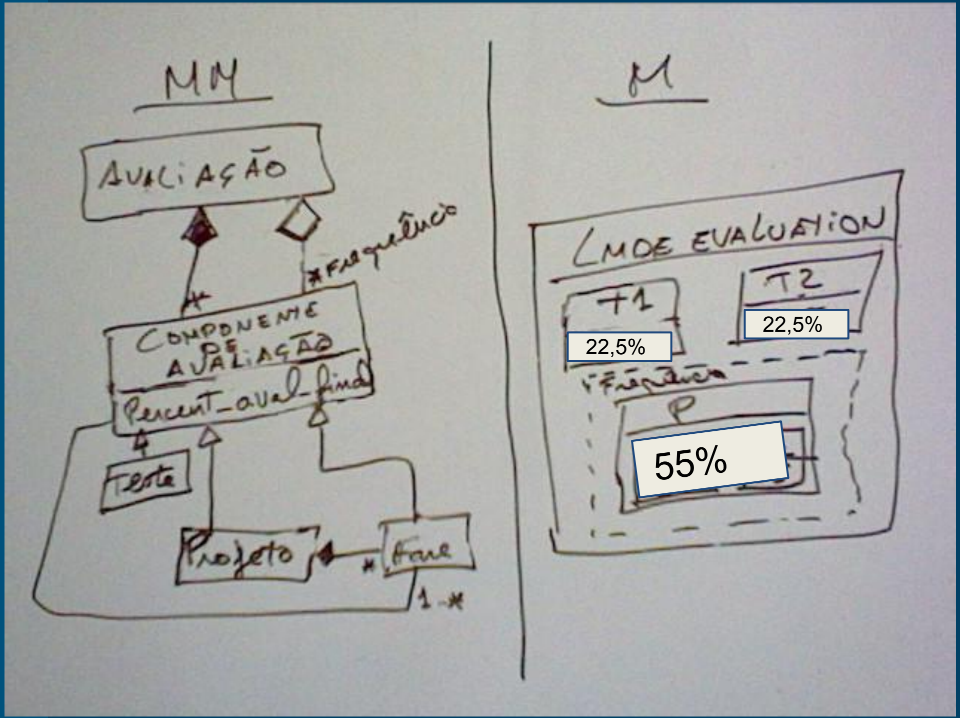
Model-Driven Engineering (MDE)
Engenharia Orientada a Modelos (EOM)

Lecture 0: Rules
by Prof. Vasco Amaral
2022/2023

Evaluation


MM
M
AVALIAÇÃO
COMPONENTE DE AVALIAÇÃO
Percent_aval_final
Teste
Projeto
Fase
1..*
LMDE EVALUATION
T1
T2
22,5%
22,5%
P
55%

Evaluation

NPF = 0,55 * NP

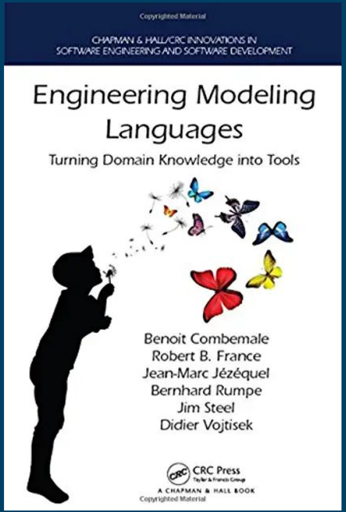
NTF = 0, 225 * NT1 + 0,225 * NT2

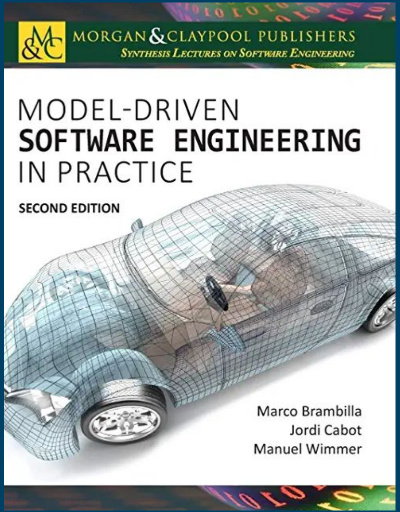
NF = NPF + NTF

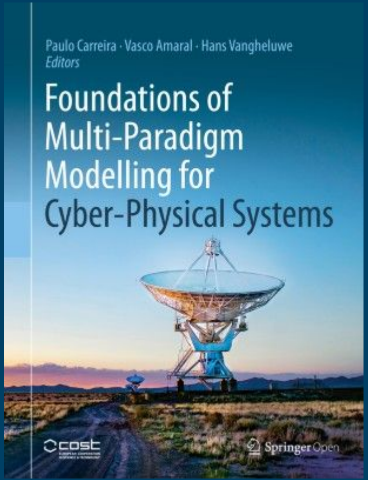
Moodle
https://moodle.fct.unl.pt/enrol/editinstance.php?courseid=7056

Enrolment key: EOM2223

Suggested Bibliography







Epsilon

01	Group Elements	19/10/2021
02	Project assignment	26/10/2021
03	First test	23/11/2021
04	Project Intermediate Submission	26/11/2021
05	Project Workshop	4/01/2022
06	Project Final Submission	5/01/2022
07	Second Test	12/01/2022

01 Group Elements

02 Project assignment

03 First test

04 Project Workshop

05 Project Final Submission

06 Second Test

Model-Driven Engineering (MDE)

Lecture 1: Models, Metamodels and the MD* landscape
by Prof. Vasco Amaral
2021/2022

Abstraction and Human Mind

The Human mind continuously re-works reality by applying cognitive processes

Abstraction and Human Mind

Abstraction: Capability of finding the commonality in many different observations.

- Generalize specific features of real objects
- Classify the objects into clusters
- Aggregate objects into more complex ones

Understanding and Predicting occurrences in the world

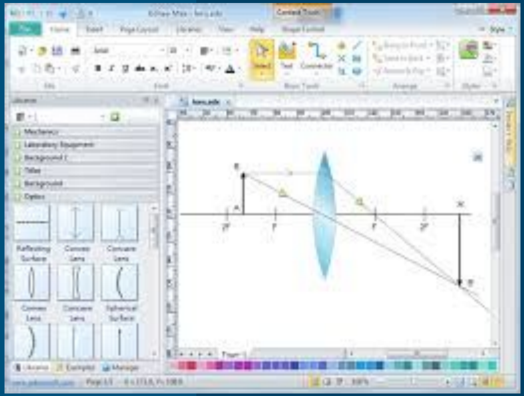
A human endeavour

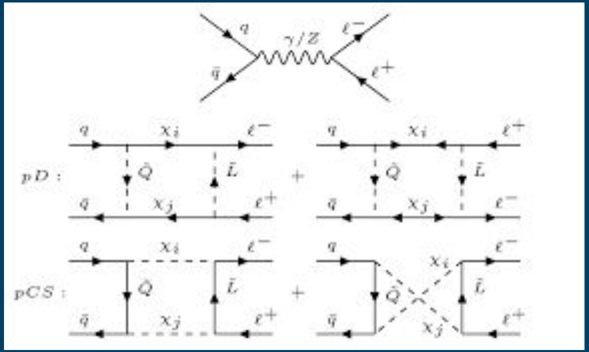
Models among us since the dawn of humanity

Humans Build models to:

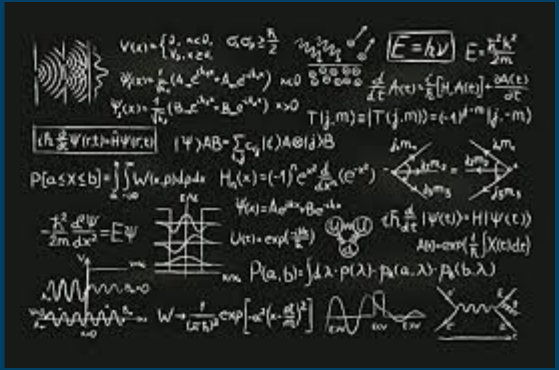
- Understand and predict occurrences in the world
- Help navigate through complex concepts
- Better understand their target of study

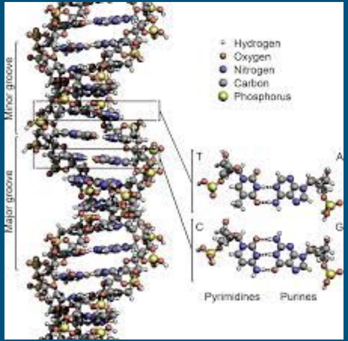
Physicists

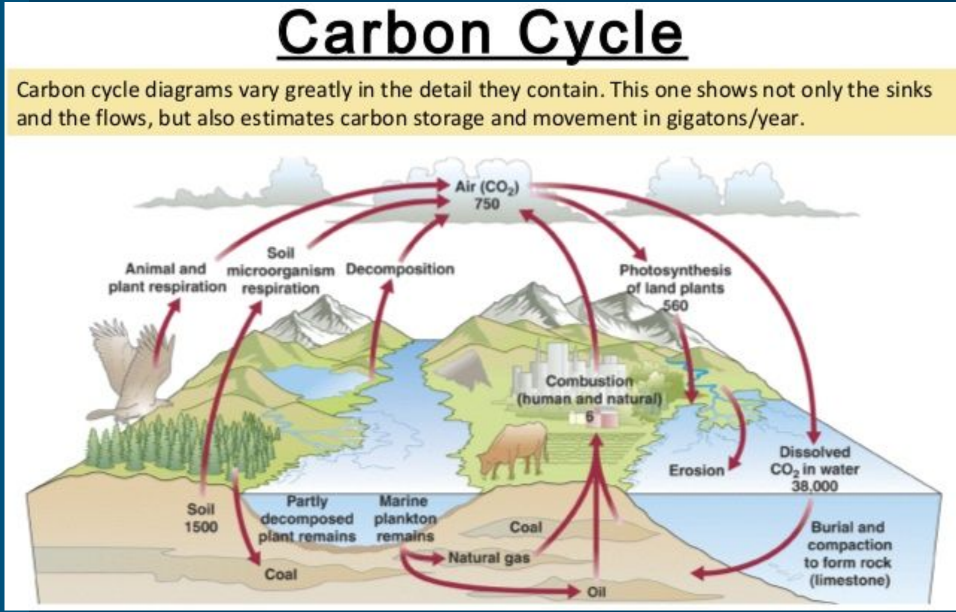




Mathematical models of a physical phenomenon

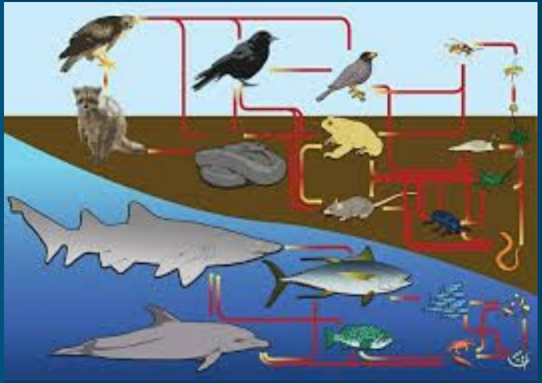


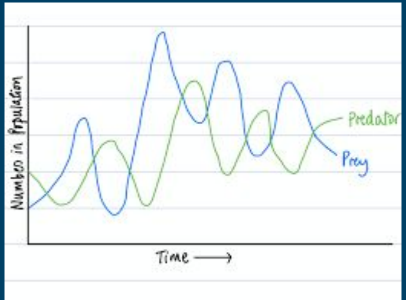


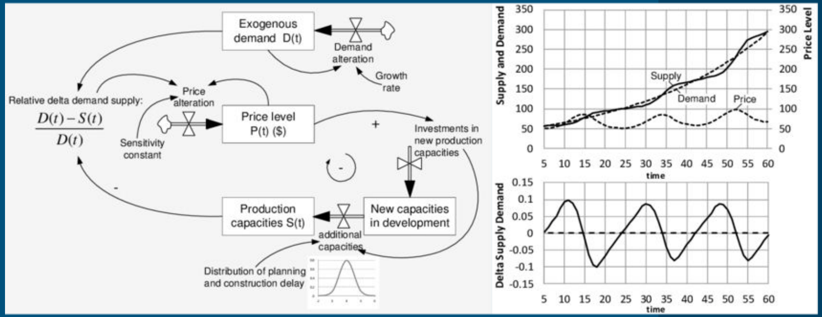


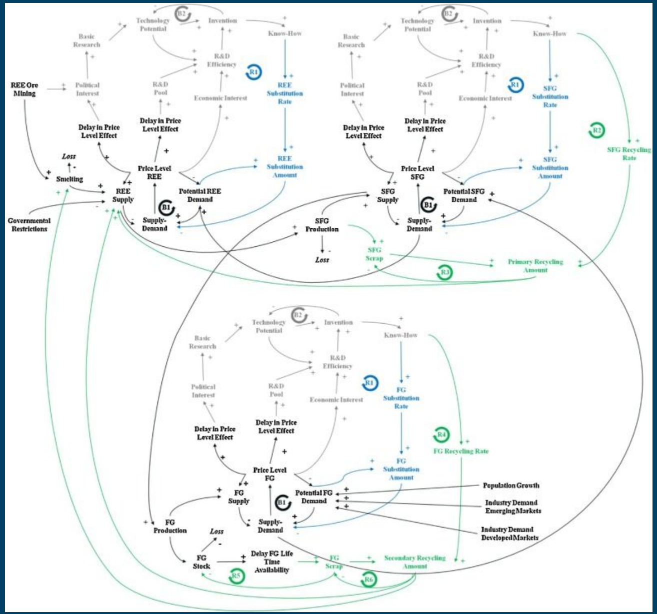
Biologists

model: the impact that a phenomenon has on particular species, cycles of materials in a ecosystem, etc...



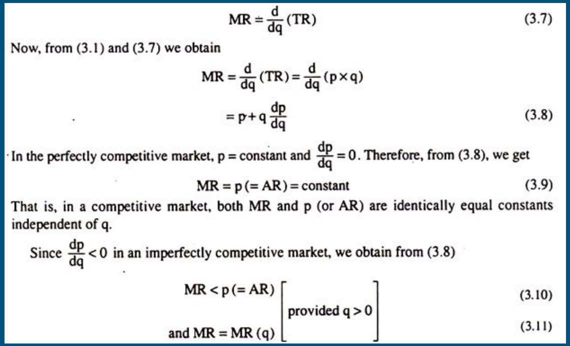






Economists

Model...



$$MR = \frac{d}{dq}(TR) \tag{3.7}$$

Now, from (3.1) and (3.7) we obtain

$$MR = \frac{d}{dq}(TR) = \frac{d}{dq}(p \times q)$$

$$= p + q\frac{dp}{dq} \tag{3.8}$$

In the perfectly competitive market, p = constant and $\frac{dp}{dq} = 0$. Therefore, from (3.8), we get

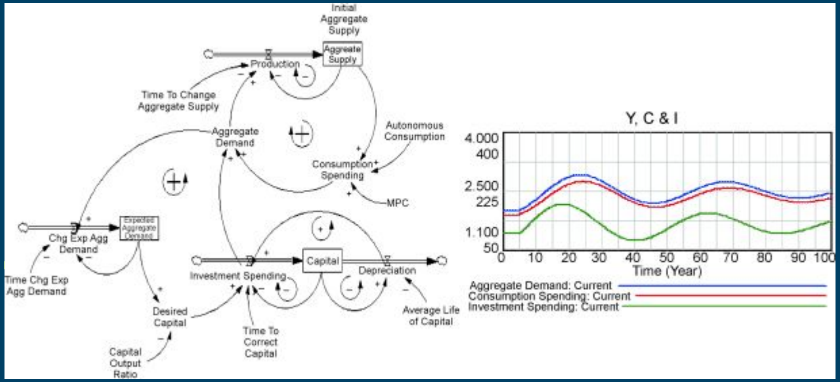
$$MR = p\,(= AR) = \text{constant} \tag{3.9}$$

That is, in a competitive market, both MR and p (or AR) are identically equal constants independent of q.

