Modeling with UML, with semantics

Till Mossakowski
Otto-von-Guericke-Universität Magdeburg

Based on a course by Alexander Knapp, Universität Augsburg

Overview

- Model-driven software design (MSDS)
  - Model-driven architecture (MDA)
- Meta Modeling
- Unified Modeling Language 2 (UML), with semantics
  - Classes and packages – semantics: sets, predicates, functions / first-order logic
  - State machines – semantics: labeled transition systems
  - Component diagrams
  - Interactions – semantics: sets of traces
  - Profiles
- Object constraint language 2 (OCL)
- Meta object facility 2 (MOF)
- Eclipse modelling framework (EMF)
- Model transformations (QVT, …)
- Specific topics

Literature

- Grady Booch, Alan Brown, Sridhar Iyengar, James Rumbaugh, Bran Selic. "**An MDA Manifesto**". MDA Journal, May 2004. http://www.ibm.com/software/rational/mda/papers.html
- Grady Booch, James Rumbaugh, Ivar Jacobson: **The Unified Modeling Language User Guide**. Addison-Wesley, 2005
- Grady Booch, James Rumbaugh, Ivar Jacobson: **The Unified Modeling Reference Manual**. Addison-Wesley, 2005
- Marco Brambilla, Jordi Cabot, Manuel Wimmer. **Model-Driven Software Engineering in Practice**. Morgan & Claypool, 2012.
- Volker Gruhn, Daniel Pieper, Carsten Röttgers. ***MDA***. Springer, 2006. [in German]
- Siegfried Nolte. **QVT Operational Mappings**. Springer, 2010.
- Kevin Lano, editor. **UML 2 - Semantics and Applications**. Wiley, 2009.
- P.H.Schmitt. **UML and its Meaning** Winter 2002/2003
- Tim Weilkiens. **Systems Engineering with SysML/UML.** Elsevier 2008
- **http://www.uml-diagrams.org/**

Software

- Eclipse modelling framework: https://www.eclipse.org/modeling/emf/
  - Modelling framework and code generation facility for MOF / MOF-based models
- Modelio: http://modelio.org
  - Tool for editing UML diagrams (and generating code)
- Hugo/RT: http://www.pst.informatik.uni-muenchen.de/projekte/hugo/
  - Verification tool for UML, translation of UML state machines to autoamata
- Heterogeneous Tool Set (Hets): http://hets.eu
  - Verification tool for various logics; translation of UML class diagrams to first-order logic
- UMLhub: http://umlhub.net
  - Git-based repository and verification tool

What is a model?

"Modeling, in the broadest sense, is the cost-effective **use of something in place of something else for some cognitive purpose**. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A **model represents reality for the given purpose**; **the model is an abstraction of reality** in the sense that it cannot represent all aspects of reality. This **allows us to deal with the world in a simplified manner**, avoiding the complexity, danger and irreversibility of reality." [Jeff Rothenberg. "The Nature of Modeling". 1989]

"Ein Modell ist seinem Wesen nach eine in Maßstab, Detailliertheit und/oder Funktionalität **verkürzte** beziehungsweise **abstrahierte Darstellung** des originalen Systems." [H. Stachowiak. *Allgemeine Modelltheorie*. 1973]

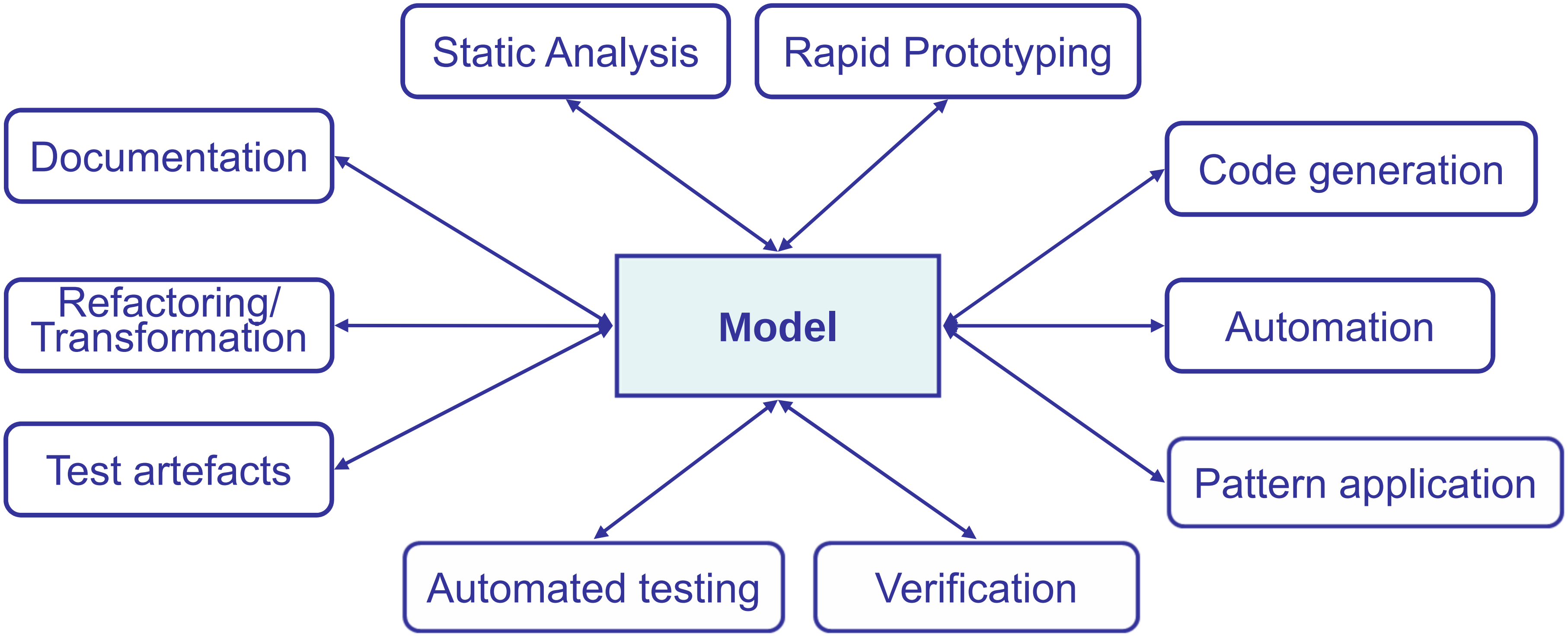
"Ein Modell ist eine vereinfachte, **auf ein bestimmtes Ziel hin ausgerichtete** Darstellung der Funktion eines Gegenstands oder des Ablaufs eines Sachverhalts, die eine Untersuchung oder eine Erforschung erleichtert oder erst möglich macht."
[H. Balzert. *Lehrbuch der Software-Technik*, Bd. 1. 2000]

