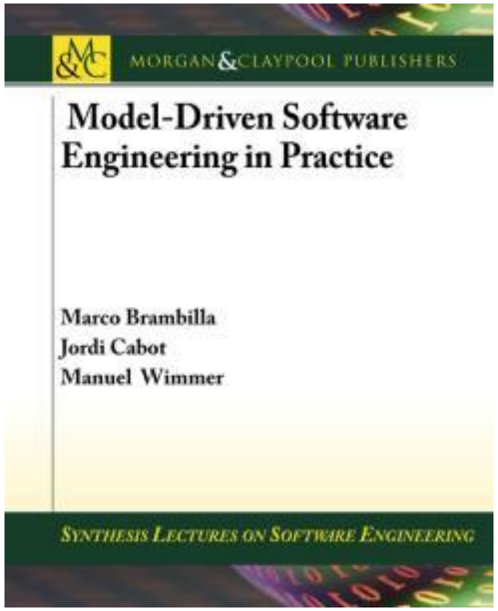
Chapter #2

MDSE PRINCIPLES

Teaching material for the book
Model-Driven Software Engineering in Practice
by Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Morgan & Claypool, USA, 2012.


MORGAN & CLAYPOOL PUBLISHERS
Model-Driven Software Engineering in Practice
Marco Brambilla
Jordi Cabot
Manuel Wimmer
SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

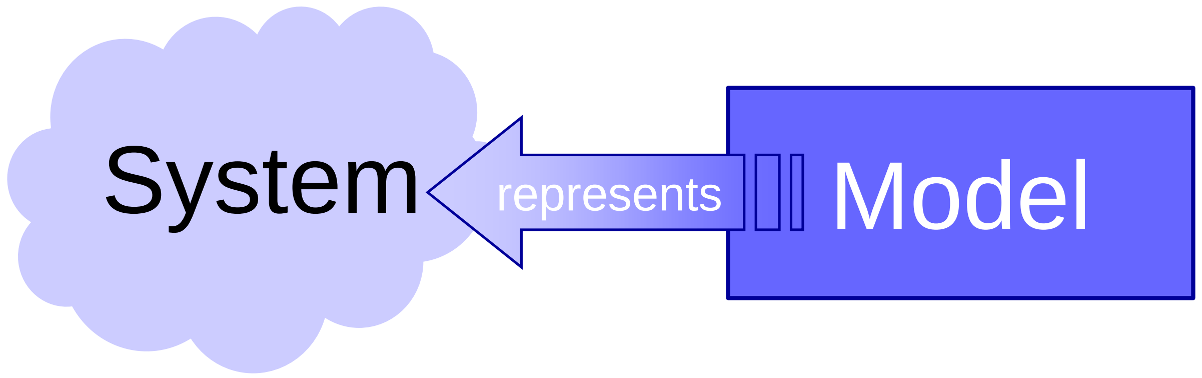
MDSE Principles

Contents

- Concepts
- Approaches
- Adoption

Models

What is a model?



Mapping Feature	A model is based on an original (=system)
Reduction Feature	A model only reflects a (relevant) selection of the original‘s properties
Pragmatic Feature	A model needs to be usable in place of an original with respect to some purpose

Purposes:

- descriptive purposes
- prescriptive purposes

MDSE aim at large

- MDSE considers models as first-class citizens in software engineering
- The way in which models are defined and managed is based on the actual needs that they will address.
- MDSE defines sound engineering approaches to the definition of
  - models
  - transformations
  - development process.

Concepts

Principles and objectives

- **Abstraction** from specific realization technologies
  - Requires modeling languages, which do not hold specific concepts of realization technologies (e.g., Java EJB)
  - Improved **portability** of software to new/changing technologies – model once, build everywhere
  - **Interoperability** between different technologies can be automated (so called Technology Bridges)

- **Automated code generation** from abstract models
  - e.g., generation of Java-APIs, XML Schemas, etc. from UML
  - Requires expressive und precise models
  - Increased **productivity** and **efficiency** (models stay up-to-date)

- **Separate development** of application and infrastructure
  - Separation of application-code and infrastructure-code (e.g., Application Framework) increases **reusability**
  - **Flexible** development cycles as well as **different development roles possible**

