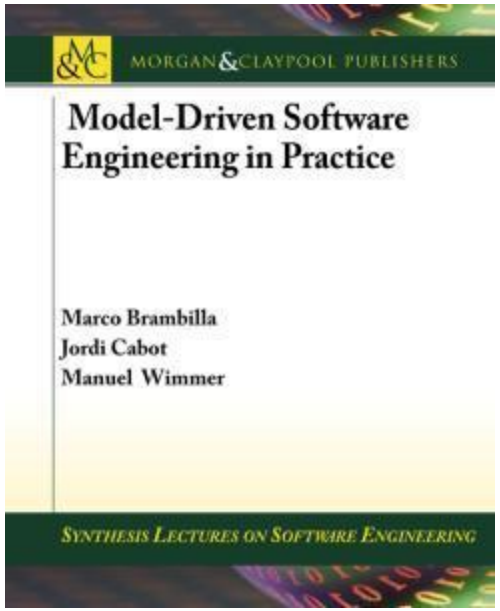
MORGAN & CLAYPOOL PUBLISHERS

CAPÍTULO 9

Transformações MODELO-Para-TEXTO

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



Apresentação: Milena Guessi e Wilmax Cruz

Resumo

- Introdução
- Geração de código baseada em linguagens de programação
- Transformações M2T baseada em geração de código
- Dominando a geração de código

Introdução

Introdução

Terminologia

- Geração de Código
  - *Wikipedia*:
    "**Geração de Código** é o processo pelo qual um gerador de código de um compilador converte um programa sintaticamente correto em uma série de **instruções** que podem ser **executadas pela máquina**."
  - *Code Generation in Action* (Herrington 2003):
    "**Geração de Código** é a técnica de usar ou escrever programas que escrevem **código-fonte".**

- Geração de Código (http://en.wikipedia.org)
  - **Engenharia de Compiladores**: componente da fase de síntese
  - **Engenharia de Software:** programa que gera código-fonte