Model engineering (1)

- Traditional rôle of models in software development
  - Used for communication purposes with the customer and within the development team (requirements specification, prototypes, etc.)
  - Used for software design
  - Specification for the programmer
  - Code visualization

- **Model engineering**
  - Models are the central artefacts in software development.
  - Models represent
    - different levels of abstraction (analysis, design, implementation);
    - different parts of the system (UI, database, business logic, system administration);
    - different concerns (security, performance, and resilience);
    - different tasks (development, testing, deployment).
  - Often, it is possible to partially generate one model from another.

Model engineering (2)



- Integration into **Model-Driven Software Development** (MDSD)

Key concepts of MDSD (1)

- **Abstraction**
  - Abstraction can be used to model applications at different levels of detail or from different perspectives.
    - Abstraction is the process of ignoring irrelevant details in order to focus on relevant ones.
    - Abstraction allows to focus on the different aspects of a system without getting lost in detail.

- **Precise modelling**
  - Models as part of the definition of a system, not just as sketches.
  - These models have well-defined semantics and can be transformed into implementation artefacts (in the same way that one compiles Java code into byte code).
  - Abstraction is not the same as imprecision
    - Using abstraction one omits specific details while being precise about those details on which one does focus.

Key concepts of MDSD (2)

- **Automation**
  - Automate the development process so that any artefact, which can be derived from information in a model, is generated (e.g., code, deployment descriptors, test cases, build scripts, other models, ...)
  - Automation can be achieved by using two main techniques:
    - Transformations automate the generation of artefacts from models.
    - Patterns automate the creation and the modification of model elements; they are typically applied interactively with a designer selecting a pattern and providing parameters (example: modelio).

- **Direct representation**
  - Modelling with languages that map their concepts to domain concepts rather than computer technology concepts
  - More direct coupling of solutions (solution domain) to problems (problem domain), leading to more accurate designs

Claimed benefits of MDSD (1)

- **Improved stakeholder communication**
  - Models omit implementation detail not relevant to understand the logical behavior of a system
  - Models are closer to the problem domain reducing the semantic gap between the concepts that are understood by stakeholders and the language in which the solution is expressed
  - Facilitates the delivery of solutions that are better aligned to business objectives
- **Improved design communication**
  - Models facilitate understanding and reasoning about systems at the design level.
  - Improved discussion making and communication about a system
- **Expertise capture**
  - Projects or organizations often depend on best practice decisions of key experts
  - Their expertise is captured in patterns and transformations
  - When sufficient documentation accompanies the transformations, the knowledge of an organization is maintained in the patterns and transformations

Claimed benefits of MDSD (2)

- **Models as long-term assets**
  - Models are important assets that capture what the IT systems of an organization do
  - High-level models are resilient to changes at the state-of-the-art platform level. They change only when business requirements change

- **Ability to delay technology decisions**
  - Early application development is focused on modeling activities
  - It is possible to delay the choice of a specific technology platform or product version until a later point when further information is available.
  - This is crucial in domains with extremely long development cycles, such as air traffic control systems

