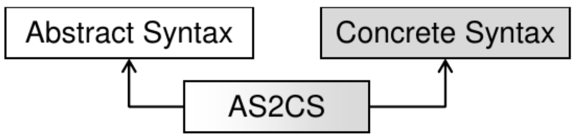
```
public IScope scope_Transition_state(Transition transition, EReference eReference) {
  // Self transitions are not allowed
  if (eReference.equals(StatemachinesPackage.Literals.TRANSITION__STATE))
        return Scopes.scopeFor(getAllowedStates(transition));
  return IScope.NULLSCOPE;
}
```

L5 M2 - Graphical Modeling Languages - Graphical Concrete Syntax

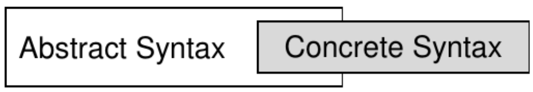
3 Graphical Concrete Syntax Approaches

- Mapping-based Approach
- Annotation-based Approach
- API-based Approach

Mapping-based Approach
Explicit mapping model between abstract syntax (i.e., the metamodel) and concrete syntax
Example: GMF (Graphical Model Framework from Eclipse), Sirius



Annotation-based Approach
The metamodel is annotated with concrete syntax information
Ecore metamodels are annotated with Graphical Concrete Syntax information
Example: Eugenia



API-based Approach
Concrete syntax is described by a programming language using a dedicated API for graphical modeling editors
Example: Graphiti