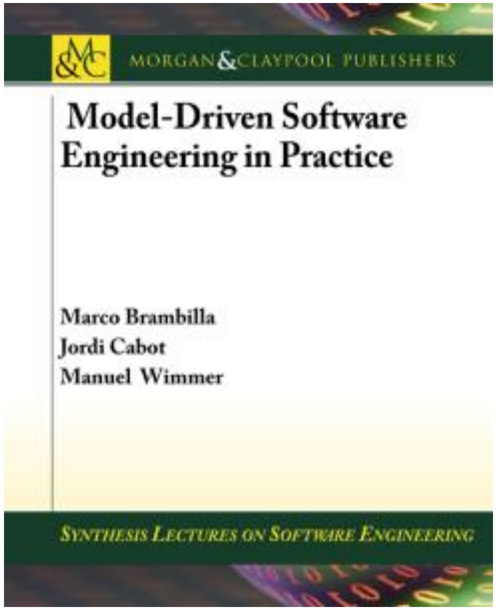
MORGAN & CLAYPOOL PUBLISHERS

CHAPTER 9

MODEL-TO-TEXT TRANSFORMATIONS

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

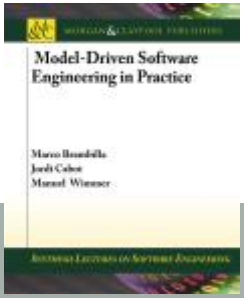
Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Content

- Introduction
- Programming Languages based Code Generation
- M2T Transformation based Code Generation
- Mastering Code Generation

INTRODUCTION



Morgan & Claypool Publishers

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

Synthesis Lectures on Software Engineering

Introduction

Terminology

- Code generation
  - *Wikipedia*:
    „**Code generation** is the process by which a compiler‘s code generator converts a syntactically-correct program into a series of **instructions** that can be **executed by a machine**.“
  - *Code Generation in Action* (Herrington 2003):
    „**Code generation** is the technique of using or writing programs that write **source code**.“

- Code generation (http://en.wikipedia.org)
  - **Compiler Engineering**: component of the synthesis phase
  - **Software Engineering:** program to generate source code

- **Résumé**: Term *Code Generation* is **overloaded!**

Introduction

Code Generation - Basic Questions

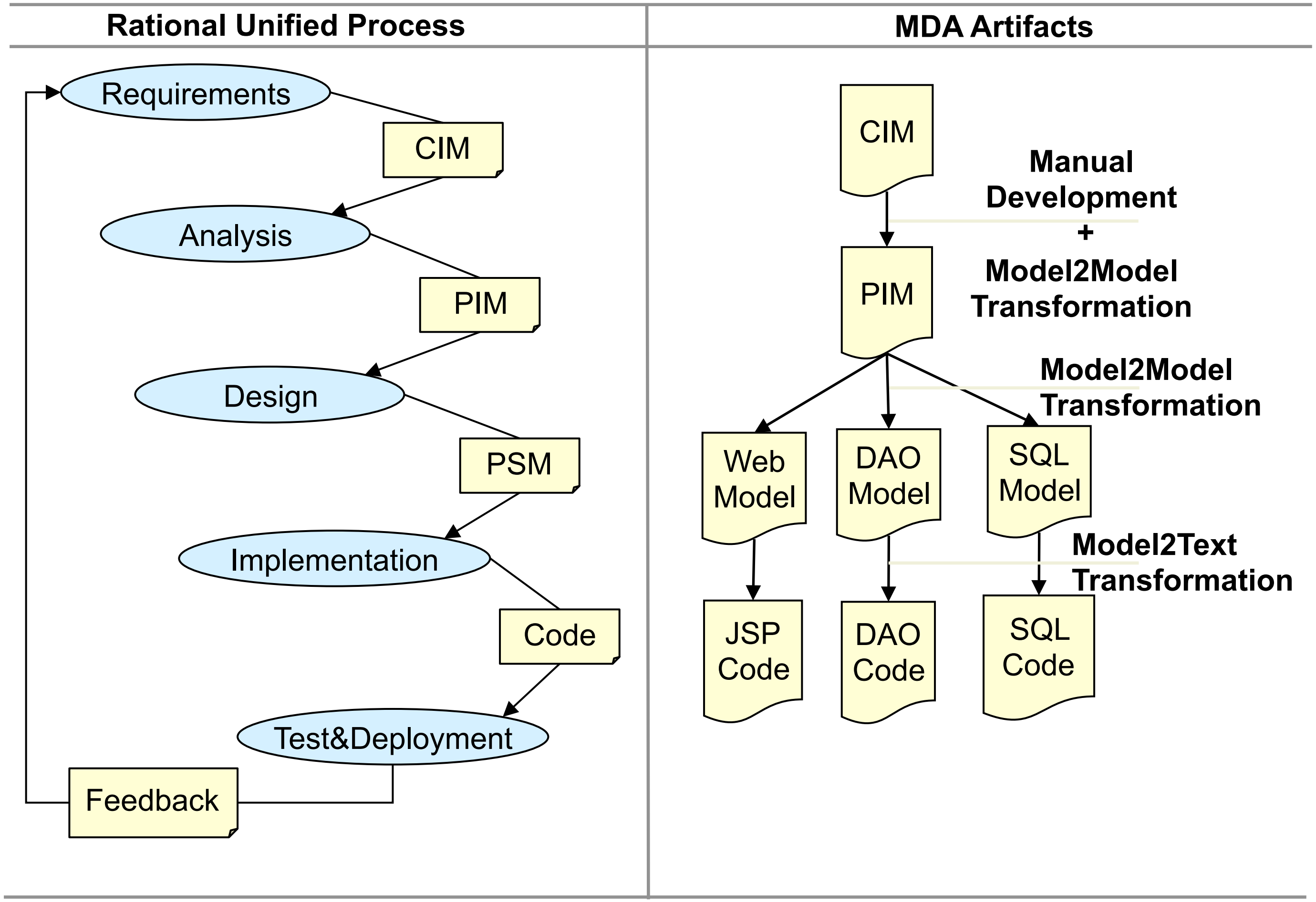
- ***How much is generated?***
  - *Which parts can be automatically generated from models?*
  - *Full or partial code generation?*