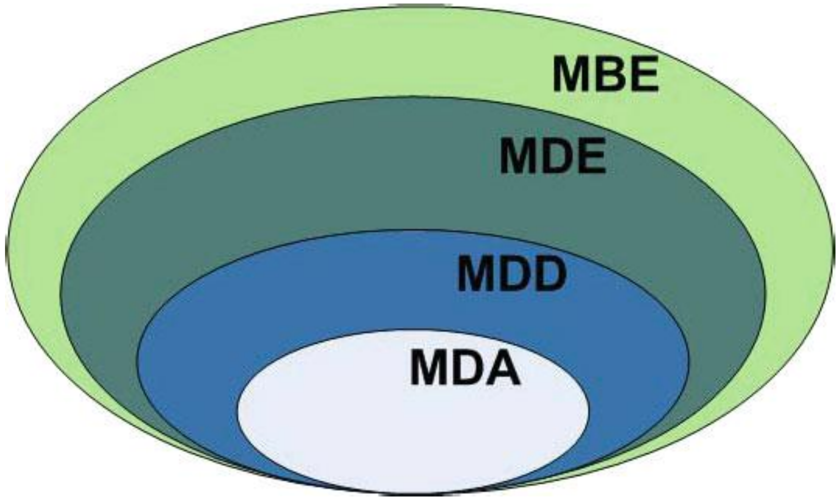
MDSE methodology ingredients

- **Concepts:** The components that build up the methodology
- **Notations:** The way in which concepts are represented
- **Process and rules:** The activities that lead to the production of the final product
- **Tools:** Applications that ease the execution of activities or their coordination

MDSE Equation

Models + Transformations = Software

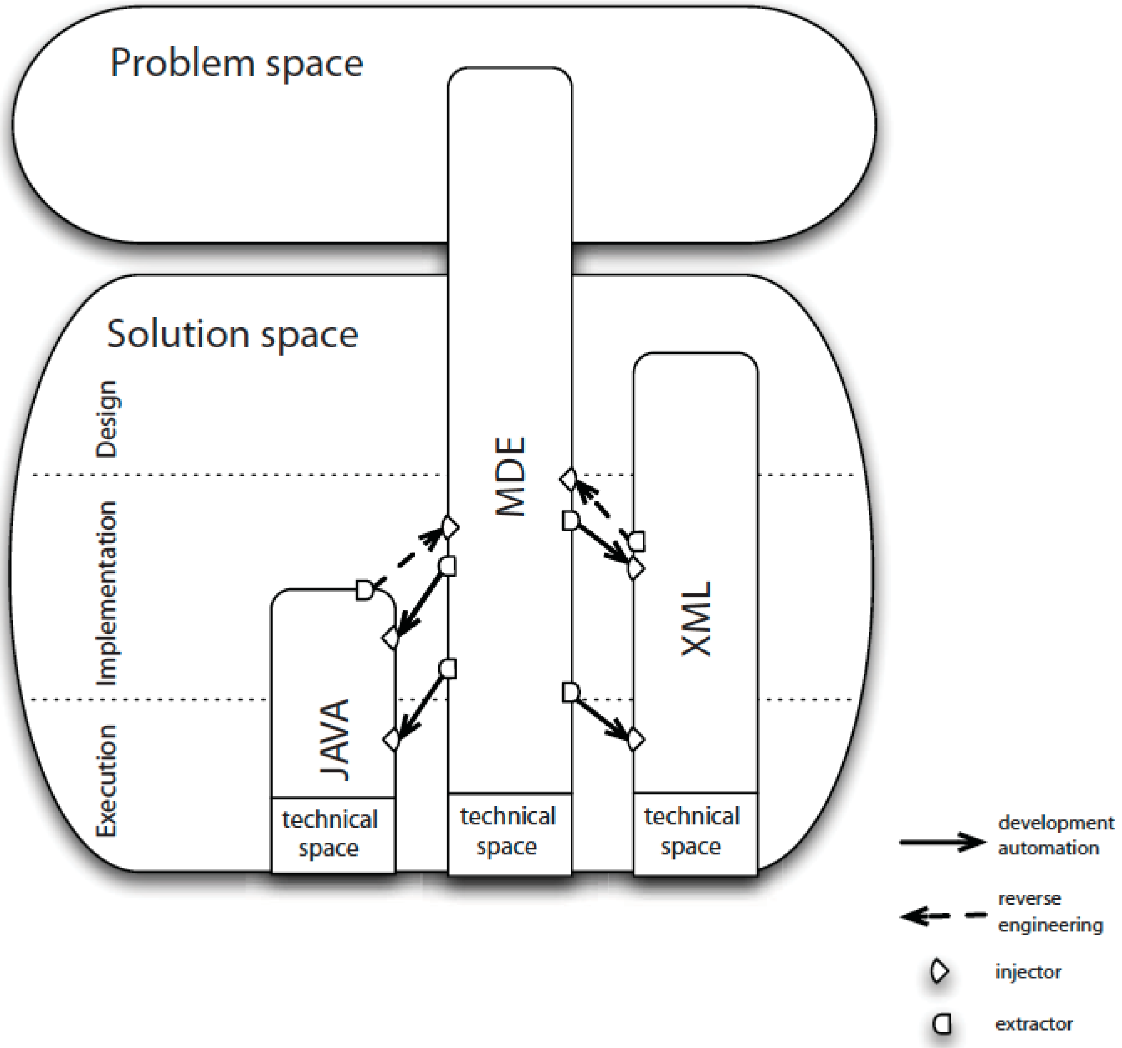
The MD* Jungle of Acronyms



- **Model-Driven Development (MDD)** is a development paradigm that uses models as the primary artifact of the development process.
- **Model-driven Architecture (MDA)** is the particular vision of MDD proposed by the Object Management Group (OMG)
- **Model-Driven Engineering (MDE)** is a superset of MDD because it goes beyond of the pure development
- **Model-Based Engineering** (or "model-based development") (**MBE**) is a softer version of ME, where models do not "drive" the process.

