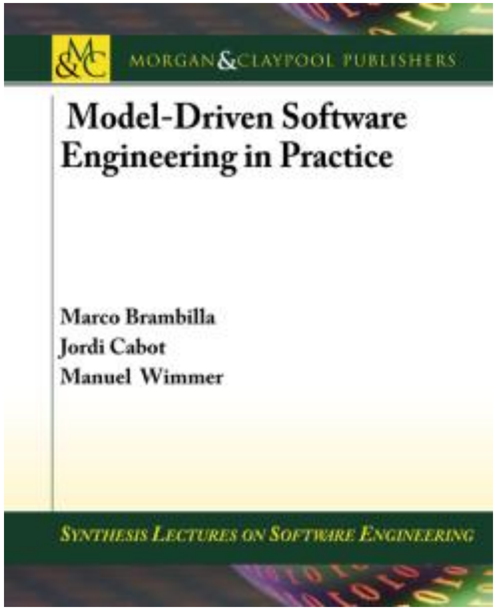
MORGAN & CLAYPOOL PUBLISHERS

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.



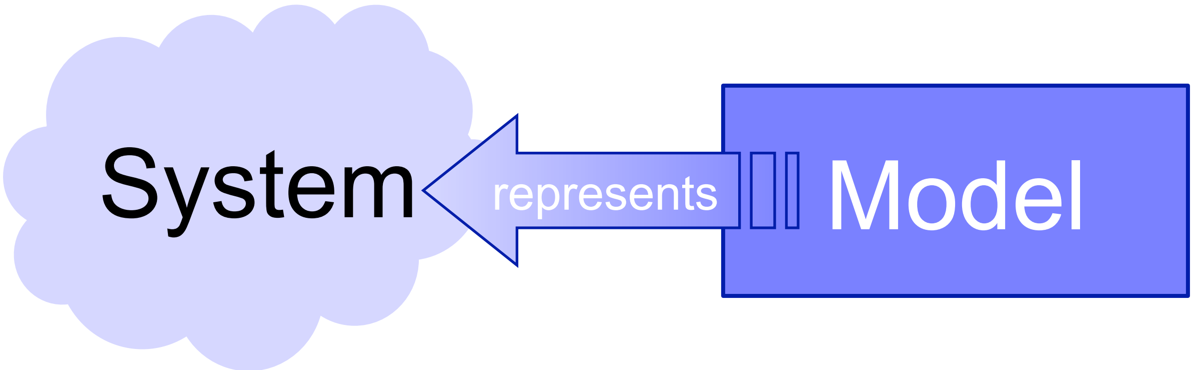
MDSE Principles

Contents

- Concepts
- Approaches
- Adoption

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

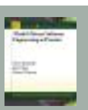
MDSE aim at large

- 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**



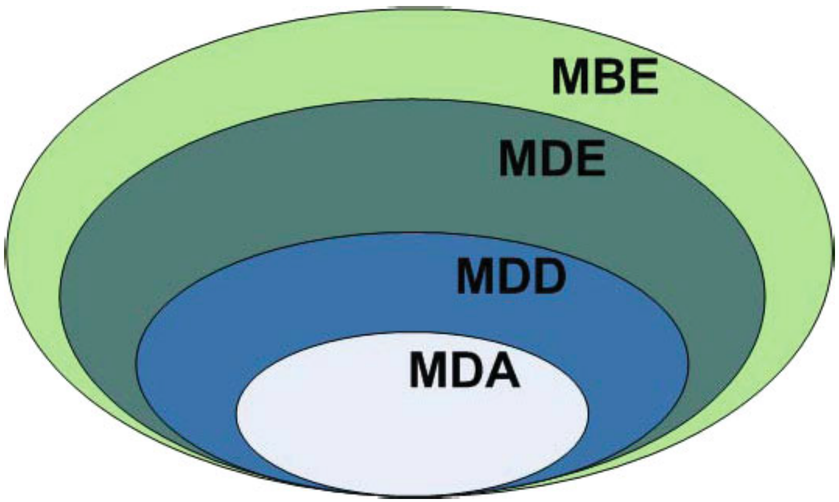
MDSE methodology ingredients

- **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

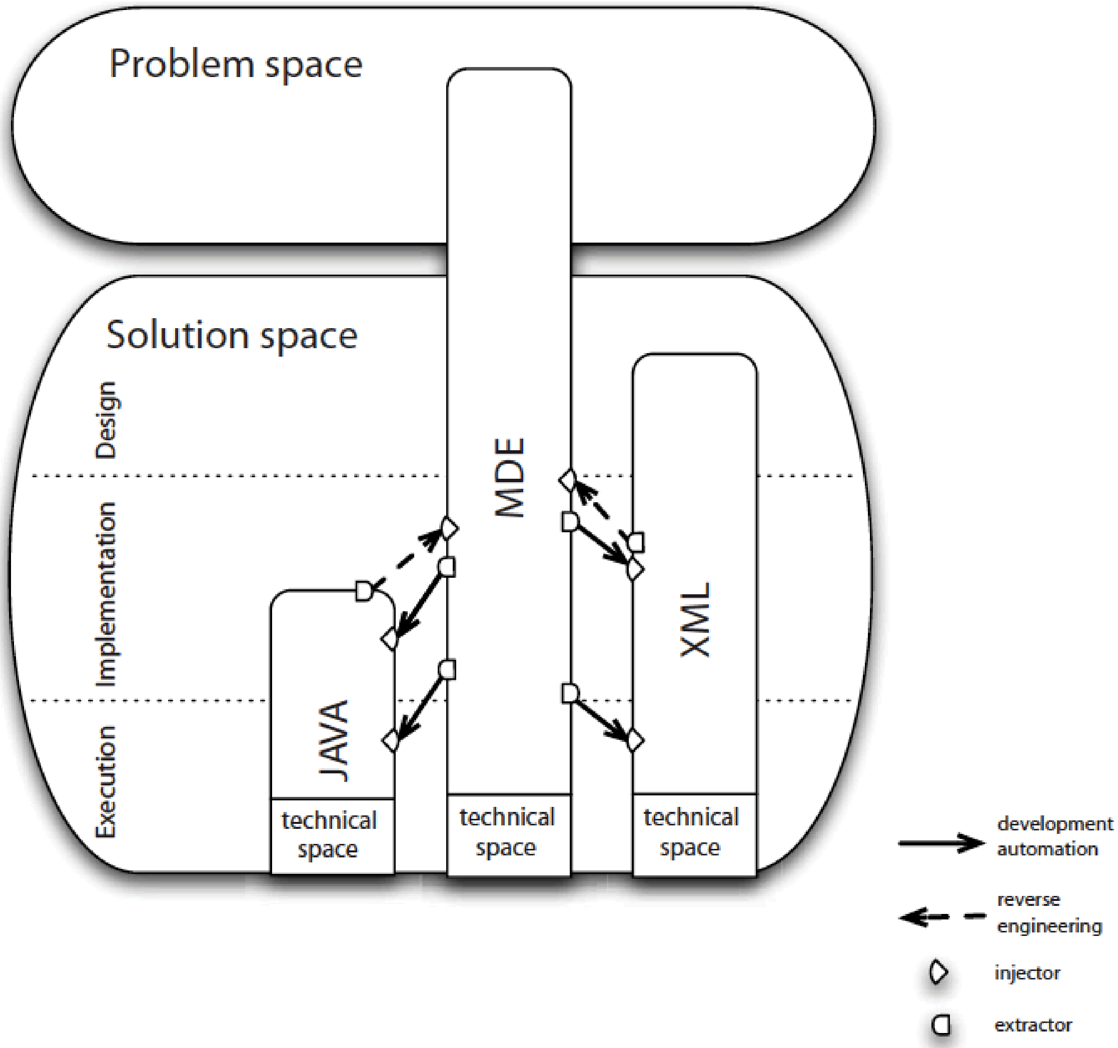
The MD* Jungle of Acronyms