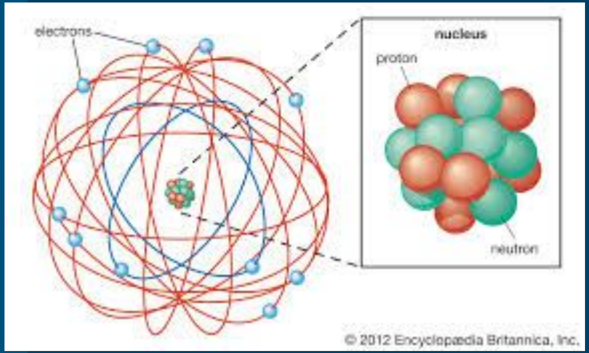
Since $\frac{dp}{dq} < 0$ in an imperfectly competitive market, we obtain from (3.8)

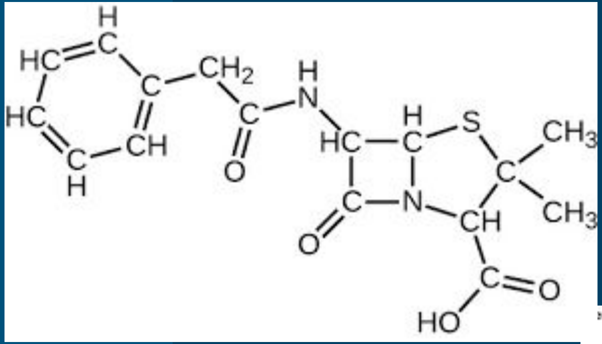
$$MR < p\,(= AR) \tag{3.10}$$

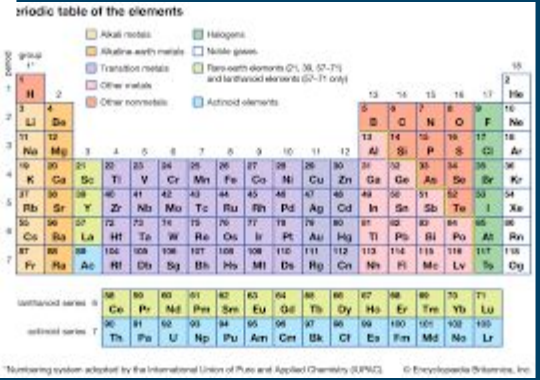
$$\text{and } MR = MR(q) \quad \left[\text{provided } q > 0\right] \tag{3.11}$$

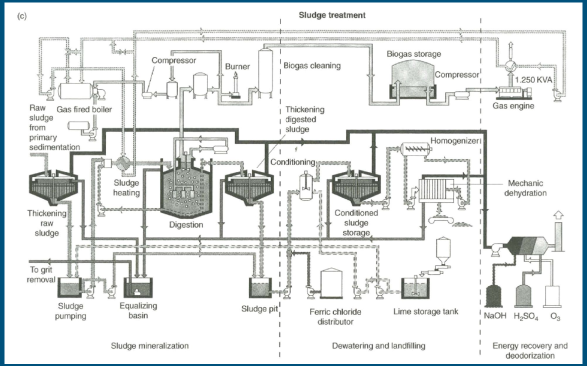


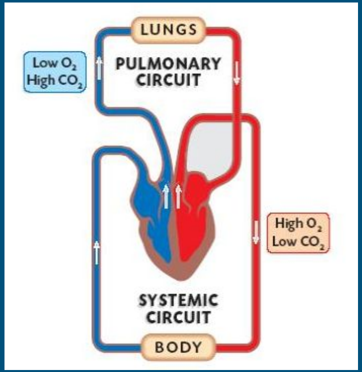
Chemical Engineers





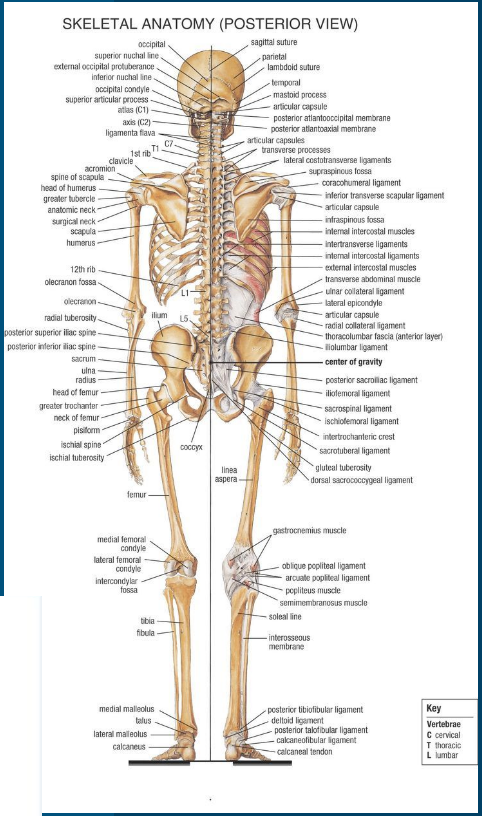


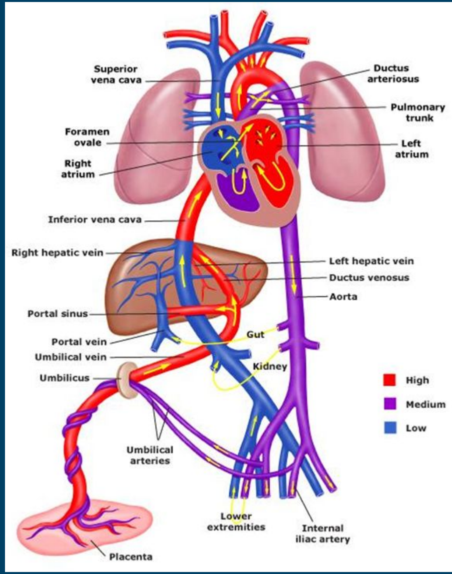


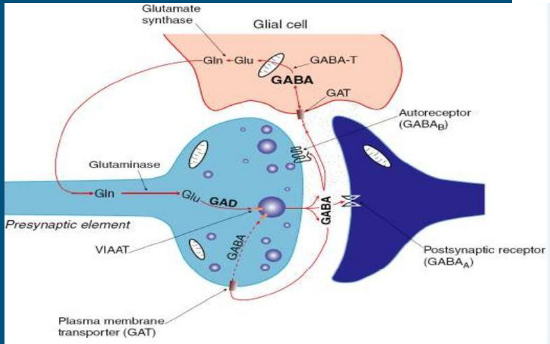


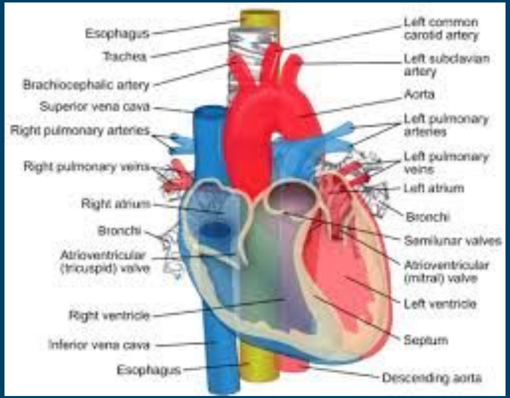
Physicians

Model human body

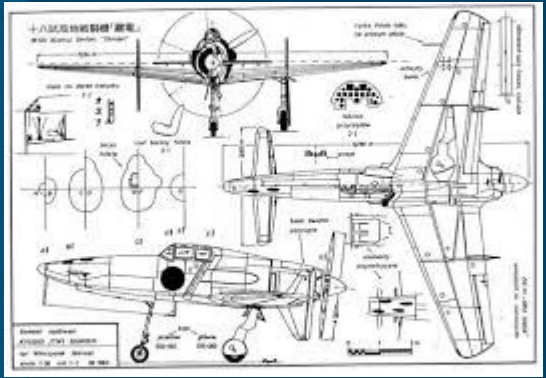


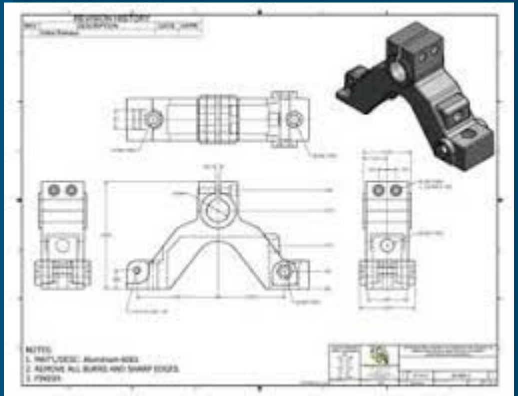






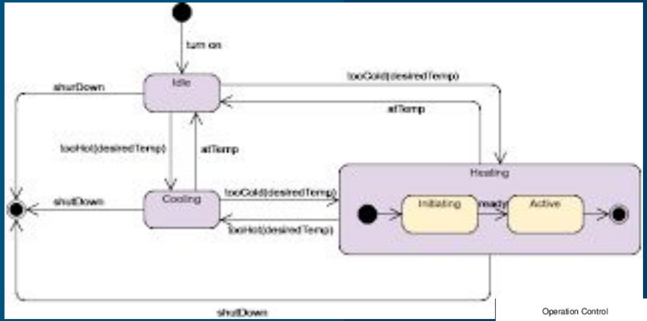
Mechanical Engineers

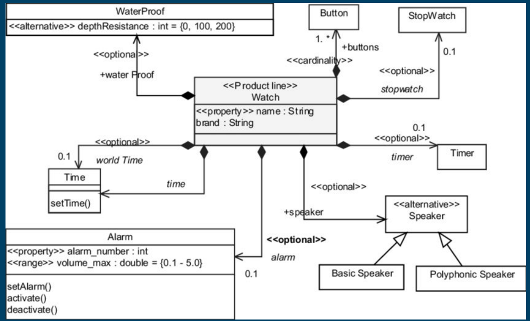


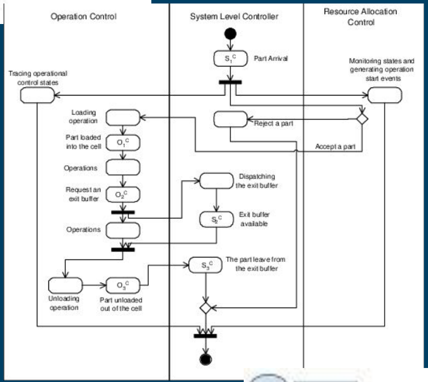


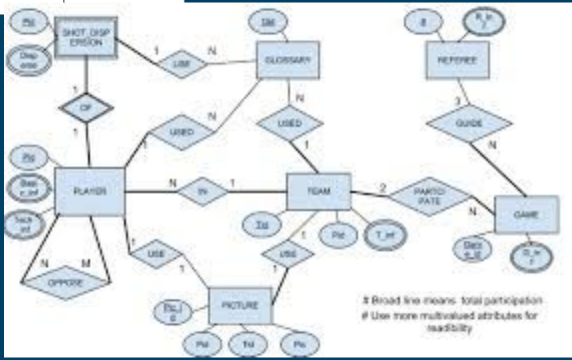
And so on…

But, hey... what about Software Engineering?!?









Also...

We will come to this topic later...

But, hey... what exactly is a Model?!?

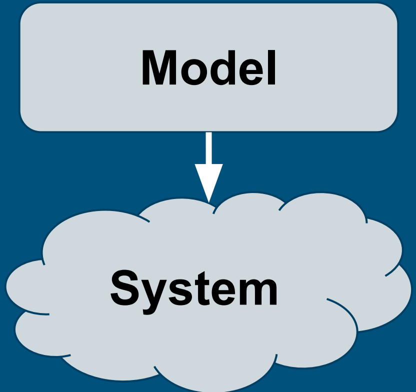
Models

What is a model?

A model is an abstraction (a simplified or partial representation) to make predictions or inferences about a reality or a given system under study (SUS).

Properties of a model

Stachowiak



mapping feature

based on an original.

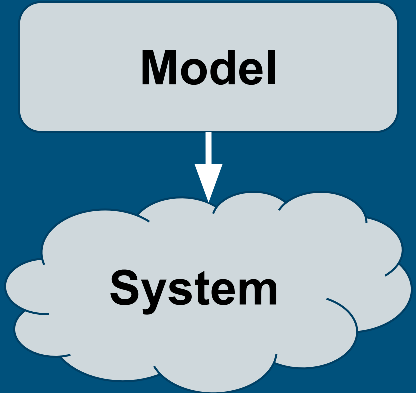
reduction feature

selection of an original's properties.

pragmatic feature

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

Properties of a model



Should **also be:**

Purposeful

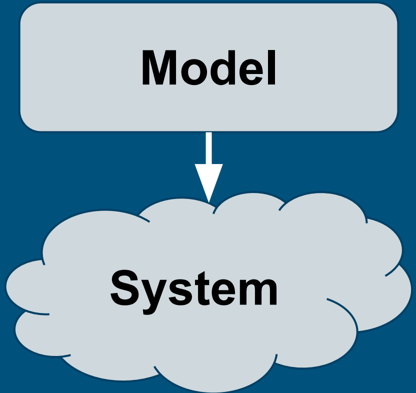
understandable

and **cost-effective**

But, hey... when and why should models be used?!?

Purposes of a model

Stachowiak



Descriptive - for describing the reality of a system or a context, used in Science to describe and predict existing phenomena of the real world.

Prescriptive - for determining the scope and details at which to study a problem, used in Engineering to describe a system to be built in the future.

Modelling in Science

Scientist use models to handle with the complexity of the phenomena

To be useful as means of communication, models need to be made **explicit**

Communicated in a language that can be understood

Modelling in Engineering

Engineers use models also to address complexity

The difference is that the phenomenon (engine, process, software building) generally does not exist at the time the model is built

Modelling in Engineering

Use of Abstraction and Separation of Concerns

Breaking down a complex system into many models in order to address all relevant concerns in a way that it is understandable, analyzable and constructable.

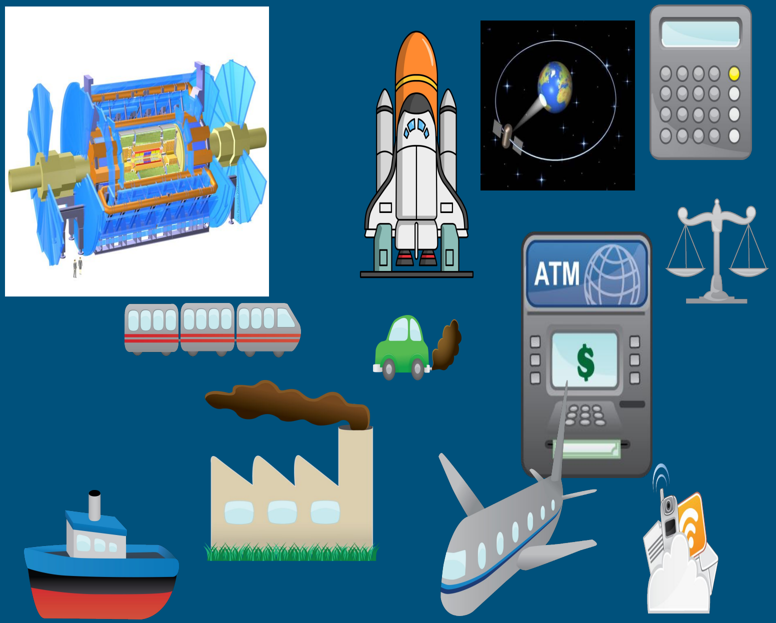
Main types of models in Software Engineering

Models that describe part/abstraction of the software to be built

Models that describe part/abstraction of the environment to be controlled, monitored, responded to, or augmented

Main purposes of models in Software Engineering

- Exploration of solutions alternatives
- Construction of the system
- Source for tests
- Help the customer to understand
- Generate code
- Customize the system
- Documentation
- Simulation



- Computers and software become tools in other domains
- Trend that domain experts define and use their own languages called Domain Specific (Modelling) Languages modelling phenomena not necessarily related to software
- There is the need for tool/methodological support to explicitly define languages
- Well defined languages enables development of reusable tools for modelling

Software Crisis

The term "**software crisis**" was coined by F. L. Bauer at the first NATO Software Engineering Conference in 1968

The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!

To put it quite bluntly: as long as there were no machines, programming was no problem at all;

when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

– Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM

"No Silver Bullet – Essence and Accidents of Software Engineering", Fred Brooks, 1986

"there is no single development (...) which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."

"we cannot expect ever to see two-fold gains every two years"

“No Silver Bullet – Essence and Accidents of Software Engineering”, Fred Brooks, 1986

Accidental Complexity

non essential to the problem solved.

Essential complexity

is inherent and unavoidable

Need for Industrial Revolution in SE

Interchangeable parts, introduced by John Hall

Assembly lines, introduced by Ransom Olds

Automated Assembly lines, (first industrial robot installed in 1961 by GM)

Craftsmen workshops

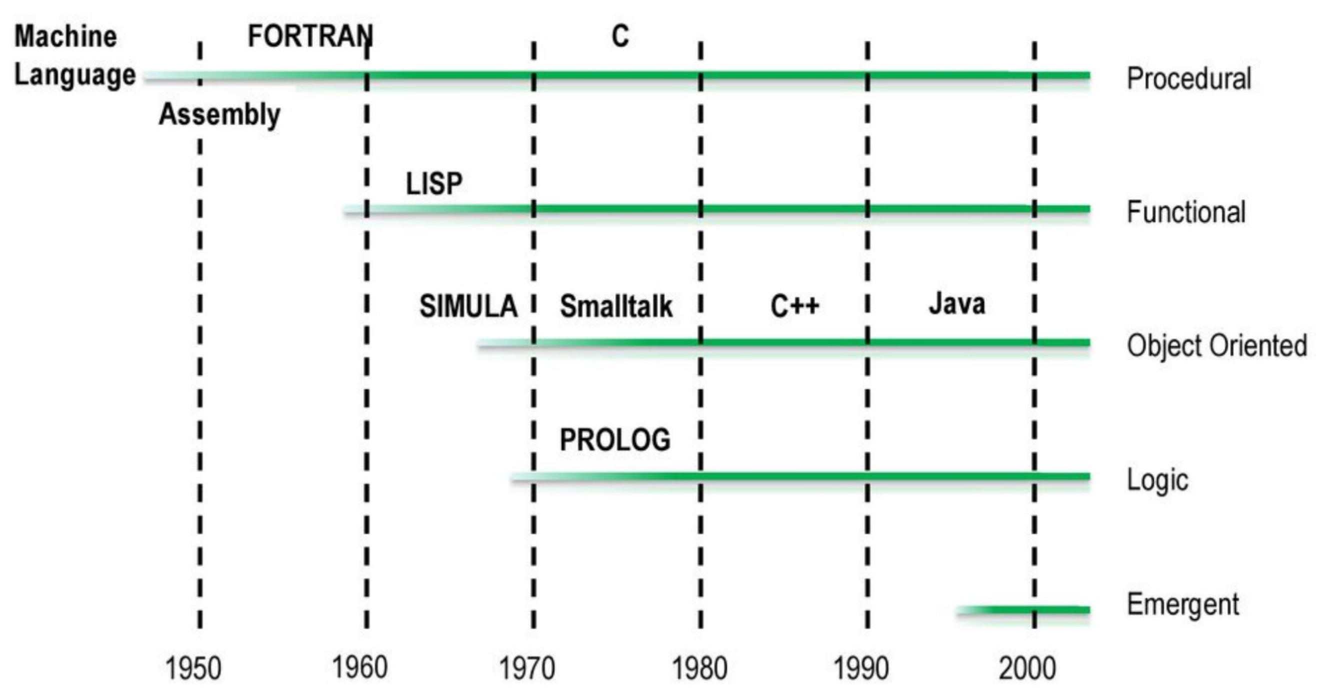
Factories

1826

1901

1980

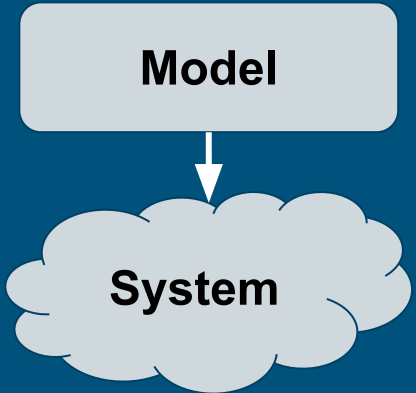
Software Factories, Jack Greenfield et.al, Wiley,2004
Generative Programming, Czarnecki. et al, Wesley 2000

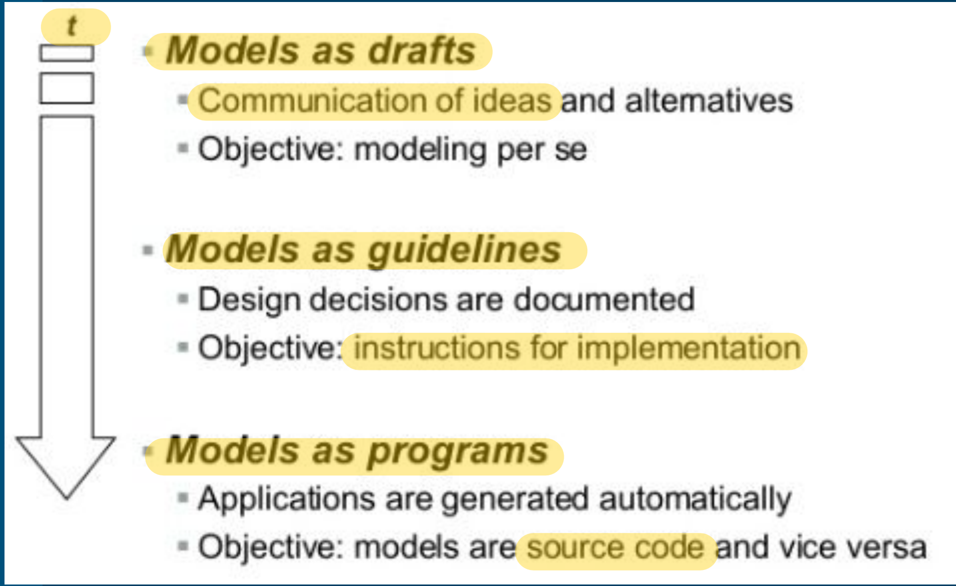


Machine Language
FORTRAN
C
Assembly
Procedural
LISP
Functional
SIMULA
Smalltalk
C++
Java
Object Oriented
PROLOG
Logic
Emergent
1950
1960
1970
1980
1990
2000