Target of MDSE

- The **Problem Domain** is defined as the field or area of expertise that needs to be examined to solve a problem.
- The **Domain Model** is the conceptual model of the problem domain
- **Technical Spaces** represent specific working contexts for the specification, implementation, and deployment of applications.



Modeling Languages

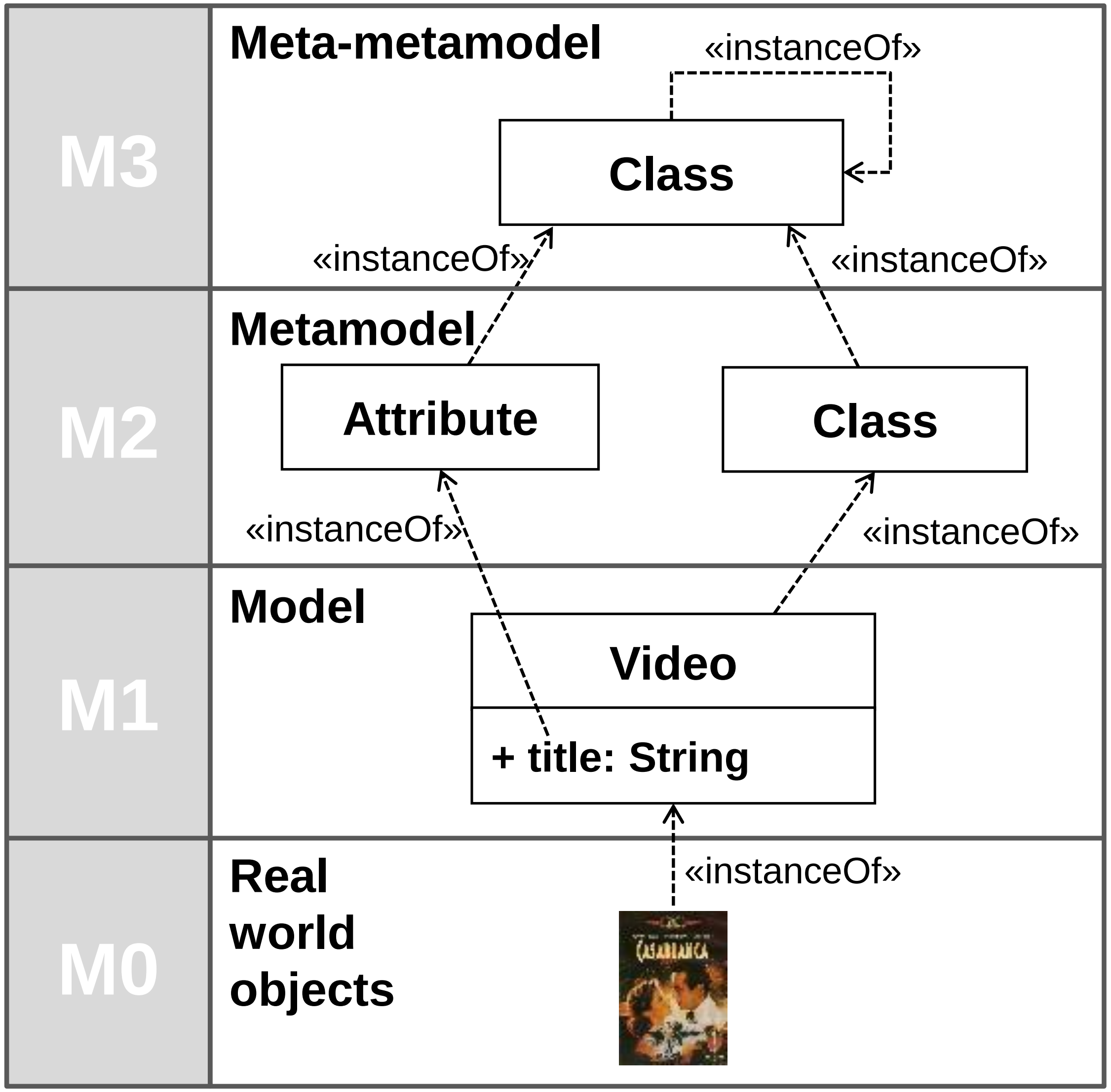
- **Domain-Specific Modeling Languages (DSMLs, DSLs):** languages that are designed specifically for a certain domain or context
- DSLs have been largely used in computer science.
- Examples: HTML, Logo, VHDL, Mathematica, SQL

- **General Purpose Modeling Languages** (GPMLs, GMLs, or GPLs): languages that can be applied to any sector or domain for (software) modeling purposes
- The typical examples are: UML, Petri-nets, or state machines

Metamodeling

- To represent the models themselves as "instances" of some more abstract models.
- **Metamodel** = yet another abstraction, highlighting properties of the model itself