- ***What is generated?***
  - *Which kind of source code to generate?*
  - *The less code to generate, the better!*

- ***How to generate?***
  - *Which languages and tools to use for developing code generators?*
  - *GPLs vs. DSLs*

Introduction

Code Generation in MDA (just an example)



Introduction

What kind of code is generated?

- **Model-to-Text**, whereas **text** may be distinguished in
  - Program code
  - Documentation
  - Test cases
  - Model serialization (XMI)

- Direct translation to machine code possible, but inconvenient, error-prone and hard to optimize
  - Reuse existing code generators
  - Using existing functionality (frameworks, APIs, components)
  - **Motto**: The less code to generate, the better!

Introduction

Example: Platform for Web application development

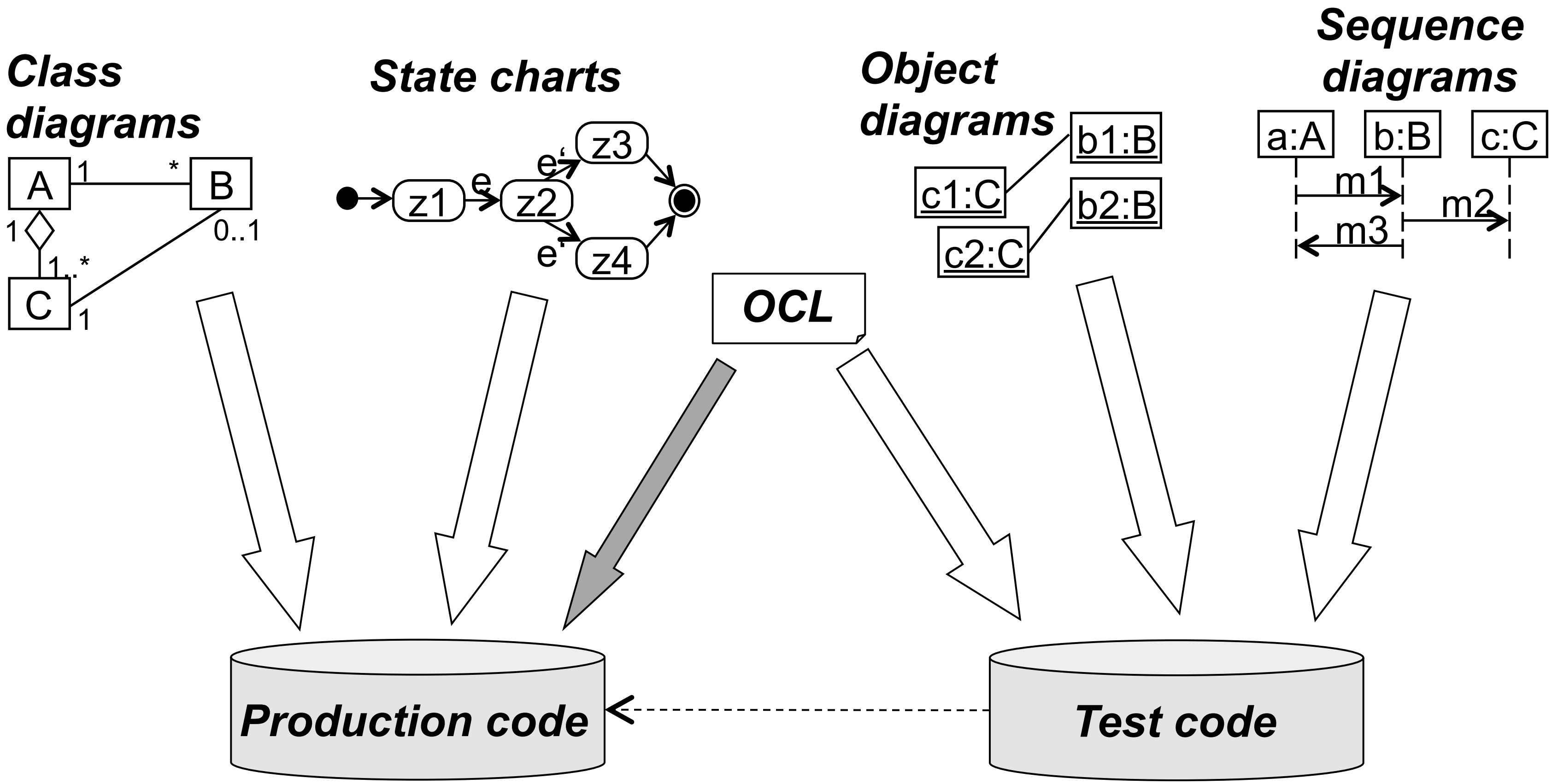
- Example: developing a code generator for Web applications

