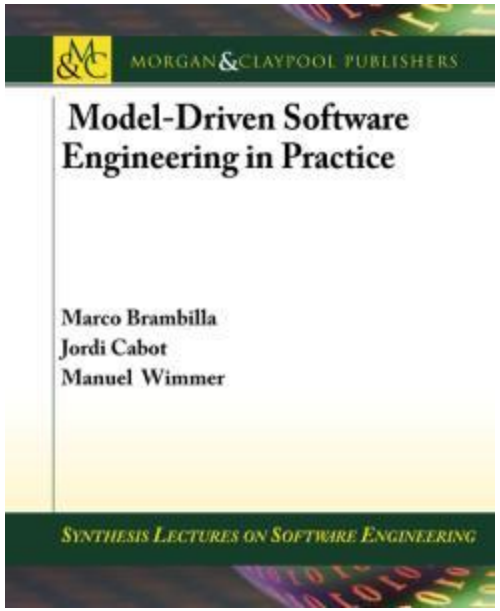
Chapter #1

INTRODUCTION

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



Introduction

Contents

- Human cognitive processes
- Models
- Structure of the book

Abstraction and human mind

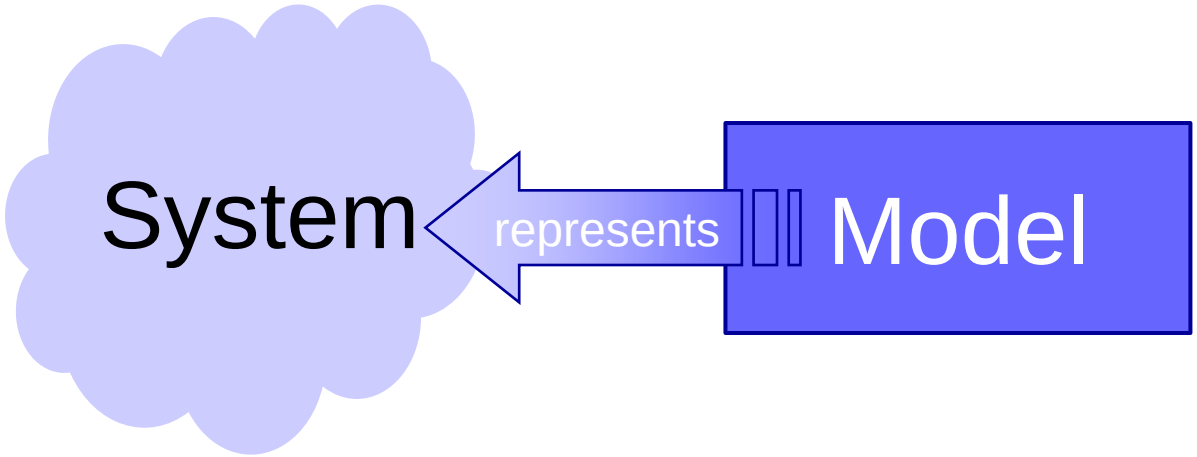
- The human mind continuously re-works reality by applying cognitive processes

- **Abstraction:** capability of finding the commonality in many different observations:
  - generalize specific features of real objects (generalization)
  - classify the objects into coherent clusters (classification)
  - aggregate objects into more complex ones (aggregation)

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

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

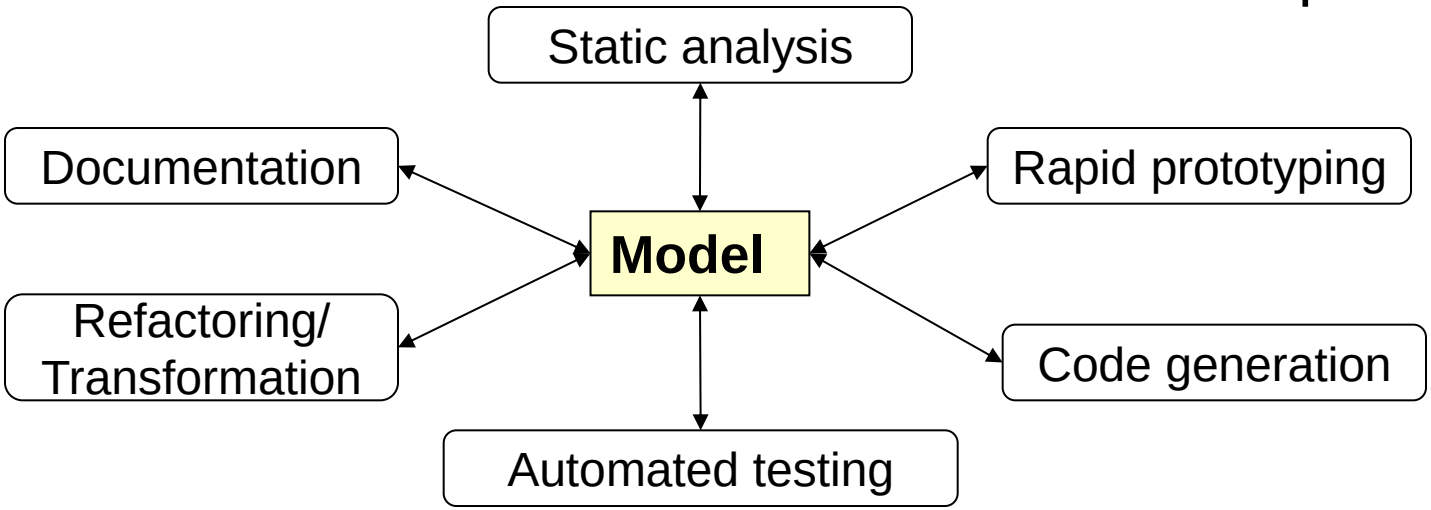
Purposes:

- descriptive purposes
- prescriptive purposes

Motivation

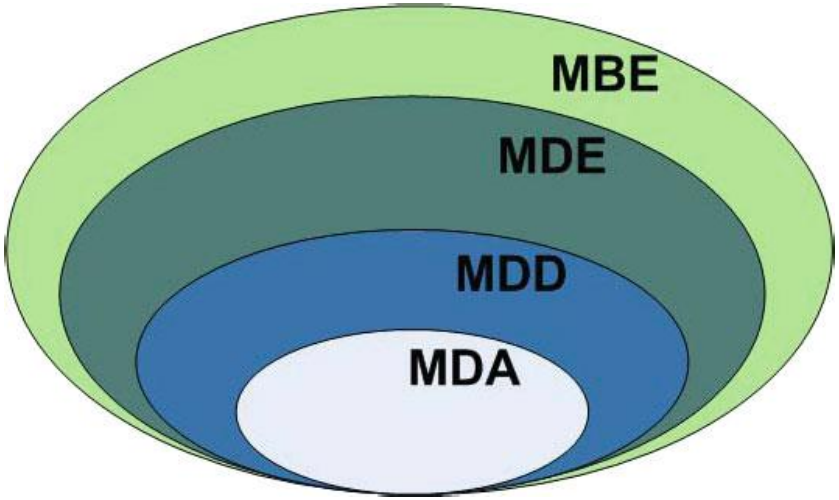
What is Model Engineering?

- Model as the **central artifact** of software development



[Illustration by Bernhard Rumpe]

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.

Motivation

Why Model Engineering?

