Model-Driven Development

Introduction

SS 2015

Egon Teiniker

Model-Driven Development

Outline

- Model-Driven Development
  - Introduction
  - Goals
  - Approach
  - Challenges

- Model Driven Architecture
  - Overview
  - Terminology

Model-Driven Development

Introduction

A significant factor behind the difficulty of developing complex software is the wide conceptual gap between the problem and the implementation domains – **problem-implementation gap**.

Bridging the gap using approaches that require **extensive handcrafting of implementation** gives rise to accidental complexities that make the development of software difficult and costly.

The growing complexity of software is the motivation behind work on **industrializing software development**.

Model-Driven Development

Introduction

Model-Driven Development (MDD) is the natural continuation of programming as we know it today.

The application of models to software development is a long-standing tradition, and has become even more popular since the development of the **Unified Modeling Language (UML)**.

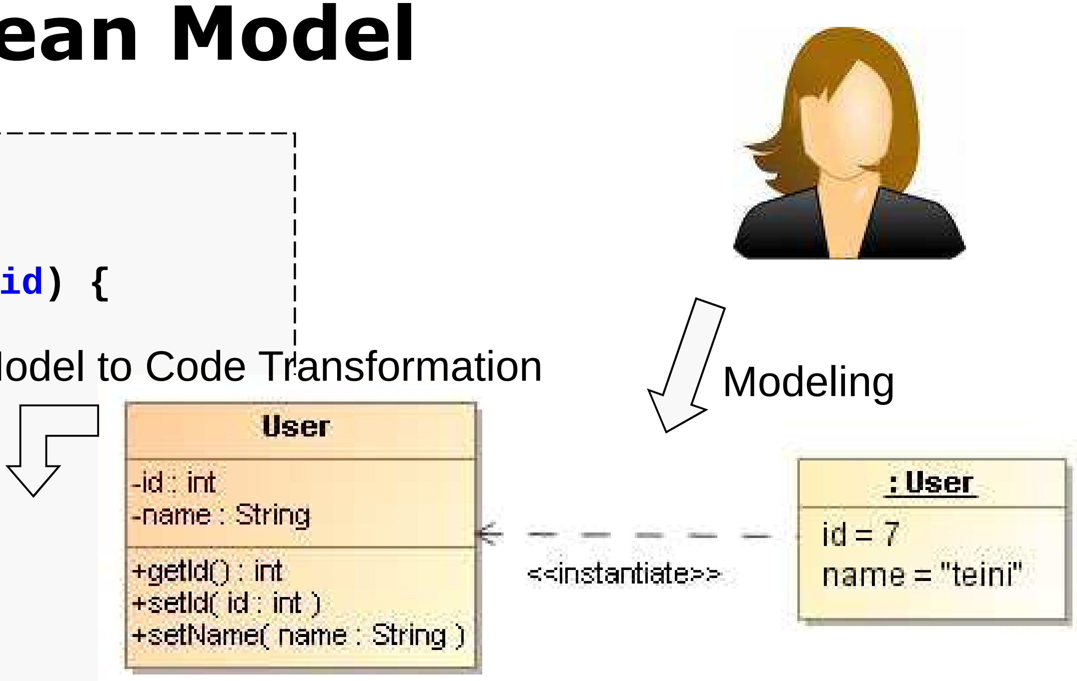
We use UML mainly as **documentation**, where the relationship between model and software implementation is only intentional but not formal. We call this flavor of model usage **model-based approach**.

Model-Driven Development

Example: JavaBean Model

```
public class User {
       private int id;
       public void setId(int id) {
           this.id = id;
       }
       public int getId(){
           return this.id;
       }

       private String name;
       public void setName(String name) {
           this.name = name;
       }
   }
```



Model-Driven Development

Introduction

MDD has an entirely different approach:

> Models do not constitute documentation, but are considered equal to code, as their implementation is automated.

This process creates a great potential for **automation of software production**, which in turn leads to **increased productivity**.

Models can also be understood by **domain experts**. **Graphical models** are often used, but **textual models** are an equally feasible option.