- **Resumindo**: O termo *Geração de Código* é **sobrecarregado!**

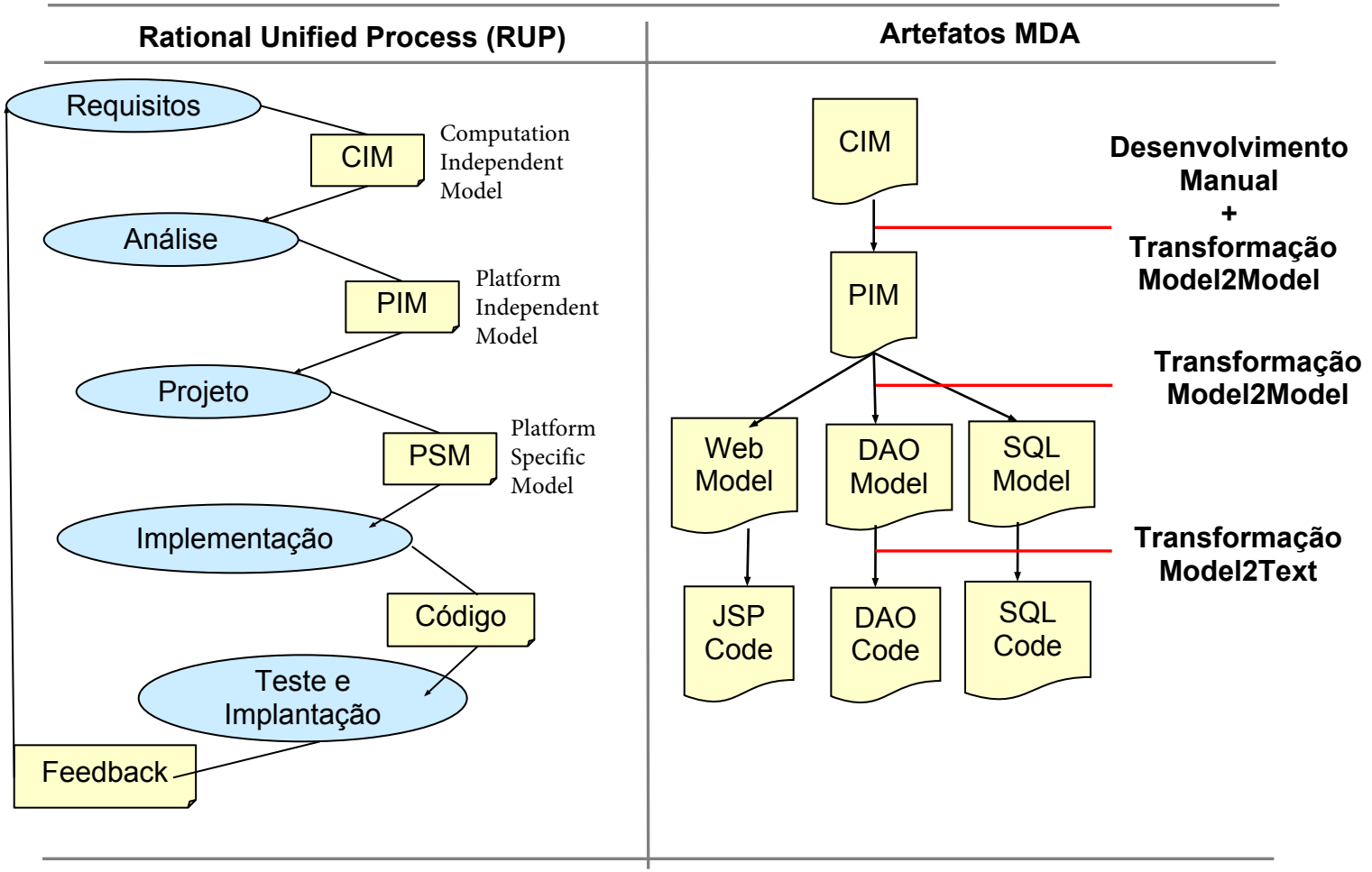
Introdução

Geração de Código - Questões Básicas

- ***Quanto é gerado?***
  - *Que partes podem ser geradas a partir de modelos?*
  - *Geração de código parcial ou total?*
- ***O que é gerado?***
  - *Que tipo de código-fonte gerar?*
  - *Quanto menos melhor!*
- ***Como gerar?***
  - *Que linguagens e ferramentas usar para construir geradores de código?*
  - *GPLs vs. DSLs*

Introdução

Geração de Código em MDA (apenas um exemplo)



Introdução

Que tipo de código é gerado?

- Modelo-para-Texto, onde **texto** pode ser do tipo
  - Código de programa
  - Documentação
  - Casos de Teste
  - Serialização de Modelo (XMI, por exemplo)

- Tradução direto para código de máquina é possível, mas incoveniente, propenso a erros e difícil de otimizar
  - Reuso de geradores de código existentes
  - Utilização de funcionalidades existentes (frameworks, APIs, componentes)
  - **Lema**: Quanto menos código para gerar melhor!

Introdução

Exemplo: Plataforma para desenvolvimento de aplicação Web

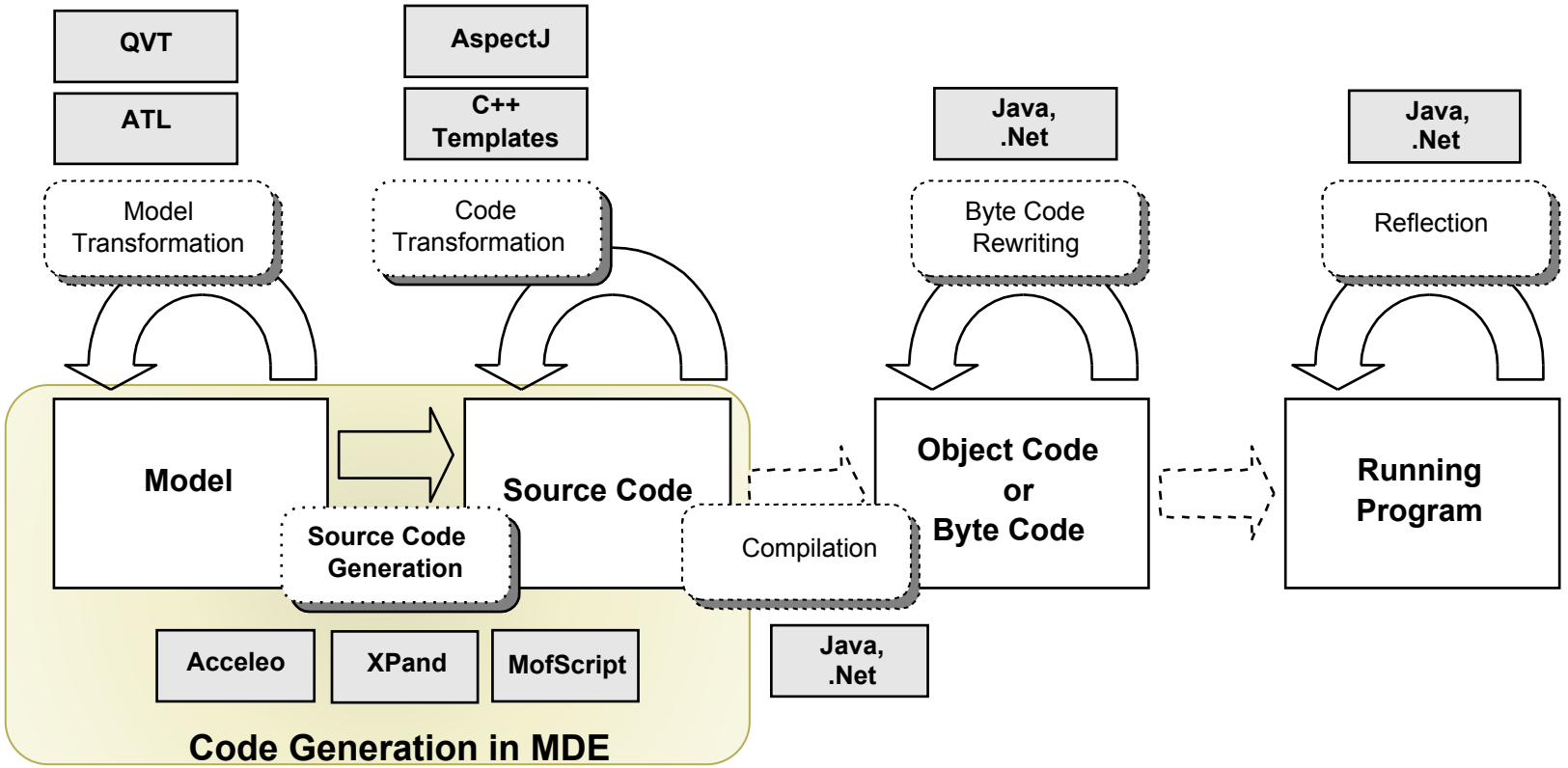
- Exemplo: desenvolvendo um gerador de código para aplicações Web