Use of Models

(levels of “commitment” to the system)





- ***Models as drafts***
  - Communication of ideas and altematives
  - 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

“no abstraction” → “no model”

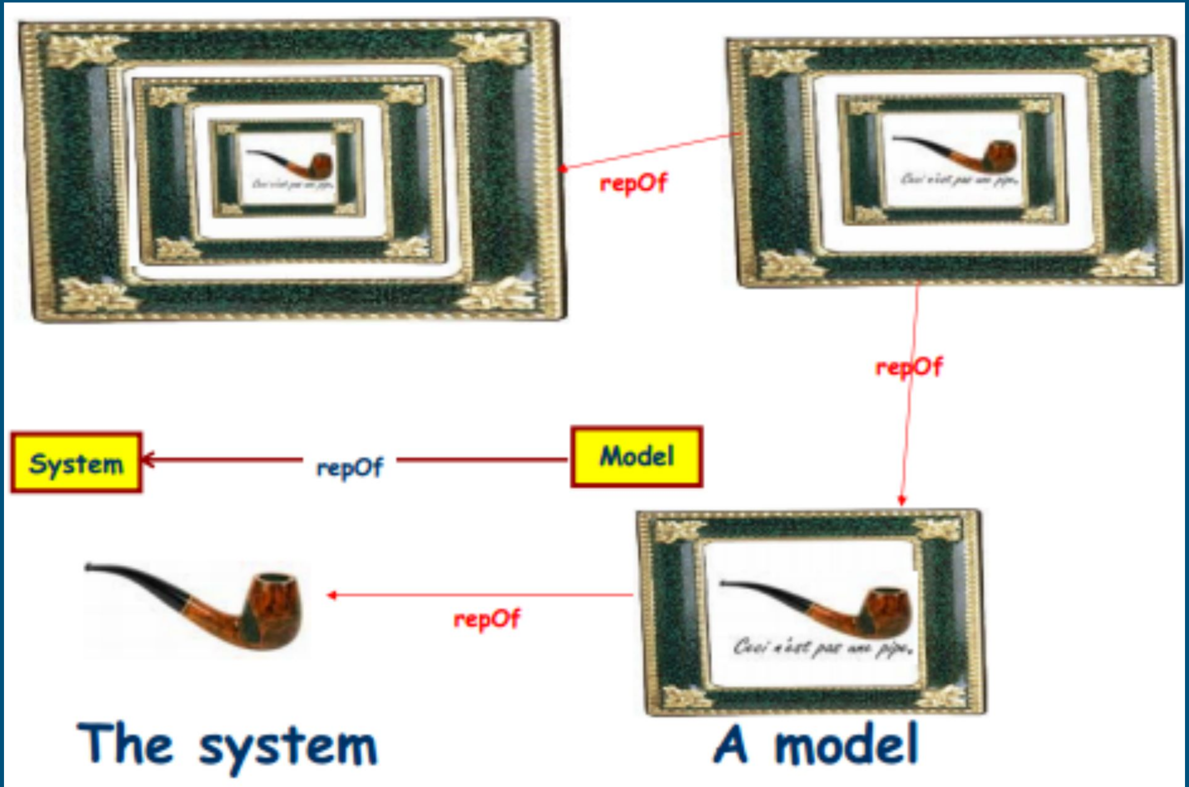
“no abstraction” → “no model”

- A copy is not a model
- Any representation of a real world subject automatically implies reduction and thus can be granted model status.

René Magritte

1928–29

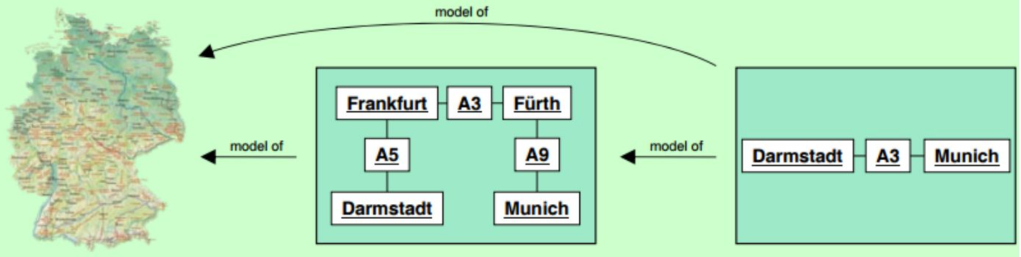
Difference between a model and System



Jean-Marie Favre:
Megamodelling and Etymology. Transformation Techniques in Software Engineering 2005

Kinds of Model Roles -Token models

Elements of a token model capture singular (as opposed to universal) aspects of the original's elements, i.e., they model individual properties of the elements in the system.

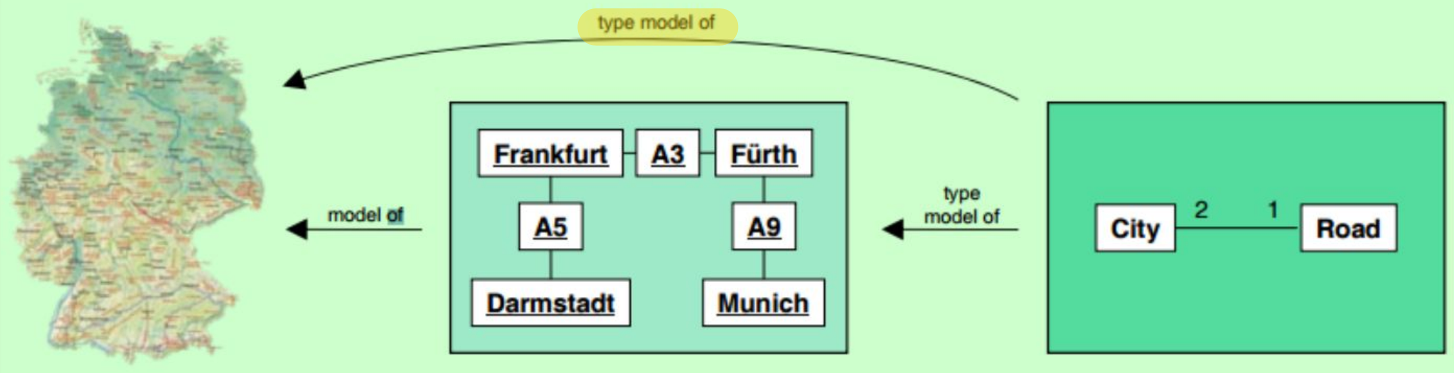


- anti-transitivity required

one-to-one representation of elements in the (relevant part of the) system.

Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

Kinds of Model Roles -Type models



- anti-transitivity required

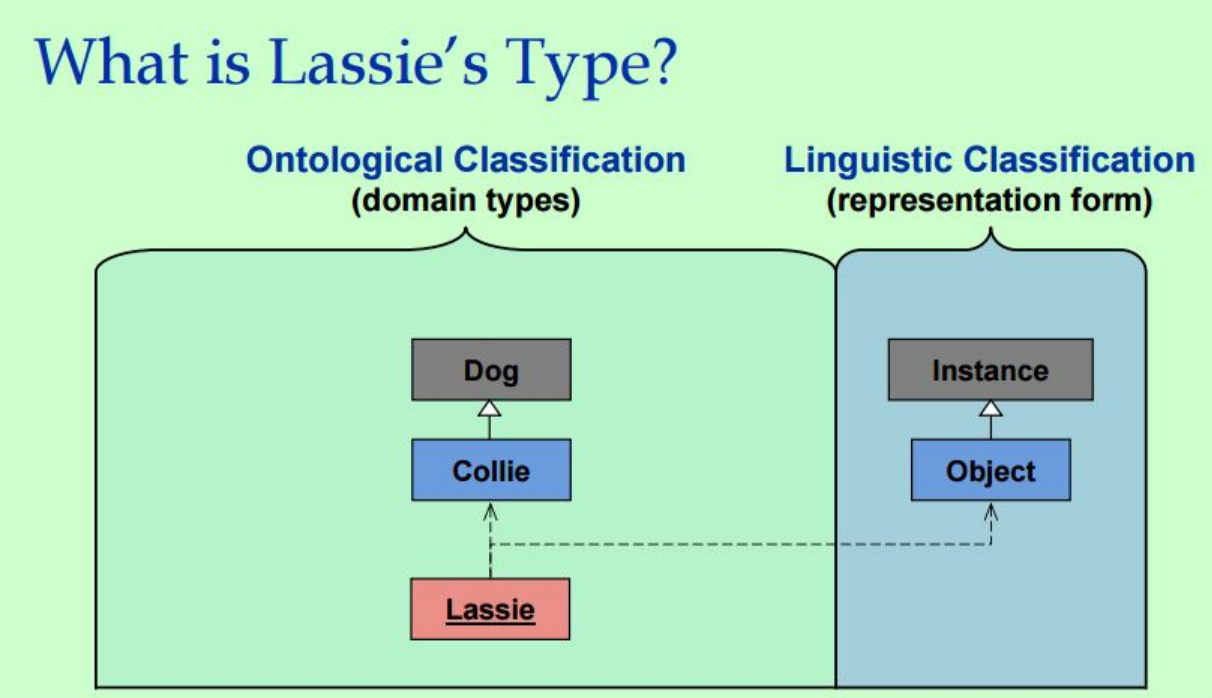
Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

A metamodel is...

Note that:

- a token model of a token model is not a metamodel.
- in order to create a metamodel we need a non-transitive relationship.

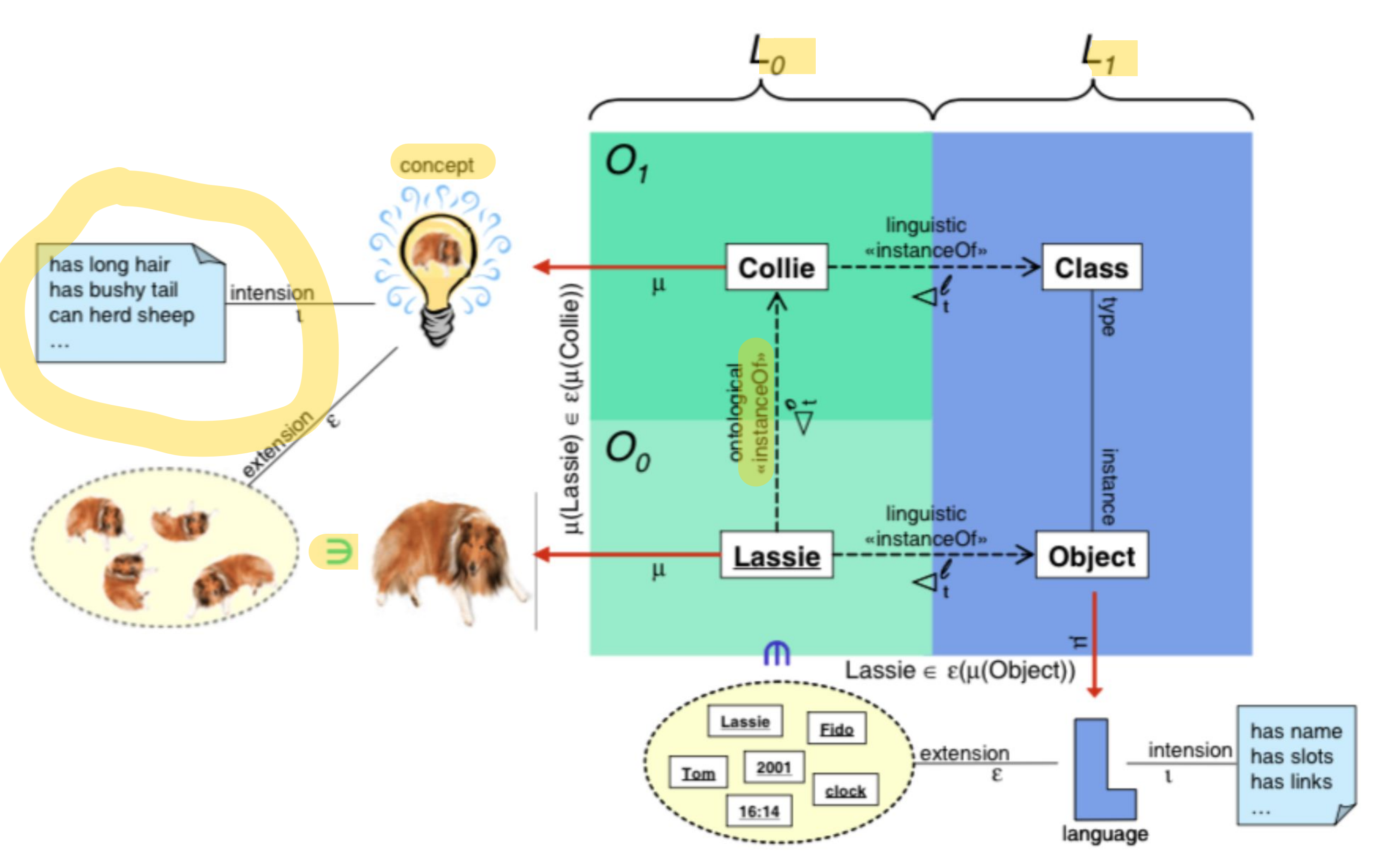
Ontological vs. Language



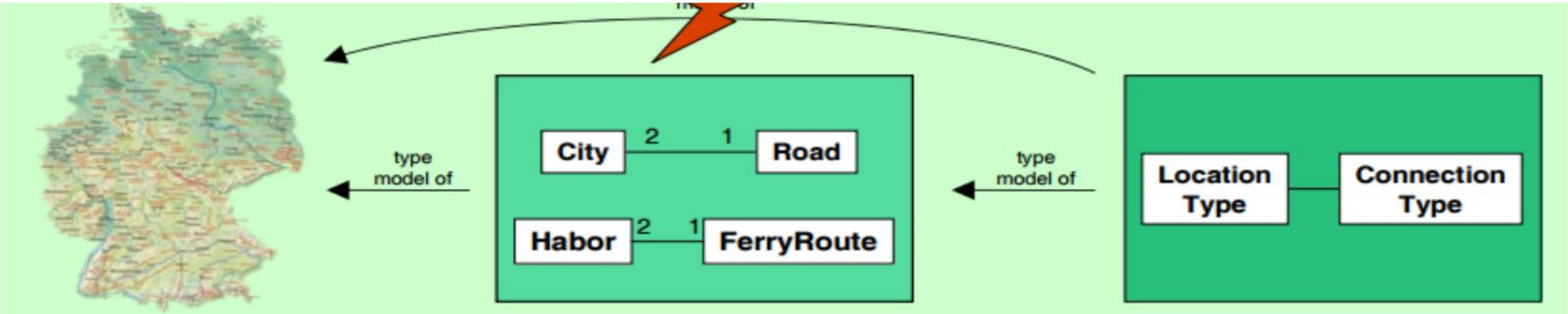
Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

Ontological vs. Language

Thomas Kühne: Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)



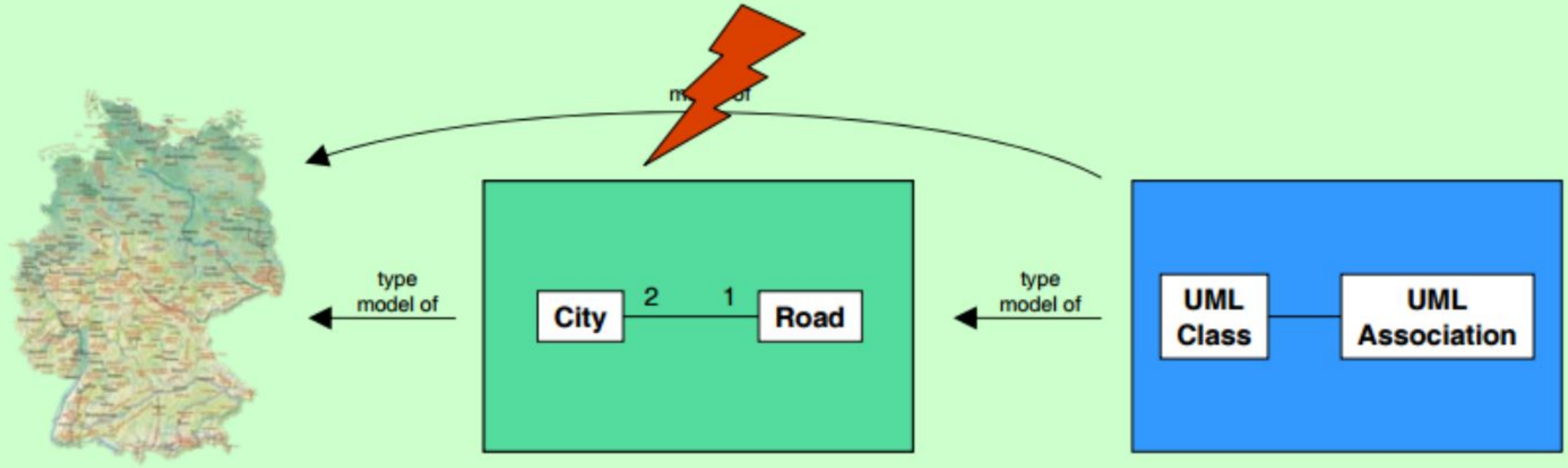
Kinds of Model Roles -Type models



- anti-transitivity through ontological types

Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

Kinds of Model Roles -Type models



- anti-transitivity through linguistic types

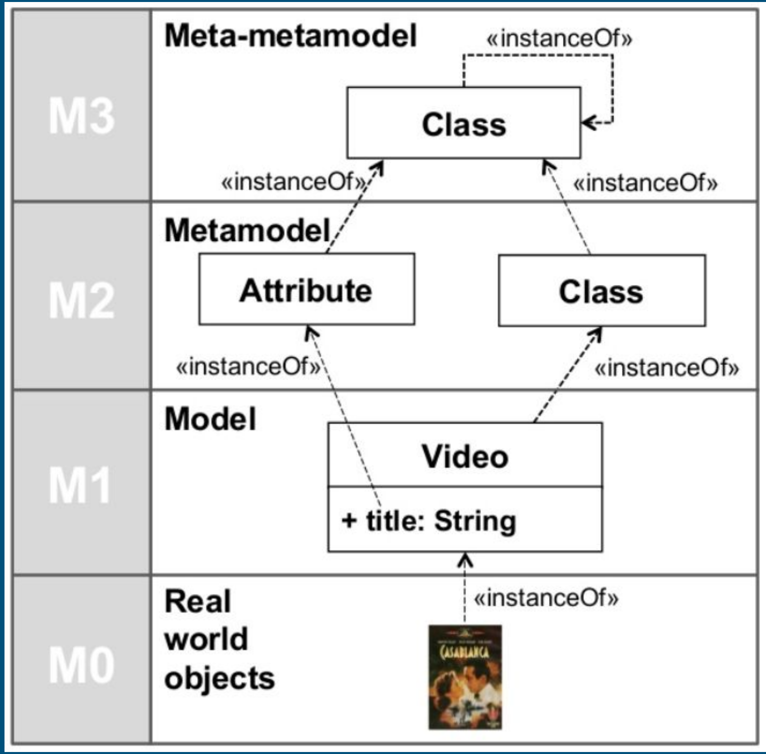
Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

A metamodel is...