- **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 becauseit 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

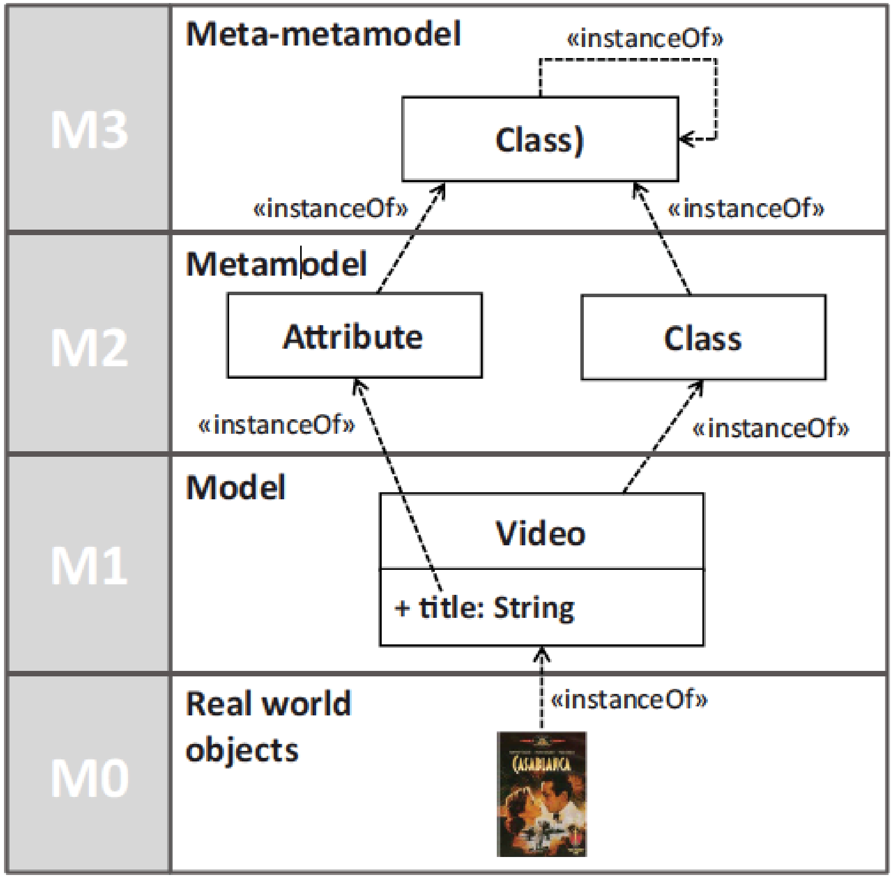
- **Domain-Specific Languages (DSLs):** languages that are designed specifically for a certain domain or context
- DSLs have been largely used in computer science. Examples: HTML, Logo, VHDL, Mathematica, SQL

- **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

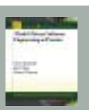
- To represent the models themselves as "instances" of some more abstract models.
- **Metamodel** = yet another abstraction, highlighting properties of the model itself