- **Traditional** usage of models in software development
  - **Communication** with customers and users (requirement specification, prototypes)
  - Support for software design, capturing of the **intention**
  - **Task specification** for programming
  - **Code visualization**, for example in TogetherJ

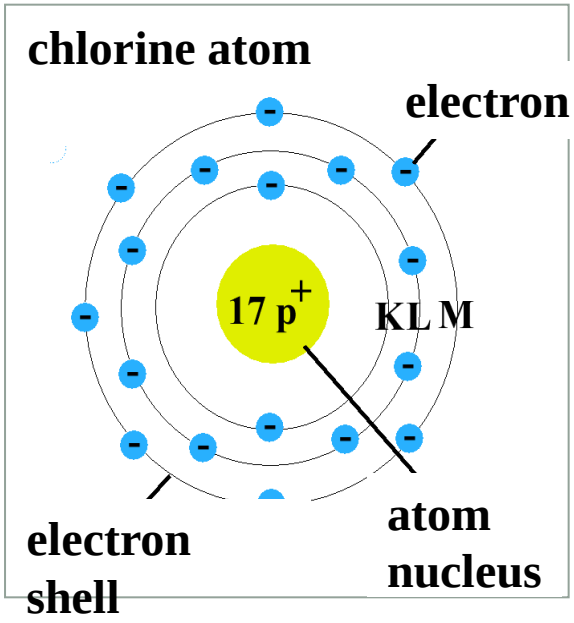
- What is the **difference** to Model Engineering?

Motivation

Usage of models

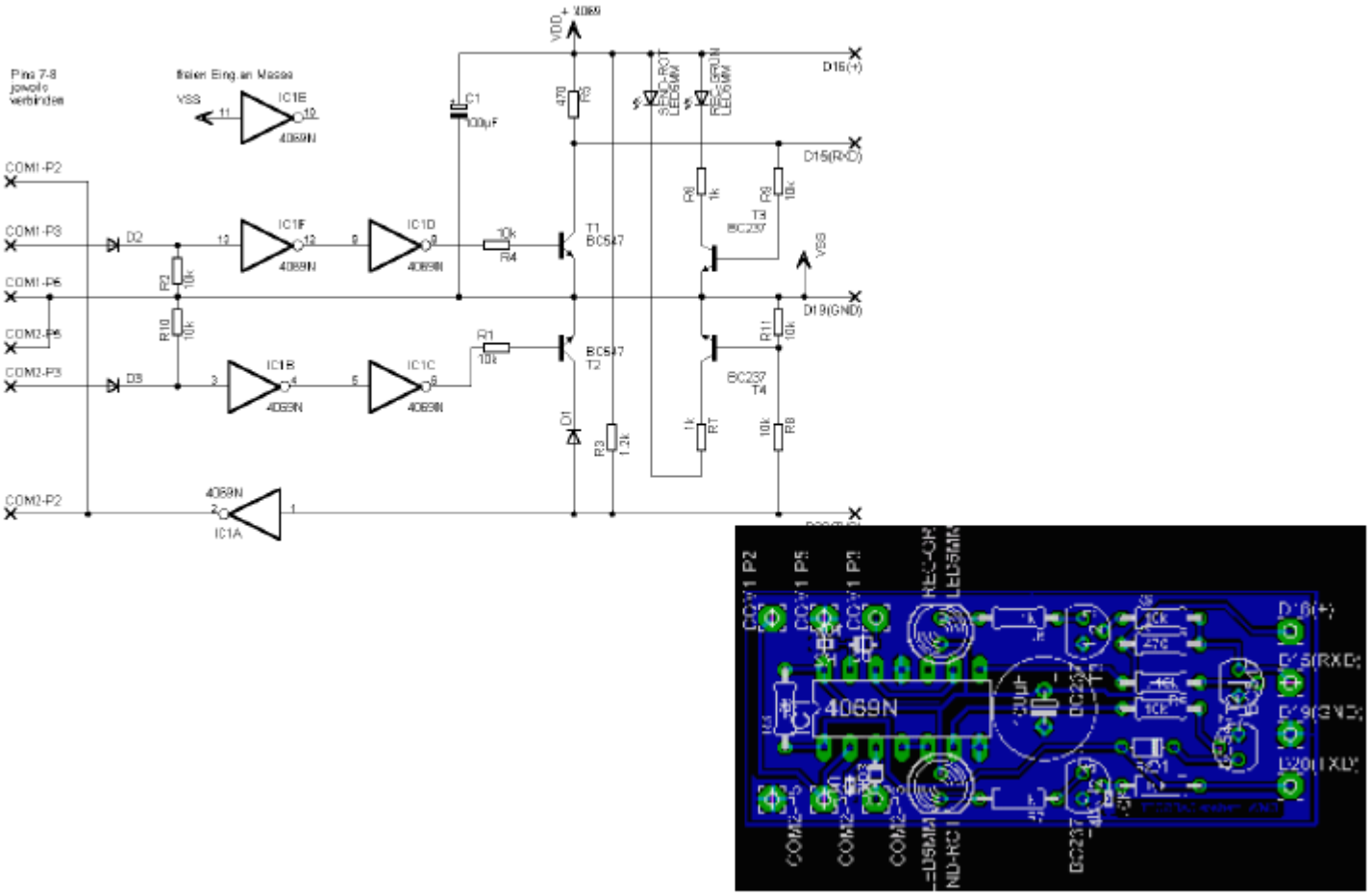
- Do not apply models as long as you have not checked the underlying **simplifications** and evaluated its **practicability**.

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



Motivation

Constructive models (Example: Electrical Engineering)

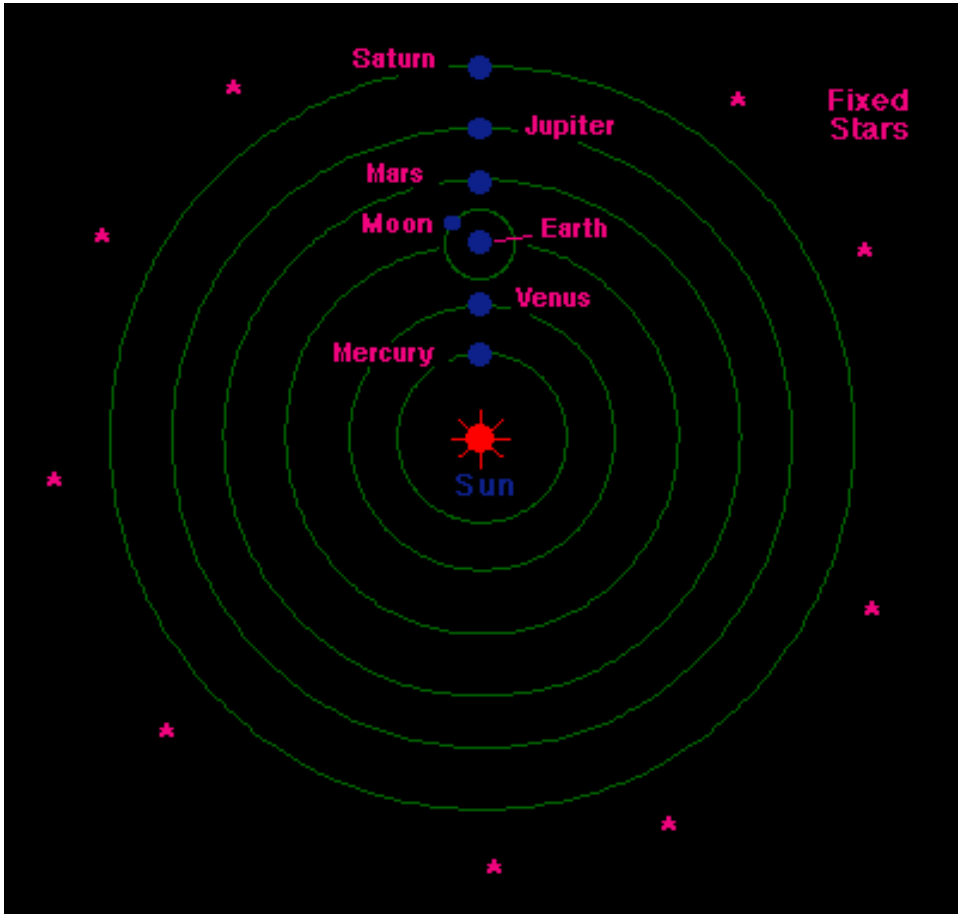


[Slide by Bernhard Rumpe]