Model-Driven Development

Introduction

The process of analyzing a problem, conceiving a solution, and expressing a solution in a high-level programming language can be viewed as an implicit form of modeling.

Software developing is essentially a model-based problem solving activity!

Writing source code is a modeling activity because the developer is modeling a solution using the **abstractions provided by a programming language**.

Model-Driven Development

MDD Goals

- **Increased development speed.**
  Runnable code can be generated from formal models using one or more transformation steps (**automation**).

- **Enhanced software quality.**
  The use of **automated transformations** and **formally-defined modeling languages** lets you enhance software quality. A software architecture will recur **uniformly** in an implementation.

- **Better maintainability.**
  Implementation aspects can be **changed in one place**, for example in the **transformation rules**. The same is true for fixing bugs in generated code.

Model-Driven Development

MDD Goals

- **Manageability of complexity through abstraction.**
  The modeling languages enable programming or configuration on a more abstract level. For this purpose, the models must ideally be described using a **problem oriented modeling language**.

- **Software product lines.**
  Architectures, modeling languages and transformations can be used to establish software product lines which lead to a higher level of **reusability**.

Model-Driven Development

MDD Approach

We can refactor the code of an existing applications so that three parts can be separated:

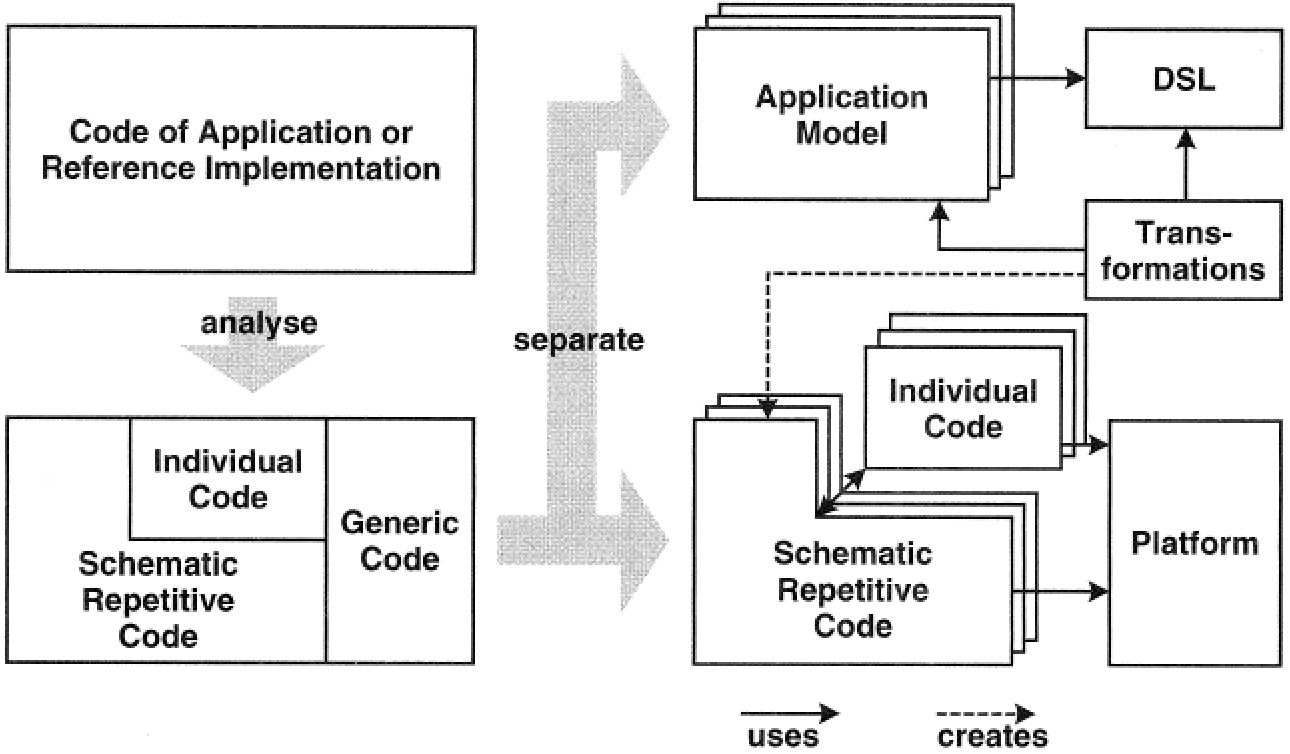
- **Generic code**
  The generic part is identical for all future applications.