- Metamodels can be used for:
  - defining new languages
  - defining new properties or features of existing information (metadata)

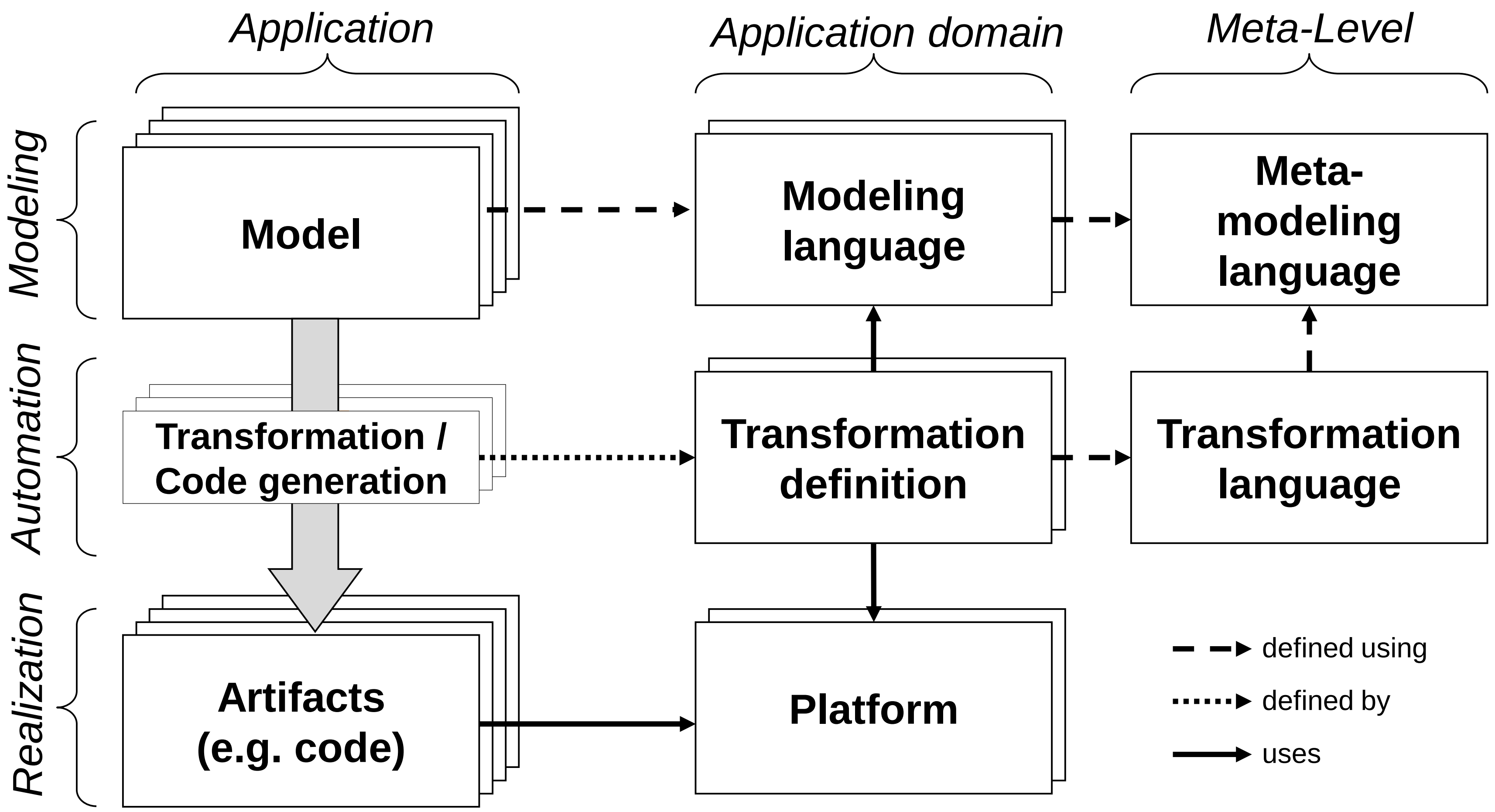


Model Transformations

- Transforming items
- MDSE provides appropriate languages for defining model transformation rules
- Rules can be written manually from scratch by a developer, or can be defined as a refined specification of an existing one.
- Alternatively, transformations themselves can be produced automatically out of some higher level mapping rules between models
  - defining a mapping between elements of a model to elements to another one (**model mapping or model weaving**)
  - automating the generation of the actual transformation rules through a system that receives as input the two model definitions and the mapping
- Transformations themselves can be seen as models!