- *Quais **opções** existem para geração de código?*
  - **Dimensões** de aplicações Web: *Content, Hypertext, Presentation*
  - **Linguagens de Programação**: *Java, C#, Ruby, PHP, …*
  - **Arquiteturas**: *2-layer, 3-layer, MVC, ActiveRecords, …*
  - **Frameworks**: *JSF, Spring, Struts, Hibernate, Ruby on Rails, ASP, …*
  - **Produtos**: *MySQL, Tomcat, WebLogic, …*

- *Quais **combinações** são apropriadas?*
  - **Experiência** ganhada em projetos anteriores
    - O que foi provado como útil?
  - **Arquiteturas de Referência**

Introdução

Resumo de técnicas de geração



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.

Introdução

Porque gerar código?

- Geração de código permite
  - Separação da **modelagem da aplicação** do **código técnico**
    - Melhora **manutenibilidade**, **extensibilidade** e **portabilidade** para novos hardware, sistemas operacionais e plataformas
  - **Prototipação rápida**
  - **Feedback antecipado e rápido** devido a demonstrações e execução de testes

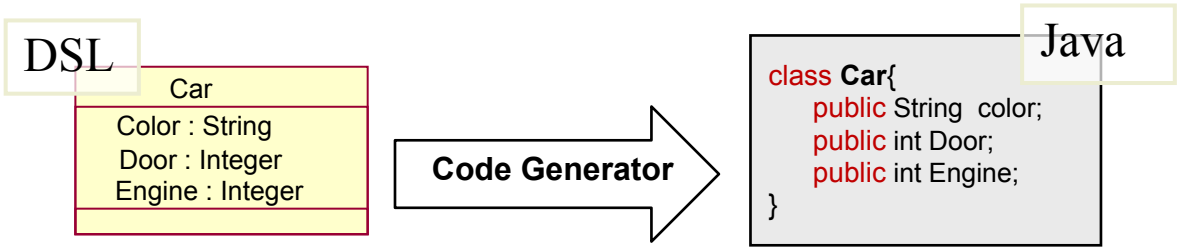
- Geração de código permite **combinar** fragmentos de código redundante **em uma fonte**
  - Exemplo: DDL, Hibernate e Java Beans
    → podem ser especificados em um Diagrama de Classes

Introdução

Porque gerar código?

- Frequentemente, nenhuma simulação “real” do modelo é possível
  - A maioria dos ambientes UML não oferece recursos para simulação
    - No entanto, eles oferecem transformações transparentes para C, C#, Java, …

- **Semântica** de linguagens de modelagem, especialmente DSMLs, frequentemente definidas por geração de código