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



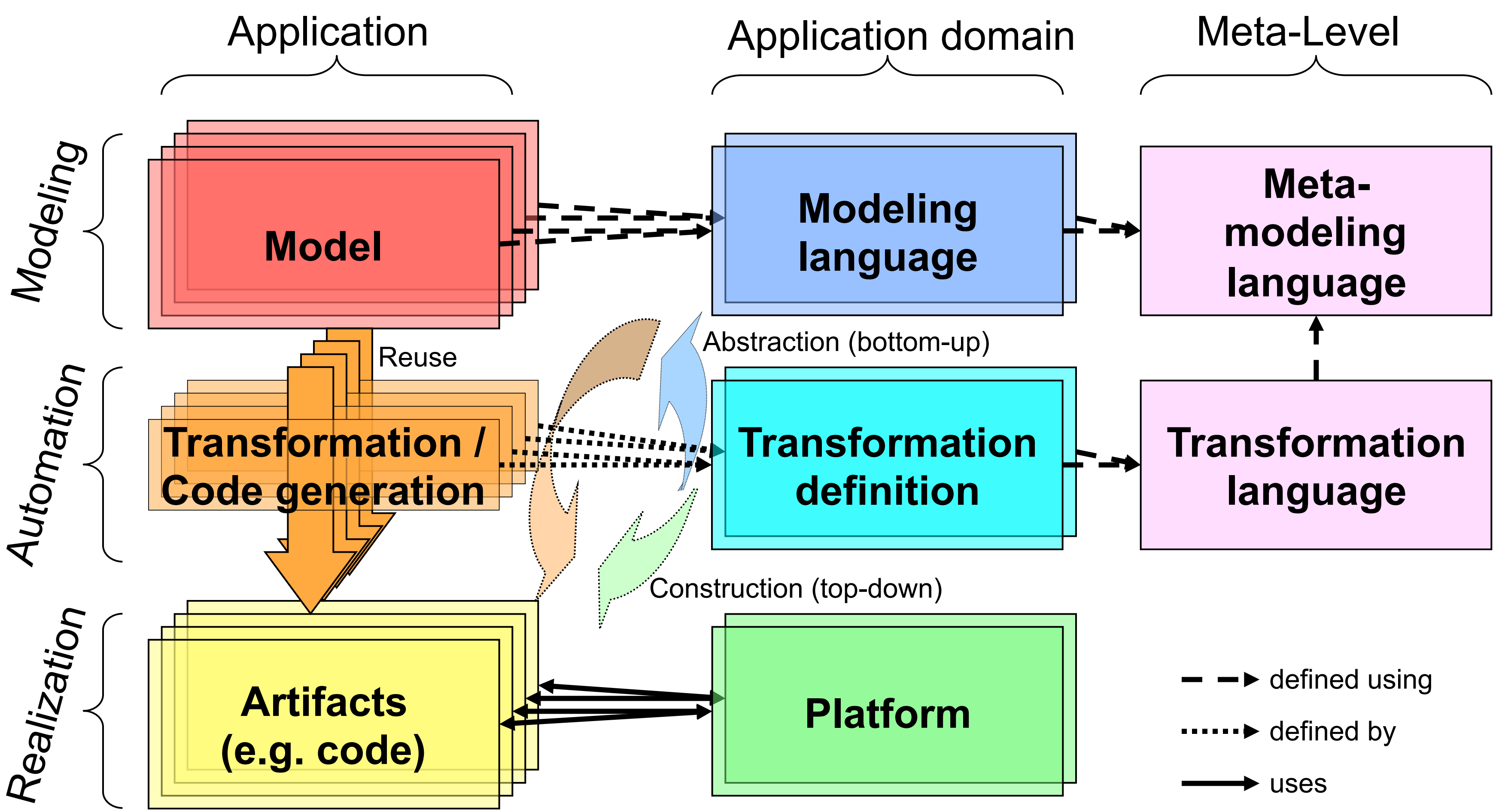
Model Transformations

- 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!!