- *What **options** exist for the to be generated code?*
  - **Dimensions** of Web applications: *Content, Hypertext, Presentation*
  - **Programming languages**: *Java, C#, Ruby, PHP, …*
  - **Architectures**: *2-layer, 3-layer, MVC, ActiveRecords, …*
  - **Frameworks**: *JSF, Spring, Struts, Hibernate, Ruby on Rails, ASP, …*
  - **Products**: *MySQL, Tomcat, WebLogic, …*

- *Which **combinations** are appropriate?*
  - **Experience** gained in earlier projects
    - What has proven useful?
  - **Reference architectures**

Introduction

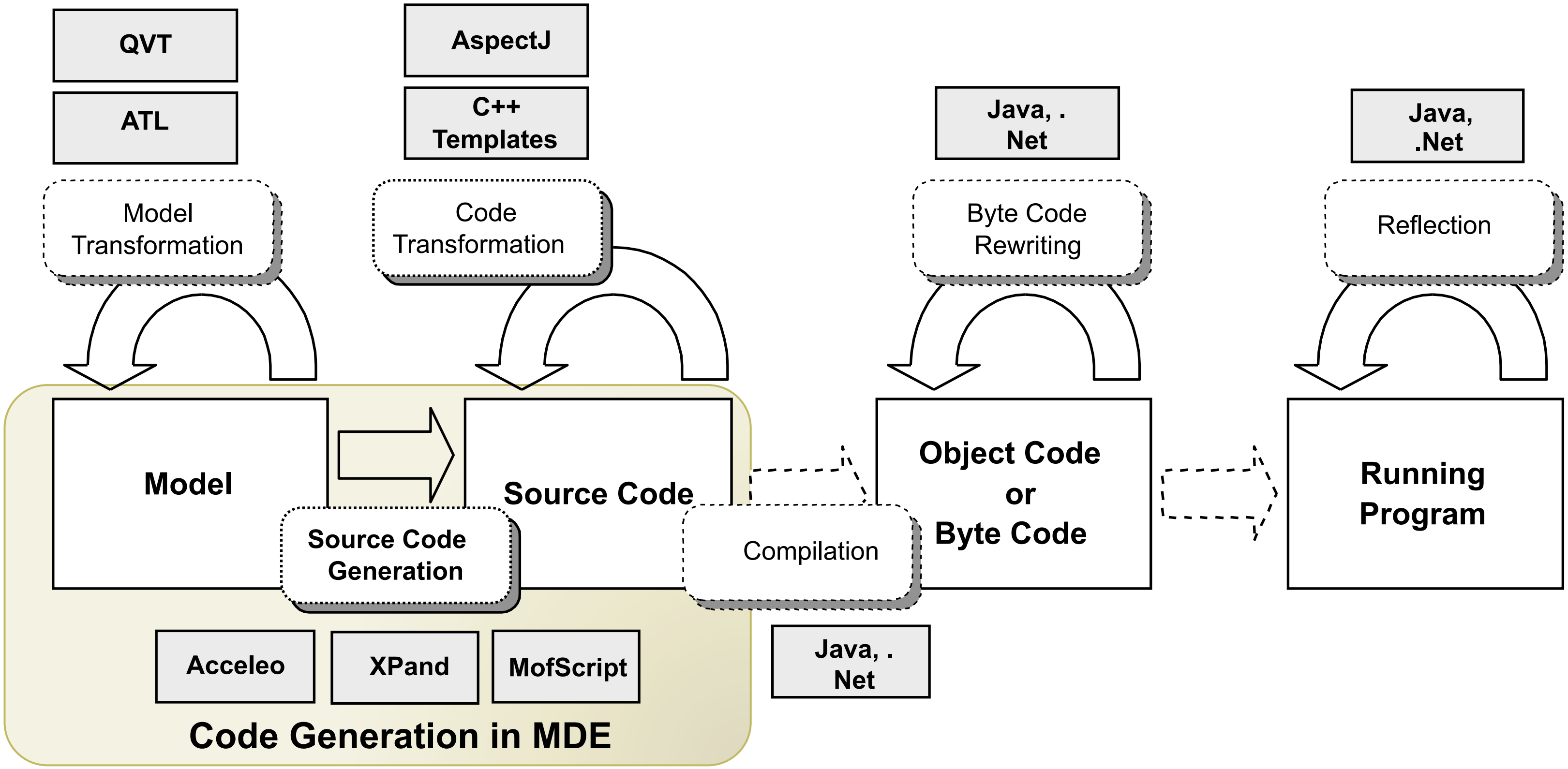
What kind of code is generated?



Picture based on Berhard Rumpe: *Agile Modellierung mit UML*. Springer, 2012.

Introduction

Overview of generation techniques



Based on Markus Völter. A catalog of patterns for program generation. In *Proceedings of the 8th European Conference on Pattern Languages of Programs (EuroPLoP'03)*, pages 285–320, 2003.

Introduction

Why code generation?

- Code generation enables
  - Separation of **application modeling** and **technical code**
  - Increasing **maintainability**, **extensibility, portability** to new hardware, operating systems, and platforms
  - **Rapid prototyping**
  - **Early and fast feedback** due to demonstrations and test runs

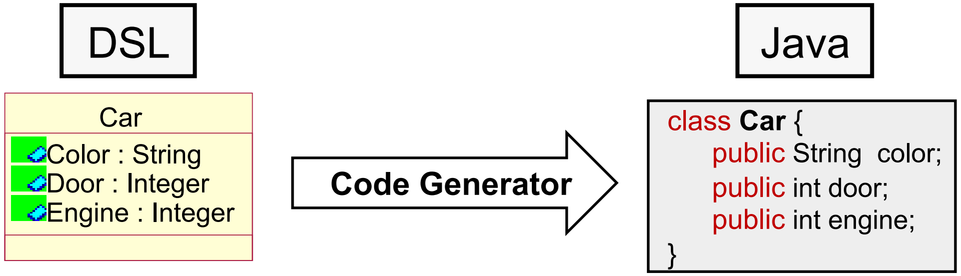
- Code generation enables to **combine** redundant code fragments **in one source**
  - Example: DDL, Hibernate, and Java Beans
    → may be specified in one UML Class Diagram

Introduction

Why code generation? – in contradiction to MDE? (1/2)

- Often **no “real” model simulation** possible
  - UML environments mostly do not provide simulation features
    - However, they provide **transparent transformation** to C, C#, Java, …
  - *UML Virtual Machines*
    - **Interpreter approach** – spare code generation for certain platforms
    - Gets a new twist with fUML!