Introdução

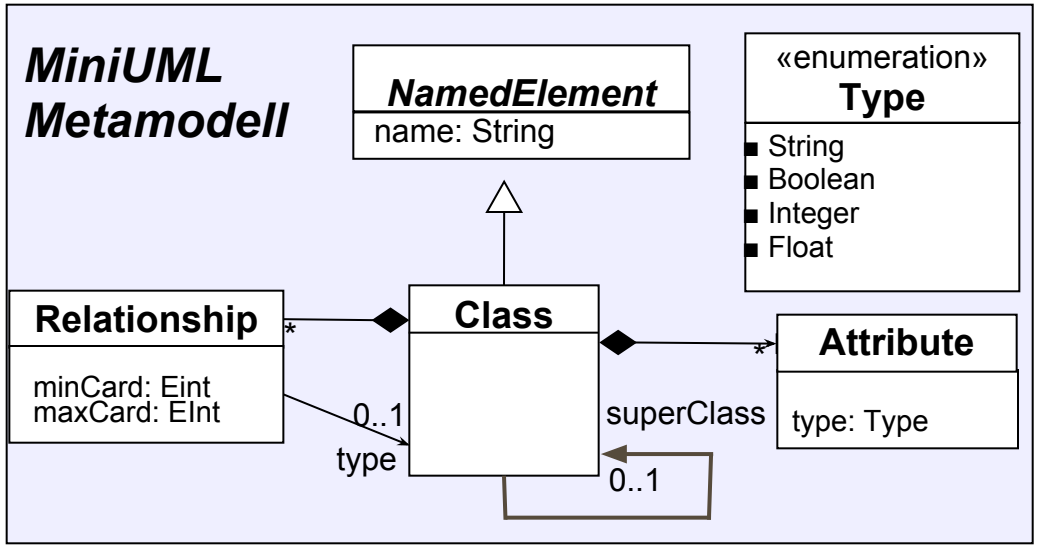
Porque gerar código?

- Ambientes de tempo de execução são projetados para linguagens de programação
  - Frameworks consagrados disponíveis (*Struts, Spring, Hibernate*, …)
  - Sistemas dependem de software existente (*Web Services, DB*)
  - Extensões para nível de código são necessárias com frequência

- **Desvantagem:** usando modelos e código em paralelo
  - Várias fontes de informação – **OUCH**!
  - Ter a mesma informação em **dois** lugares pode resultar em inconsistências, por exemplo, considerando a manutenibilidade dos sistemas.

Introdução

Exemplo: MiniUML_2_MiniJava



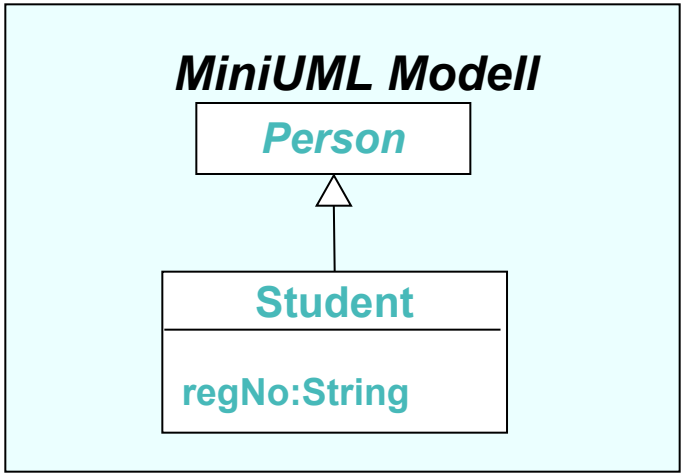
MiniJava Grammar

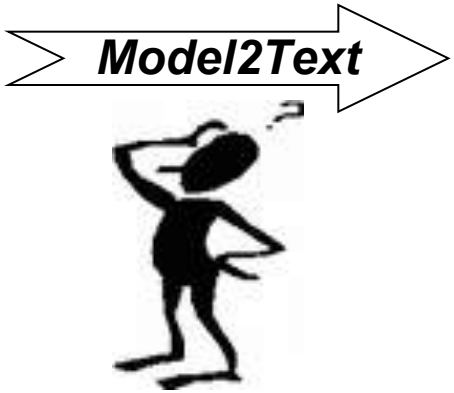
```
ClassDec := Modifier "class" Identifier
["extends" Identifier] ClassBody;

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(){
  …
  }
}
```