Motivation

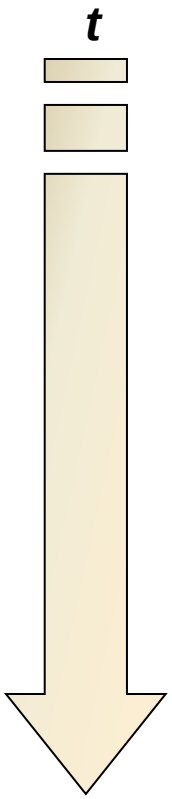
Declarative models (Example: Astronomy)

- Heliocentric model by Kopernikus



Motivation

Application area of modeling

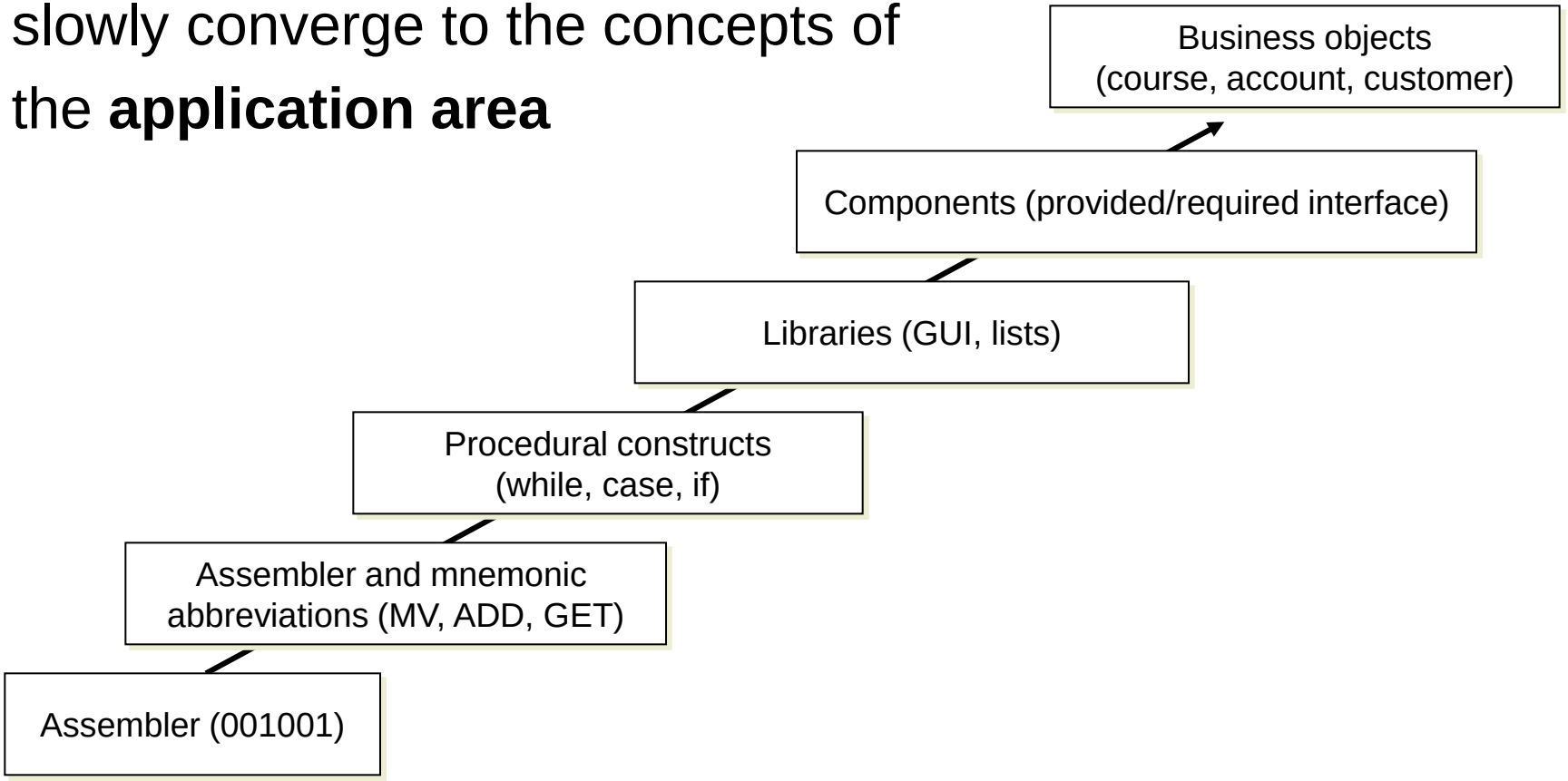


- ***Models as drafts***
  - Communication of ideas and alternatives
  - Objective: modeling per se
- ***Models as guidelines***
  - Design decisions are documented
  - Objective: instructions for implementation
- ***Models as programs***
  - Applications are generated automatically
  - Objective: models are source code and vice versa

Motivation

Increasing abstraction in software development

- The **used artifacts of software development** slowly converge to the concepts of the **application area**



[Illustration by Volker Gruhn]

Structure of the book

PART 1: MDSE Foundations

- **1 Introduction**
- 1.1 Purpose and Use of Models
- 1.2 Modeling for Software Development
- 1.3 How to Read this Book

- **2 MDSE Principles**
- 2.1 MDSE Basics
- 2.2 Lost in Acronyms: The MD* Jungle
- 2.3 Overview of the MDSE Methodology
- 2.3.1 Overall Vision
- 2.3.2 Target of MDSE: Domains, Platforms,Technical Spaces, and Scenarios
- 2.3.3 Modeling Languages
- 2.3.4 Metamodeling
- 2.3.5 Transformations
- 2.3.6 Model Classification
- 2.4 MDSE Adoption in Industry
- 2.5 Tool Support
- 2.5.1 Drawing Tools vs Modeling Tools
- 2.5.2 Model-Based vs Programming-Based MDSE Tools
- 2.5.3 Eclipse and EMF
- 2.6 Criticisms of MDSE

Structure of the book

PART 1: MDSE Foundations (continued)

- **3 MDSE Use Cases**
- 3.1 Automating Software Development
- 3.1.1 Code Generation
- 3.1.2 Model Interpretation
- 3.1.3 Combining Code Generation and Model Interpretation
- 3.2 System Interoperability
- 3.3 Reverse Engineering

- **4 Model-Driven Architecture (MDA)**
- 4.1 MDA Definitions and Assumptions
- 4.2 The Modeling Levels: CIM, PIM, PSM
- 4.3 Mappings
- 4.4 General Purpose and Domain-Specific Languages in MDA
- 4.5 Architecture-Driven Modernization

- **5 Integration of MDSE in your Development Process**
- 5.1 Introducing MDSE in your Software Development Process
- 5.1.1 Pains and Gains of Software Modeling
- 5.1.2 Socio-Technical Congruence of the Development Process
- 5.2 Traditional Development Processes and MDSE
- 5.3 Agile and MDSE
- 5.4 Domain-Driven Design and MDSE
- 5.5 Test-Driven Development and MDSE
- 5.5.1 Model-Driven Testing
- 5.5.2 Test-Driven Modeling

Structure of the book

PART 1: MDSE Foundations (continued)

- **6 Modeling Languages at a Glance**
- 6.1 Anatomy of Modeling Languages
- 6.2 General Purpose vs Domain-Specific Modeling Languages
- 6.3 General-Purpose Modeling: The Case of UML
- 6.4 UML Extensibility: The MiddleWay Between GPL and DSL
- 6.5 Overview on DSLs (Domain Specific Languages)
- 6.5.1 Principles of DSLs
- 6.5.2 Some Examples of DSLs
- 6.6 Defining Modeling Constraints (OCL)

PART 2: MDSE Technologies

- **7 Developing yourOwn Modeling Language**
- 7.1 Metamodel-Centric Language Design
- 7.1.1 Abstract Syntax
- 7.1.2 Concrete Syntax
- 7.1.3 Language Ingredients at a Glance
- 7.2 Example DSML: sWML
- 7.3 Abstract Syntax Development
- 7.3.1 Metamodel Development Process
- 7.3.2 Metamodeling in Eclipse
- 7.4 Concrete Syntax Development
- 7.4.1 Graphical Concrete Syntax (GCS)
- 7.4.2 Textual Concrete Syntax (TCS)