- **Semantics** of modeling languages, especially DSMLs, often defined by code generation



```
class Car {
      public String  color;
      public int door;
      public int engine;
}
```

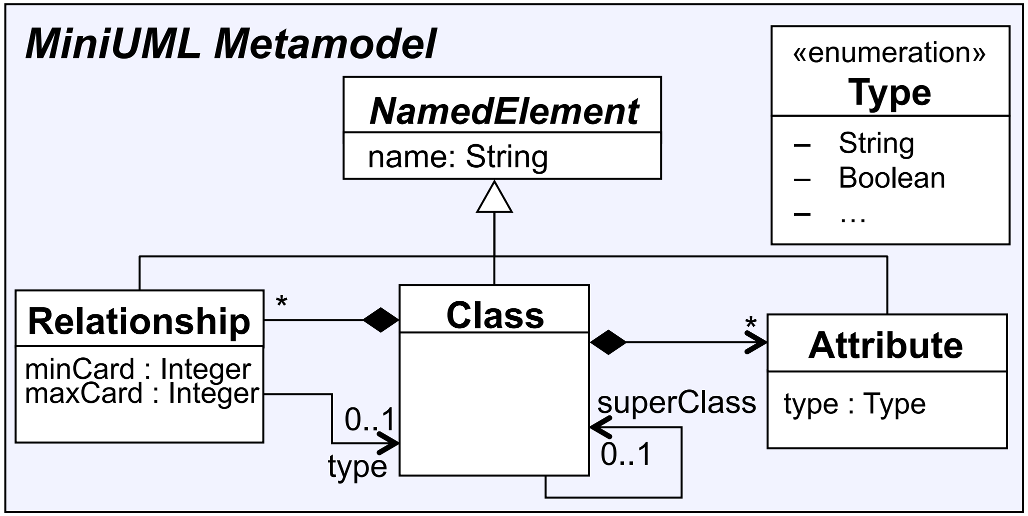
Introduction

Why code generation? – in contradiction to MDE? (2/2)

- Runtime environments are designed for programming languages
  - Established frameworks available (*Struts, Spring, Hibernate*, …)
  - Systems depend on existing software (*Web Services, DB*)
  - Extensions for code level often required (*Logging*)
- **Disadvantage:** using models and code in parallel
  - No single source of information – **OUCH**!
  - Having the same information in **two** places may lead to inconsistences, e.g., consider maintainability of systems

Introduction

Example: MiniUML_2_MiniJava



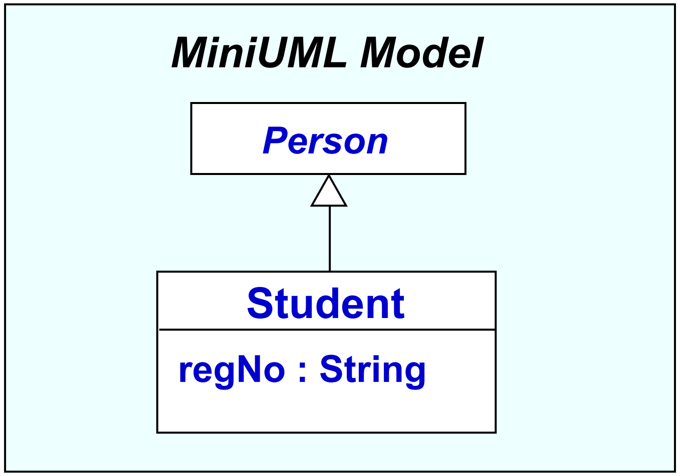
MiniJava Grammar

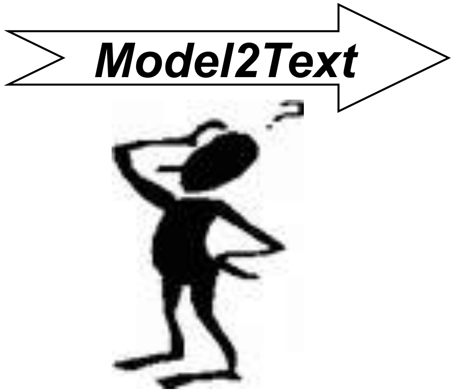
```
ClassDec  := Modifier “class” Identifier [“extends”
             Identifier] ClassBody;

AttributeDec := Modifier Type Identifier“;”;

MethodDec  := Modifier ReturnType Identifier “(”
              ParamList “)” “{” MethodBody “}”;

Identifier  := {“a”-”z” | “A”-”Z” | “0”-”9”};
```





MiniJava Code

```
class Student extends Person{
  private String regNo;
  public void setRegNo(…){…}
  public String getRegNo(){…}
}
```

PROGRAMMING LANGUAGES BASED CODE GENERATION

MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

Programming languages

Introduction – Code generation with Java based on EMF

- Code generation may be realized using a traditional general purpose programming language, e.g., Java, C#, …