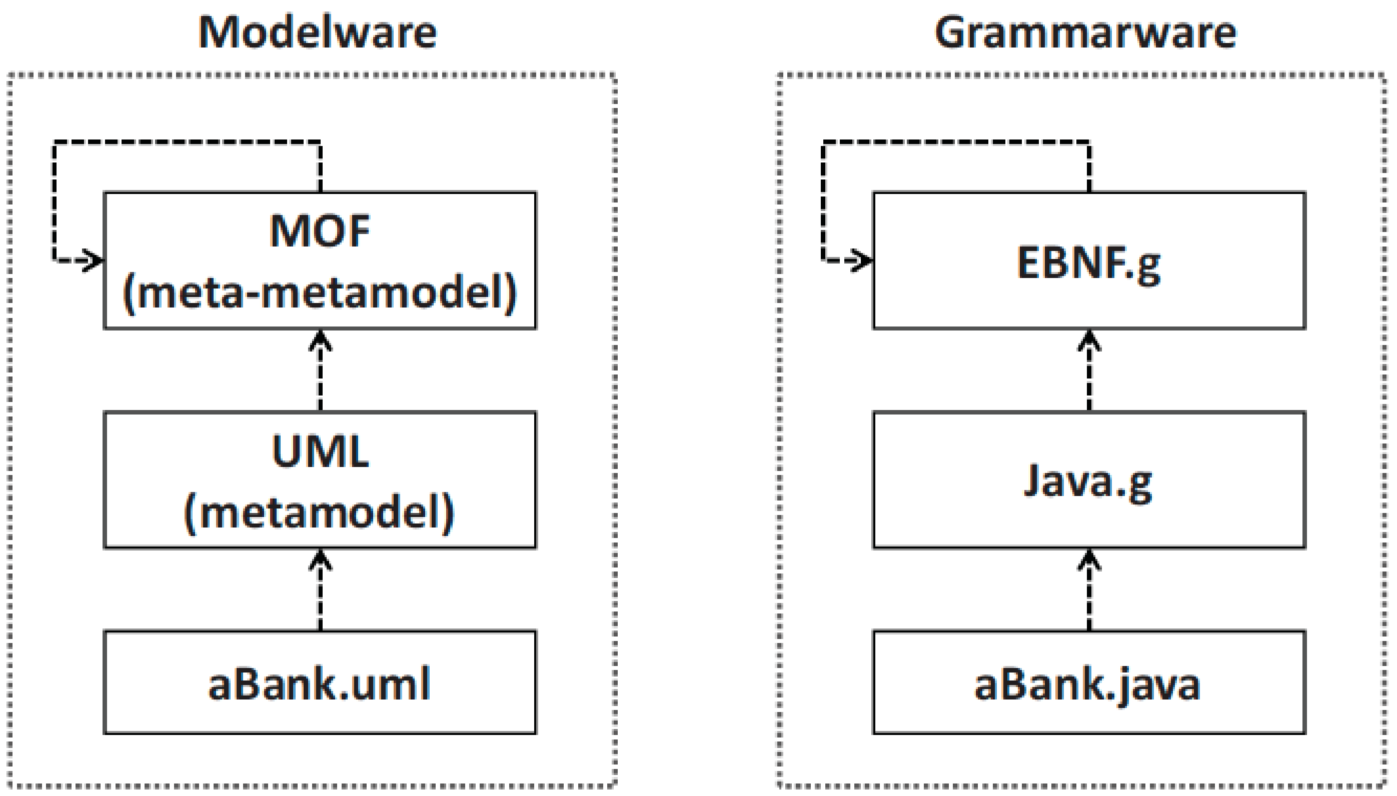
Concepts

Model Engineering basic architecture



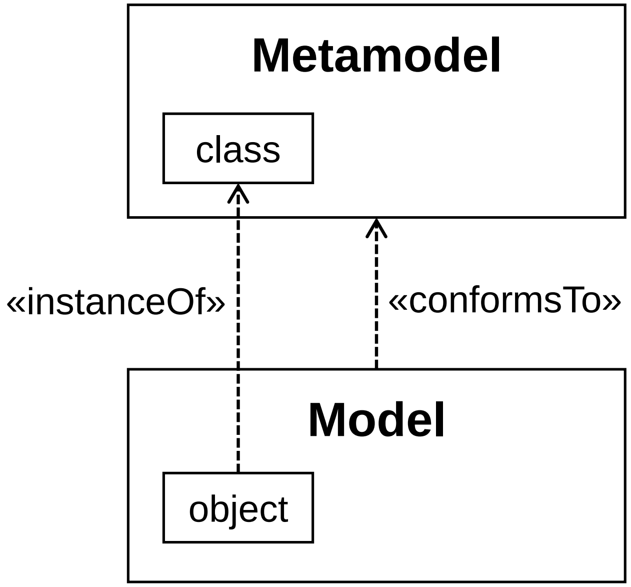
Modelware vs. Grammarware

- Two technical spaces



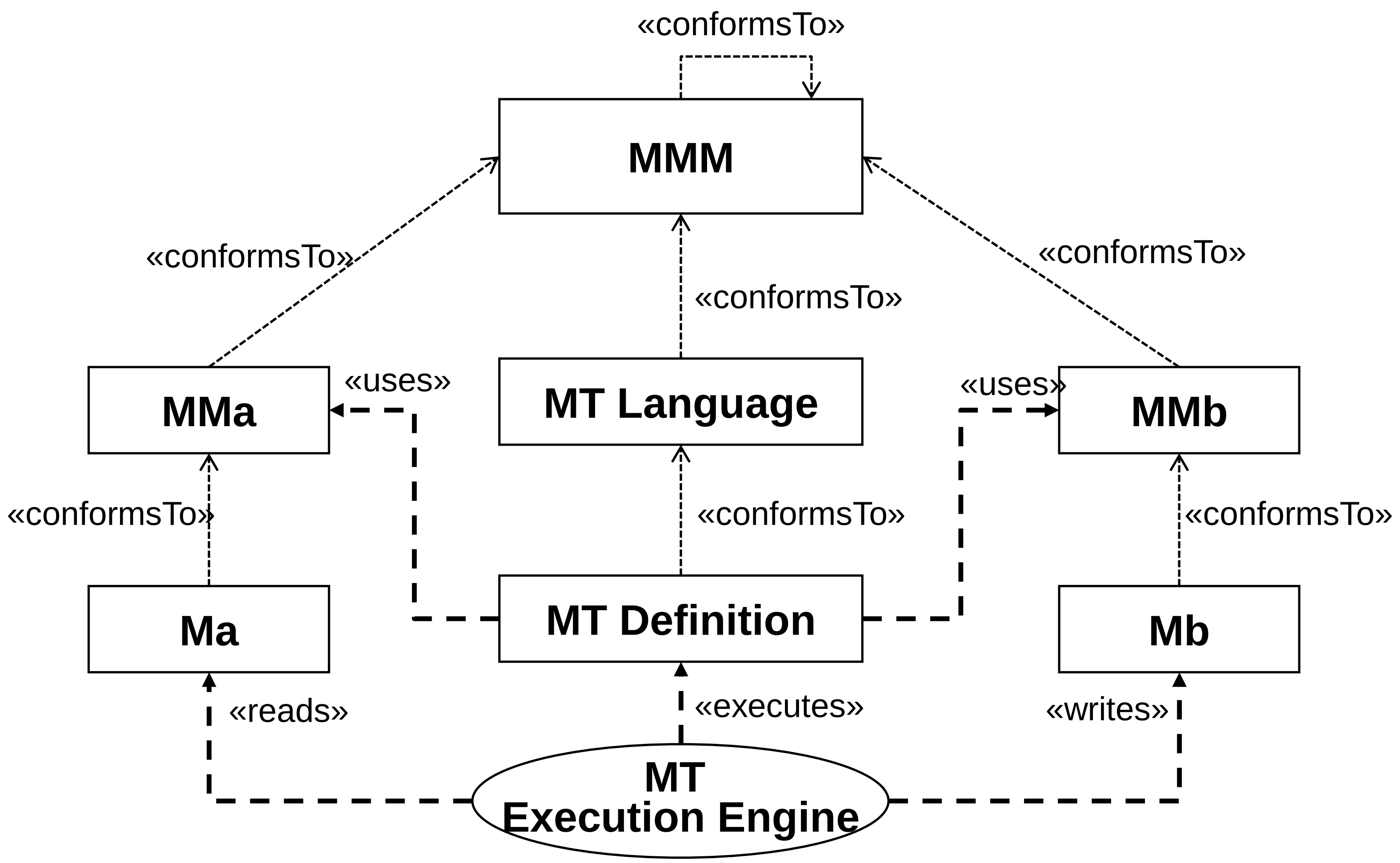
InstanceOf vs. ConformsTo

- Conformance is between models
- Instantiation is between model elements