A metamodel is a model of models (type-model with non-transitive relationship to the SUS)

- A model is an instance of a metamodel
- implies that a metamodel is a model of another model.

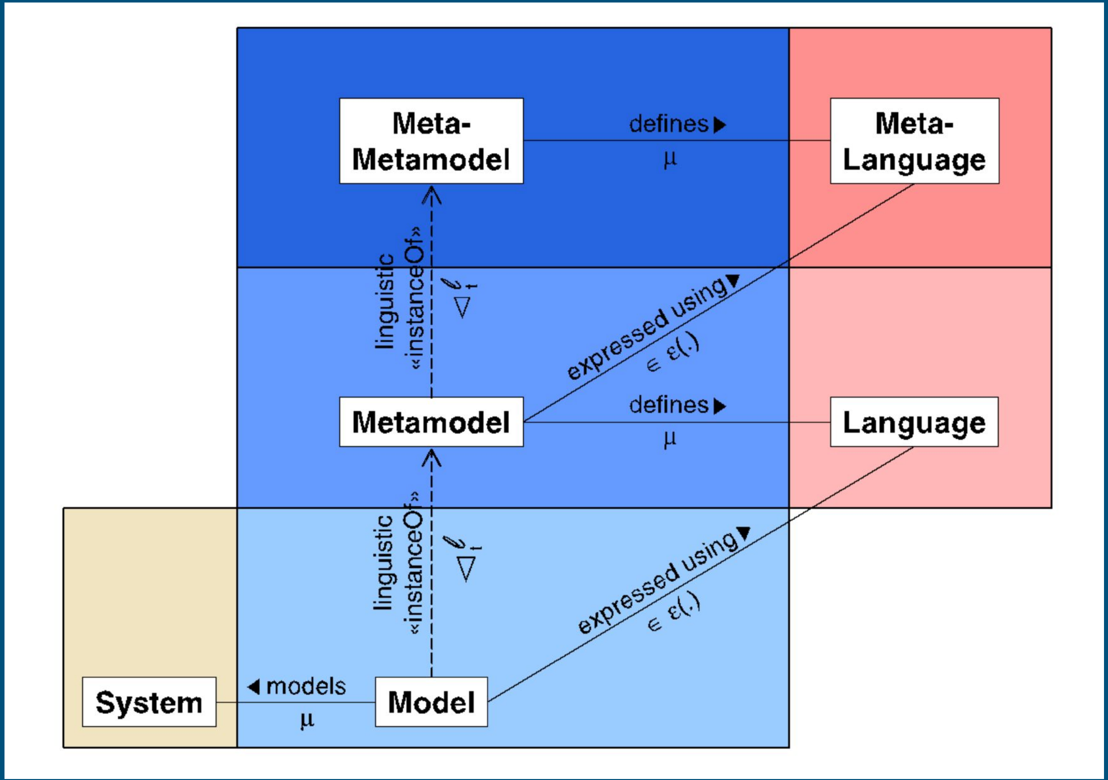
OMG's modelling stack



Can be used for:

- Defining new languages
- Defining new properties or features of existing information (metadata)

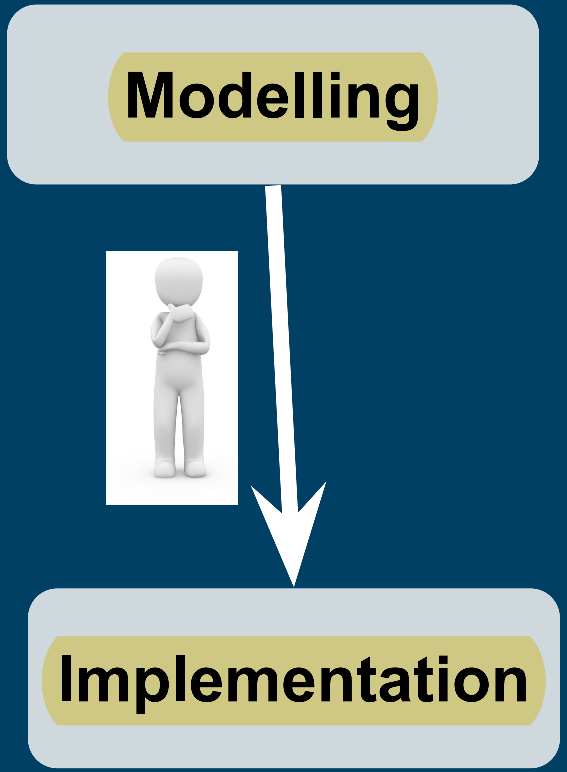
Language mechanism stack



Thomas Kühne:
Matters of (Meta-)Modeling. Software and System Modeling 5(4): 369-385 (2006)

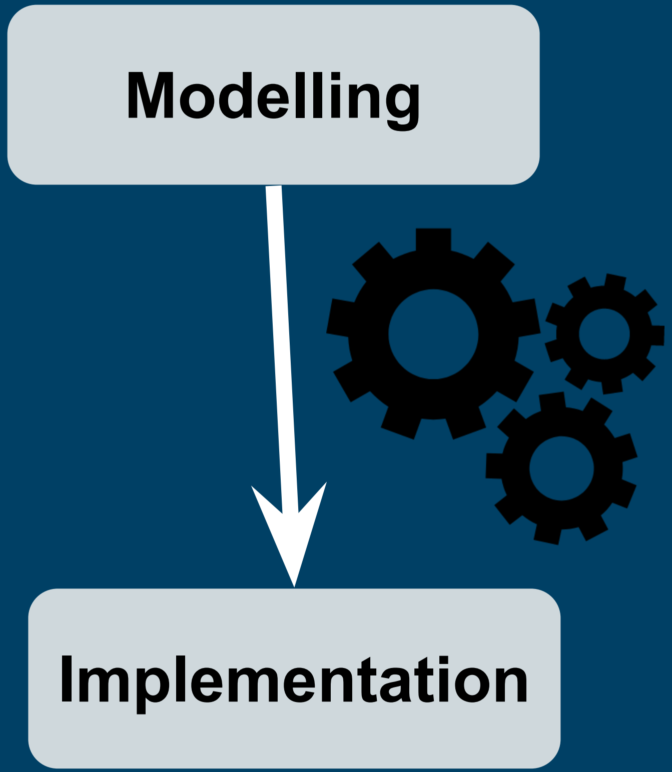
The Modelling Gap

The tradition...



Modelling Gap

Automation is needed



Model-Driven Engineering

MDE is a methodology for advantages of modelling to Software Engineering. Comprises:

- Concepts - components that build up the methodology
- Notations - how concepts are represented
- Process Rules - Activities that lead to the final product
- Tools - that ease activities and coordination

Model-Driven Engineering

Use of sound engineering approaches to the definition of models, transformations and development process.

Considers the models as first class citizens,

Software

In Programming (Wirth):

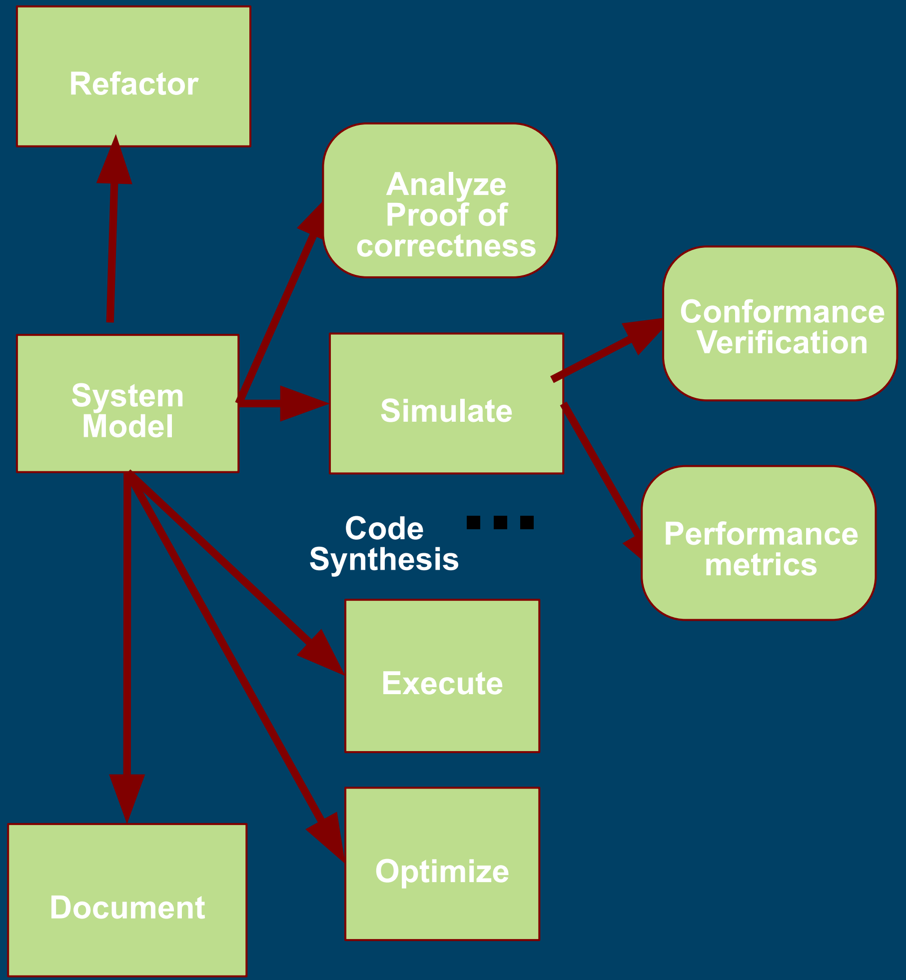
Algorithms + Data Structures = Programs

In MDE:

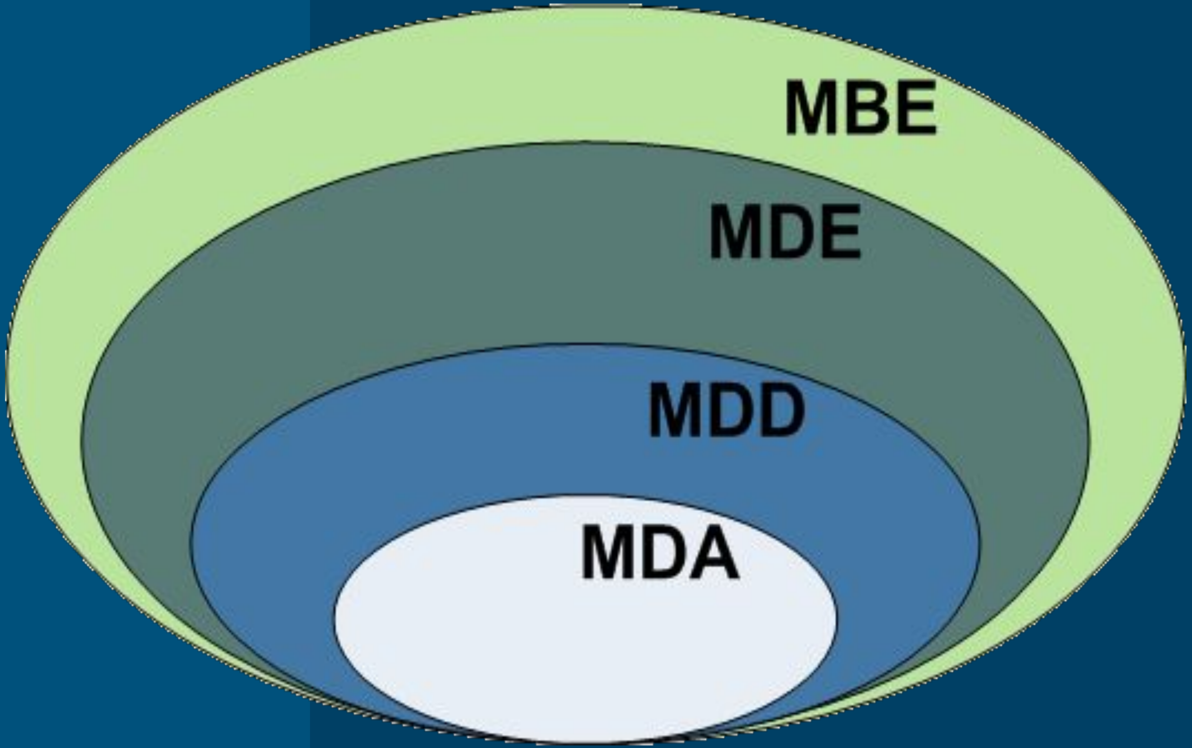
Models + Transformations = Software

Model-Driven Roadmap

The model as a central artefact



MD*



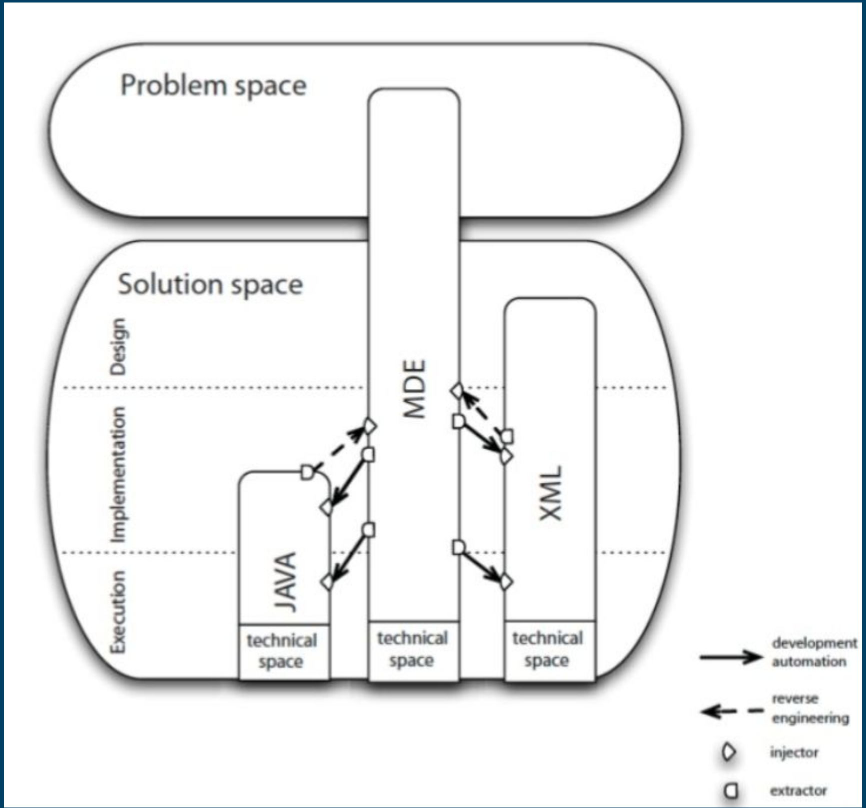
Taken from Master thesis of David Ameller (supervised by Xavier Franch)

MD*

- MBE – Model-Based Engineering
  - Process in which software models play an important role although they are not necessarily the key artifacts of the development (i.e. they do NOT “drive” the process)
- MDE – Model-Driven Engineering
  - Goes beyond of the pure development activities and encompasses other model-based tasks of a complete software engineering process (e.g. the model-based evolution the system or the model-driven reverse engineering of a legacy system).
- MDD – Model-Driven Development
  - Development paradigm that uses models and transformation (which also have models) as the primary artifact of the development process. Usually, in MDD, the implementation is (semi)automatically generated from the models.
- MDA – Model-Driven Architecture
  - OMG’s particular vision of MDD and thus relies on the use of OMG standards.

Taken from Master thesis of David Ameller (supervised by Xavier Franch)

MDE Coverage



MDE Coverage

Problem Domain - field or area of expertise where to solve a problem

Domain Model - Conceptual problem of the problem domain

Technical spaces- specific working contexts for specification, implementation and deployment of applications

Software Language Engineering

SLE is the application of a systematic, disciplined and quantifiable approach to the development, usage, and maintenance of software languages.

Software Language Engineer

One task of a language engineer is to develop languages that make the job of creating software easier.

Another task is to create a language that will support the language end-user (also known as domain expert) efficiently and effectively.

General purpose

General Purpose

Specific Purpose

- **match the user's mental model** of the problem domain
- **maximally constrain** the user (to the problem at hand)

  ⇒ easier to learn

  ⇒ avoid errors
- **separate** domain-expert's work from analysis/transformation expert's work

Anecdotal evidence of 5 to 10 times speedup

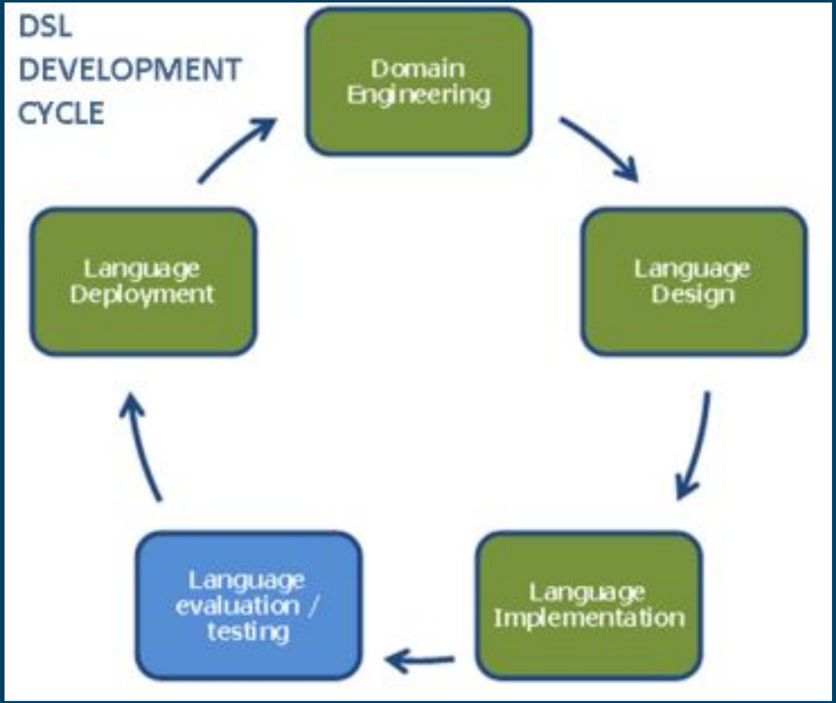
Steven Kelly and Juha-Pekka Tolvanen. Domain-Specific Modeling: Enabling Full Code Generation. Wiley, 2008.