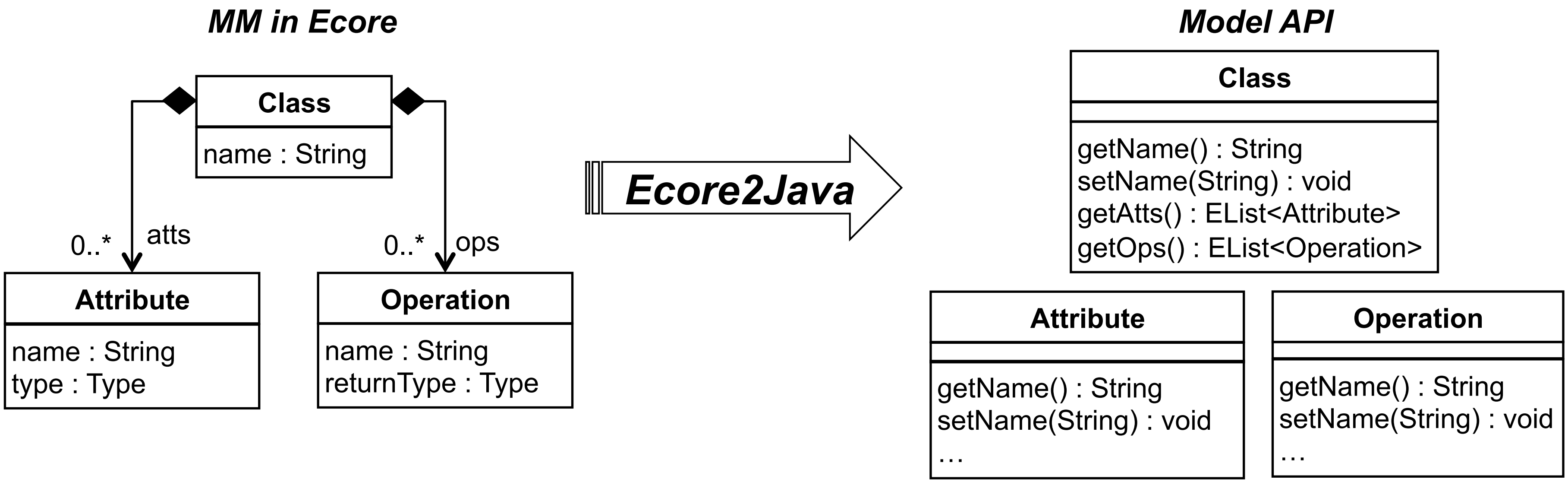
- Models are **de-serialized** to an in-memory *object graph*
  - Pre-defined XMI de-serialzer provided by meta-modeling frameworks
  - Out-of-the-box support in EMF

- **Model API** eases processing of models
  - Generated automatically from metamodels
    - In EMF: .ecore -> .genmodel -> Java code
  - If metamodel not available, you may use ***reflection***

Programming languages

Model APIs for processing models

- **Example**: Ecore-based metamodel and automatically generated Java code (shown as UML Class Diagram)



Programming languages

Code generation with Java: phases of code generation

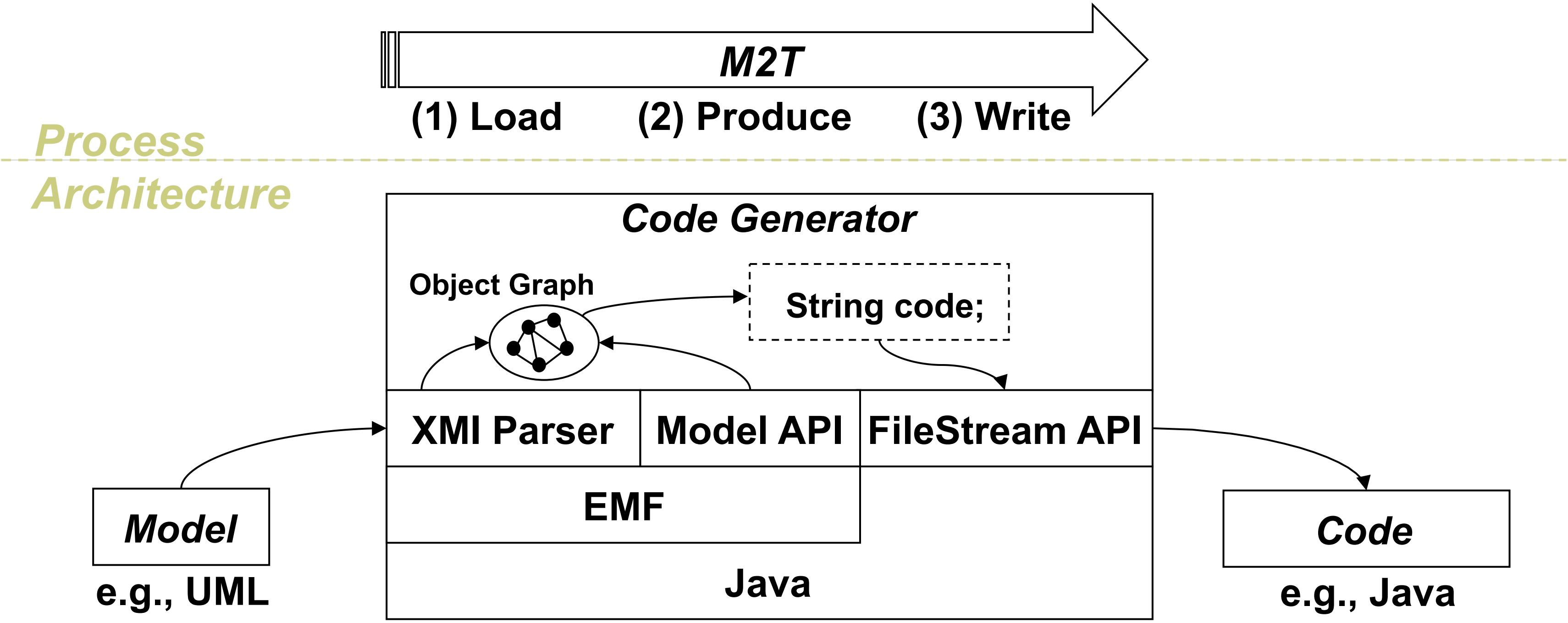
1. ***Load models***
   - Load XMI file into memory