Model Transformations

MOF and transformation setting



Types of models

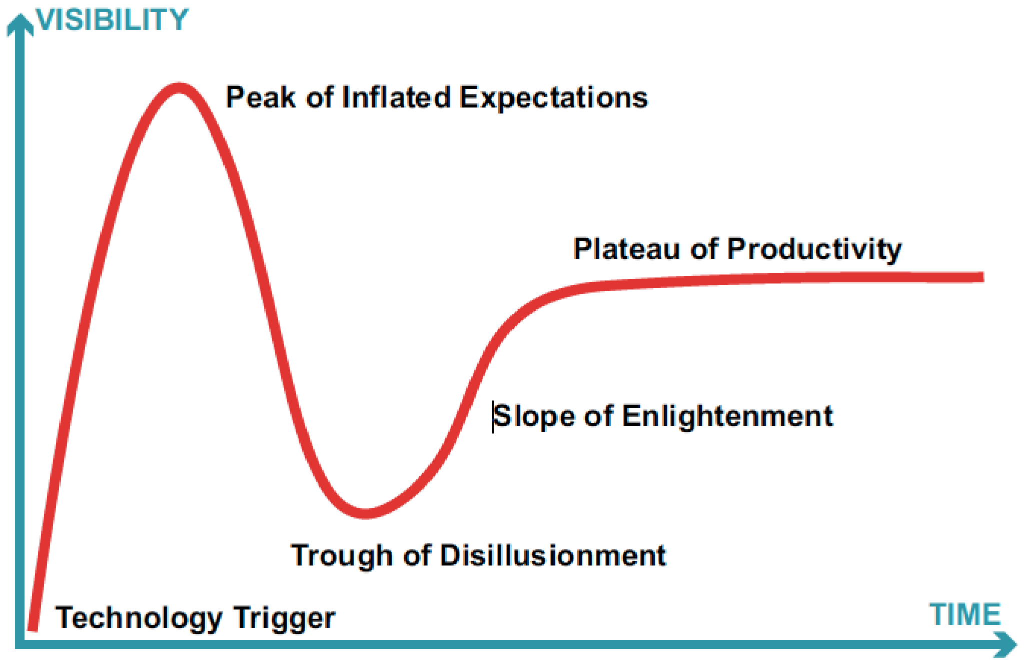
- **Static models:** Focus on the static aspects of the system in terms of managed data and of structural shape and architecture of the system.
- **Dynamic models:** Emphasize the dynamic behavior of the system by showing the execution.