Structure of the book

PART 2: MDSE Technologies (continued)

- **8 Model-to-ModelTransformations**
- 8.1 Model Transformations and their Classification
- 8.2 Exogenous, Out-Place Transformations
- 8.3 Endogenous, In-Place Transformations
- 8.4 Mastering Model Transformations
- 8.4.1 Divide and Conquer: Model Transformation Chains
- 8.4.2 HOT: Everything is a Model, Even Transformations!
- 8.4.3 Beyond Batch: Incremental and Lazy Transformations
- 8.4.4 Bi-Directional Model Transformations

- **9 Model-to-TextTransformations**
- 9.1 Basics of Model-Driven Code Generation
- 9.2 Code Generation Through Programming Languages
- 9.3 Code Generation Through M2T Transformation Languages
- 9.3.1 Benefits of M2T Transformation Languages
- 9.3.2 Template-Based Transformation Languages: an Overview
- 9.3.3 Acceleo: An Implementation of the M2T Transformation Standard
- 9.4 Mastering Code Generation
- 9.5 Excursus: Code Generation Through M2M Transformations and TCS

Structure of the book

PART 2: MDSE Technologies (continued)

- **10 Managing Models**
- 10.1 Model Interchange
- 10.2 Model Persistence
- 10.3 Model Comparison
- 10.4 Model Versioning
- 10.5 Model Co-Evolution
- 10.6 Global Model Management
- 10.7 Model Quality
- 10.7.1 Verifying Models
- 10.7.2 Testing and Validating Models
- 10.8 Collaborative Modeling

- **11 Summary**

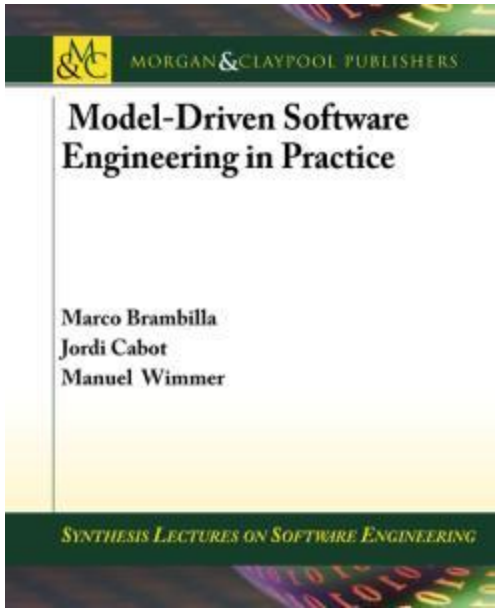
- Bibliography

- Authors' Biographies

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.



MDSE Principles

Contents

- Concepts
- Approaches
- Adoption

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 and 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

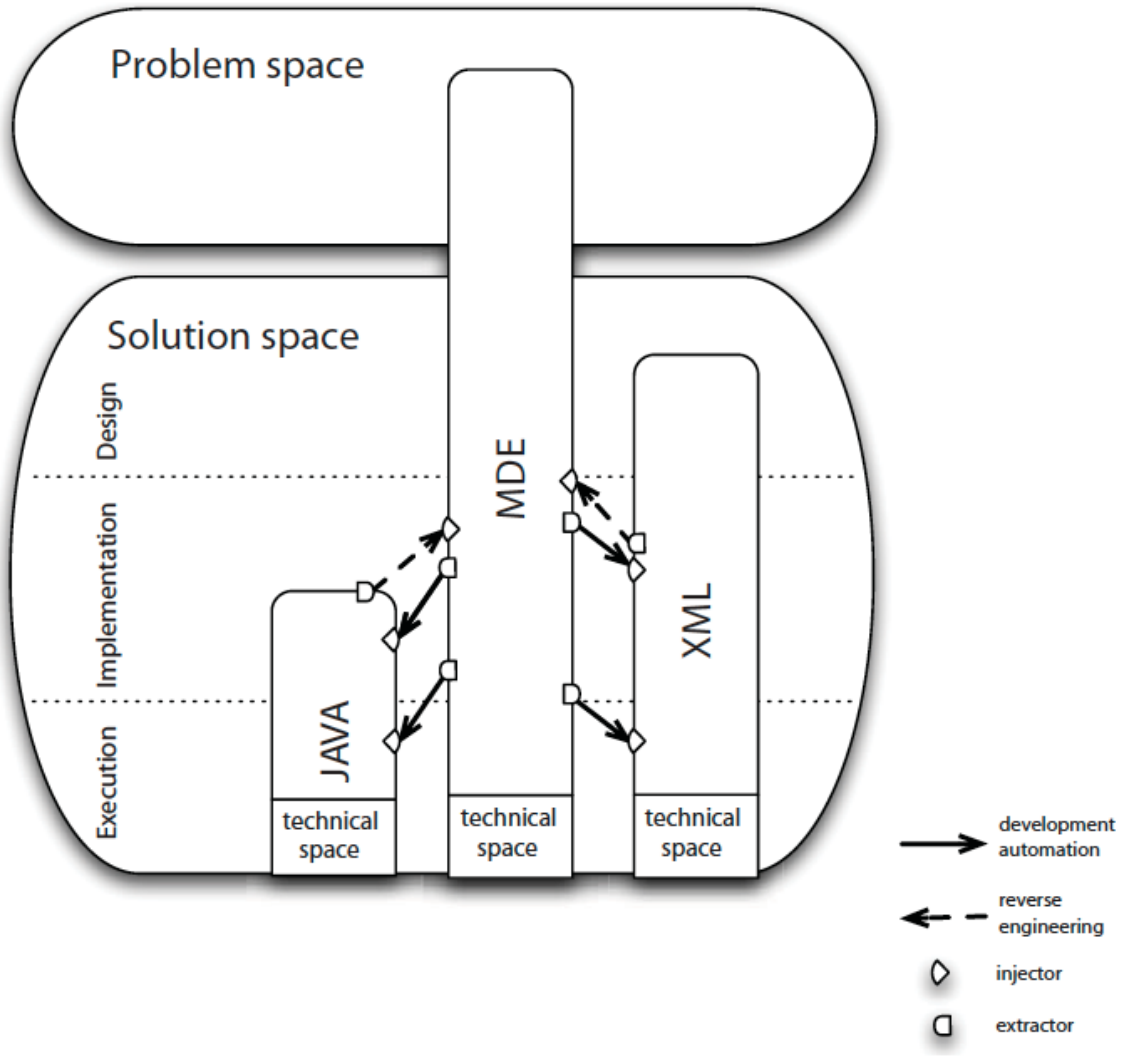
- **Concepts:** The components that build up the methodology
- **Notations:** The way in which concepts are represented
- **Process and rules:** The activities that lead to the production of the final product
- **Tools:** Applications that ease the execution of activities or their coordination