2. ***Process models and produce code***
   - Process models by traversing the model structure
   - Use model information to produce code
   - Save code into String variable

3. ***Write code***
   - Persist String variable to a file using streams

Programming languages

Code generation with Java: Process and Architecture



Programming languages

Running Example solved in Java

(1) Load

```java
ResourceSet resourceSet = new ResourceSetImpl();
Resource resource = resourceSet.getResource(URI.create("model.miniUML"));
TreeIterator treeIter = resource.getAllContents();
```

Get all model elements

```java
while (treeIter.hasNext()) {
   Object object = treeIt.next();
   if (!object instanceof Class) continue;
   Class cl = (Class) object;
   String code = "class „ + cl.getName() + "{";
   // generate Constructor: code += …
   // generate Attributs: code += …
   // generate Methods: code += …
   code += "}";
```

Query values via model API

(2) Produce

```java
  try {
    FileOutputStream fos = new FileOutputStream(cl.getName() +".java“);
    fos.write(code.getBytes());
    fos.close();
  } catch (Exception e) {…}
}
```

Create a file for each class

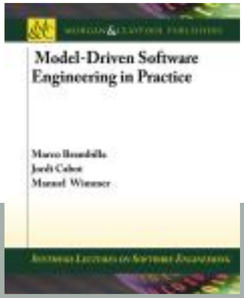
(3) Write

Programming languages

Summary

- **Advantages**
  - No new languages have to be learned
  - No additional tool dependencies
- **Disadvantages**
  - Intermingled static/dynamic code
  - Non-graspable output structure
  - Lack of declarative query language
  - Lack of reusable base functionality

M2T TRANSFORMATION BASED CODE GENERATION



MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING

M2T Transformation Languages…

…are Template based

- **Templates** are a well-established technique in software engineering
  - Application domains: Text processing, Web engineering, …
  - Example:

E-Mail Text

Dear **Homer Simpson**,
Congratulations! You have won …

Template Text

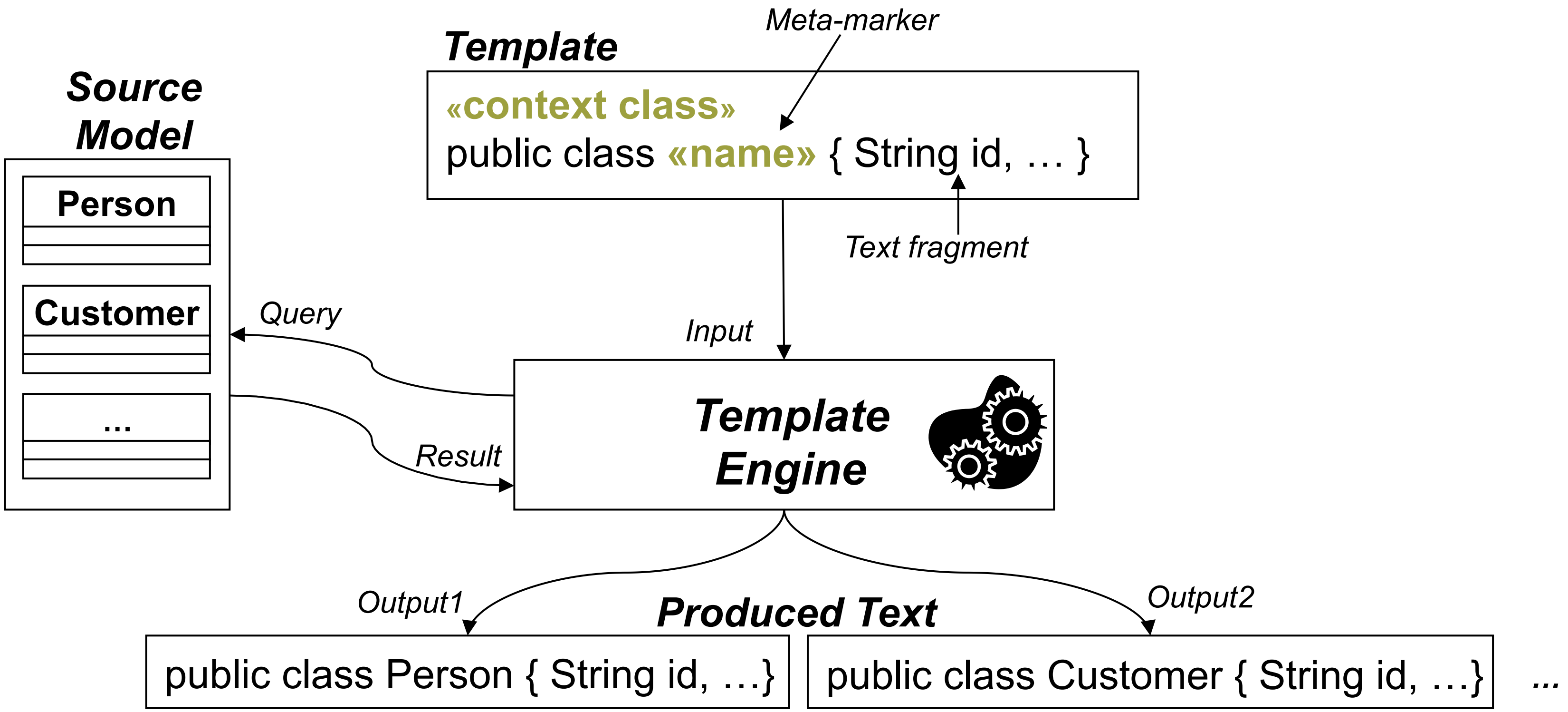
Dear **«firstName» «lastName»**,
Congratulations! You have won …