- Just think about UML!

MDSE industry

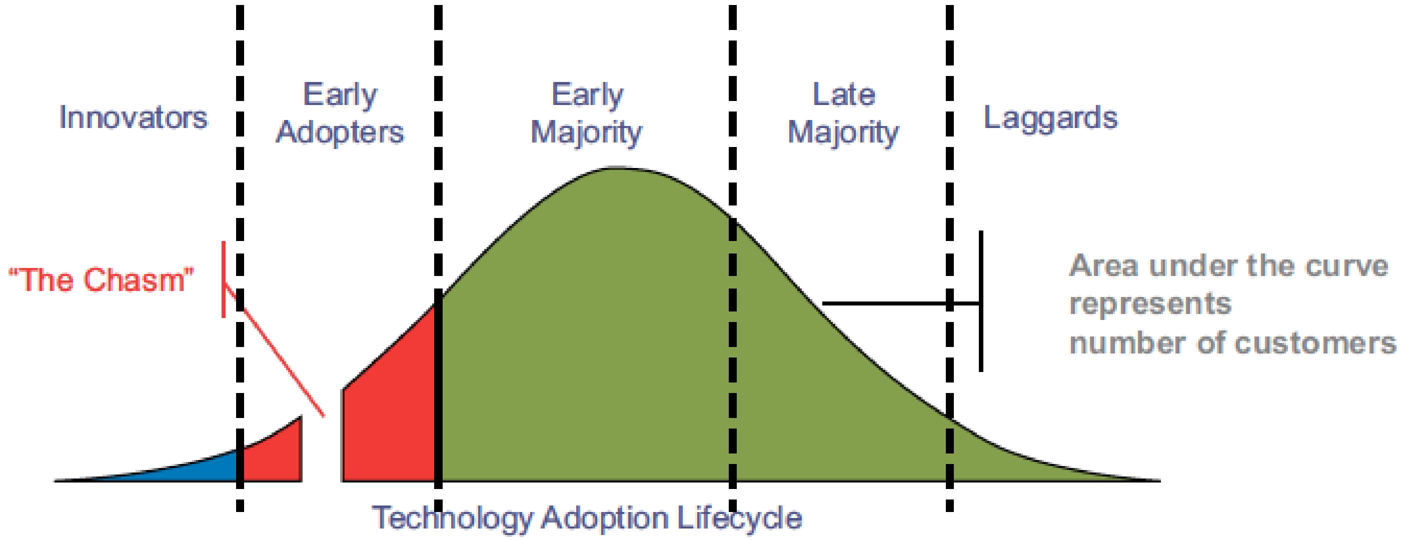
Adoption and acceptance (hype)

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



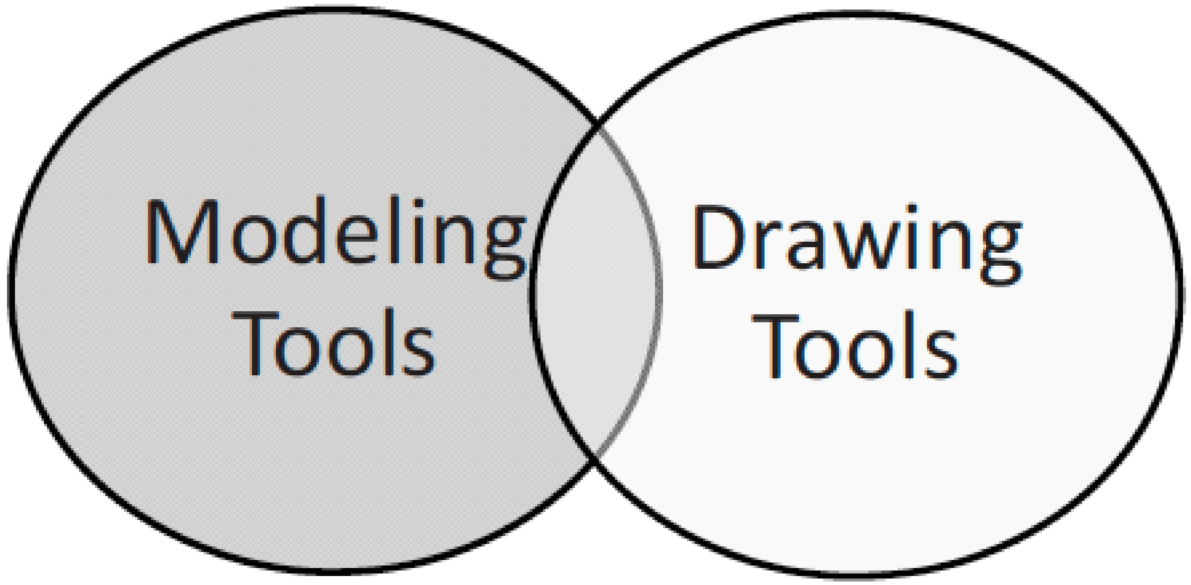
MDSE Industry (2)