- **Schematic repetitive code**
  The schematic part is not identical for all applications, but possesses the same systematics.

- **Individual code**
  The application specific part cannot be generalized.

Model-Driven Development

MDD Approach



Model-Driven Development

MDD Challenges

The major challenges when realizing the MDD vision can be grouped into the following categories:

- **Modeling language challenges** arise from concerns associated with providing support for creating and using problem-level abstractions in modeling languages, and for rigorously analyzing models.
- **Separation of concerns challenges** arise from problems associated with modeling systems using multiple, overlapping viewpoints that utilize possibly heterogeneous languages.

MDD Challenges

- **Model manipulation and management challenges** arise from problems associated with
  - defining, analyzing, and using model transformations
  - maintaining traceability links among model elements to support model evolution and roundtrip engineering
  - maintain consistency among viewpoints
  - tracking versions

Model-Driven Architecture

Overview

The **Object Management Group (OMG)** launched the **Model-Driven Architecture (MDA)** as a framework of standards in 2001.

MDA advocates modeling systems from three viewpoints:

- The **computation independent viewpoint** focuses on the environment in which the system of interest will operate in and the required features of the systems. Modeling a system from this viewpoint results in a **computation independent model (CIM)**.

Model-Driven Architecture

Overview

- The **platform independent viewpoint** focuses on the aspects of system features that are not likely to change from one platform to another.
  A **platform independent model (PIM)** is used to present this viewpoint.
- The **platform specific viewpoint** provides a view of a system in which platform specific details are integrated with the elements in a PIM.
  This view of a system is described by a **platform specific model (PSM)**.

Model-Driven Architecture

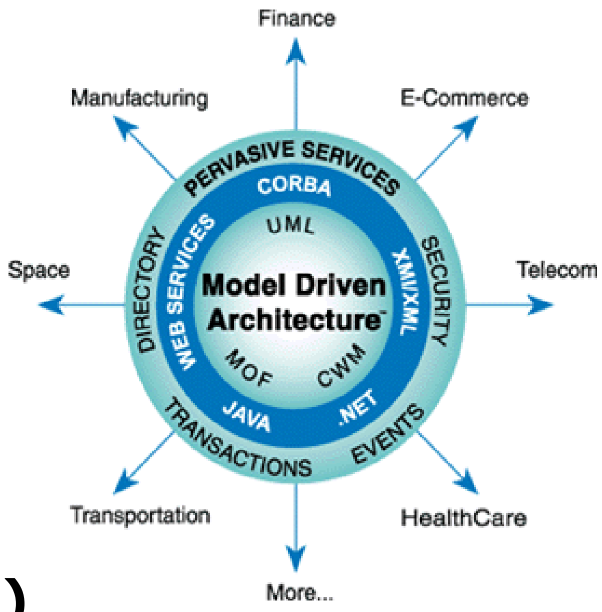
Overview

The pillars of MDA are:

- **Meta Object Facility (MOF)**
  A language for defining the abstract syntax of modeling languages.

- **Unified Modeling Language (UML)**

- **Query, View, Transformation (QVT)**
  A standard for specifying and implementing model transformations.



Model-Driven Architecture

Terminology

- **Domain**
  A domain is a bounded field of interest or knowledge.

- **Platform**
  The OMG defines a platform as a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns.
  Examples of platforms are:
  - JavaEE
  - Microsoft’s .NET
  - CORBA Component Model

Model-Driven Architecture

Terminology

- **Model**
  A model is an abstract representation of a system’s structure, function or behavior.