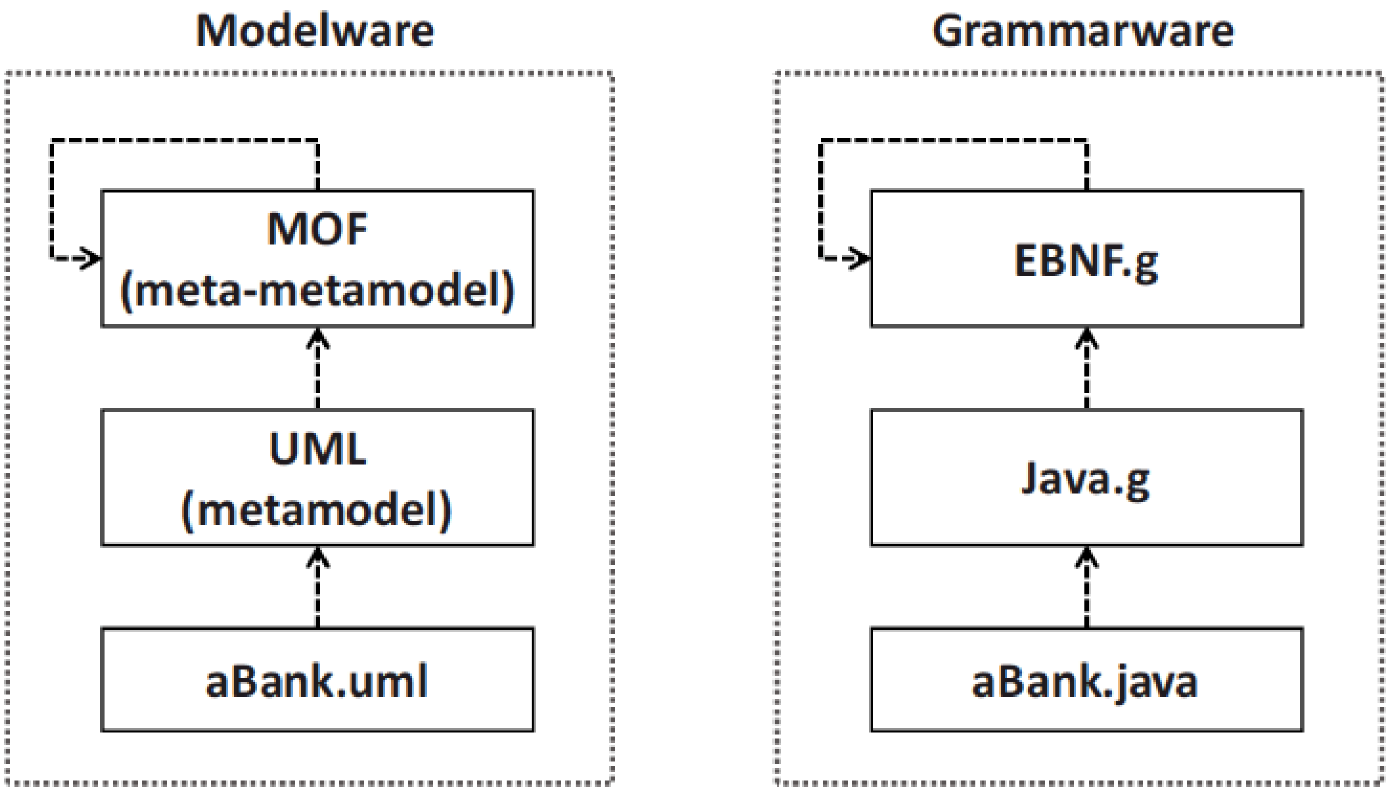
Concepts

Model Engineering basic architecture



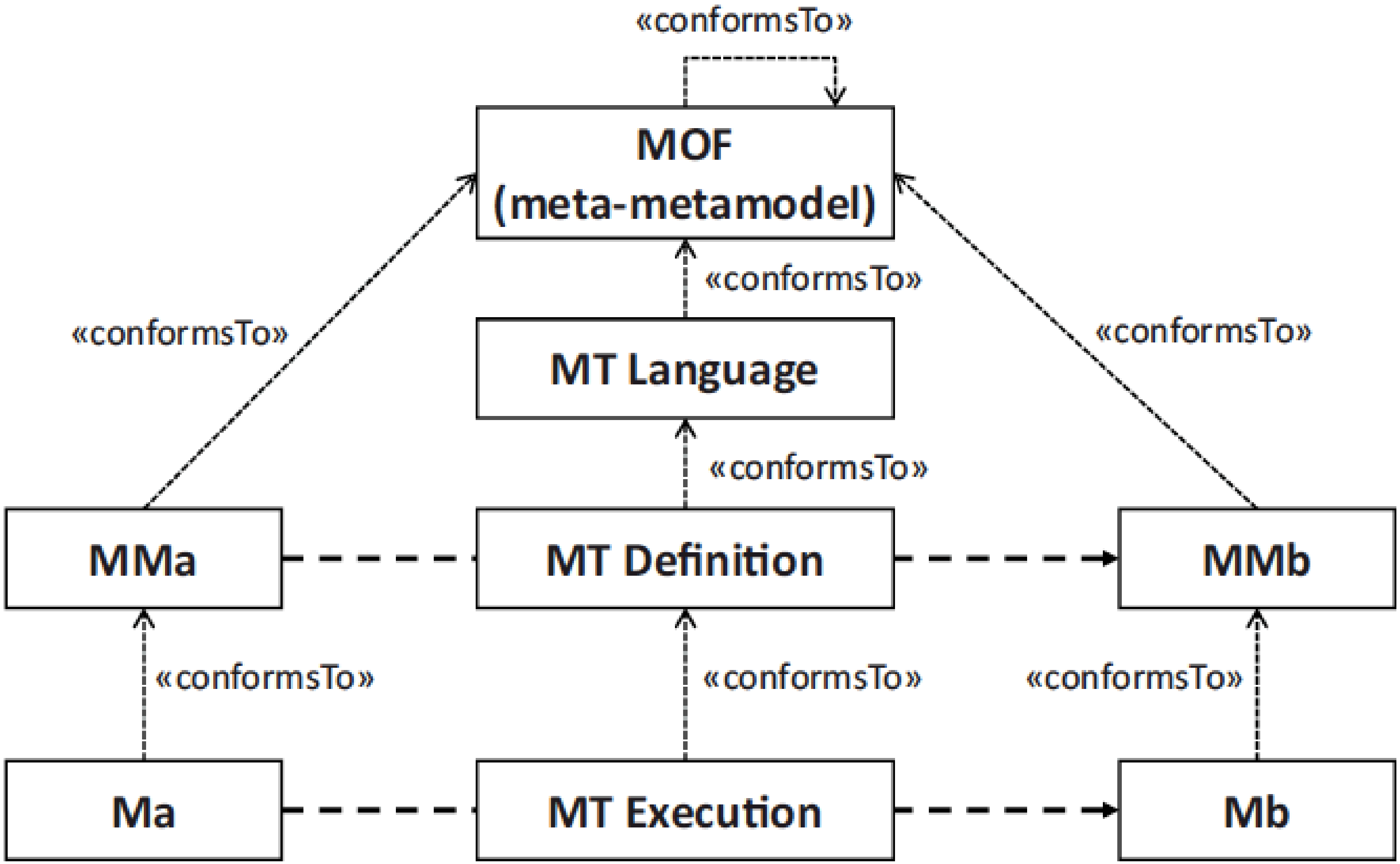
Modelware vs. Grammarware

- 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!