Laurent Safa. The practice of deploying DSM, report from a Japanese appliance maker trenches. In Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), pp. 185-196, 2006.

taken from Hans Vangheluwe Lecture on Domain-Specific Languages at DSM-TP summer school

SLE Process

(as systematic)



Ankica Barisic, Vasco Amaral, Miguel Goulão:
Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures 51: 118-157 (2018)

Modeling Languages

DS(M)Ls

E.g. HTML, Logo, VHDL, Mathematica, SQL

GP(M)Ls

E.g. UML, Petri-nets, Statecharts

In MDE:

Models + **Transformations** = Software

Model Transformations

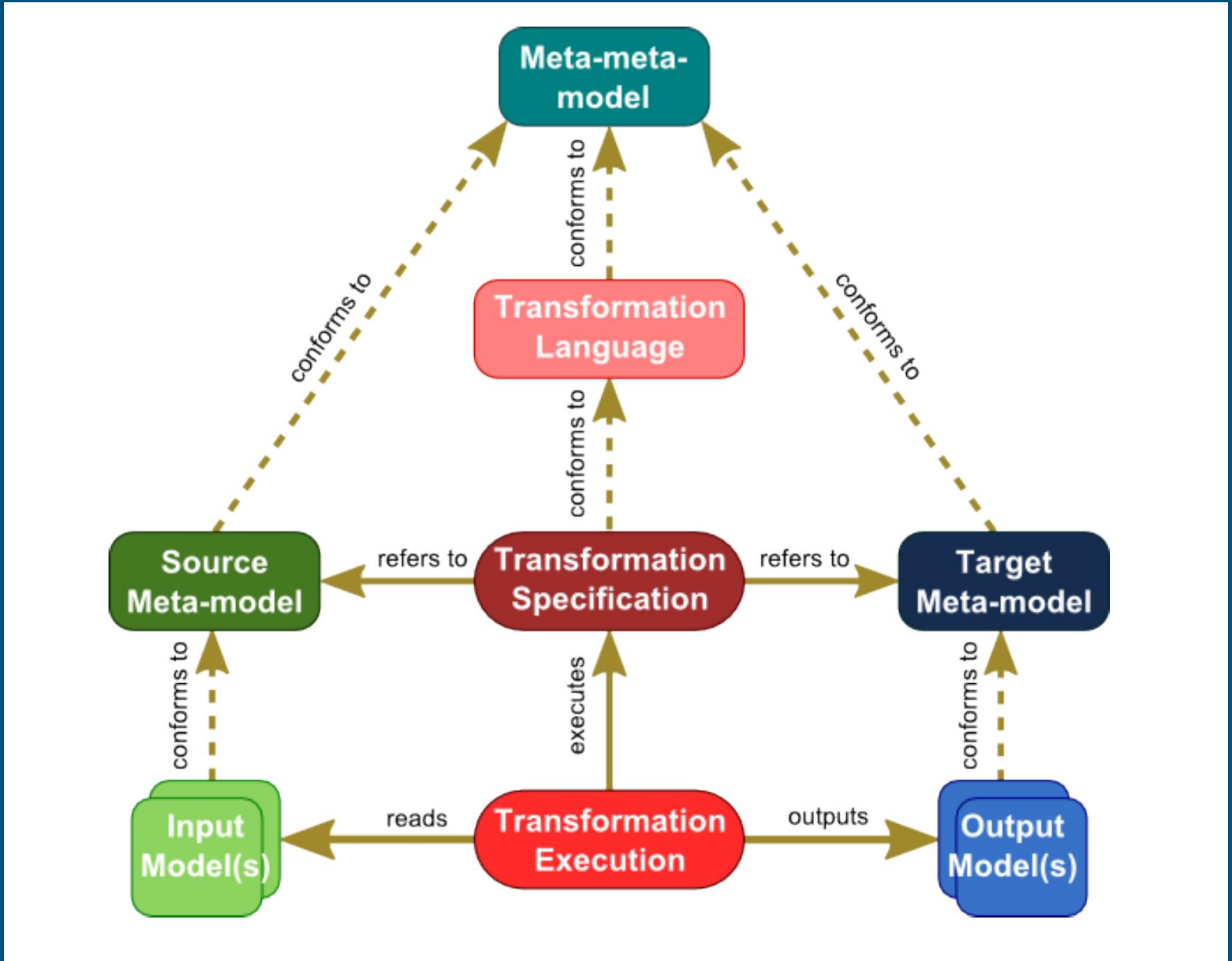
Definition
(later in the course we will refine this knowledge)

A model transformation is the automatic manipulation of input models to produce output models, that conform to a specification and has a specific intent

Model Transformations

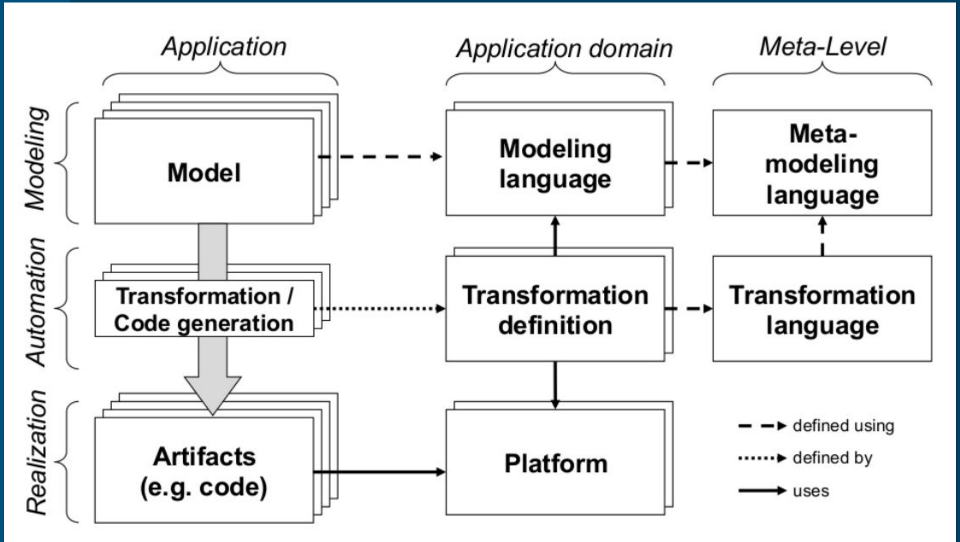
MDE provides appropriate languages for defining model transformation rules

Transformations can be seen as models



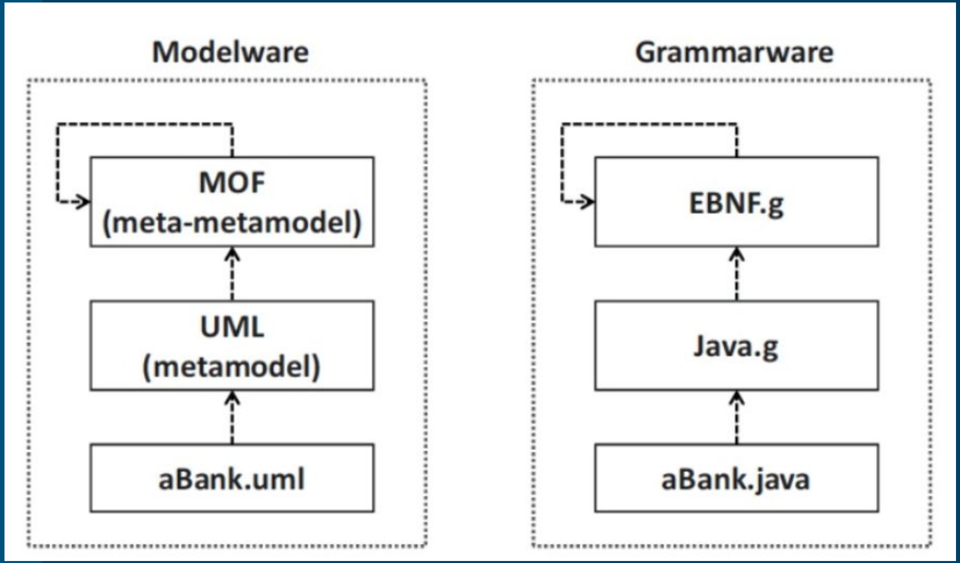
Meta-meta-model
conforms to
conforms to
conforms to
Transformation Language
conforms to
Source Meta-model
refers to
Transformation Specification
refers to
Target Meta-model
conforms to
executes
conforms to
Input Model(s)
reads
Transformation Execution
outputs
Output Model(s)

MDE Architecture



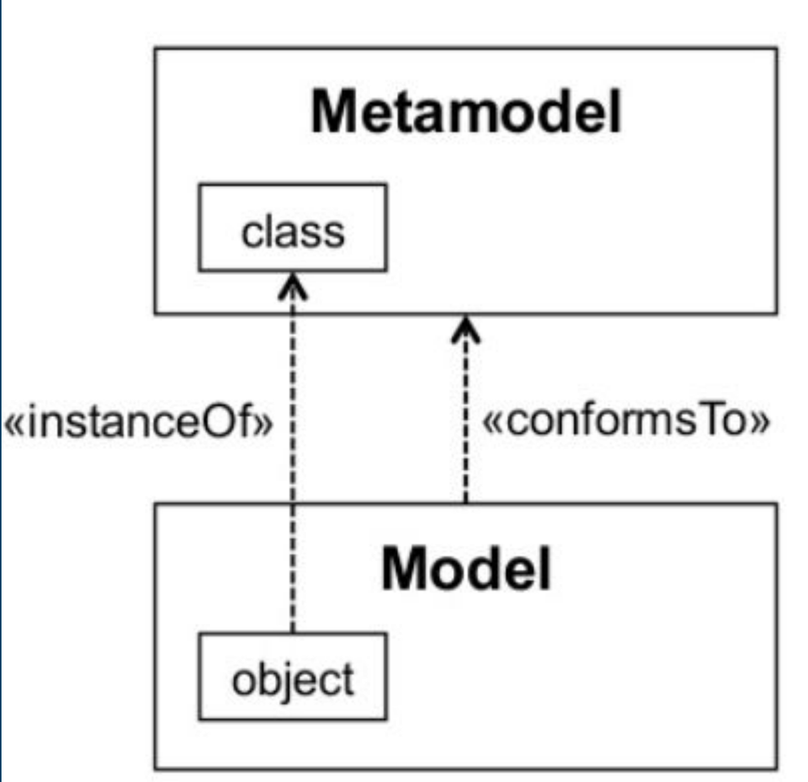
Two Technical Spaces

Modelware vs. Grammarware



InstanceOf vs. ConformsTo

Conformance is between models
Instantiation is between model elements



Types of Models

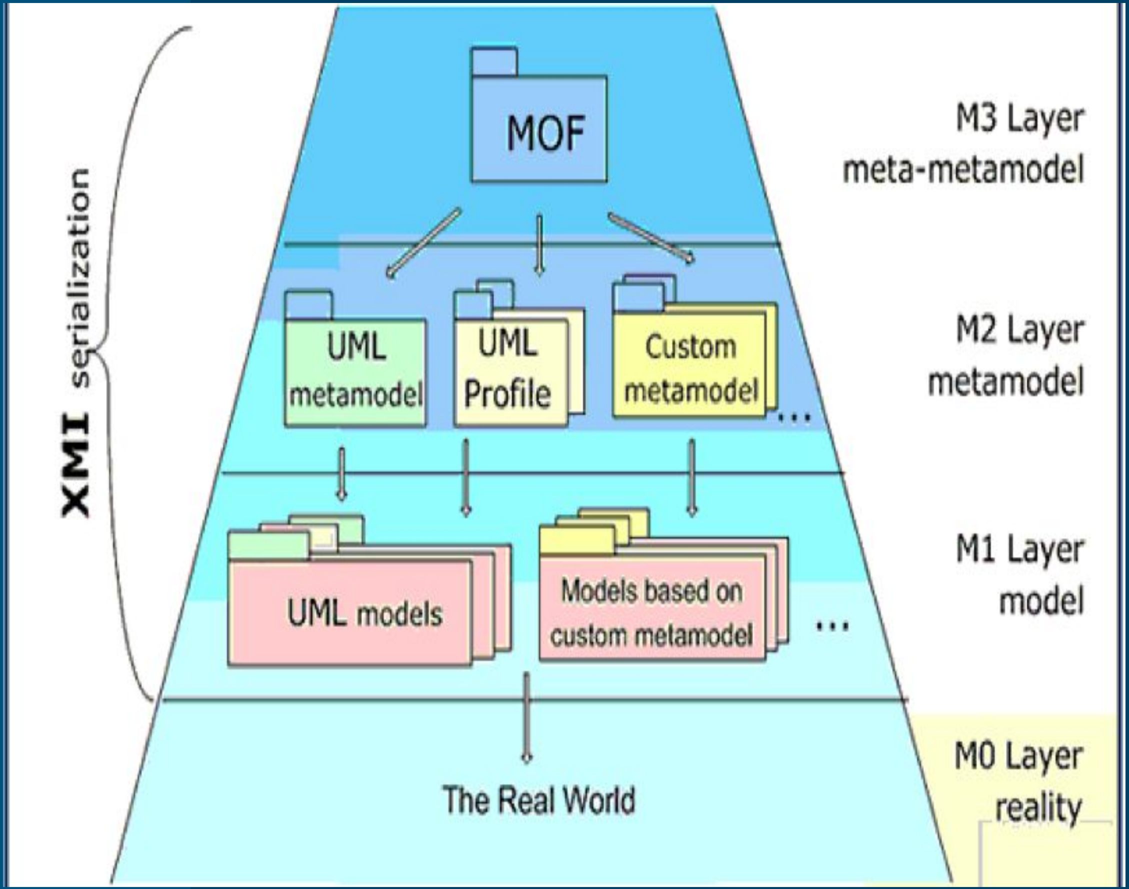
Static models - describing in terms of structural shape and architecture of the system

Dynamic models - describing dynamic behavior of the system by showing the execution

MDE Approaches

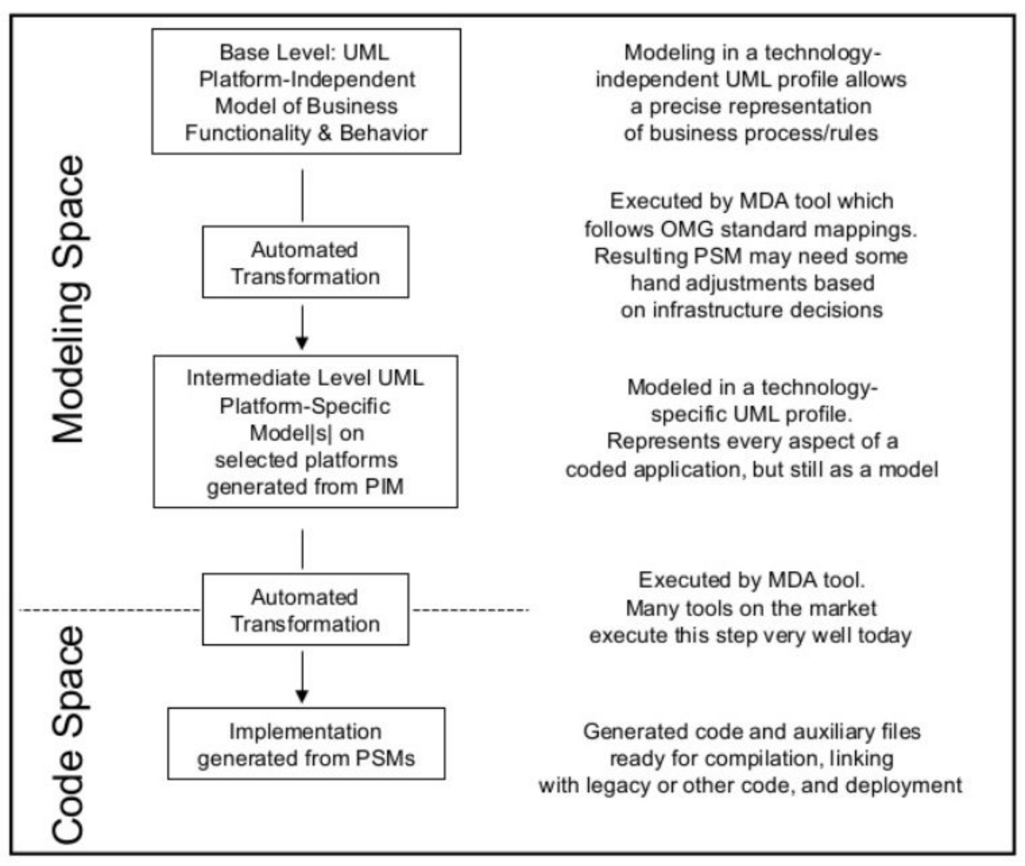
OMG MDA

OMG's 4 layer architecture



MDE Approaches

OMG MDA



MDE Approaches

OMG MDA

- **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.