Geração de Código baseada em linguagens de programação

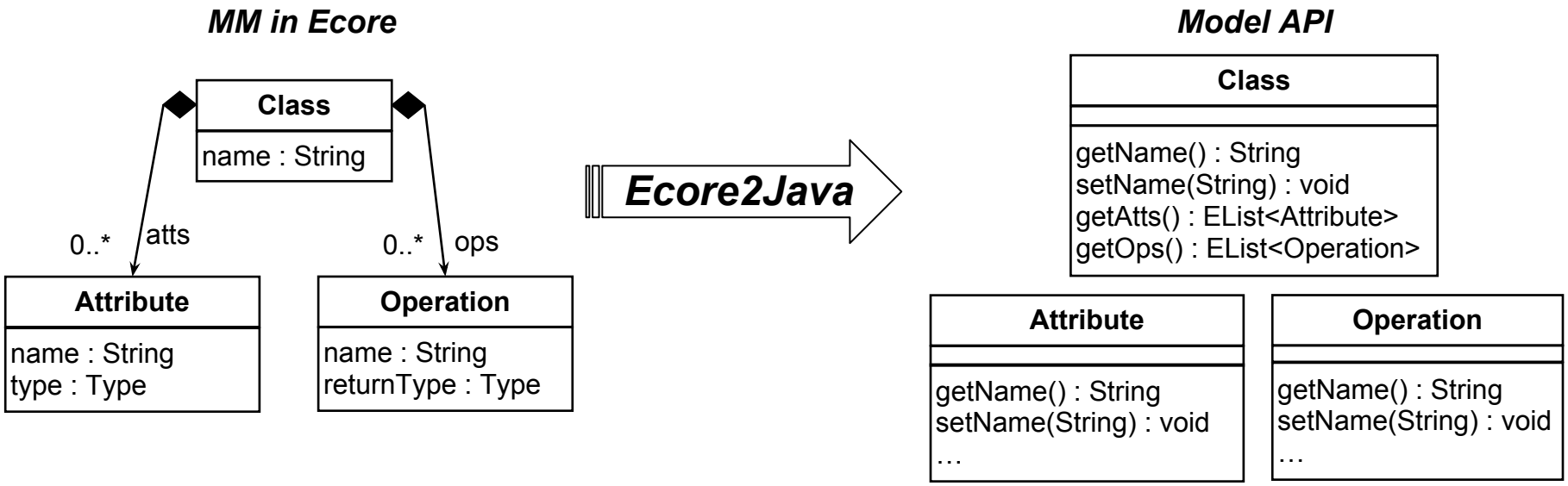
Linguagens de Programação

- Geração de código pode ser alcançada com linguagens de programação de propósito geral tradicionais, por exemplo, Java, C#, …
- Modelos são desconstruídos em um *grafo de objeto* na memória
  - XMI desconstrutor pré-definido é fornecido por framework de meta-modelagem
- **Model API** facilita o processamento de modelos
  - Gerado automaticamente a partir de metamodelos
    - No EMF: .ecore -> .genmodel -> Java code

Linguagens de Programação

Model APIs para o processamento de modelos

- **Exemplo**: Ecore-based metamodelo e código Java gerado automaticamente (mostrado como um Diagrama de Classe UML)



Linguagens de Programação

Geração de código com Java: fases da geração de código

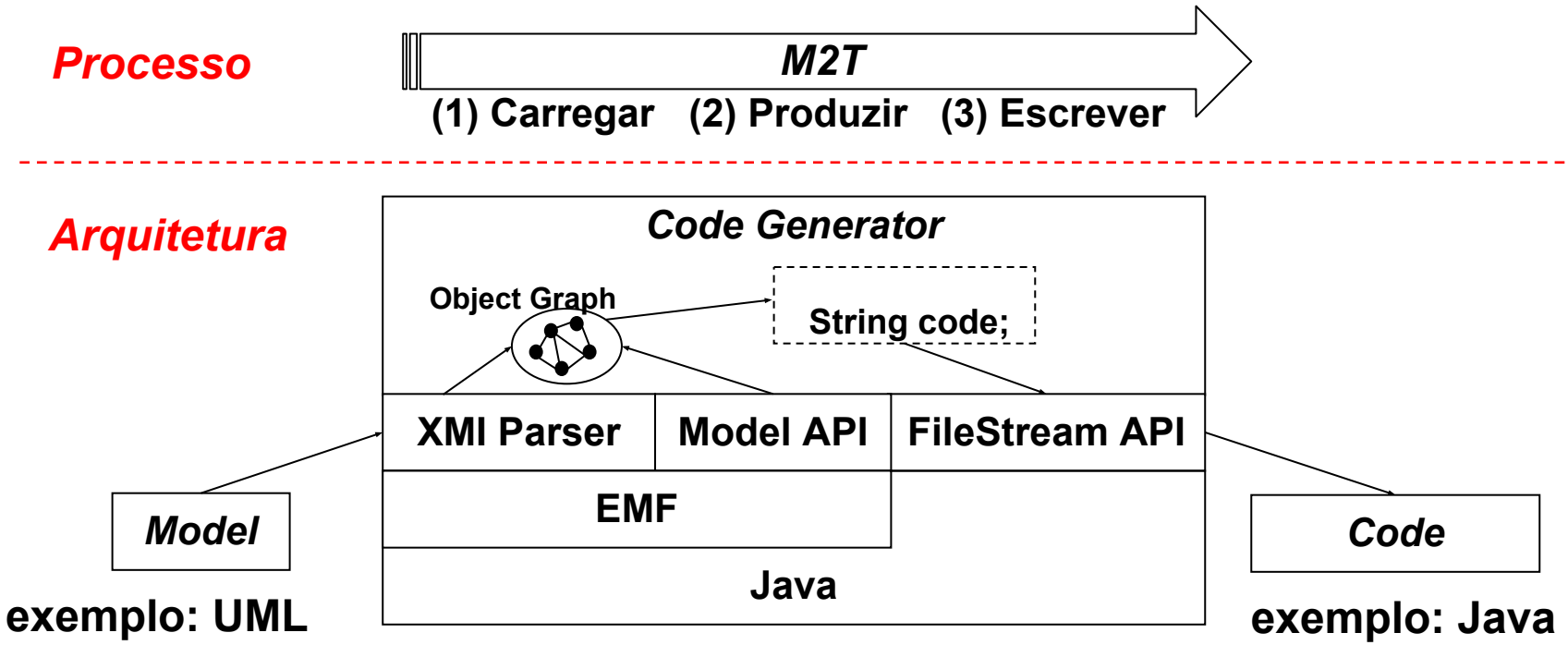
1. ***Carregar modelos***
   - Carregue o arquivo XMI na memória