MDSE Equation

Models + Transformations = Software

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

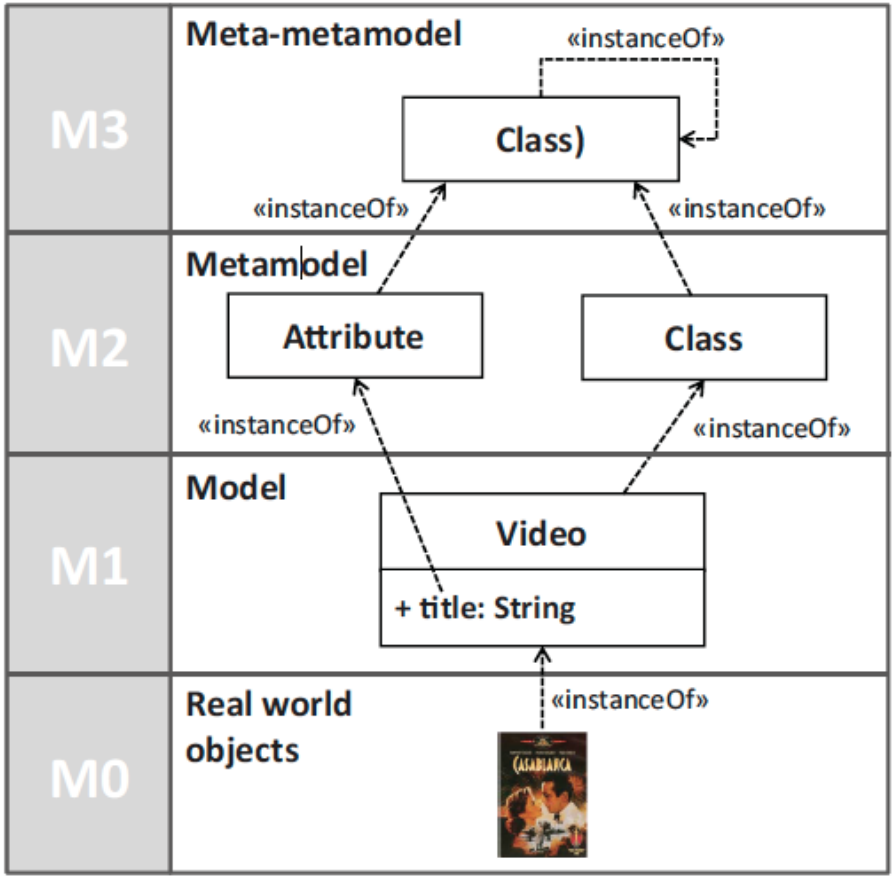
- **Domain-Specific Languages (DSLs):** languages that are designed specifically for a certain domain or context
- DSLs have been largely used in computer science. Examples: HTML, Logo, VHDL, Mathematica, SQL

- **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)

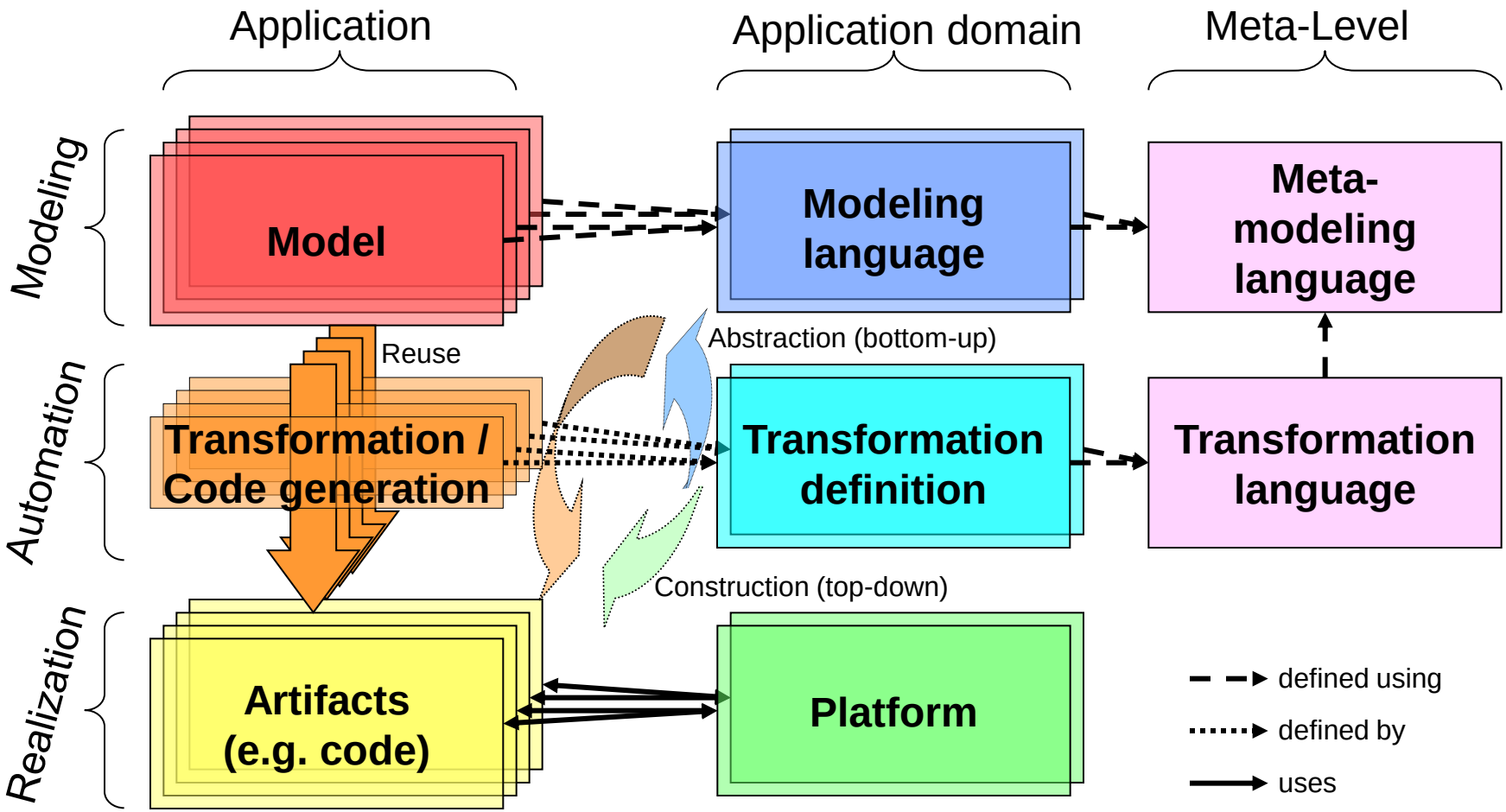


Model Transformations

- 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!!