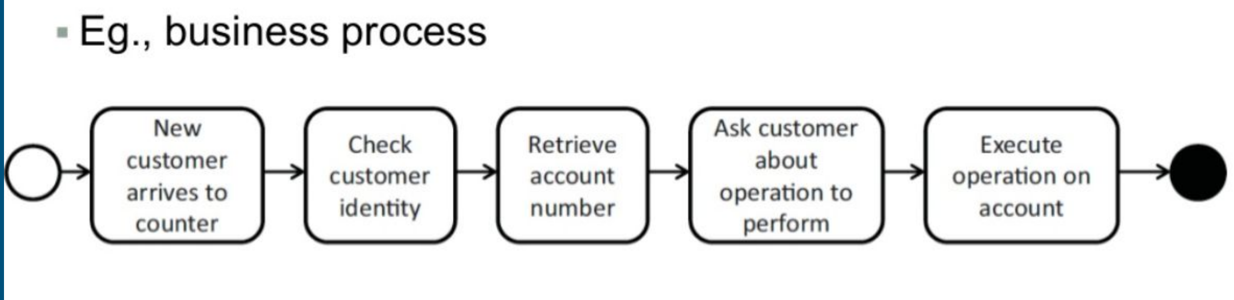
MDE Approaches

OMG MDA

- **Interoperability** through platform independent models
  - Standardization initiative of the Object Management Group (**OMG**), based on OMG Standards, particularly **UML**
  - Counterpart to CORBA on the modeling level: interoperability between different platforms
  - Applications which can be installed on different platforms → portability, no problems with changing technologies, integration of different platforms, etc.
- **Modifications to the basic architecture**
  - Segmentation 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; cf. QVT) and model-to-code transformation (PSM-Code)
  - Platform development is not taken into consideration – in general industry standards like J2EE, .NET, CORBA are considered as platforms

MDA

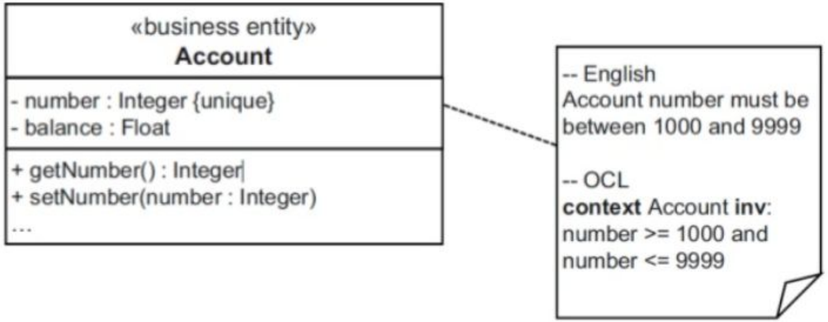
Example of CIM



MDA

Example of PIM

- Specification of structure and behaviour of a system, abstracted from technological details

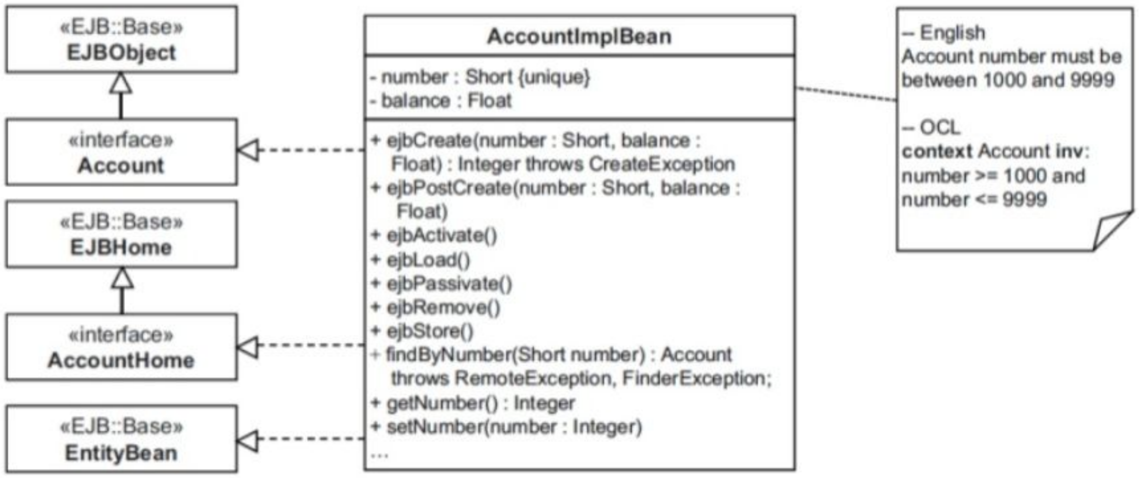


- Using UML(optional)

- Abstraction of structur and behaviour of a system by PIM simplifies the following:
  - Validation of correctness of the model
  - Creation of implementations on different platforms
  - Tool support during implementation

MDA

Example of PSM



- Specifies how the functionality described in the PIM is realized on a certain platform
- Using a UML-Profile for the selected platform, e.g., EJB

OMG Standards

- CORBA - Common Object Request Broker Architecture
  - Language- and platform-neutral interoperability standard (similar to WSDL, SOAP and UDDI)
- UML - Unified Modeling Language
  - Standardized modeling language, industry standard
- CWM - Common Warehouse Metamodel
  - Integrated modeling language for data warehouses
- MOF – Meta Object Facility
  - A standard for metamodels and model repositories
- XMI - XML Metadata Interchange
  - XML-based exchange of models
- QVT – Queries/Views/Transformations
  - Standard language for model-to-model transformations

MDA

- Problems when using **UML** as PIM/PSM
  - Method bodies?
  - Incomplete diagrams, e.g. missing attributes
  - Inconsistent diagrams
  - *For the usage of the UML in Model Engineering special guidelines have to be defined and adhered to*
- Different requirements to **code generation**
  - get/set methods
  - Serialization or persistence of an object
  - Security features, e.g. Java Security Policy
  - *Using adaptable code generators or PIM-to-PSM transformations*
- **Expressiveness** of the UML
  - UML is mainly suitable for “generic” software platforms like Java, EJB, .NET
  - Lack of support for user interfaces, code, etc.
  - *MDA tools often use proprietary extensions*

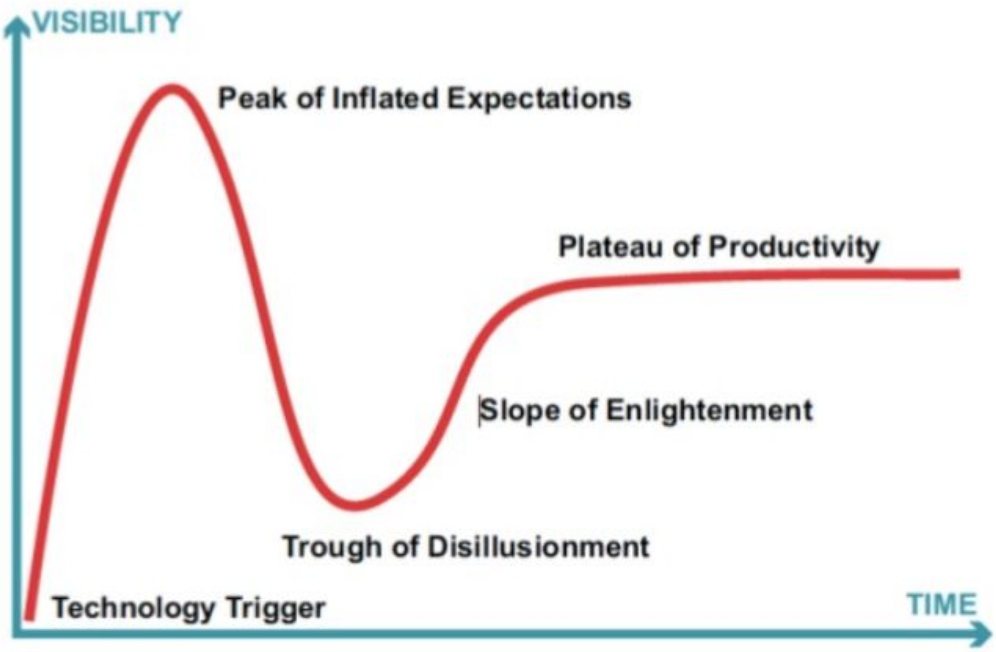
MDA

- Many UML tools are expanded to MDA tools
  - UML profiles and code generators
  - Stage of development partly still similar to CASE: proprietary UML profiles and transformations, limited adaptability
- Advantages of MDA
  - Standardization of the Meta-Level
  - Separation of platform independent and platform specific models (reuse)
- Disadvantages of MDA
  - 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)

MDE

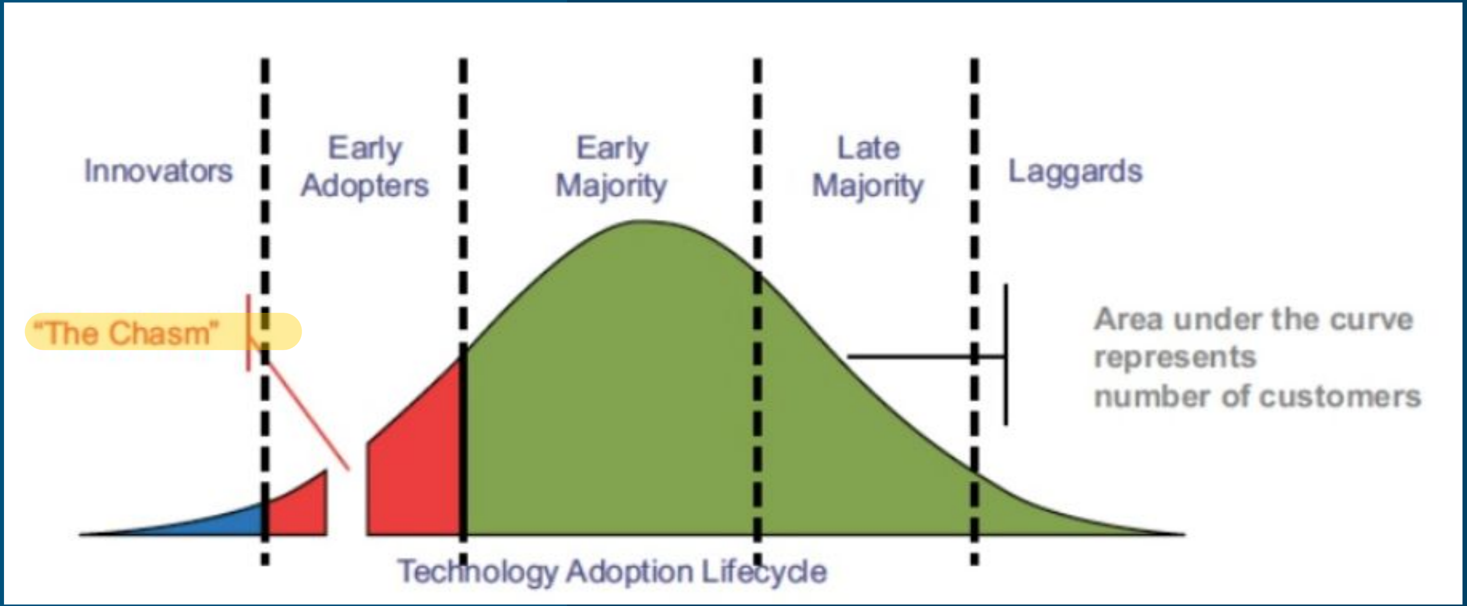
Adoption

- Not yet mainstream in all industries
- Strong in core industry (defense, avionics, …)



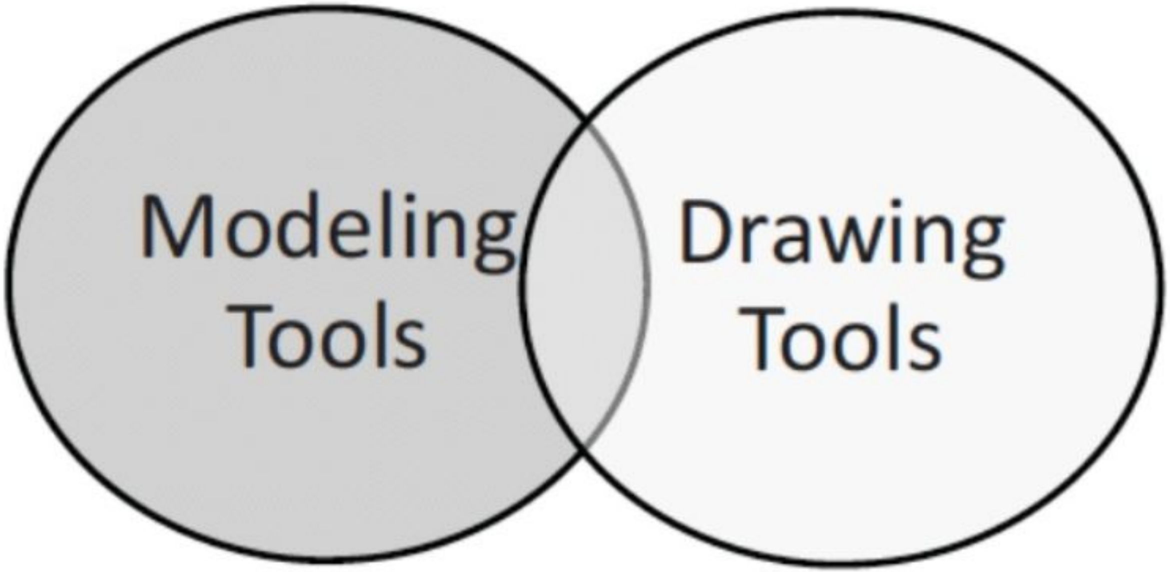
MDE

Adoption



MDE

- Drawing vs. modeling



Eclipse

- Eclipse Modeling Framework
- Full support for metamodeling and language design
- Fully MD (vs. programming-based tools)
- Used in this course!

Criticism

- Critical Statements of Software Developers
- »When it comes down to it, the real point of software development is cutting code«
- »Diagrams are, after all, just pretty pictures«
- »No user is going to thank you for pretty pictures; what a user wants is software that executes«

M. Fowler, "UML Distilled", 1st edition, Addison Wesley, 1997

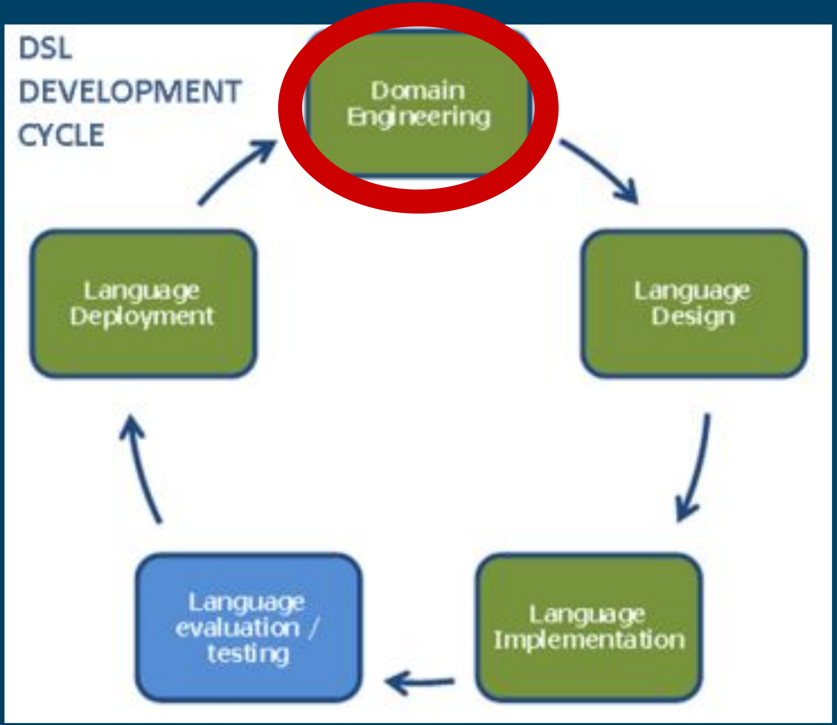
Dealing with Criticism

- »When it comes down to it, the real point of software development is cutting code«
  - To model or to program, that is not the question!
  - Instead: Talk about the right abstraction level
- »Diagrams are, after all, just pretty pictures«
  - Models are not just notation!
  - Instead: Models have a well-defined syntax in terms of metamodels
- »No user is going to thank you for pretty pictures; what a user wants is software that executes«
  - Models and code are not competitors!
  - Instead: Bridge the gap between design and implementation by model transformations

M. Fowler, "UML Distilled", 1st edition, Addison Wesley, 1997 (revisited in 2009)

SLE Process

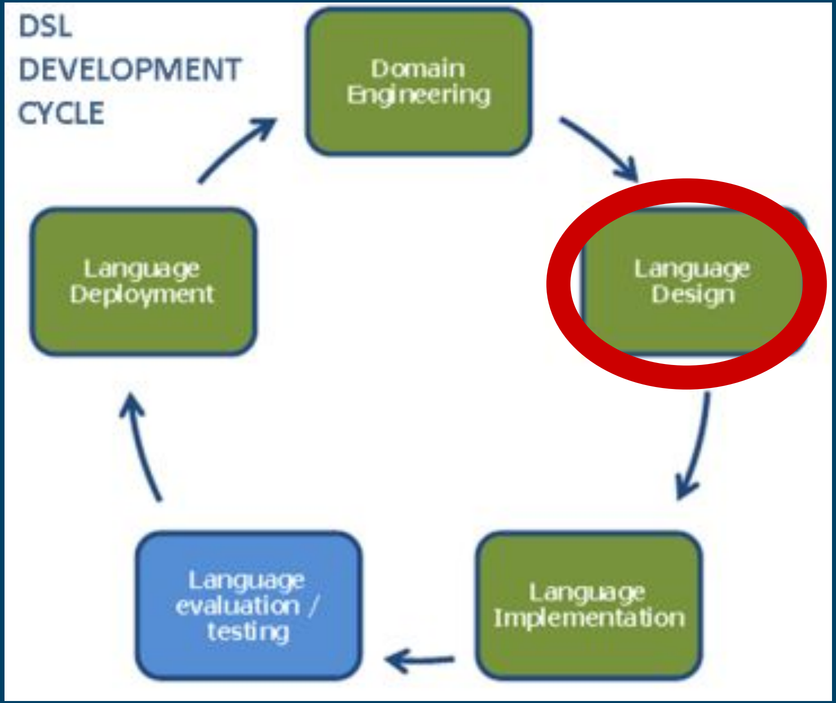
Domain Engineering



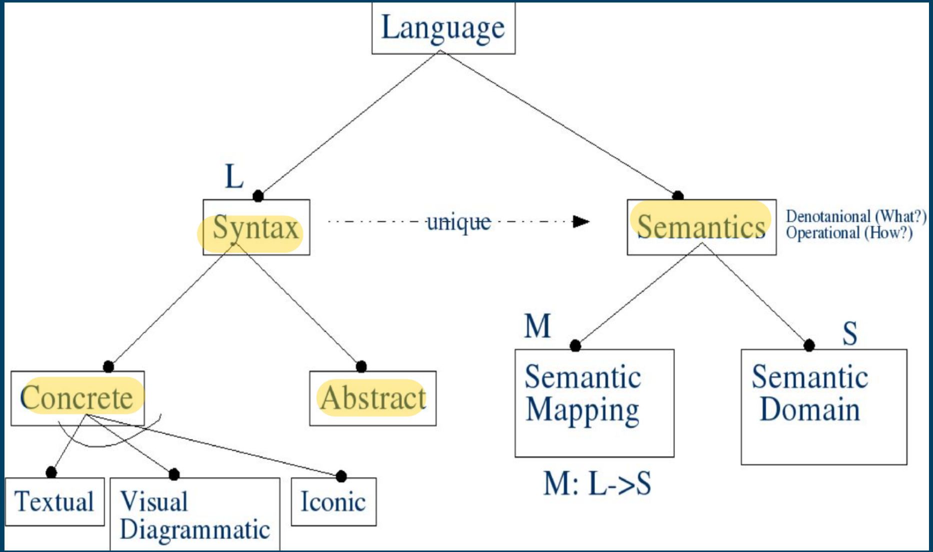
Ankica Barisic, Vasco Amaral, Miguel Goulão:
Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures 51: 118-157 (2018)

SLE Process

Language Design

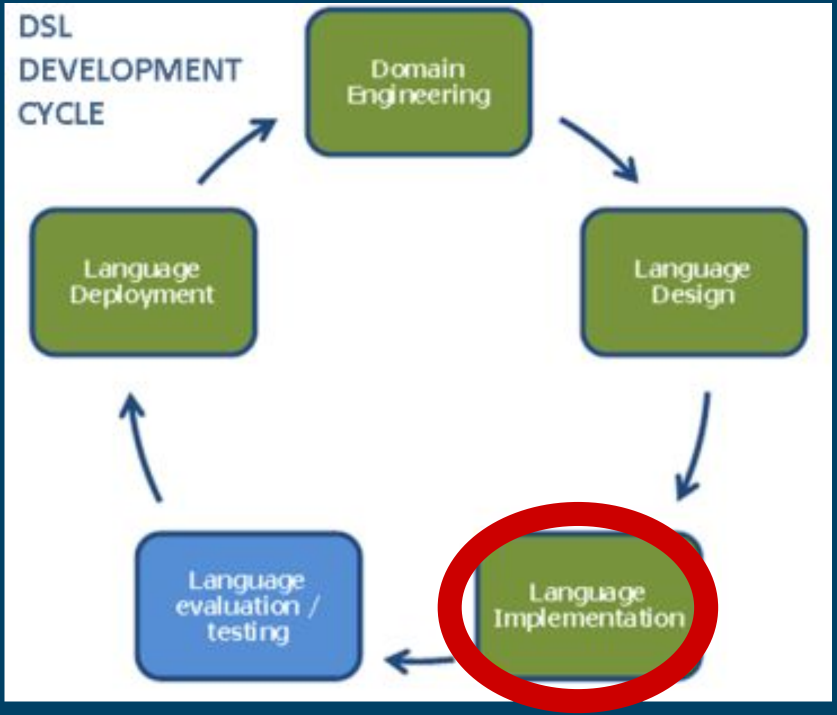


What is a Language?