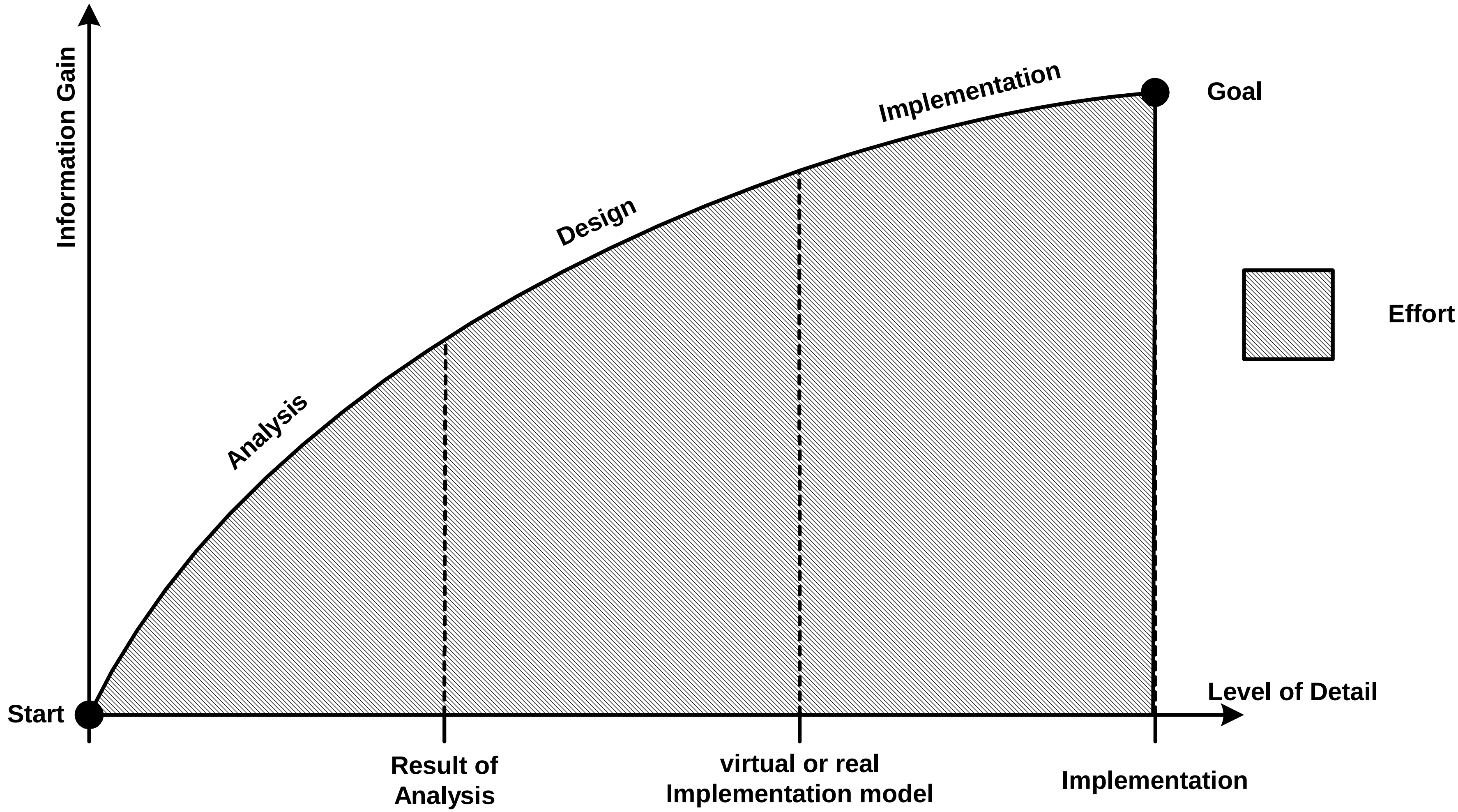
Claimed benefits of MDSD (3)

- **Increased productivity**
  - Generation of code and artefacts from models
  - Careful planning needs to ensure that there is an overall cost reduction.
- **Maintainability**
  - MDSD helps to develop maintainable architectures where changes are made rapidly and consistently, enabling more efficient migration of components onto new technologies.
  - Keeping the high-level models free of implementation detail makes it easier to handle changes in the underlying platform technology and its technical architecture.
  - A change in the technical architecture of the implementation is made by updating a transformation.
- **Reuse of legacy**
  - One can consistently model existing legacy platforms.
  - Reverse transformations from the components
  - Migrating the components to a new platform or generating wrappers to enable the legacy component to be accessed via integration technologies such as Web services.

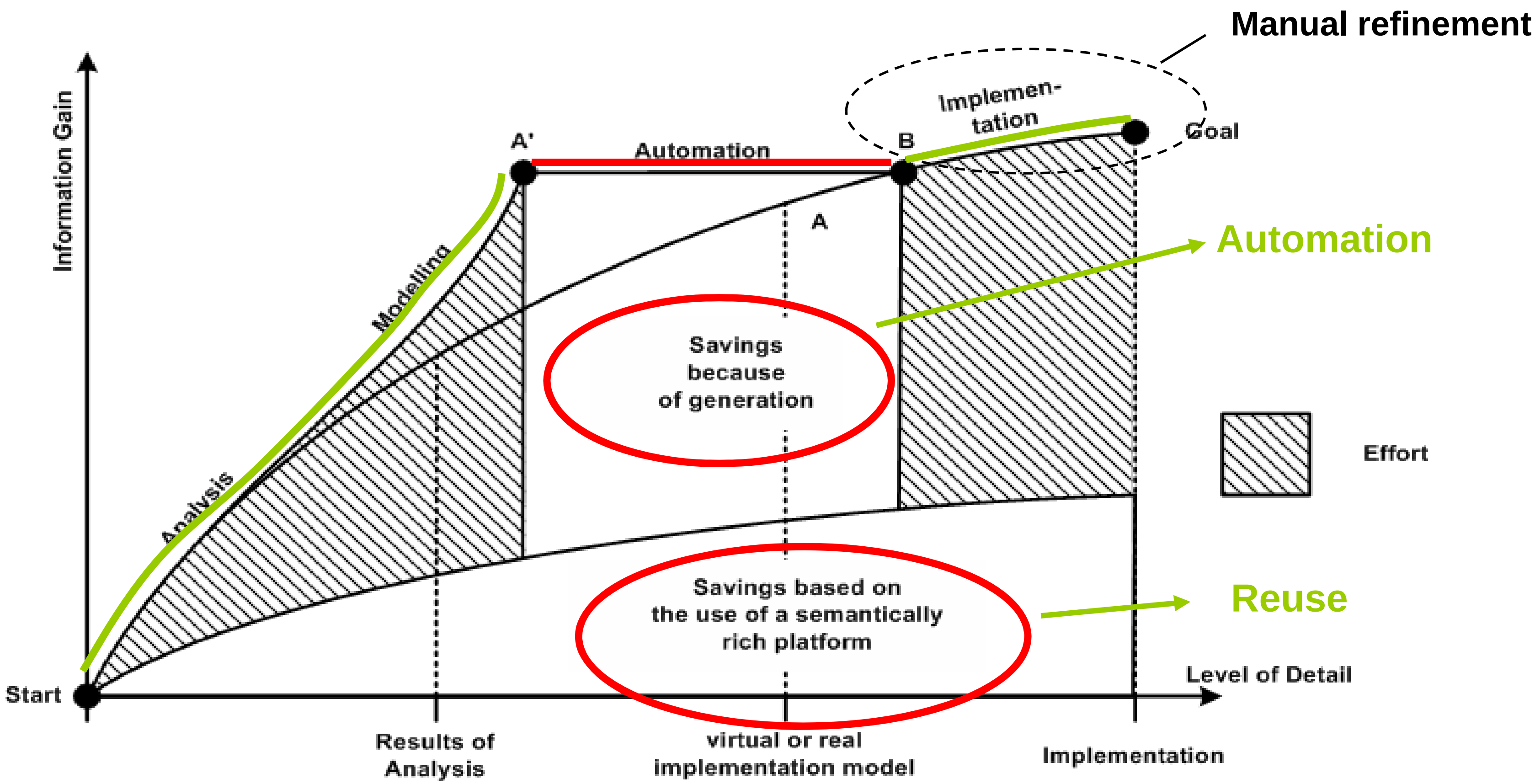
Claimed benefits of MDSD (4)