Adoption Lifecycle


Innovators
Early Adopters
Early Majority
Late Majority
Laggards
"The Chasm"
Area under the curve represents number of customers
Technology Adoption Lifecycle

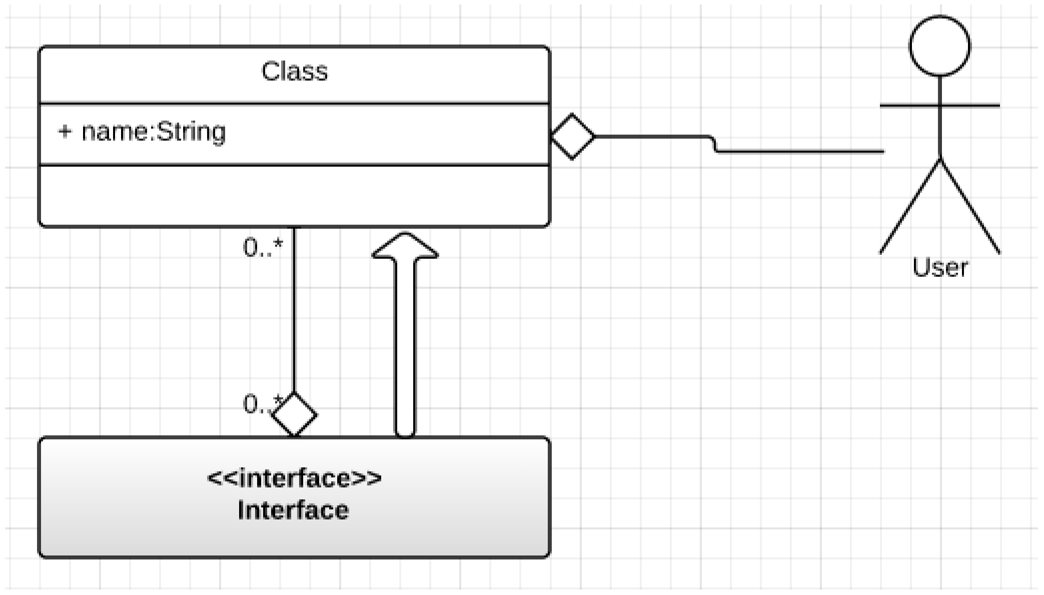
Tool support

- Drawing vs. modeling



Tool support

- Drawing vs. modeling



getAllRectangularShapes vs. getAllClasses

Model-based vs. Programming-based MDSE Tools

- Model-based: developed using MDSE principles → one should apply the principles ones advocates
- Programming-based: developed using traditional coding techniques

Eclipse and EMF

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

Conclusion

Modeling in the last century

- 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

Conclusion

Modeling in the new millennium – Much has changed!

- »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

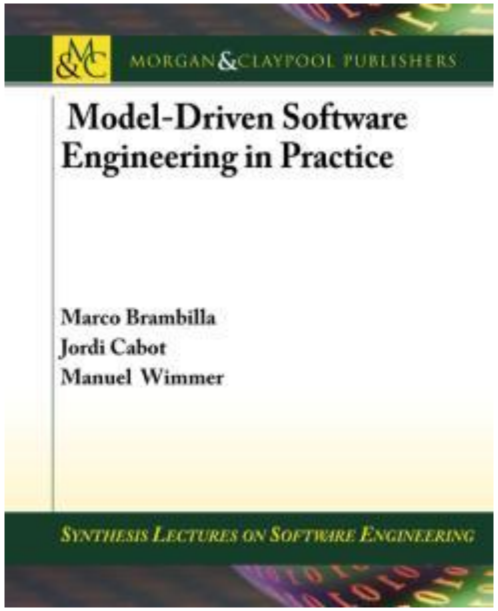
M. Fowler, "UML Distilled", 1st edition, Addison Wesley, 1997
(revisited in 2009)

MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.

www.mdse-book.com
www.morganclaypool.com
or buy it on www.amazon.com


Morgan & Claypool Publishers

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

Synthesis Lectures on Software Engineering