- **Metamodel**
  It is absolutely mandatory to be clear about the structure of a domain, so that one can formalize this structure or its relevant parts. The metamodel compasses the **abstract syntax** and the **static semantics** of a language, and is an instance of the meta meta model.

Model-Driven Architecture

Terminology

- **Meta Meta Model**
  The metamodel must itself have a metamodel that defines the concepts available for metamodeling. This is the role of the meta meta model.

- **Abstract and Concrete Syntax**
  While the **concrete syntax** of a language specifies what a parser for the language accepts, the **abstract syntax** merely specifies what the language's structure looks like.
  It's interesting that various concrete syntax forms can have a common abstract syntax.

Model-Driven Architecture

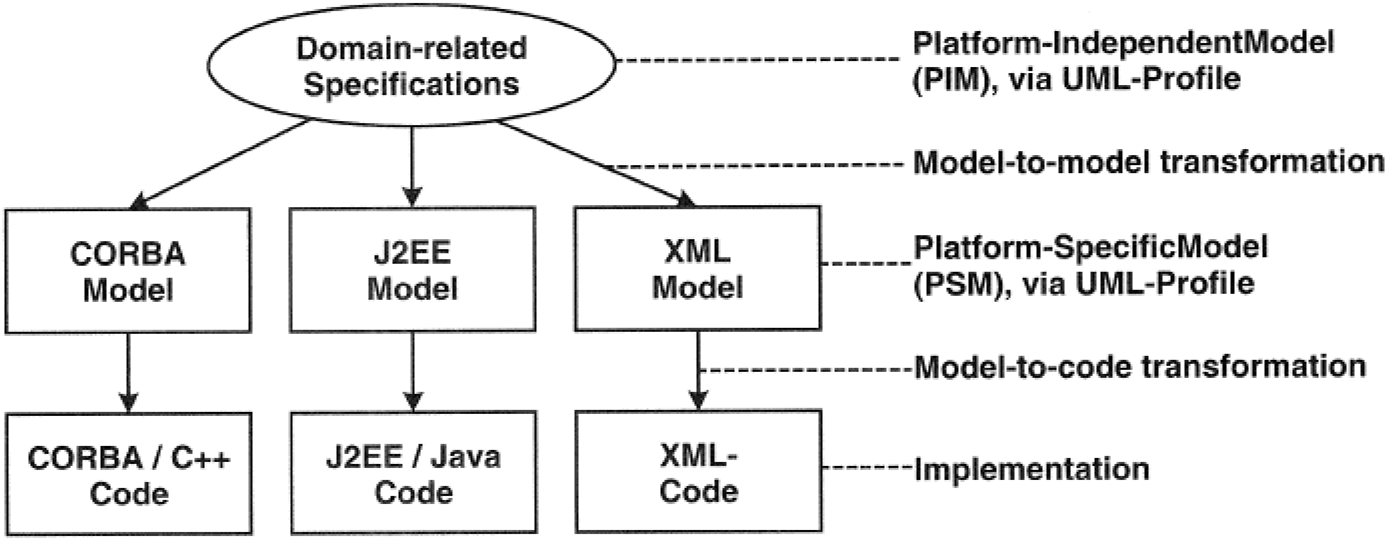
Terminology

- **Platform-Independent Model (PIM)**
  Domain-related specifications are defined in PIMs.
  A formal modeling language is used that is specific to the **concepts of the domain** to be modeled.
  These domain-specific descriptions are completely independent of the later implementation on the target platform.

- **Platform-Specific Model (PSM)**
  Via model transformation PSMs are created from the PIMs.
  These PSMs contain the **target platform's specific concepts**.
  The implementation for a concrete target platform is then generated with another transformation based on one or more PSMs.