- **Adaptability**
  - Adding or modifying a business function is simplified since the investment in automation was already made.
- **Consistency**
  - Manually applying coding practices and architectural decisions is an error prone activity.
- **Repeatability**
  - ROI from developing the transformations increases each time they are reused.
  - The use of tried and tested transformations
    - increases the predictability of developing new functions;
    - reduces the risk since the architectural and technical issues were already resolved.

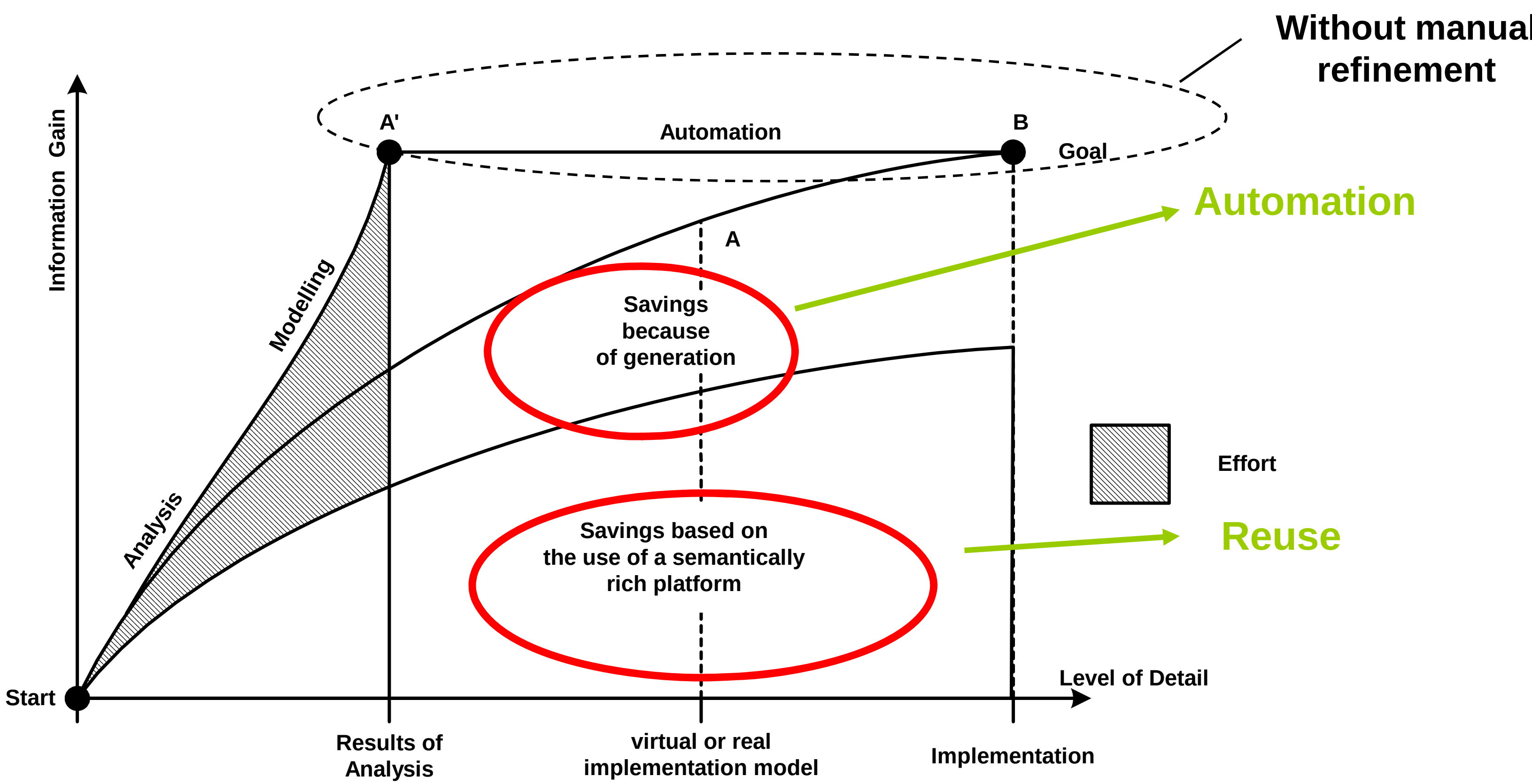
“Normal” software development



MDSD effort (stage 1)