Concepts

Consequences or Preconditions

- Modified **development process**
  - Two levels of development – application and infrastructure
    - Infrastructure development involves modeling language, platform (e.g. framework) and transformation definition
    - Application development only involves modeling – efficient reuse of the infrastructure(s)
  - Strongly simplified application development
    - Automatic code generation replaces programmer
    - Working on the code level (implementation, testing, maintenance) becomes unnecessary
    - *Under which conditions is this realistic … or just futuristic?*
- New **development tools**
  - Tools for language definition, in particular meta modeling
  - Editor and engine for model transformations
  - Customizable tools like model editors, repositories, simulation, verification, and testing tools

Approaches
Overview

- Considered Approaches
  - Computer Aided Software Engineering (CASE)
  - Executable UML
  - Model Driven Architecture (MDA)
  - Architecture Centric Model Driven Software Development (AC-MDSD)
  - MetaCASE
  - Software Factories

- Distinguishing features
  - Special objectives and fields of application
  - Restrictions or extensions of the basic architecture
  - Concrete procedures
  - Specific technologies, languages, tools

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

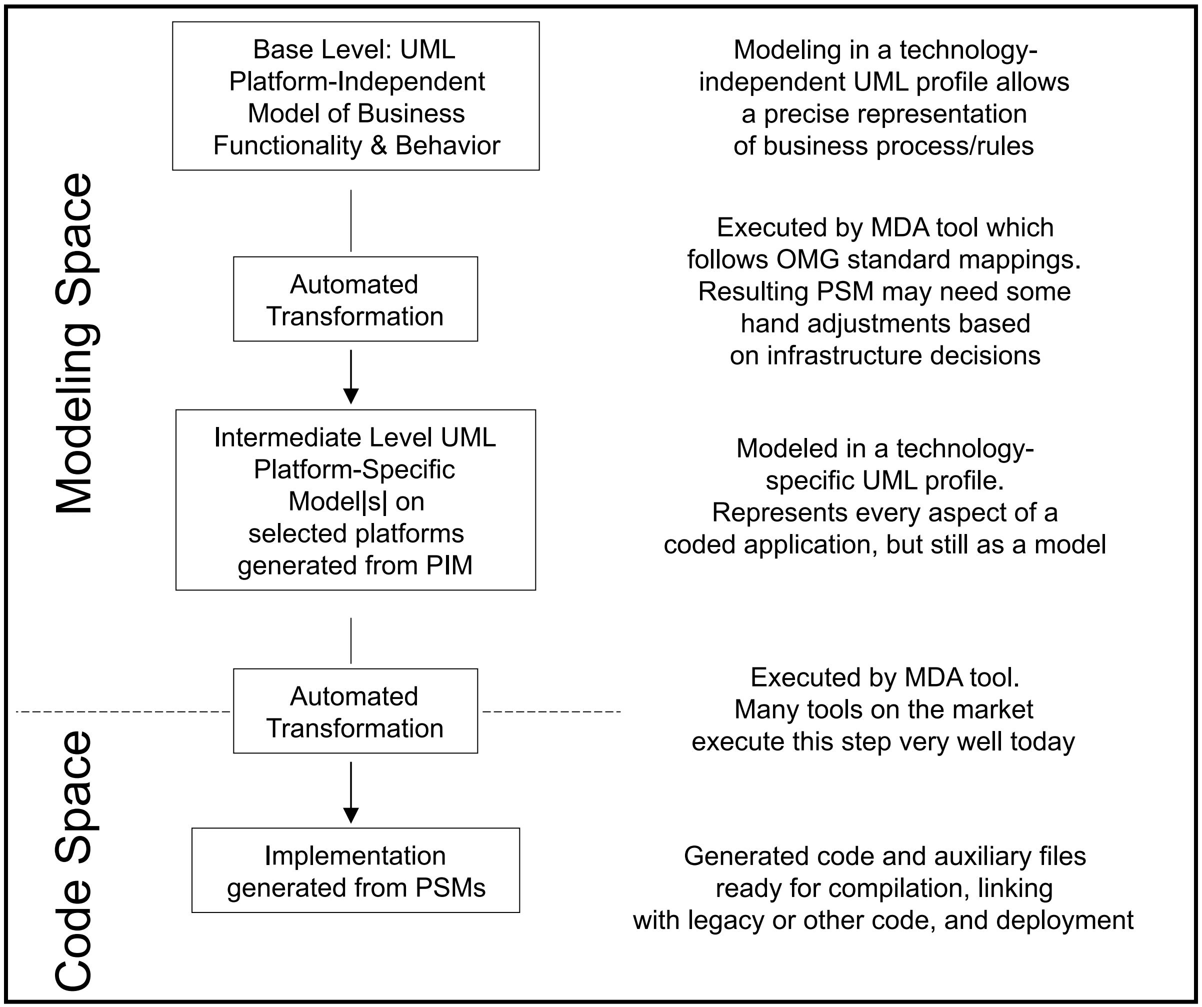
MDA

- **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/]