2. ***Processar modelos e produzir código***
   - Processe modelos cruzando a estrutura do modelo
   - Use informações do modelo para produzir código
   - Salve o código em uma variável String

3. ***Escrever código***
   - Salve a variável String para um arquivo usando *streams*

Linguagens de Programação

Geração de código com Java: Processo e Arquitetura



Linguagens de Programação

Exemplo prático resolvido em Java

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

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 += "}";

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

(1) Carregar

Pegue todos os elementos do modelo

Consulte valores com a API do modelo

(2) Produzir

Crie um arquivo para cada classe

(3) Escrever

Linguagens de Programação

Resumo

- **Vantagens**
  - Nenhuma linguagem nova precisou ser aprendida
  - Nenhuma dependência de ferramenta acrescentada

- **Desvantagens**
  - Código estático e dinâmico misturados
  - Falta de uma linguagem de consulta declarativa
  - Falta de funcionalidades básicas
  - Estrutura de saída obscura (*ungraspable*)

Geração de código baseada em transformações M2T

Milena

Linguagens de Transformação M2T...

…são baseadas em *templates*

- ***Templates*** são uma técnica bem estabelecida em engenharia de software
  - Domínios de aplicação: processamento de texto, engenharia Web,...
  - Exemplo:

Texto de E-Mail

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

Texto do Template

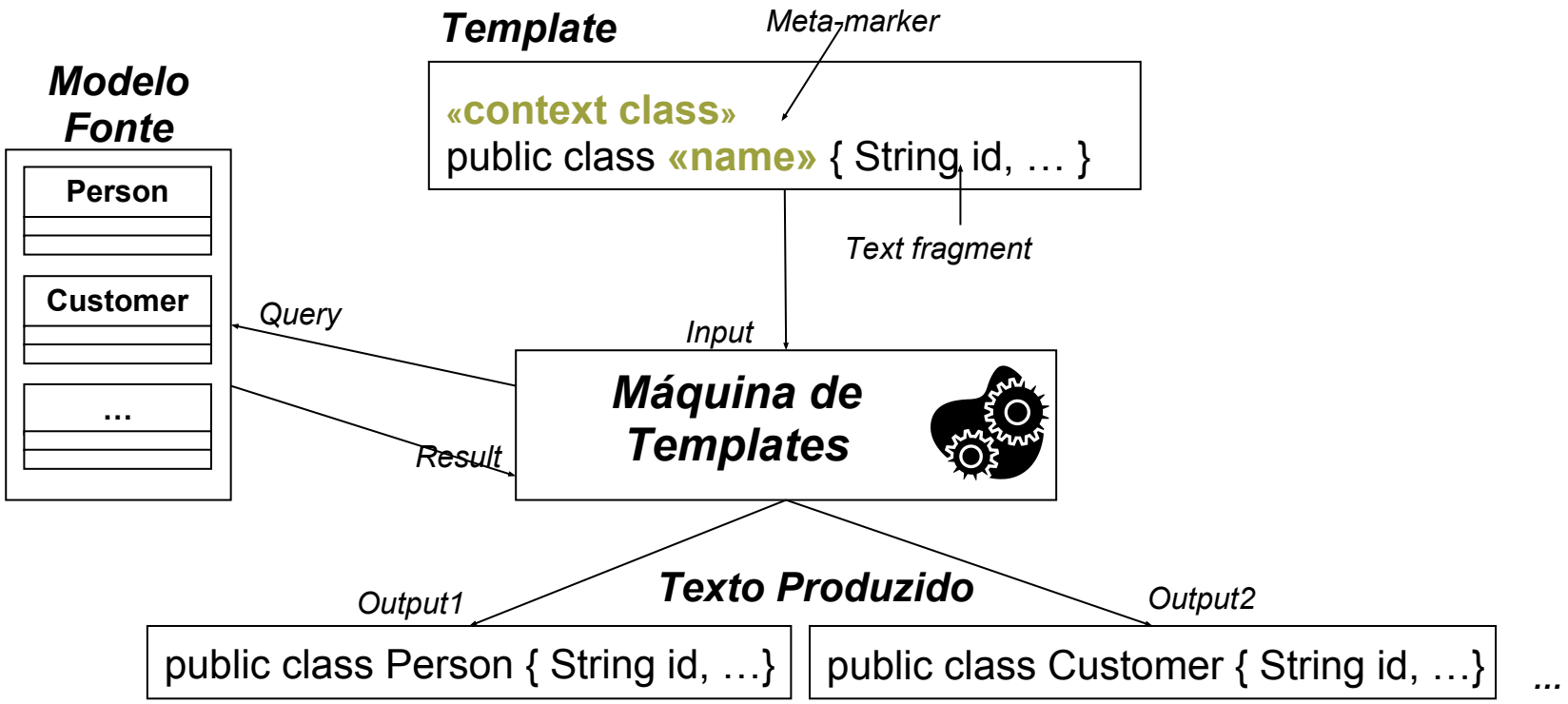
Dear **«firstName» «lastName»**,
Congratulations! You have won …