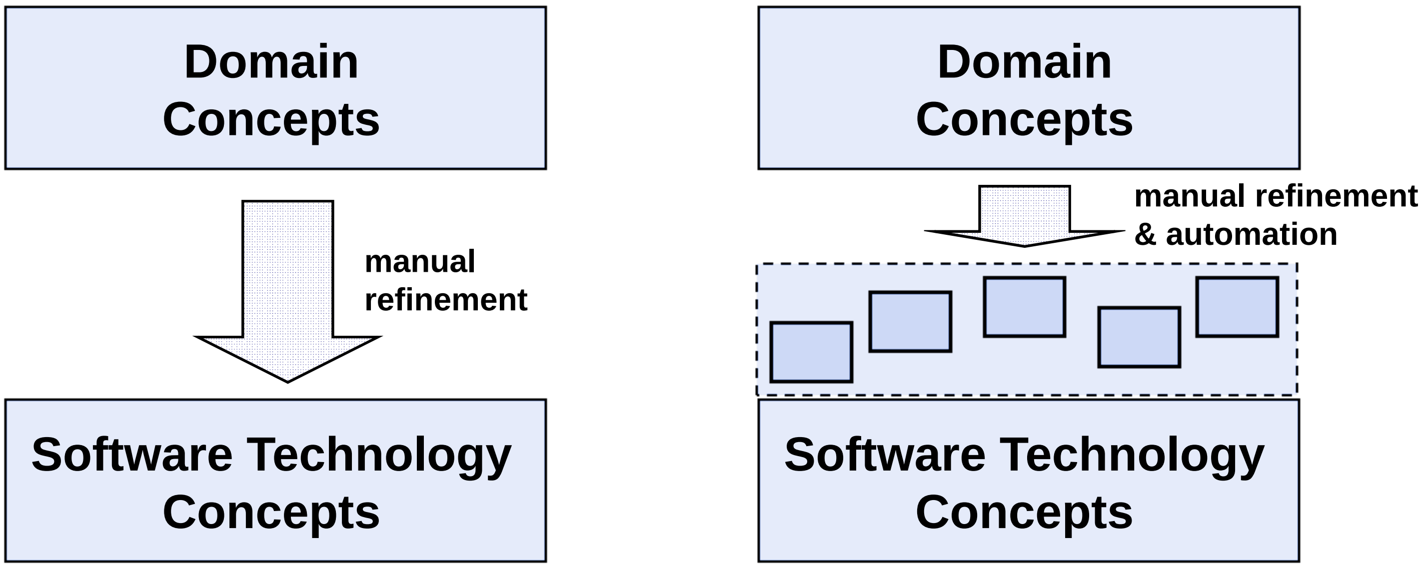


MDSD effort (stage 2)