Approaches

MDA development cycle



Modeling Levels

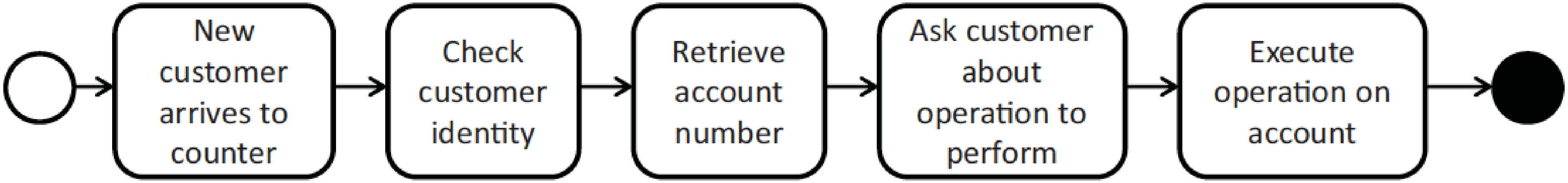
CIM, PIM, PSM

- 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

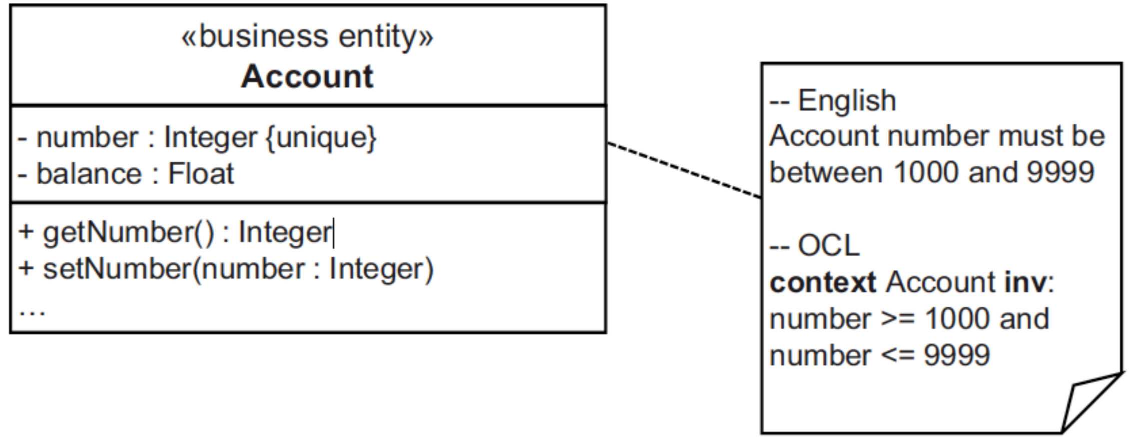
- Eg., business process



Modeling levels

MDA Platform Independent Model (PIM)

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

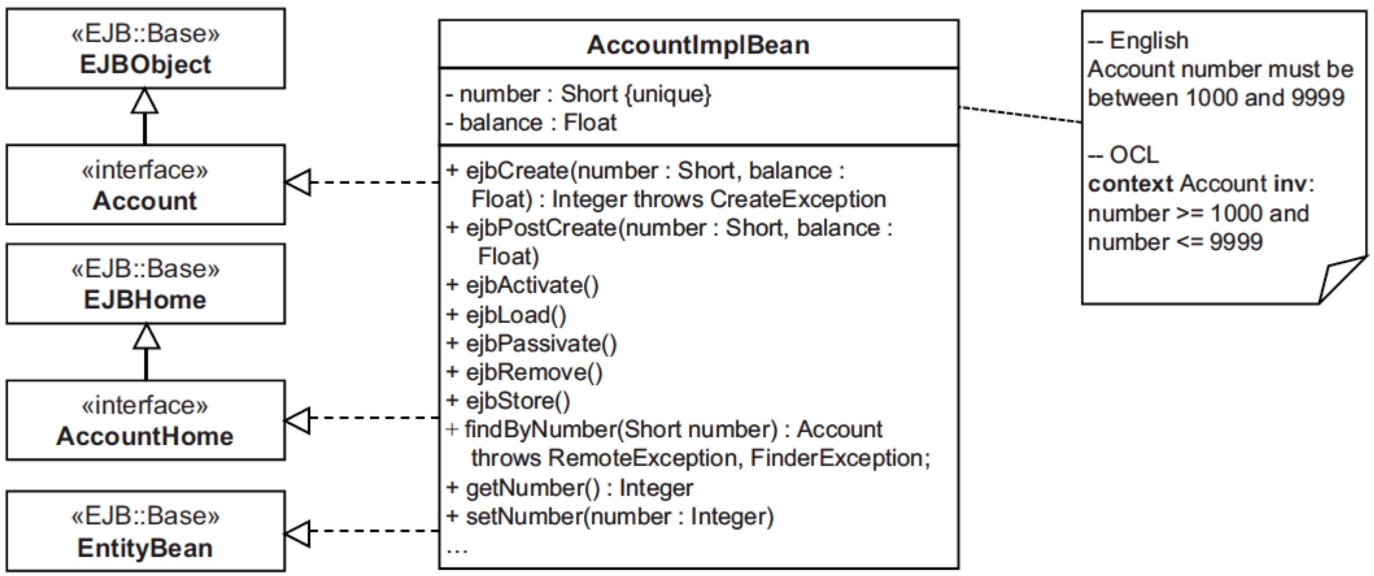


- Using the UML(optional)

- Abstraction of structur 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)