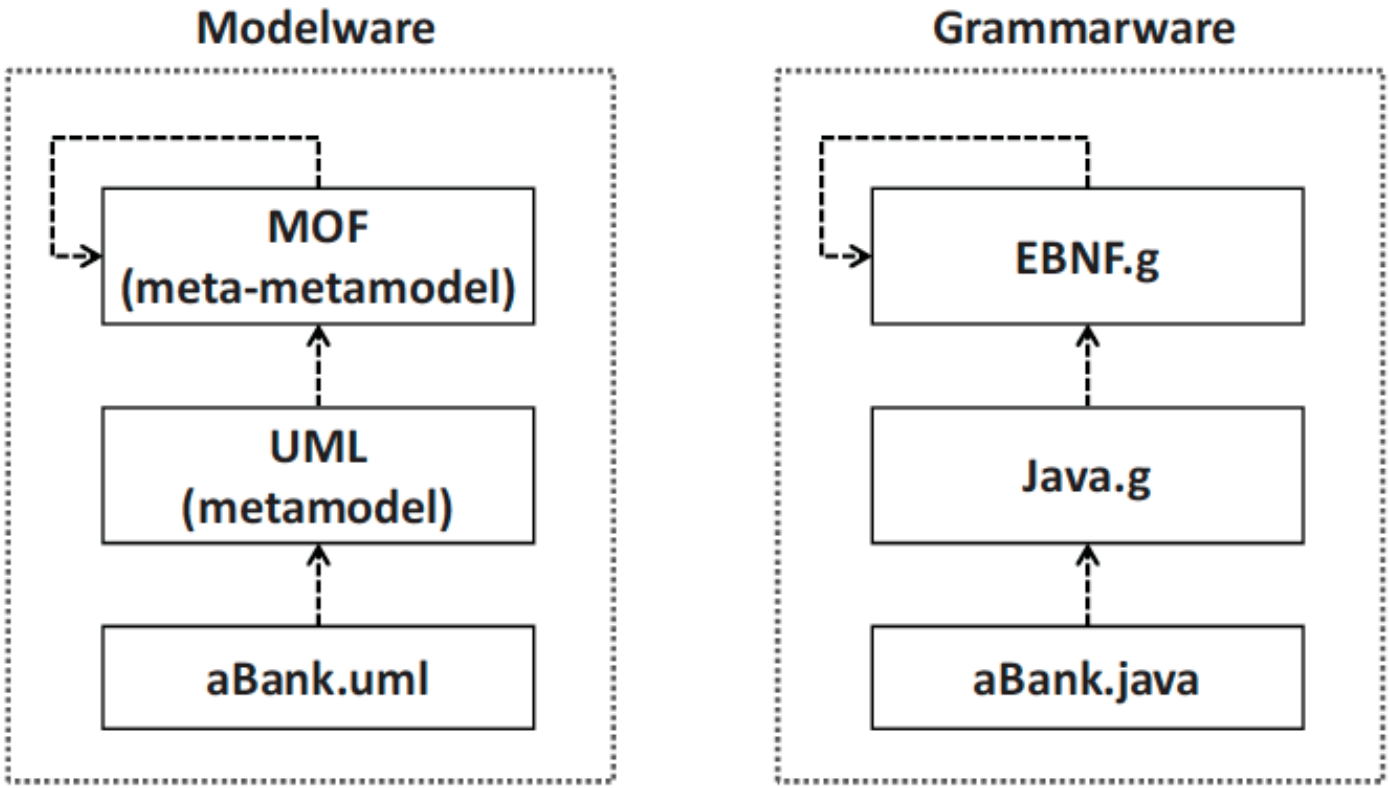
Concepts

Model Engineering basic architecture



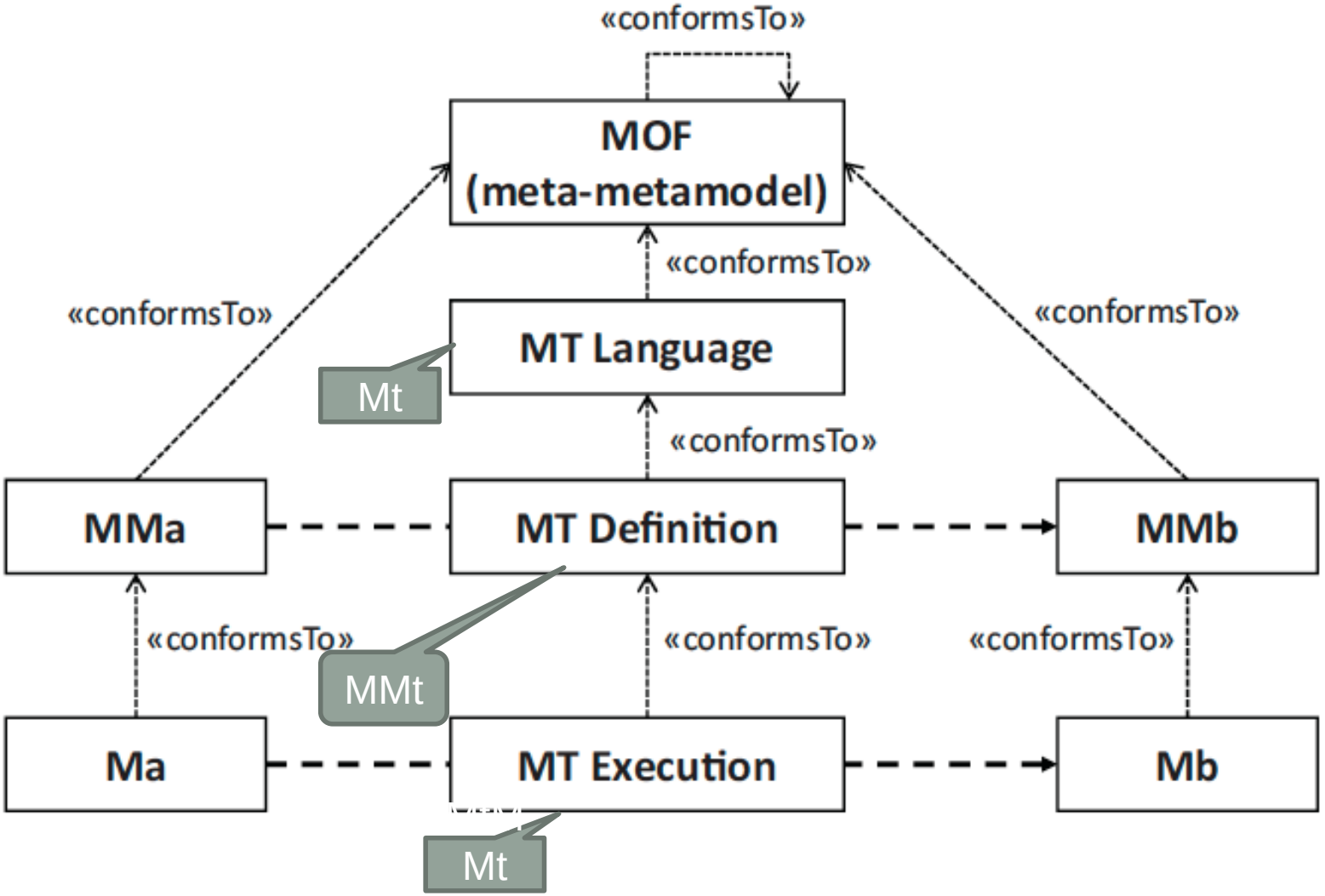
Modelware vs. Grammarware

- Two technical spaces



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!

Approaches

CASE

- **Historic** approach (end of 20th century)
- **Example**: Computer Associates' AllFusion Gen
  - Supports the Information Engineering Method by James Martin by a series of diagram types (incl. user interface)
  - Fully automated code generation for one architecture (3-Tier) and plenty of execution platforms (Mainframe, Unix, .NET, J2EE, different databases, …)
  - Advantage/Disadvantage: no handling with the target platform required/possible
- Different **implementation versions of the basic architecture**
  - Meta-Level often not supported / not accessible
  - Modeling language often fixed, tool specific versions
  - Execution platform often not considered or fixed
- **Advantages**
  - Productivity, development and maintenance costs, quality, documentation
- **Disadvantages**
  - Proprietary (version of a) modeling language
  - Tool interoperability nonexistent
  - Strongly dependent on the tool vendor regarding execution platforms, further development
  - Tools are highly complex

Approaches

Executable UML

- **“CASE with UML”**
  - **UML-Subset**: Class Diagram, State Machine, Package/Component Diagram, as well as
  - UML Action Semantic Language (ASL) as programming language
- **Niche product**
  - Several specialized vendors like Kennedy/Carter
  - Mainly used for the development of Embedded Systems
- One **part of the basic architecture** implemented
  - Modeling language is predetermined (**xUML**)
  - Transformation definitions can be adapted or can be established by the user (via ASL)
- **Advantages** compared to CASE
  - Standardized modeling language based on the UML
- **Disadvantages** compared to CASE
  - Limited extent of the modeling language

[S.J. Mellor, M.J. Balcer: Executable UML: a foundation for model-driven architecture. Addison-Wesley, 2002]

Approaches

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; compare 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

[www.omg.org/mda/]

Modeling Levels

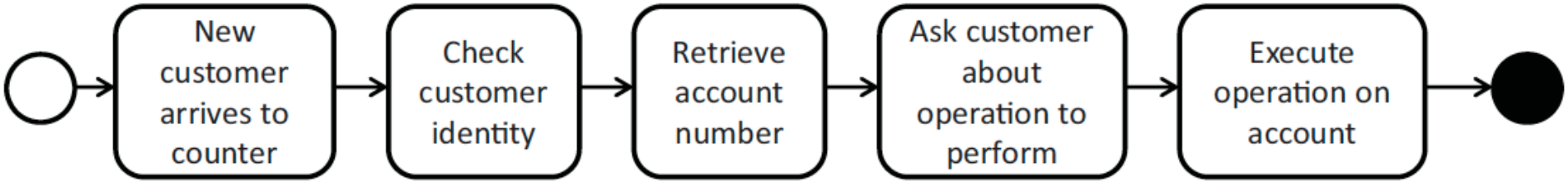
CIM, PIM, PSM

- Computation independent (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 (PIM): define the behavior of the systems in terms of stored data and performed algorithms, without any technical or technological details;
- Platform-specific (PSM): define all the technological aspects in detail.