SLE Process

Implementation



Pick the most adequate tool for your purpose

Visual?

- Modelling workbenches (ideal for prototyping languages):
  - EMF/GMF; GME; DSL Tools; MetaEdit+; AtomPM
- UML Stereotypes?

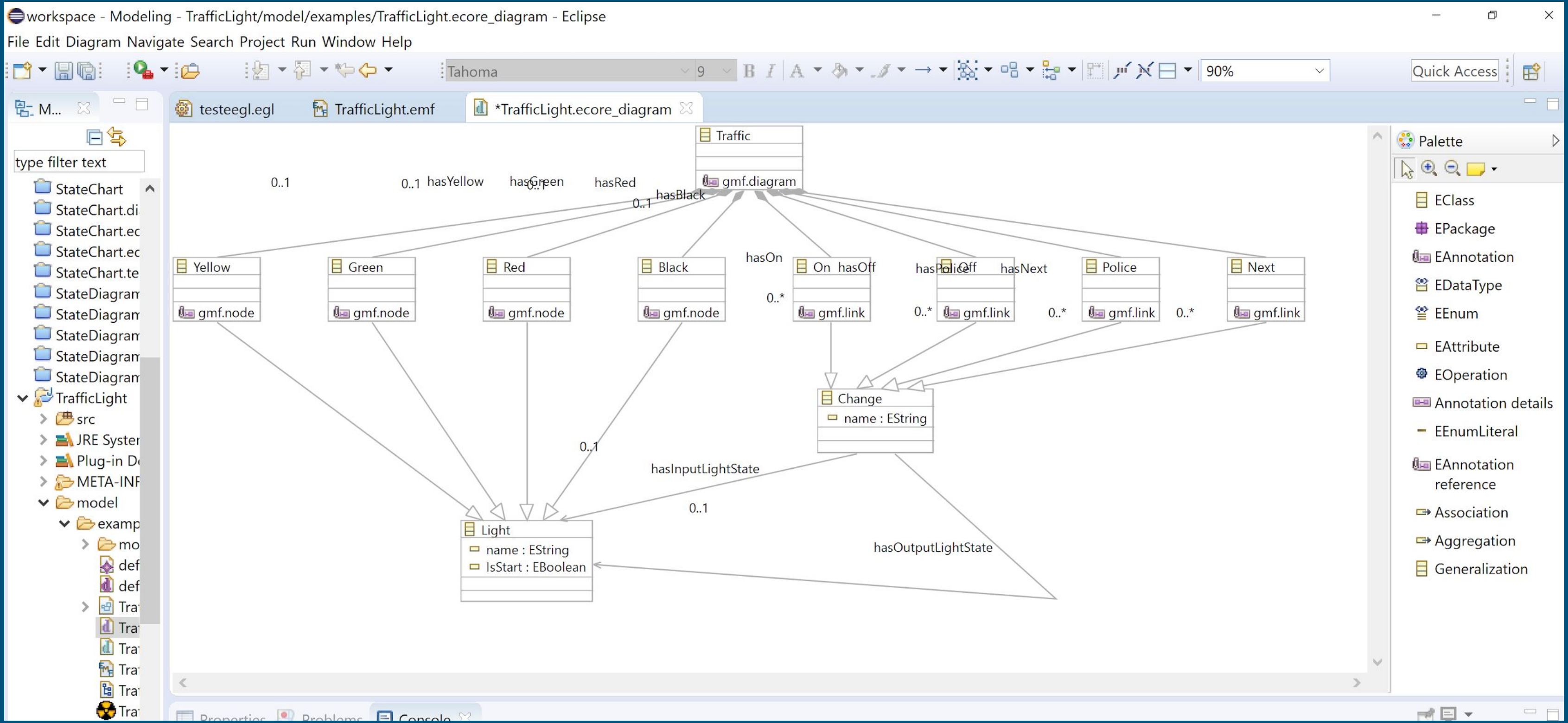
Textual?

- Macro
- Embedded Language? (Ex. Ruby)
- Standalone:
  - Traditional Compiler approach
  - Modelling Workbenches: MPS (JetBrains); Xtext Eclipse;...