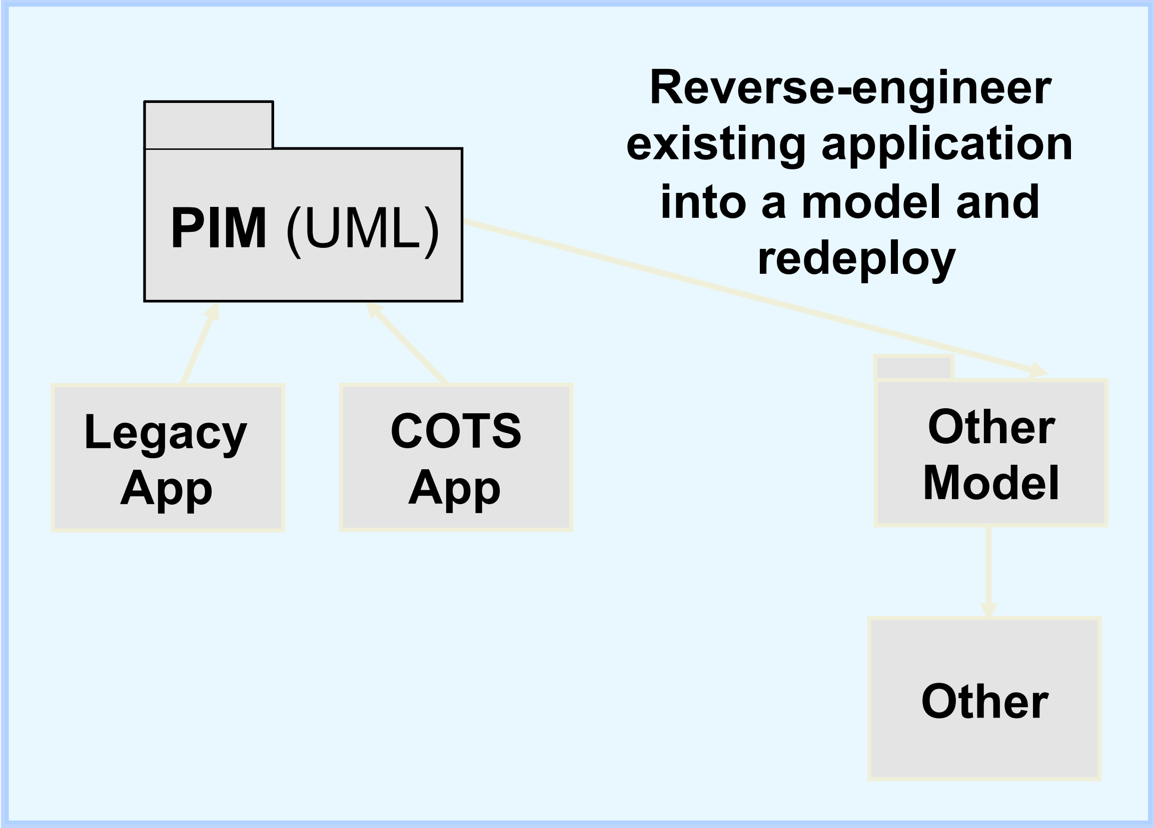
- Specifies how the functionality described in the PIM is realized on a certain platform
- Using a UML-Profile for the selected platform, e.g., EJB

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



Approaches

Excursus: 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

Approaches

MDA with UML

- 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*

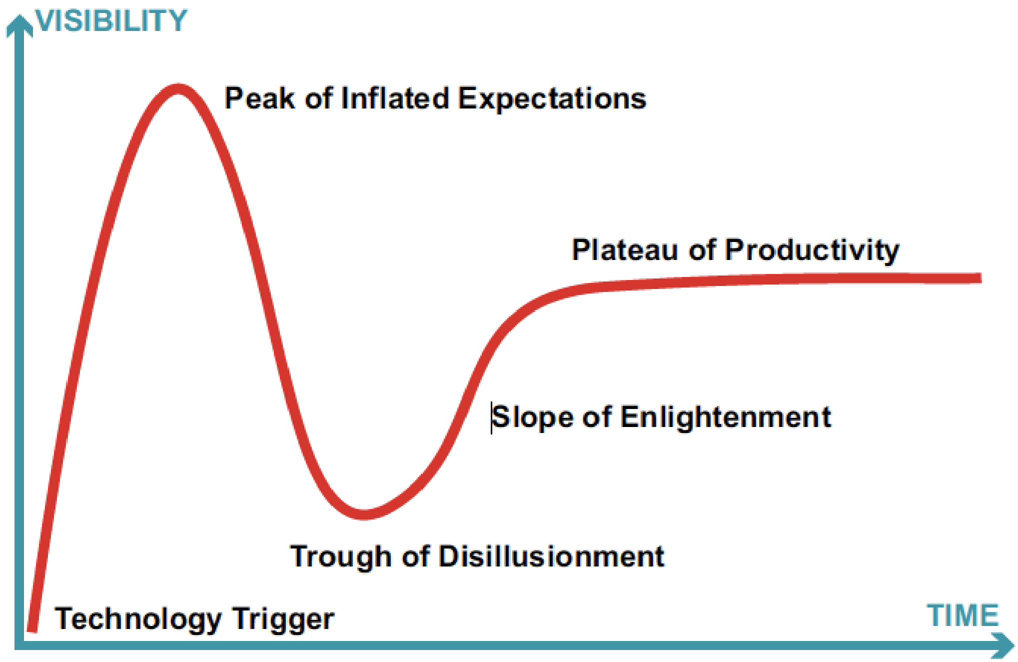
Approaches
MDA

- 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)

MDSE industry

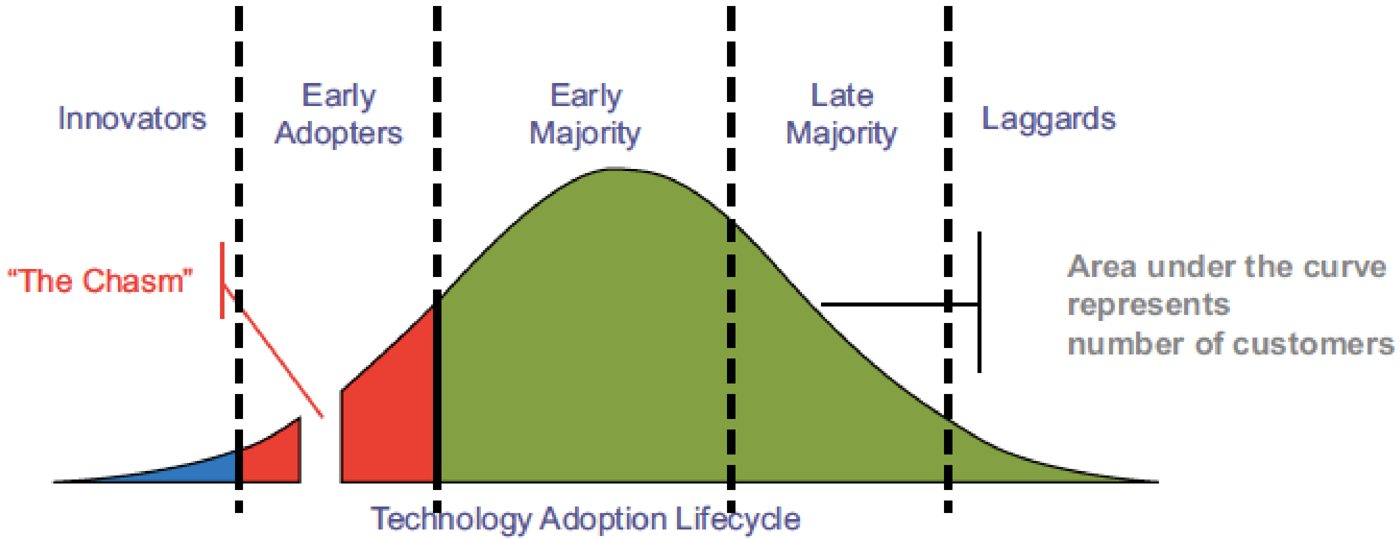
Adoption and acceptance (hype)