Modeling levels

CIM

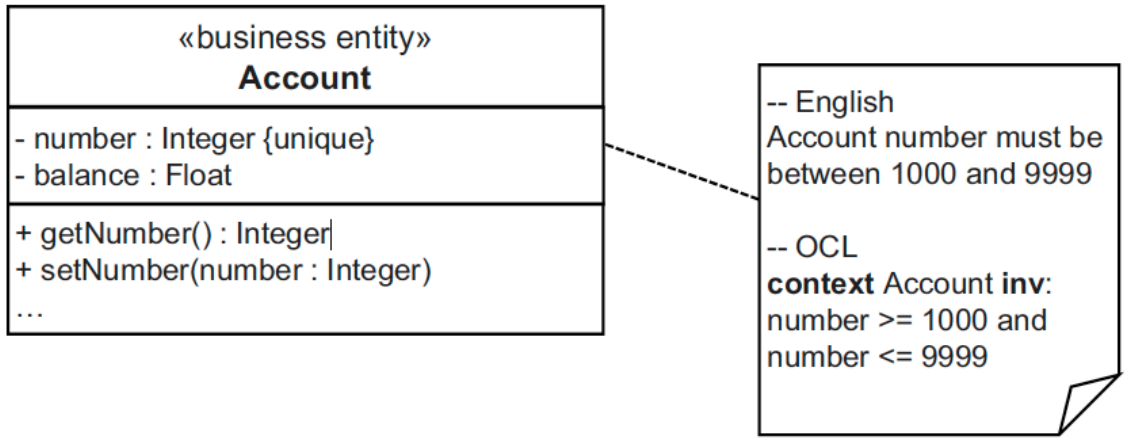
- Eg., business process



Modeling levels

MDA Platform Independent Model (PIM)

- specification of structure and behaviour of a system, abstracted from technologicical details

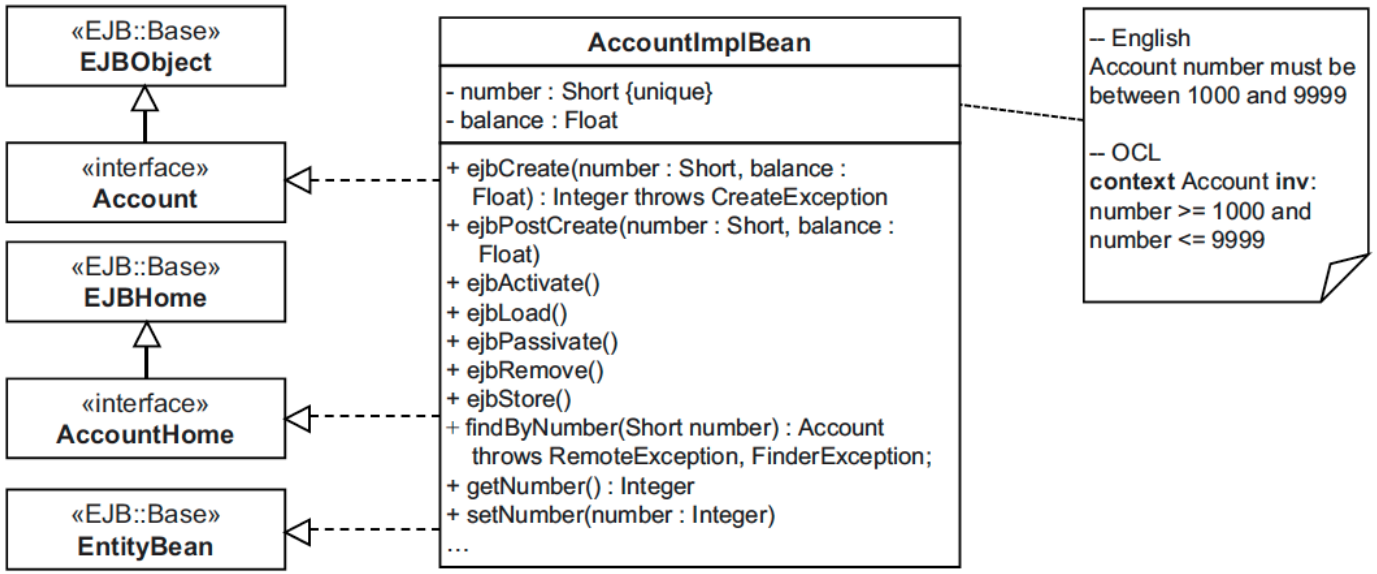


- Using the UML(optional)

- Abstraction of structure and behaviour of a system with the PIM simplifies the following:
  - Validation for correctness of the model
  - Create implementations on different platforms
  - Tool support during implementation

Modeling levels

MDA Platform Specific Model (PSM)



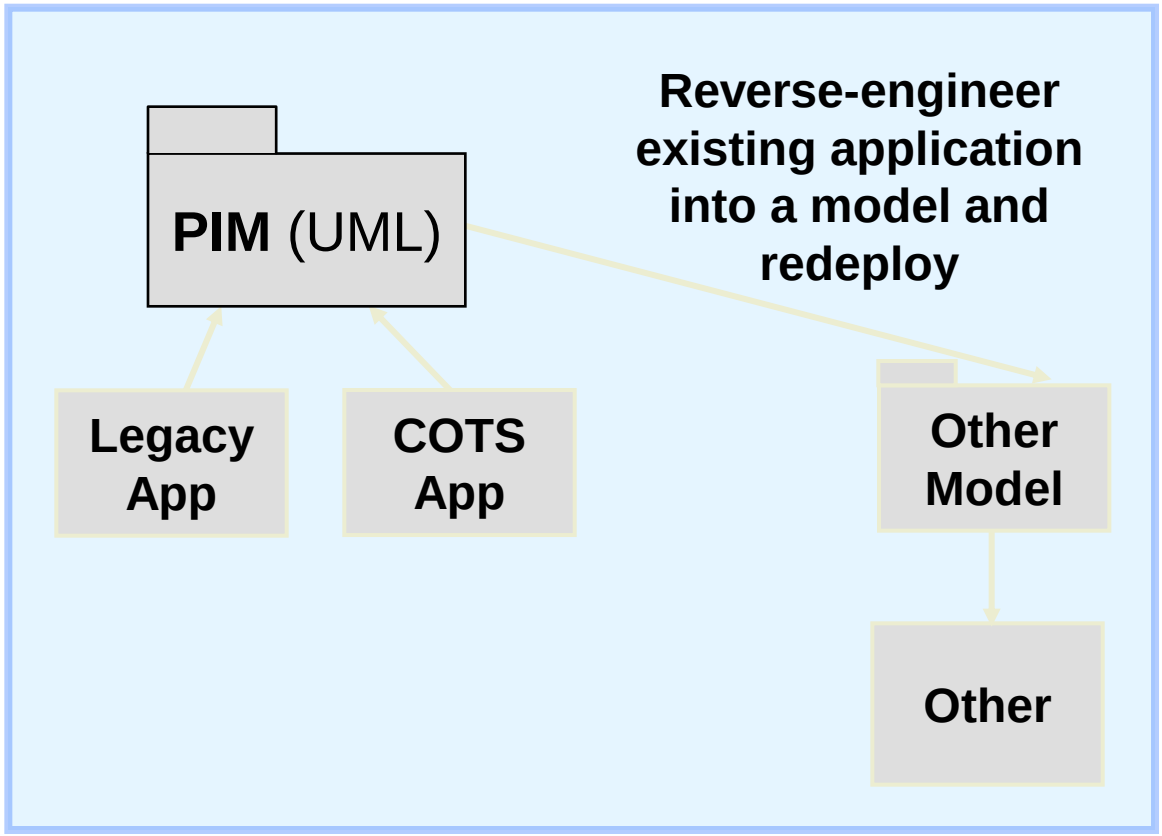
- Specifies how the functionality described in the PIM is realized on a certain platform
- Using a UML-Profile for the selected platform, e.g., EJB