Model-Driven Architecture

Terminology


Domain-related Specifications
Platform-IndependentModel (PIM), via UML-Profile
Model-to-model transformation
CORBA Model
J2EE Model
XML Model
Platform-SpecificModel (PSM), via UML-Profile
Model-to-code transformation
CORBA / C++ Code
J2EE / Java Code
XML-Code
Implementation

Model-Driven Architecture

Terminology

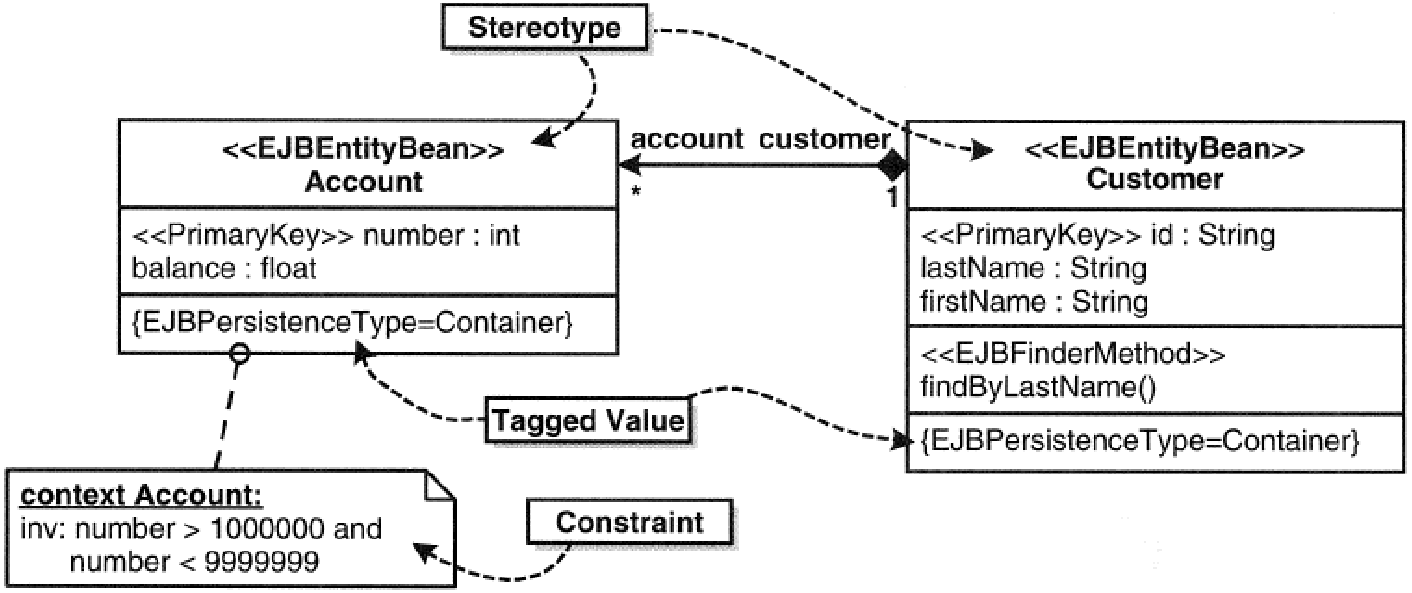
- **Transformations**
  A Transformation maps models to the respective next level. Transformation rules should be defined between two metamodels.
  - **Model-to-Model transformation.**
  - **Model-to-Code transformation.**

- **UML Profiles**
  UML profiles are the standard mechanism for expanding the vocabulary of UML. They contain language concepts that are defined via basic UML constructs such as classes and associations, **stereotypes**, **tagged values**, and **constraints**.

Model-Driven Architecture

Terminology

Example: UML Profile



FAQs

Introduction

- Describe the **goals of Model-Driven Development**.
- Describe the **three parts of code** in which an existing application can be refactored (including sketch).
- Describe the Model-Driven Development **Challenges.**
- Explain the three **viewpoints of a software system** which are specified by the Model-Driven Architecture.
- Explain the concept and usage of **UML Profiles**.

References

- *Thomas Stahl, Markus Völter*
  Model-Driven Software Development
  Wiley 2006

- *Bran Selic*
  The Pragmatics of Model-Driven Development
  IEEE Software, 2003

- *Robert France, Bernhard Rumpe*
  Model-driven Development of Complex Software:
  A Research Roadmap
  Future of Software Engineering, FOSE 2007