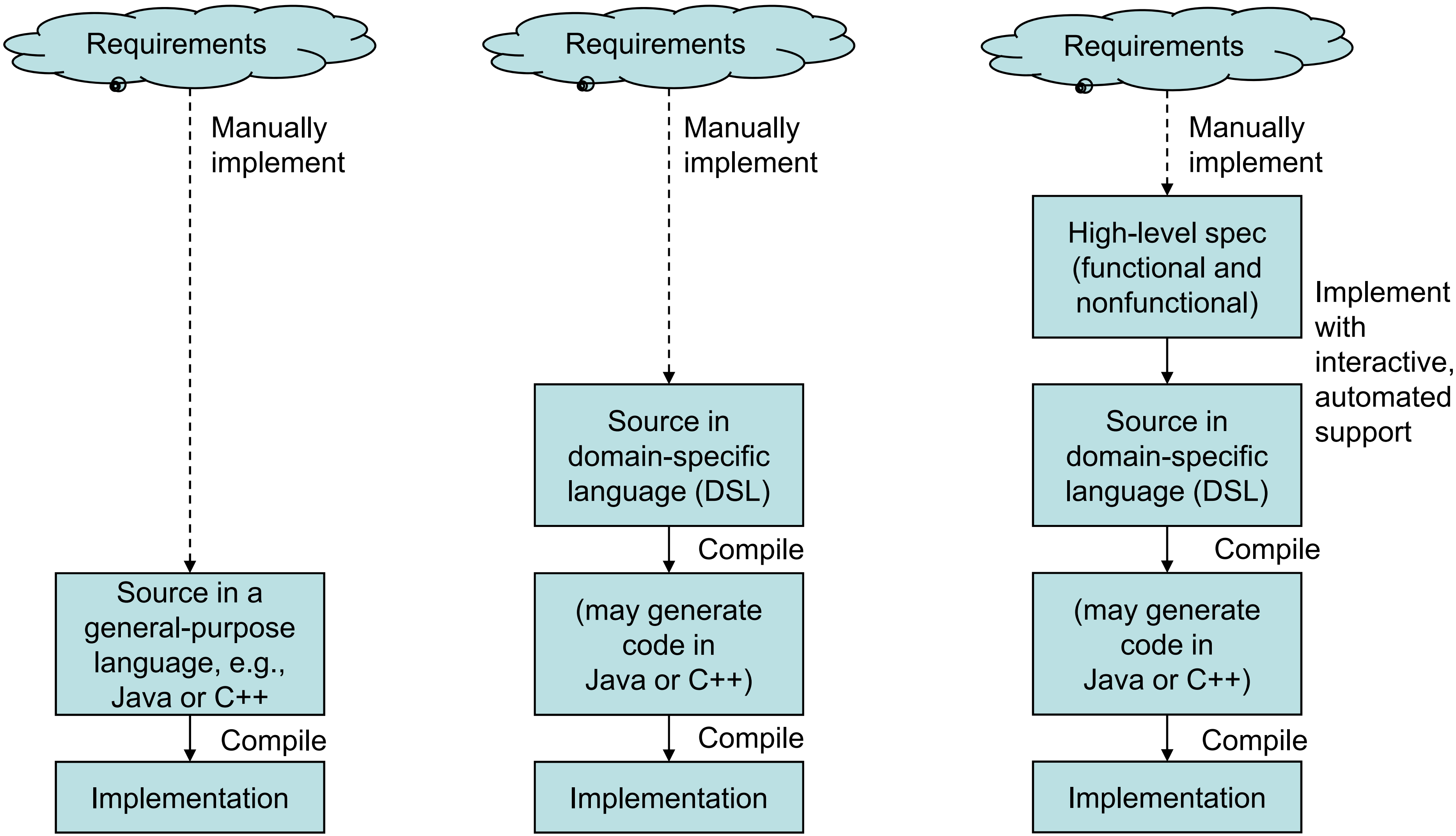
Model-driven software development

- Makes software development more **domain related** opposed to **computing related**

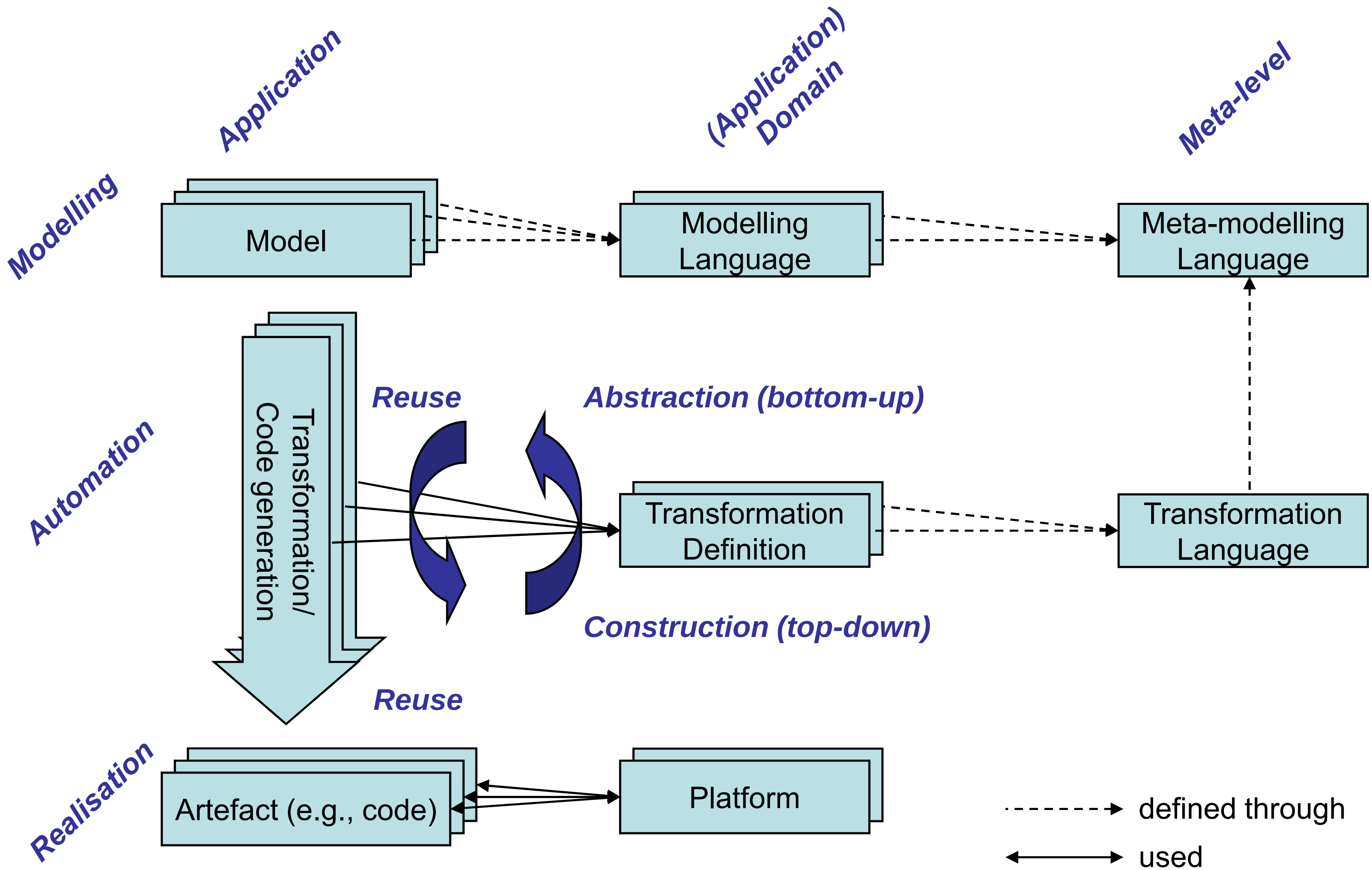


- Narrows the semantic gap between business models and IT
- Re-use of components (assets)
- Generation techniques reduce time-to-market
- Makes software development more efficient