- Components of a template-based approach
  - **Templates**
    - **Text fragments** and **embedded meta-markers**
  - **Meta-markers** query an additional data source
    - Have to be **interpreted** and **evaluated** in contrast to text fragments
    - Declarative model query: query languages (OCL, XPath, SQL)
    - Imperative model query: programming languages (Java, C#)
  - **Template engine**
    - Replaces meta-markers with data at runtime and produces output files

M2T Transformation Languages

Core Architecture

- Template-based approach at a glance



Template

Meta-marker

«context class»
public class «name» { String id, … }

Text fragment

Source Model

Person

Customer

…

Query

Result

Input

Template Engine

Output1

Produced Text

Output2

public class Person { String id, …}

public class Customer { String id, …}

…

M2T Transformation Languages

Benefits

- **Separated** static/dynamic code
  - Templates separate **static** code, i.e., normal text, from **dynamic** code that is described by meta-markers
- **Explicit** output structure
  - **Primary structure** of the template is the **output structure**
  - Computation logic is embedded in this structure
- **Declarative** query language
  - **OCL** is employed to query the input models
- **Reusable** base functionality
  - Support for reading in models, serialize text to files, …

M2T Transformation Languages

Approaches

- A bunch of template languages for M2T transformation available
  - JET, JET2
  - **Xpand, Xtend**
  - **MOFScript**
  - **Acceleo**
  - XSLT
  - …

Acceleo

Introduction

- ***Acceleo*** is a mature **implementation** of the ***OMG M2T transformation standard***
  - Acceleo website: http://www.eclipse.org/acceleo/
  - M2T Transformation standard*:* http://www.omg.org/spec/MOFM2T

- **Template-based** language
  - Several meta-markers for useful for code generation available

- Powerful **API** supporting
  - OCL
  - String manipulation functions
  - …

- Powerful **tooling supporting**
  - Editor, debugger, profiler, traceability between model and code, …

Acceleo

Language Concepts

- **Module** concept is provided
  - Imports the metamodels for the input models
  - Act as container for templates

- A **template** is always **defined** for a particular **meta-class**
  - Plus an optional **pre-condition** to filter instances
  - Templates may **call** each other
  - Templates may **extended** each other
  - Templates contain text and provided meta-markers

Acceleo

Language Concepts

- Several meta-markers (called *tags*) are supported
- **File** Tag: To open and close files in which code is generated
- **For/If** Tag: Control constructs for defining loops and conditions
- **Query** Tag: Reusable helper functions
- **Expression** Tag: Compute values that are embedded in the output
- **Protected** Tag: Define areas that are not overridden by future generation runs

Acceleo

Example

Import metamodel (root package)

Query

Open output file

Static Text

Template Call

Loop

Close output file

```
[module generateJavaClass('http://smvcml/1.0')]

[query public getter(att : Attribute) : String = 'get'+att.name.toUpperFirst() /]

[query public returnStatement(type: String) : String = if type = 'Boolean'
   then 'return true;' else '...' endif  /]

[template public javaClass(aClass : Class)]

[file (aClass.name.toUpperFirst()+'.java', false, 'UTF-8')]
package entities;

import java.io.Serializable;

public class [aClass.name/] implements Serializable {

[for (att : Attribute | aClass.atts) separator ('\n')]
[javaAttribute(att)/]
[/for]

[for (op : Operation | aClass.ops) separator ('\n')]
[javaMethod(op)/]
[/for]

}
[/file]
[/template]

…
```

Template definition

Meta class

Expression

Protected Area

```
…

[template public javaAttribute(att : Attribute)]
            private [att.type/] [att.name/];

            public [att.type/] [att.getter()/]() {
                        return [att.name/];
            }
  …
[/template]

[template public javaMethod(op : Operation)]
            public [op.type/] [op.name/]() {
                        // [protected] (op.name)]
                        // Fill in the operation implementation here!
                        [returnStatement(op.type)/]
                        // [/protected]
            }
[/template]
```

Marco Brambilla, Jordi Cabot, Manuel Wimmer.
Model-Driven Software Engineering In Practice. Morgan & Claypool 2012.

Acceleo

Protected Areas

- **Protected areas are not overriden** by the **next generator run**

- They are **marked by comments**

- Their content is **merged** with the newly produced code
  - If the right place cannot be found, warning is given!

- **Example**

```
public boolean checkAvailability(){
    // Start of user code checkAvailability
    // Fill in the operation implementation here!
    return true;
    // End of user code
}
```

MASTERING
CODE GENERATION

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

Abstracting Templates

- To ensure that generated code is **accepted** by developers (cf. *Turing test for code-generation*), familiar code should be generated
  - Especially when only a **partial** code generation is possible!