- Componentes de uma abordagem baseada em *templates*
  - ***Templates***
    - **Fragmentos de texto** e **meta-marcadores embutidos**
    - **Meta-marcadores** consultam uma fonte de informação adicional
    - Precisa ser **interpretada** e **avaliada** ao contrário de fragmentos de texto regulares
    - Consultas declarativas ao modelo: linguagens de consulta (OCL, XPath, SQL)
    - Consultas imperativas ao modelo: linguagens de programação (Java, C#)
  - **Mecanismo de Padrão**
    - Substitui meta-marcadores com dados em tempo de execução e produz arquivos de saída

Linguagens de Transformação M2T

Arquitetural Central

- Visão geral da abordagem baseada em *Templates*



Linguagens de Transformação M2T

Benefícios

- Código estático e dinâmico **separados**
  - *Templates* separam código **estático**, ou seja, texto regular, de código **dinâmico** que é descrito por meta-marcadores
- Estrutura de saída **explícita**
  - **Estrutura primária** do *template* é a estrutura de saída
  - Lógica de computação está embutida nessa estrutura
- Linguagem de consulta **declarativa**
  - **OCL** é empregada para consultar modelos de entrada
- Funcionalidade básica **reutilizável**
  - Apoio para leitura em modelos, produção em série de texto para arquivos, …

Linguagens de Transformação M2T

Abordagens

- Várias linguagens de *template* para transformações M2T estão disponíveis
  - JET, JET2
  - **XPAND**
  - **MOFScript**
  - **Acceleo**
  - XSLT
  - …

Acceleo

Introdução

- ***Acceleo*** é uma **implementação** madura do ***padrão OMG para transformação M2T***
  - Disponível em: http://www.eclipse.org/acceleo/
  - Padrão para transformação M2T: http://www.omg.org/spec/MOFM2T
- Linguagem **baseada em *templates***
  - Vários meta-marcadores para geração de código disponíveis
- Poderosa **API** apoia
  - OCL
  - Funções para manipulação de strings
  - …
- Poderosa ferramenta apoiando
  - Edição, debugger, profiler, rastreabilidade entre modelo e código, …

Acceleo

Conceitos da linguagem

- Conceito de **módulo**
  - Importa metamodelos para os modelos de entrada
  - Atua como um conteiner para os *templates*

- Um ***template*** é sempre **definido** para uma **meta-classe** particular
  - Além de uma pré-condição opcional para filtrar instâncias
  - *Templates* podem **chamar** uns aos outros
  - *Templates* podem **estender** uns aos outros
  - *Templates* contém texto e meta-marcadores oferecidos

Acceleo

onceitos da linguagem

- Apoio de vários meta-marcadores (chamados *tags*, etiquetas)
  - **File**: para abrir e fechar arquivos nos quais código é gerado
  - **For/If**: controla construtores para definição de laços e condições
  - **Query**: funções de ajuda reutilizáveis
  - **Expression**: computa valores que estão embutidos na saída
  - **Protected**: define regiões que não são sobrescritas em futuras execuções

Acceleo

Exemplo

```
[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 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]
```

Import metamodel (root package)

Query

Open output file

Static Text

Template definition

Meta class

Template Call

Expression

Loop

Protected Area

Close output file

Acceleo

Regiões Protegidas

- **Regiões protegidas não são sobrescritas** em **execuções futuras do gerador**
- São **marcadas por comentários**
- O seu conteúdo é **combinado** ao novo código produzido
  - Se o lugar certo não pode ser encontrado é dado um aviso!
- **Exemplo**

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


Dominando Geração de Código

Abstraindo *Templates*

- Para garantirar que o código gerado é **aceito** pelos desenvolvedores (cf. *Teste de Turing para geração de código*), código familiar deve ser gerado
  - Especialmente quando apenas geração **parcial** de código é possível!