Approaches

MDA Reverse Engineering / Roundtrip Engineering

- Re-integration onto new platforms via Reverse Engineering of an existing application into a PIM und subsequent code generation

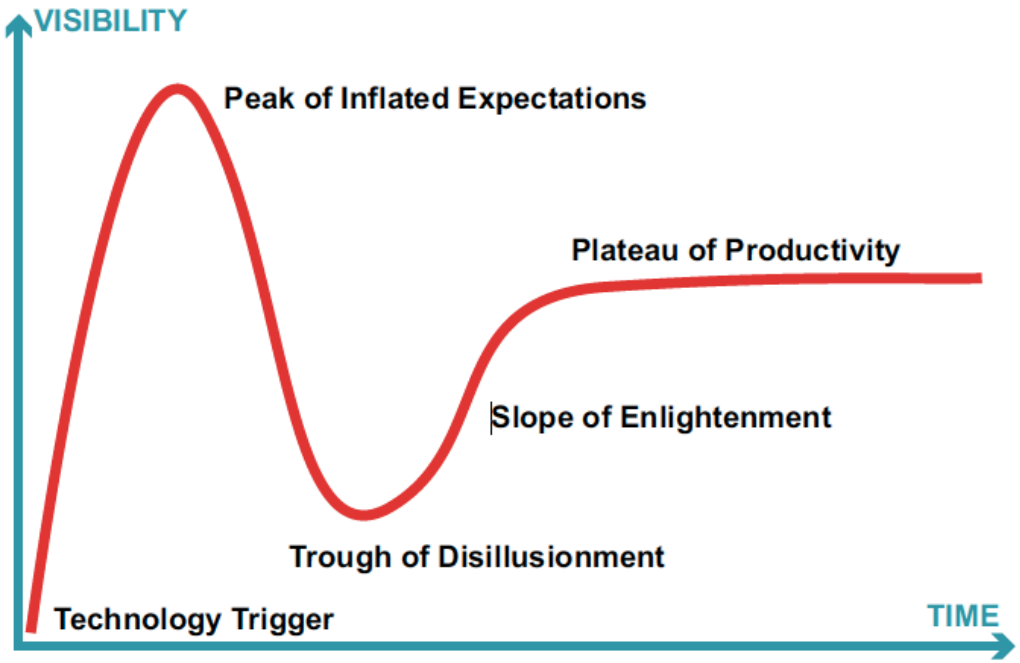
- MDA tools for Reverse Engineering automate the model construction from existing code



MDSE in Industry

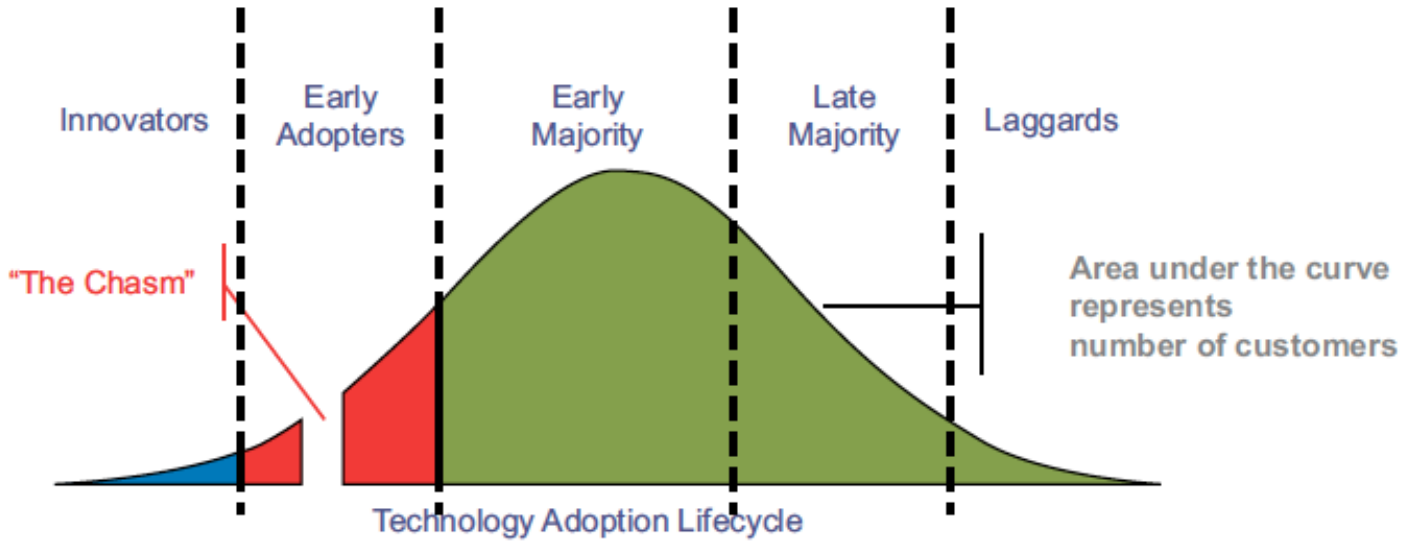
Adoption and acceptance (hype)

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



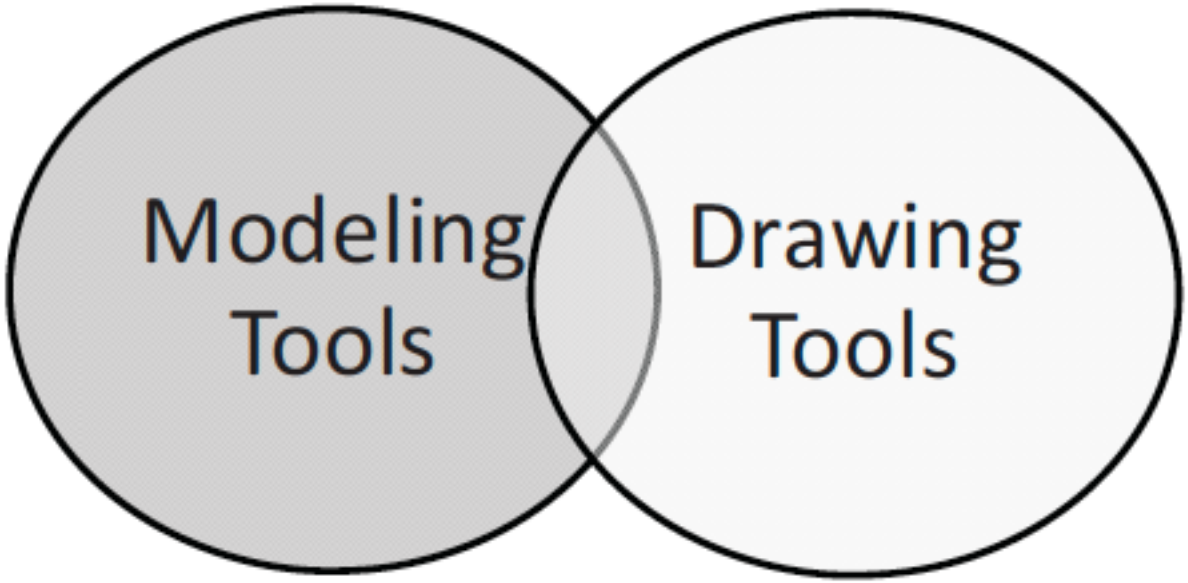
MDSE Industry (2)

Adoption



Tool support

- Drawing vs. modeling



Eclipse and EMF

- EMF (Eclipse Modeling Framework) is the core methodology in Eclipse o support MDE.
- Full support for metamodeling and language design
- Fully MD (vs. programming-based tools)
- Used in this course!

eclipseCON EUROPE 2012
Starts in less than 3 weeks
Ludwigsburg, Germany
October 23 - 25
Register Now

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