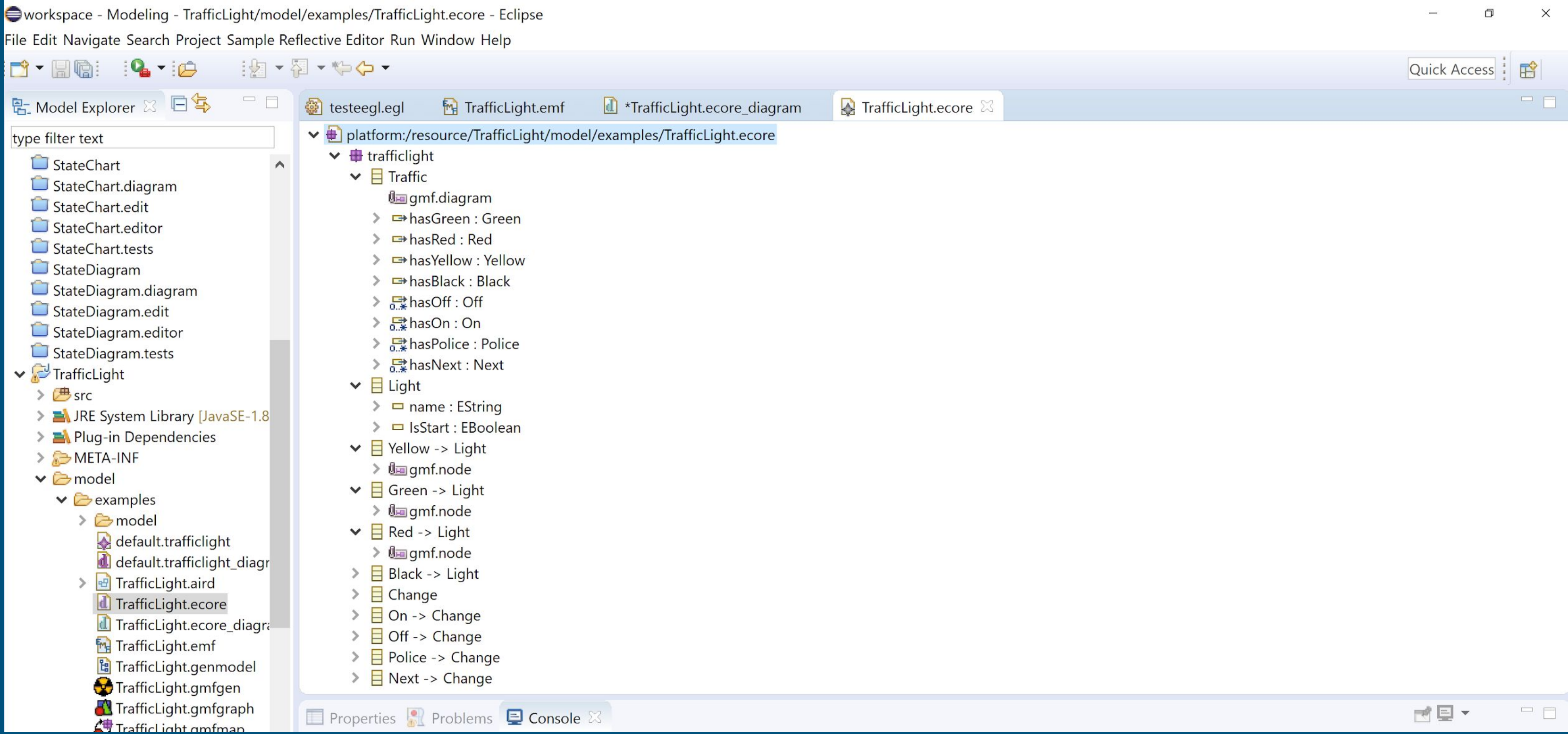
MDD approach

Lets use Eclipse GMF/EMF with the Epsilon toolset

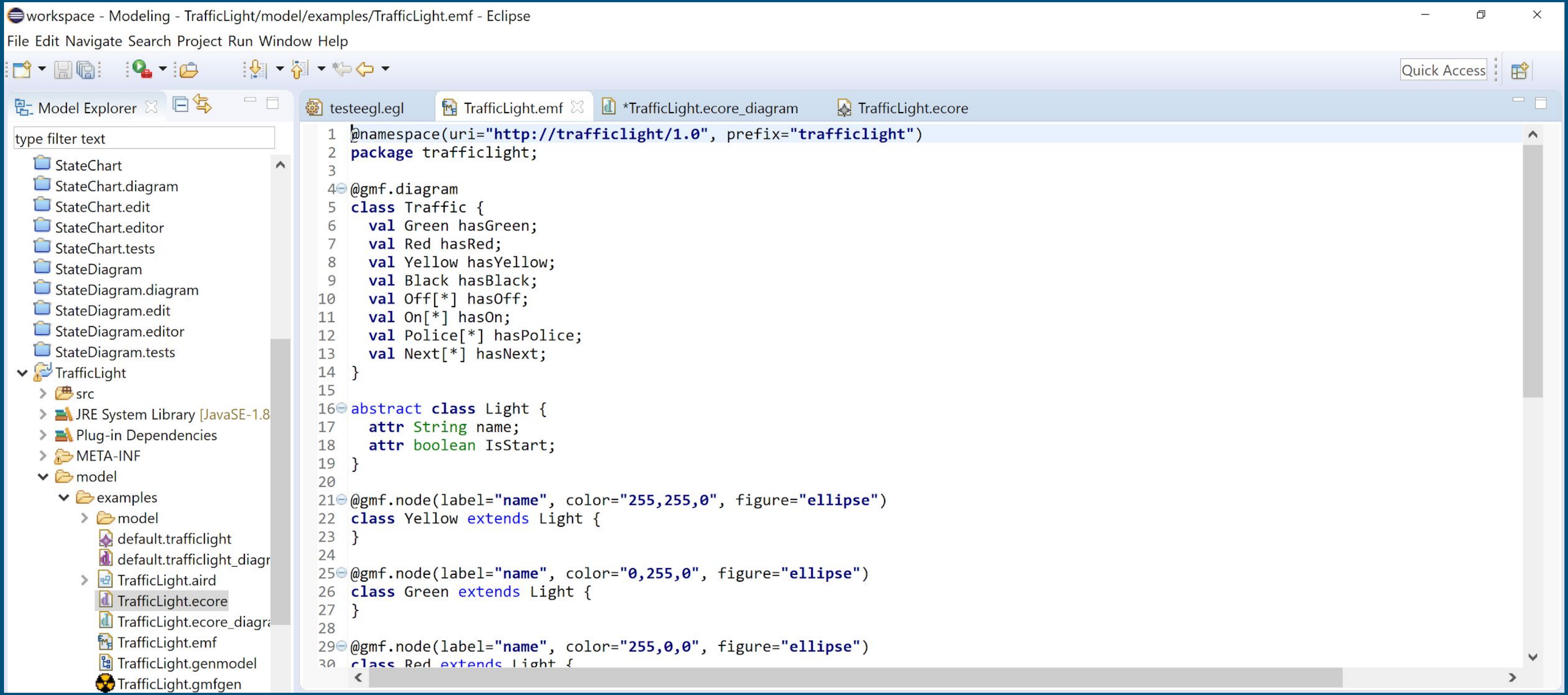
The Abstract Syntax - Ecore Metamodel



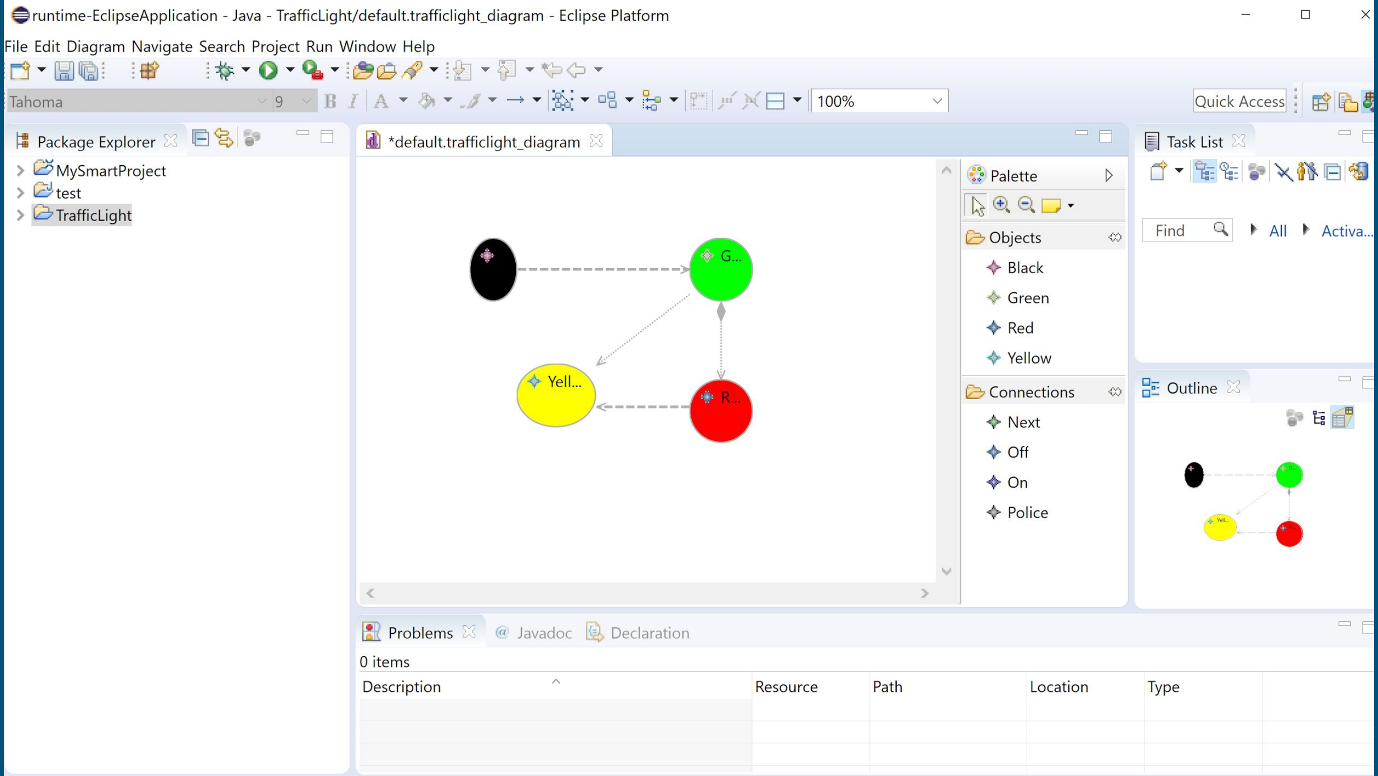
The Abstract Syntax - Ecore Metamodel



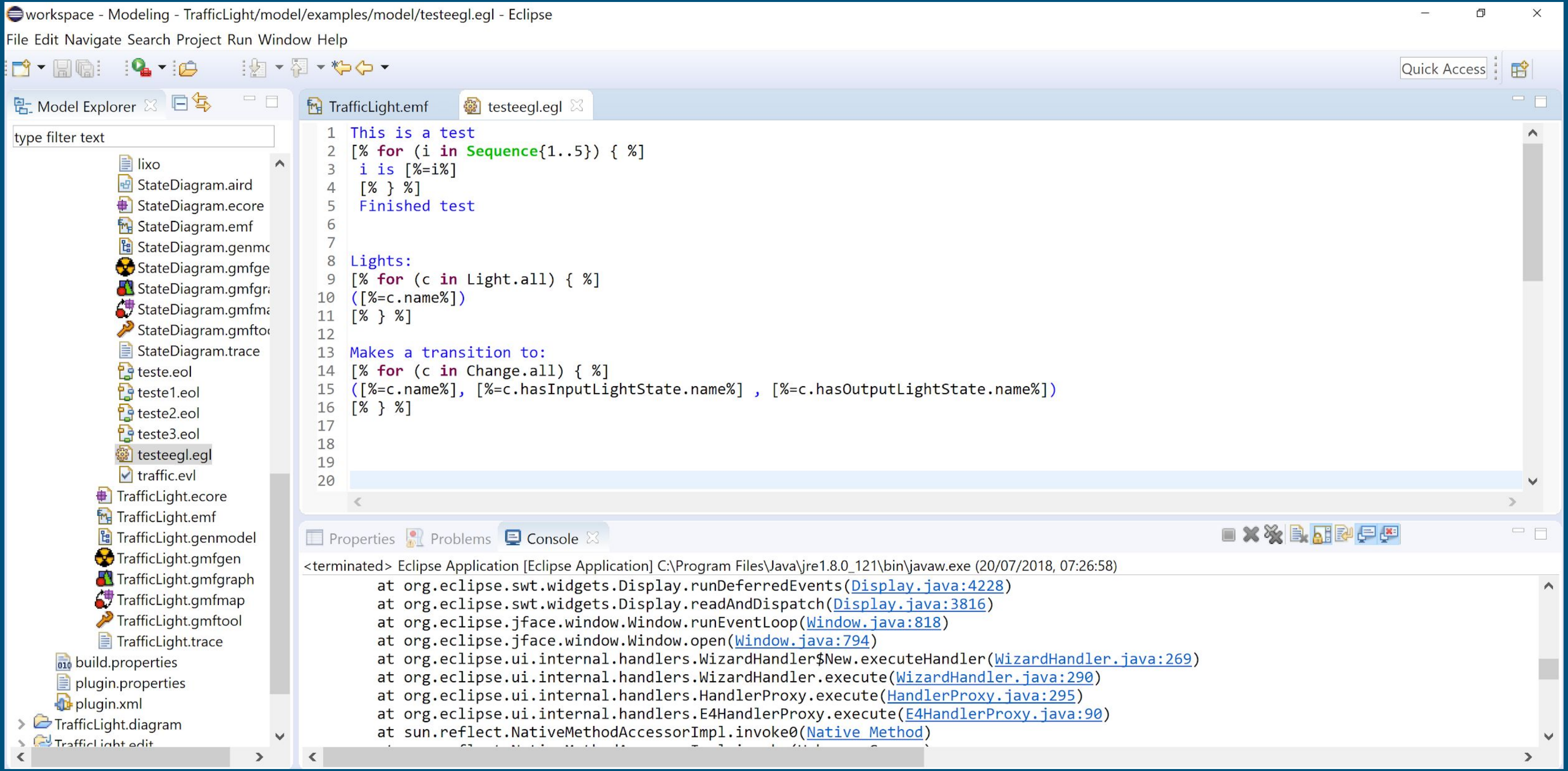
The Concrete Syntax



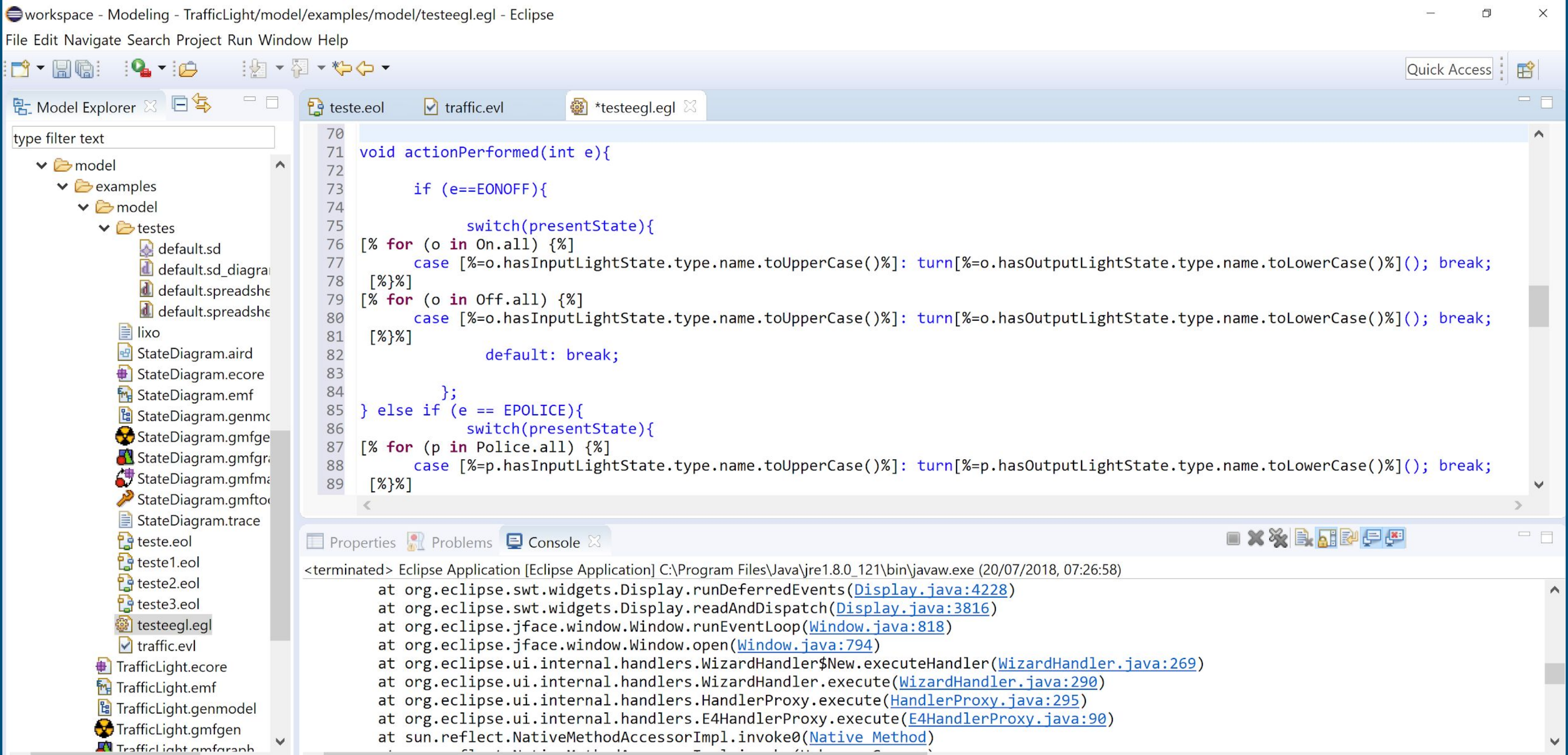
Model to Code - Just push a button



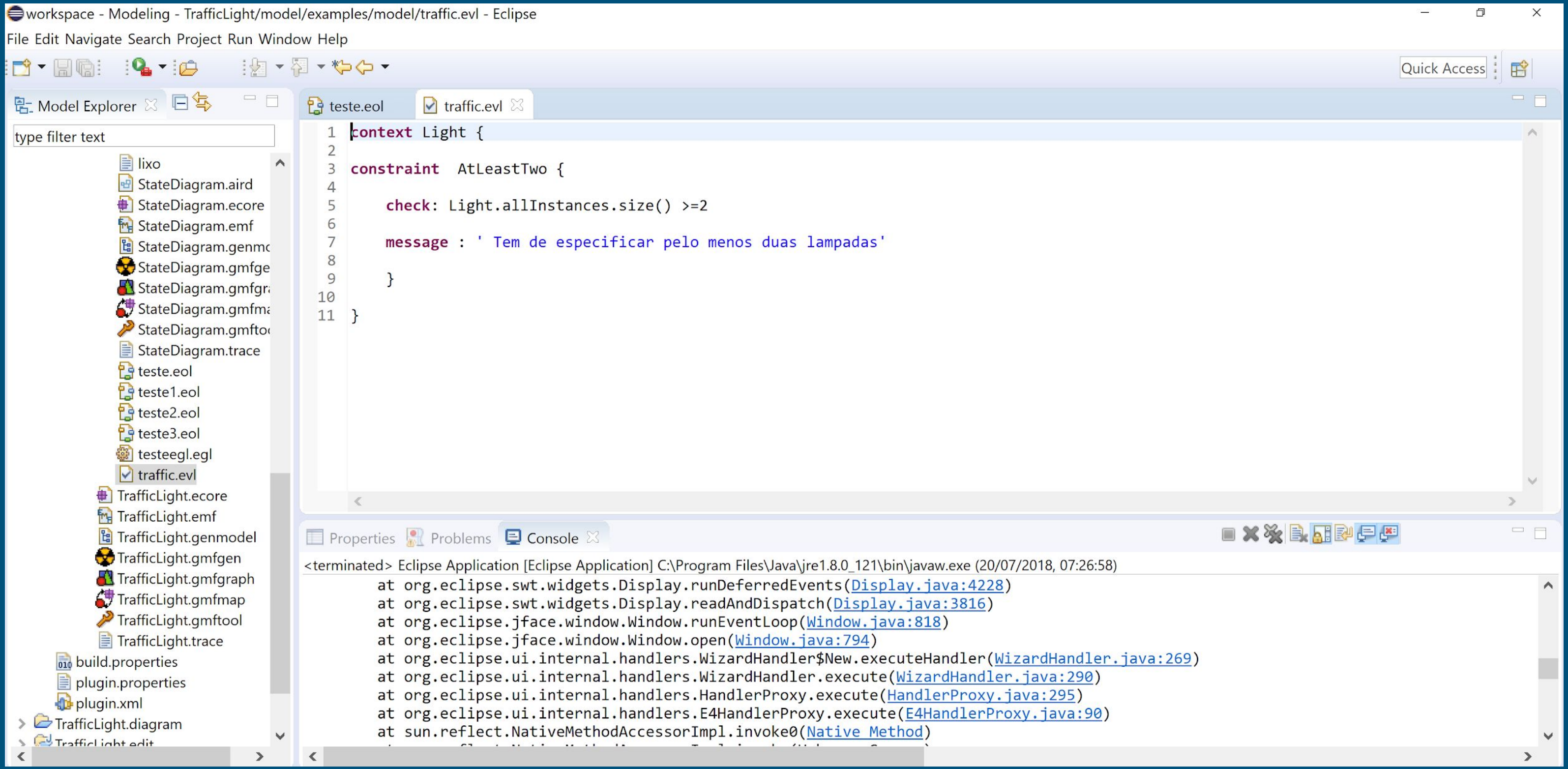
Model to Code - Epsilon Generation Language (Template based)



Model to Code - Epsilon Generation Language (Template based) Arduino



Model to Code - Epsilon Validation Language (Template based)



Model to Code - Epsilon Validation Language (Template based)

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;

public class TrafficLight extends JFrame implements ActionListener {
    JButton b1, b2, b3;

    Signal green = new Signal(Color.green);
    Signal yellow = new Signal(Color.yellow);
    Signal red = new Signal(Color.red);

    public TrafficLight(){
        super("Traffic Light");
        getContentPane().setLayout(new GridLayout(2, 1));
        b1 = new JButton("Red");
        b2 = new JButton("Yellow");
        b3 = new JButton("Green");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

        green.turnOn(false);
        yellow.turnOn(false);
        red.turnOn(true);

        JPanel p1 = new JPanel(new GridLayout(3,1));
        p1.add(red);
        p1.add(yellow);
        p1.add(green);
        JPanel p2 = new JPanel(new FlowLayout());
        p2.add(b1);
```

Transformation rules (ETL)

```
rule Tree2Node

  transform t : Tree!Tree
  to n : Graph!Node {

  n.name = t.label;

  // If t is not the top tree
  // create an edge connecting n
  // with the Node created from t's parent

  if (t.parent.isDefined()) {
    var e : new Graph!Edge;
    e.source ::= t.parent;
    e.target = n;
  }
}
```

Model transformation intent classification

Refinement
- Refinement
- Synthesis
  - Serialization

Abstraction
- Abstraction
- Reverse Engineering
- Restrictive Query
- Approximation

Semantic Definition
- Translational Semantics
- Simulation

Language Translation
- Translation
- Migration

Constraint Satisfaction
- Model Finding
- Model Generation

Analysis

Editing
- Model Editing
- Optimization
- Model Refactoring
- Normalization
  - Canonicalization

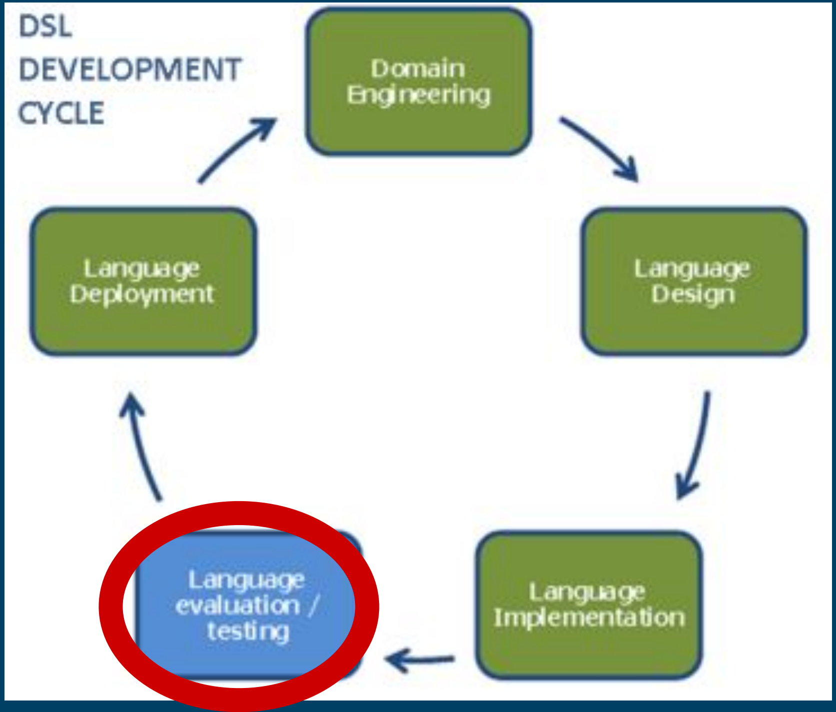
Model Visualization
- Animation
- Rendering
- Parsing

Model Composition
- Model Merging
- Model Matching
- Model Synchronization

L. Lúcio, M. Amrani, J. Dingel, L. Lambers, R. Salay, G. Selim, E. Syriani and M. Wimmer. Model transformation intents and their properties. *Journal on Software and Systems Modeling*: 10.1007/s10270-014-0429-x. Springer (2014).

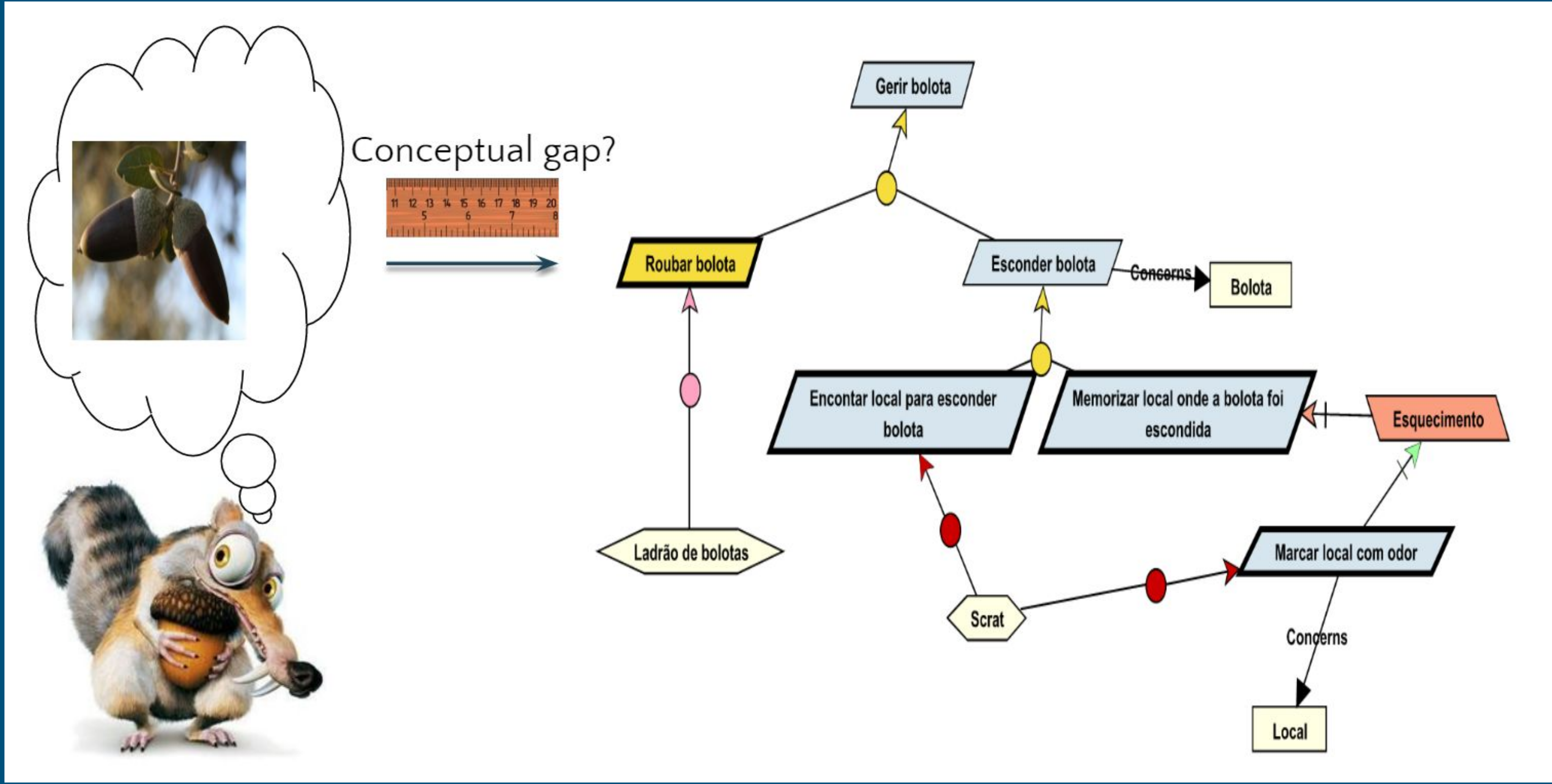
SLE Process

Evaluation



Gabriel, P., M. Goulão, and V. Amaral, "Do Software Languages Engineers Evaluate their Languages?", XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010), ISBN: 978-9978-325-10-0, Cuenca, Ecuador, Universidad del Azuay, pp. 149-162, 2010

The language has to empower its user... or he will end up using something else



What strategies are available to us?

Constructive approaches:
- Our own expertise and common sense
- Usability heuristics such as the "Physics of notations"

Evaluation-based approaches:
- "Traditional" usability evaluations
- User monitoring while using the DSML

Language usage tasks

writing
reading
interpretation
comprehension
memorization
problem solving

How is SLE doing?

SLE misses:

- More examples of successful industry projects showing SLE
- Maturity in some stages
- Language Composition/Integration
- Language Evolution
- Improved Supporting Tools
- Cost estimation strategies
- ...

Thank you!

Contact: vma@fct.unl.pt