- **Abstract** code generation templates **from reference code** to have known structure and coding guidelines considered

- ***Acceleo*** supports dedicated **refactorings** to **transform code into templates**
  - E.g., substitute String with Expression Tag

Generating step-by-step

- **Divide code generation process** into **several steps**
  - Same applies as for M2M transformations!

- **Transformation chains** may use a **mixture** of M2M and M2T transformations
  - To keep the gap between the models and the code short

- If code generators **exists**, try to produce their required input format with simpler M2M or M2T transformations
  - E.g., code generator for flat state machines, transform composite state machines to flat ones and run existing code generator

Separating transformation logic from text

- **Separete** complex transformation **logic** from **text** fragements

- Use ***queries*** or ***libraries*** that are imported to the M2T transformation

- By this, templates get more **readable** and maintainable

- Queries may be **reused**

Mastering code layout

- **Code layout** is determined by the **template layout**

- Challenging to produce code layout when several control structures such as loops and conditionals are used in the template
  - Special escape characters for line breaks used for enhancing the reabability of the template are provided

- Alternative
  - Use **code beautifiers** in a post-processing step
  - Supported by Xpand for Java/XML out-of-the-box

Model/code synchronization issues

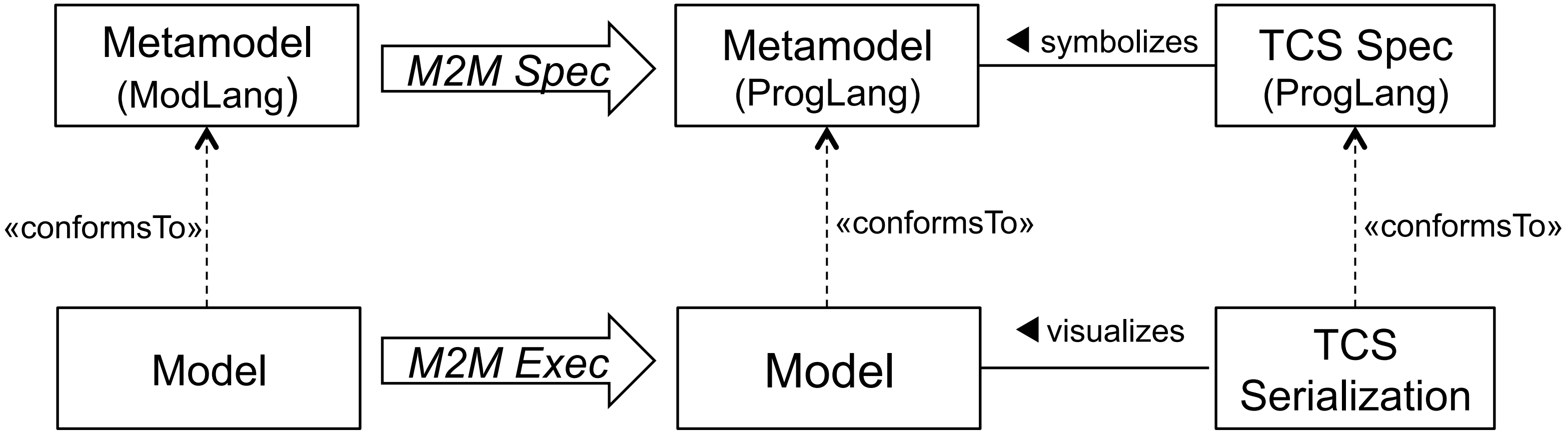
- **Protected areas** help saving manually added code in succeeding generator runs

- Code contained in protected areas is **not always automatically** integrated in the newly generated code
  - Assume a method is renamed on model level
  - Where to place the code of the method implementation?
  - Which identifier to use for identifying a protected area?
  - Natural or artificial identifiers?

- Model refactorings may be replayed on the code level before the next generator run is started
  - Code in protected areas may also reflect the refactorings!

Code Generation = M2M + TCS?

- Code Generation achievable through applying a ***M2M transformations*** to a ***programming language metamodel***
- If a **TCS** is available for the programming language metamodel, the resulting ***model*** may be directly ***serialized*** into text
- **Only recommended** when
  - programming language metamodel + TCS are already **available**
  - **full** code generation is possible

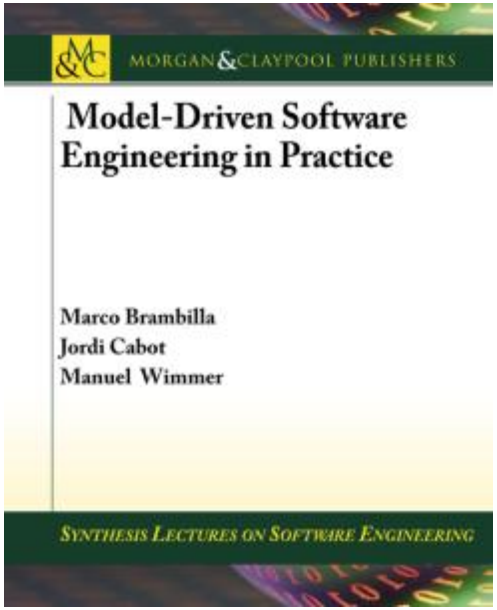


MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE

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

www.mdse-book.com
www.morganclaypool.com



MORGAN & CLAYPOOL PUBLISHERS

Model-Driven Software Engineering in Practice

Marco Brambilla
Jordi Cabot
Manuel Wimmer

SYNTHESIS LECTURES ON SOFTWARE ENGINEERING