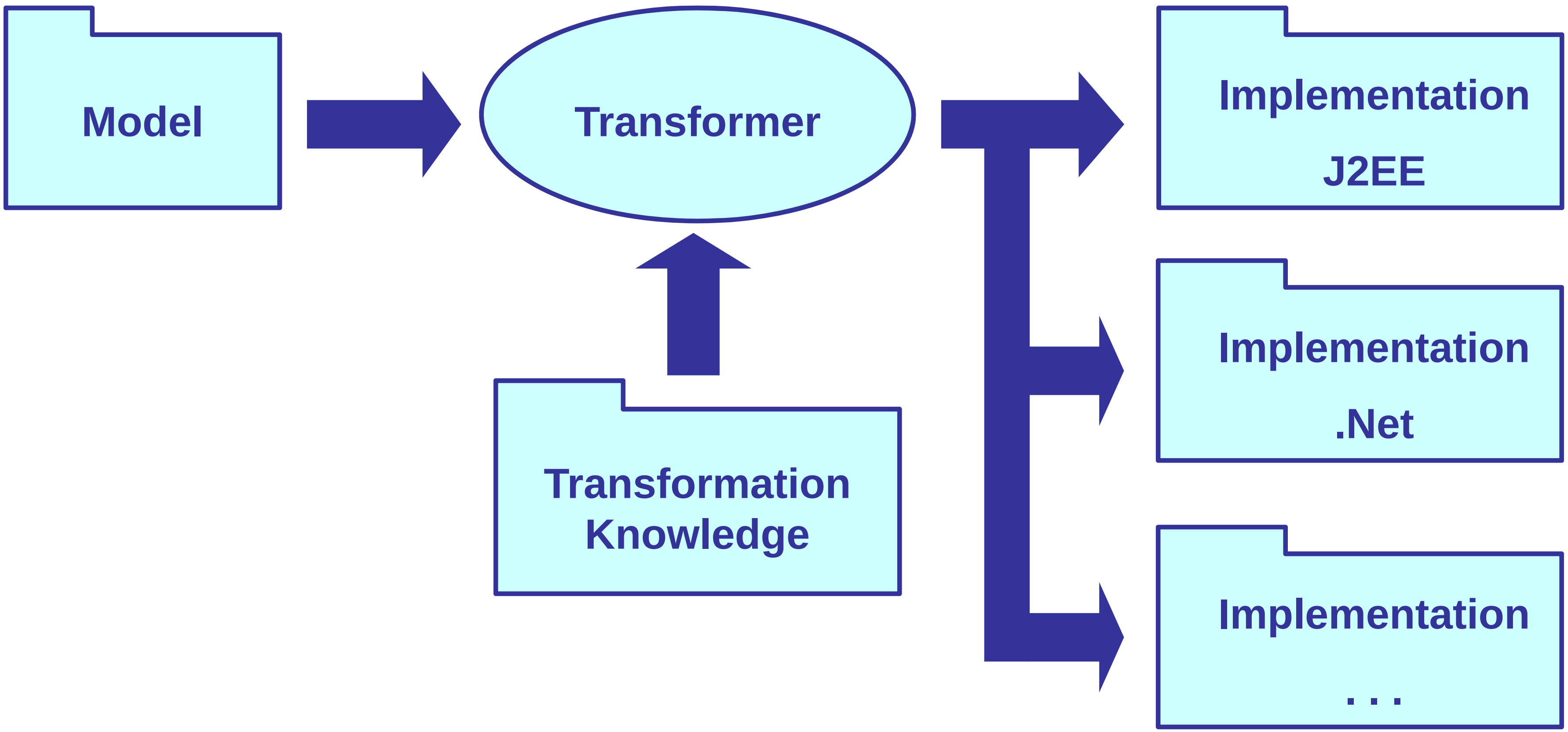
Automation in software development



MDSD: Basic architecture

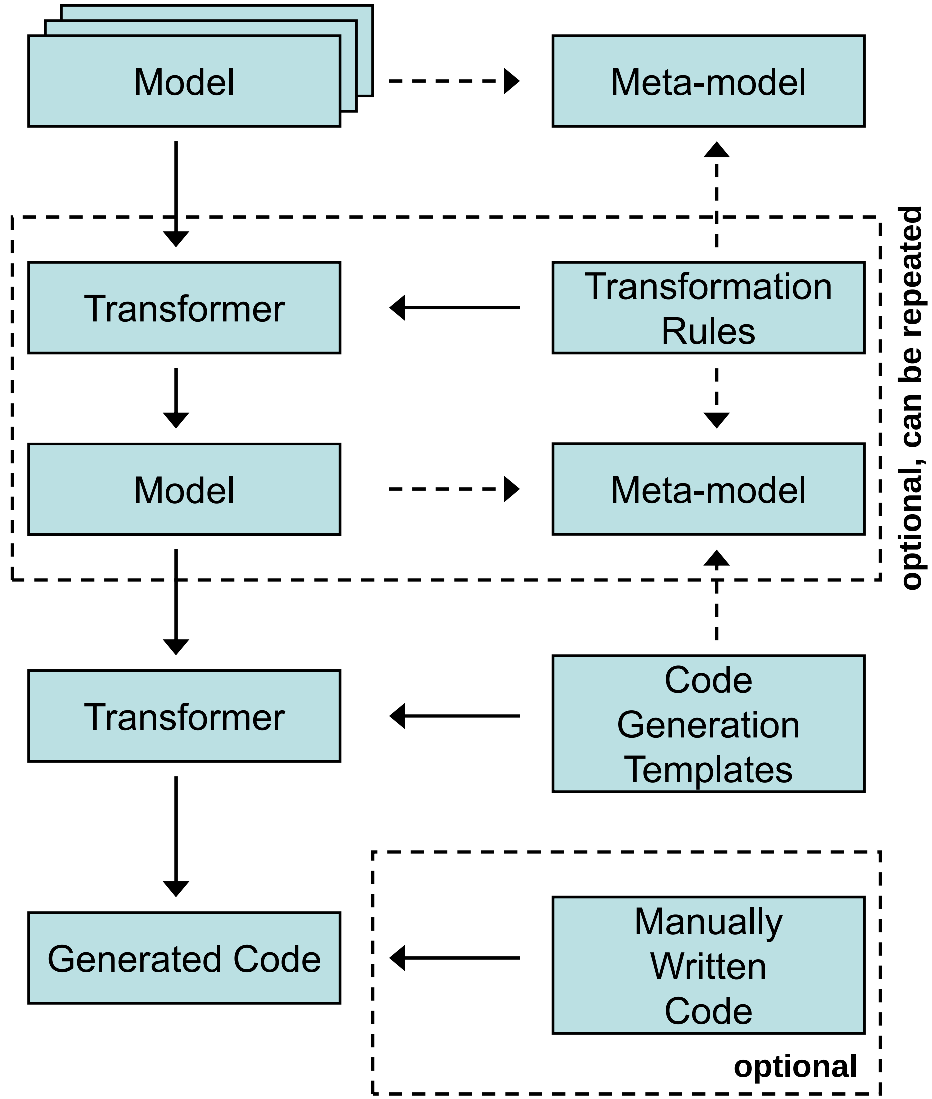


MDSD: A bird's view

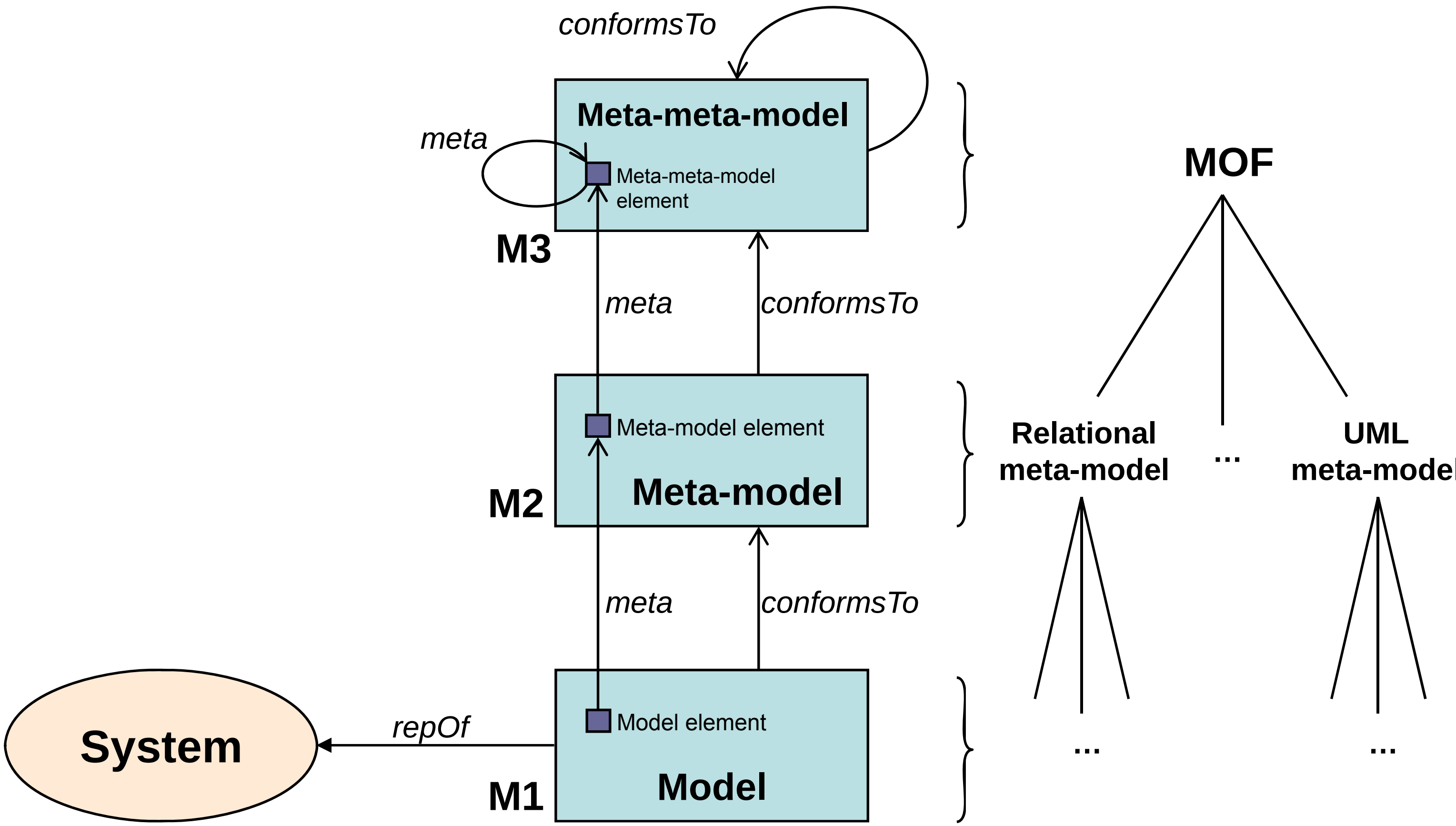


How is MDSD realised?

- Developer develops **model(s)**, expressed using a DSL, based on certain meta-model(s).
- Using **code generation templates**, the model is transformed into executable code.
  - Alternative: Interpretation
- Optionally, the **generated code is merged** with manually written code.
- One or more **model-to-model transformation steps** may precede code generation.



(Meta-)Model hierarchy



(Meta-)Model hierarchy: Example