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



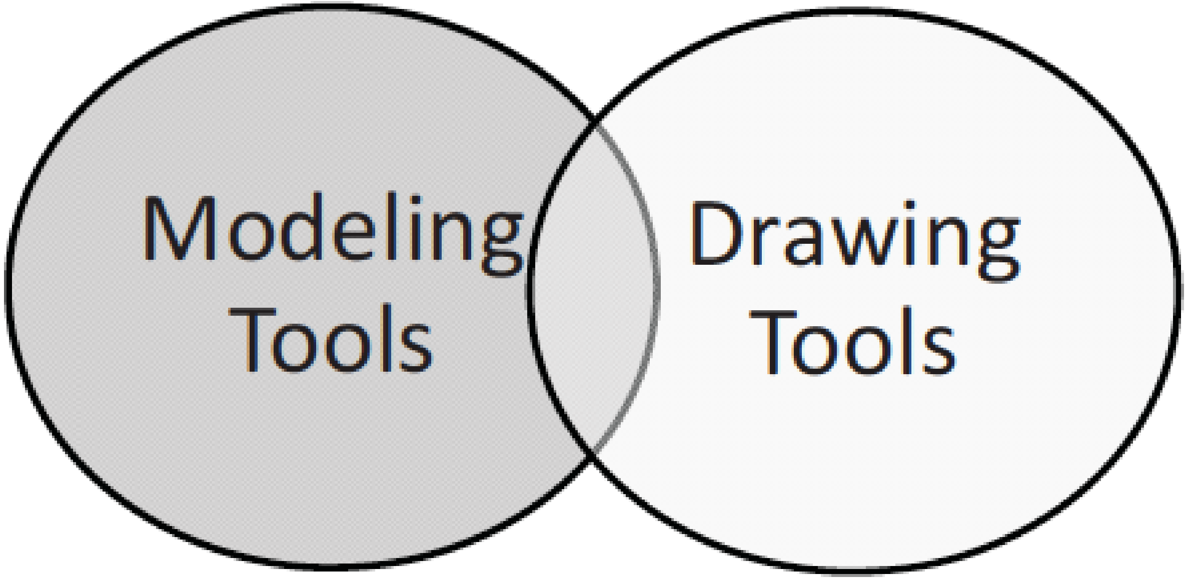
MDSE Industry (2)

Adoption



Tool support

- Drawing vs. modeling



Approaches

AC-MDSD

- Efficient reuse of architectures
  - Special attention to the efficient reuse of infrastructures/frameworks (= architectures) for a series of applications
  - Specific procedure model
    - Development of a reference application
    - Analysis in individual code, schematically recurring code and generic code (equal for all applications)
    - Extraction of the required modeling concepts and definition of the modeling language, transformations and platform
  - Software support (www.openarchitectureware.org)
- Basic architecture almost completely covered
  - When using UML profiles there is the problem of the method bodies
  - The recommended procedure is to rework these method bodies not in the model but in the generated code
- Advantages compared to MDA
  - Support for platform- and modeling language development
- Disadvantages compared to MDA
  - Platform independence and/or portability not considered

Approaches

MetaCASE/MetaEdit+

- Free configurable CASE
  - Meta modeling for the development of domain-specific modeling languages (**DSL**s)
  - **The focus** is on the ideal support of the **application area**, e.g. mobile-phone application, traffic light pre-emption, digital clock – Intentional Programming
  - Procedural method driven by the DSL development
- Support in particular for the **modeling level**
  - Strong Support for meta modeling, e.g. graphical editors
  - Platform development not assisted specifically, the usage of components and frameworks is recommended
- **Advantages**
  - Domain-specific languages
- **Disadvantages**
  - Tool support only focuses on graphical modeling

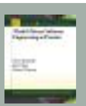
[www.metacase.com]

Approaches

Software Factories

- **Series production** of software products
  - Combines the ideas of different approaches (MDA, AC-MDSD, MetaCASE/DSLs) as well as popular SWD-technologies (patterns, components, frameworks)
  - Objective is the automatically processed development of software product series, i.e., a series of applications with the same application area and the same infrastructure
  - The SW-Factory as a marketable product
- Support of the **complete basic architecture**
  - Refinements in particular on the realization level, e.g. deployment
- **Advantages**
  - Comprehensive approach
- **Disadvantages**
  - Approach not clearly delimited (similar MDA)
  - Only little tool support

[J. Greenfield, K. Short: Software Factories. Wiley, 2004]



Eclipse and EMF

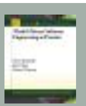
- Eclipse Modeling Framework
- Full support for metamodeling and language design
- Fully MD (vs. programming-based tools)
- Used in this course!

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