- **Abstraia** padrões de geração de código **a partir de código de referência** para ter estrutura conhecida e diretrizes de codificação consideradas

- ***Acceleo*** apoia ***refactorings*** dedicados para **transformar código em *templates***
  - Por exemplo, substituir *String* por uma etiqueta *Expression*

Geração passo a passo

- **Divide** processo de geração de código em vários passos
  - O mesmo se aplica para transformações M2M!
- **Cadeias de transformação** podem usar uma **mistura** de transformações M2M e M2T
  - Para manter a lacuna entre modelo e código estreita
- Se geradores de código **existem**, tente produzir o formato de entrada exigido com transformações M2M ou M2T mais simples
  - Por exemplo, gerador de código para máquinas de estado simples, transforme máquinas de estado compostas em máquinas de estado simples e execute o gerador de código

Separando transformação lógica do texto

- **Separe** lógica de transformação complexa de fragmentos de **texto**
- Use ***queries*** ou ***libraries*** que são importadas para a trasformação M2T
- Com isso, *templates* são mais **fáceis de ler** e manter
- *Queries* podem ser **reutilizadas**

Dominando Aparência do Código

- **Aparência do código** é determinada pelo ***template* de aparência**

- É desafiador produzir a aparência do código quando várias estruturas como laços e condições são usadas no *template*
  - Caracteres especiais para quebra de linhas são fornecidos para melhorar a capacidade leitura do padrão

- Alternativa
  - Usar **embelezadores de código** em um passo pós-processamento
  - Apoiado por Xpand para Java/XML *out-of-the-box*

Esteja ciente dos problemas de sincronização do modelo/código

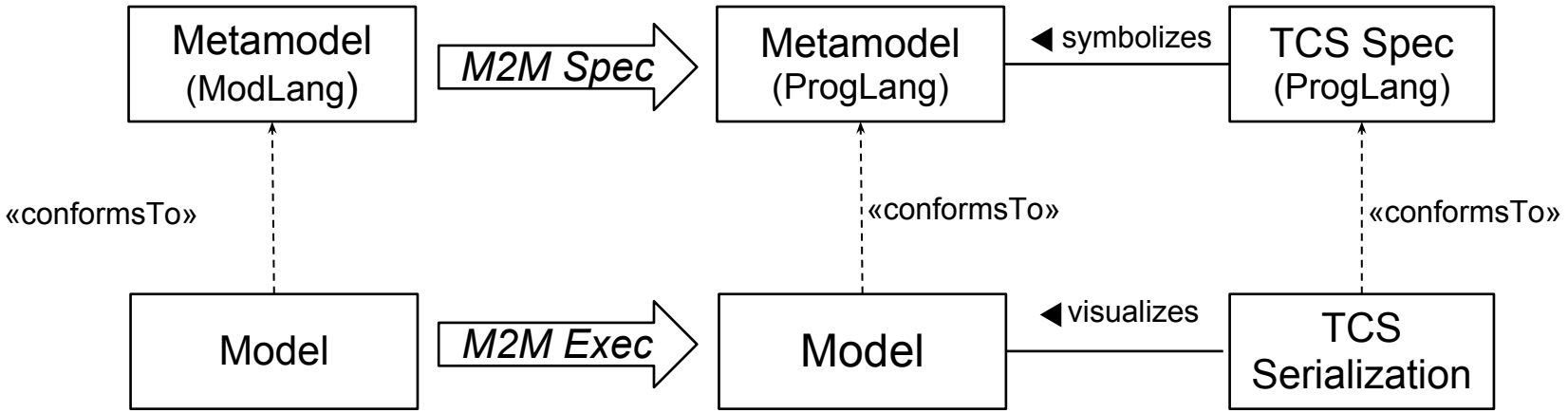
- **Regiões protegidas** ajudam a salvar código em execuções subsequentes

- Código contido em regiões protegidas nem sempre serão automaticamente integrados no código gerado mais recente
  - Considere que um método é renomeado no novo modelo
  - Onde colocar o código da implementação do método?
  - Quais identificadores usar para determinar uma região protegida?
  - Identificadores naturais ou artificiais?

- Refatoração do modelo pode ser executada em nível de código antes da próxima execução do gerador ser iniciada
  - Código em áreas protegidas também pode refletir essas refatorações!

Geração de código por M2M + TCS

- Geração de código pode ser alcançada aplicando-se ***transformações M2M*** em um **metamodelo de linguagem de programação**
- Se um **TCS** estiver disponível para o metamodelo da linguagem de programação, o **modelo** resultante pode ser diretamente **sequenciado** em texto
- Apenas **recomendado** quando
  - metamodelo da linguagem de programação + TCS estiverem **disponíveis**
  - é possível gerar código **completamente**



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