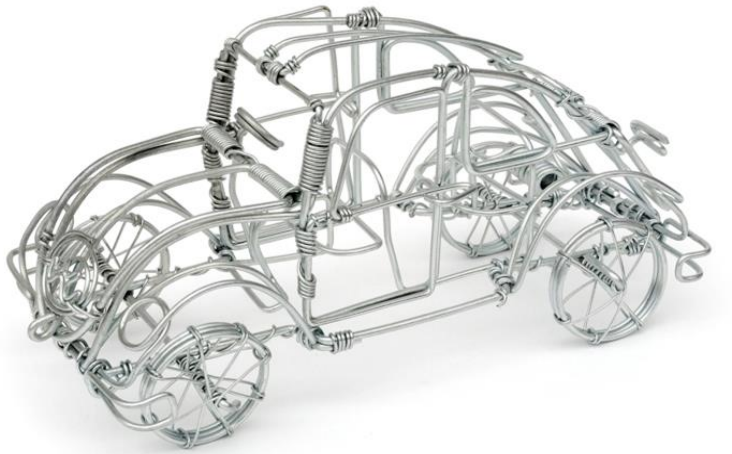
Introduction to Code Generation for Model-Driven Software Development (MDSD)

Special thanks to Sven Jörges, formerly at TU Dortmund.

MDSD Concept

Models in Software Development



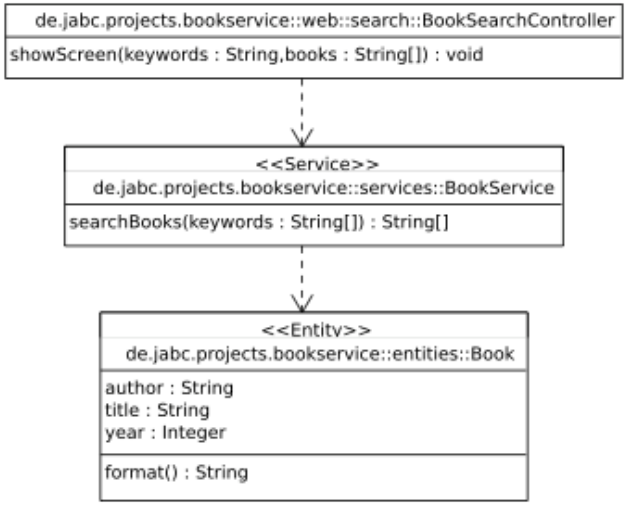
- no clear definition available
- Examples:
- OMG: *„Modeling is the designing of software applications before coding. [...] A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper.“*

- Mellor & Clark: *„A model is a coherent set of formal elements describing something (for example, a system, bank, phone, or train) built for some purpose that is amenable to a particular form of analysis [...]“*
- Bran Selic: *„A reduced representation of some system or process, which emphasizes properties that are of interest to a given set of concerns“*

abstraction; reduction to aspects (concerns); for a goal

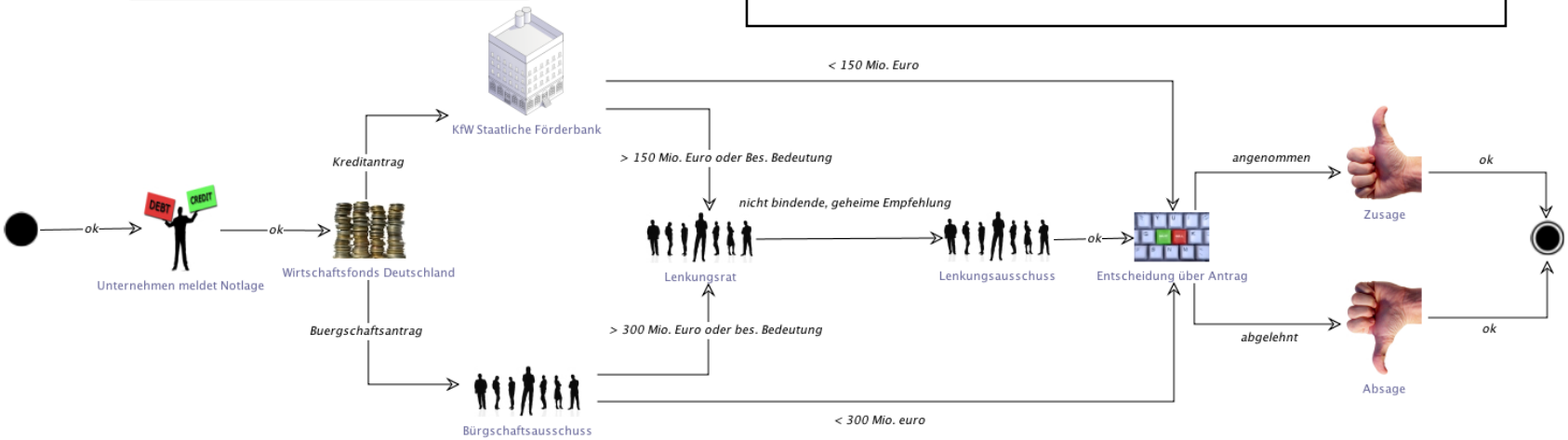
MDSD Concept

Model Examples




```
entity Customer {
  String firstName
  String lastName
  String email
  Address shippingAddress
  Order* orders
}

entity Product {
  String description
  Double price
}
```

```
entity Address {
  String line1
  String line2
  String city
  String postcode
  String state
  String country
}
```



MDSD Concept

Key Ideas/Goals of MDSD

- models as **development artifacts / „first citizens“**
- **automatic code generation**:
  - to concrete target platform (Java EE, .NET, mobile phone, ...)
  - maybe preceeded by: model refinement, model transformations, ...
- **domain specific languages**:
  - models use elements of a language tailored for a specific application domain
  - domain expertise can better be incorporated
  - code generators can be more restricted

MDSD Concept

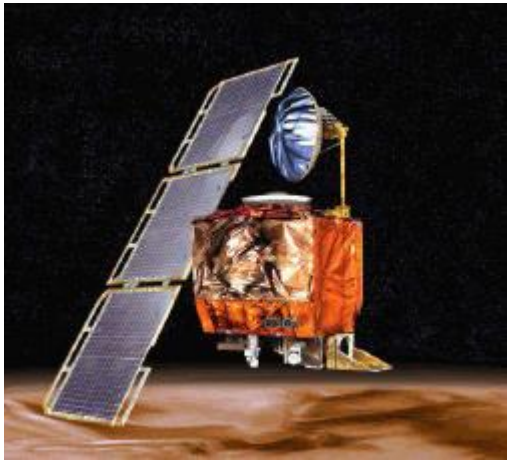
Code Generation: Advantages

- higher code quality: no manual coding errors
- less repetitive work (e.g. boilerplate code)
- translate a model to another target platform = choice/implementation of an appropriate code generator

The Vision:
Complete, running system „at the push of a button“.

MDSD Concept

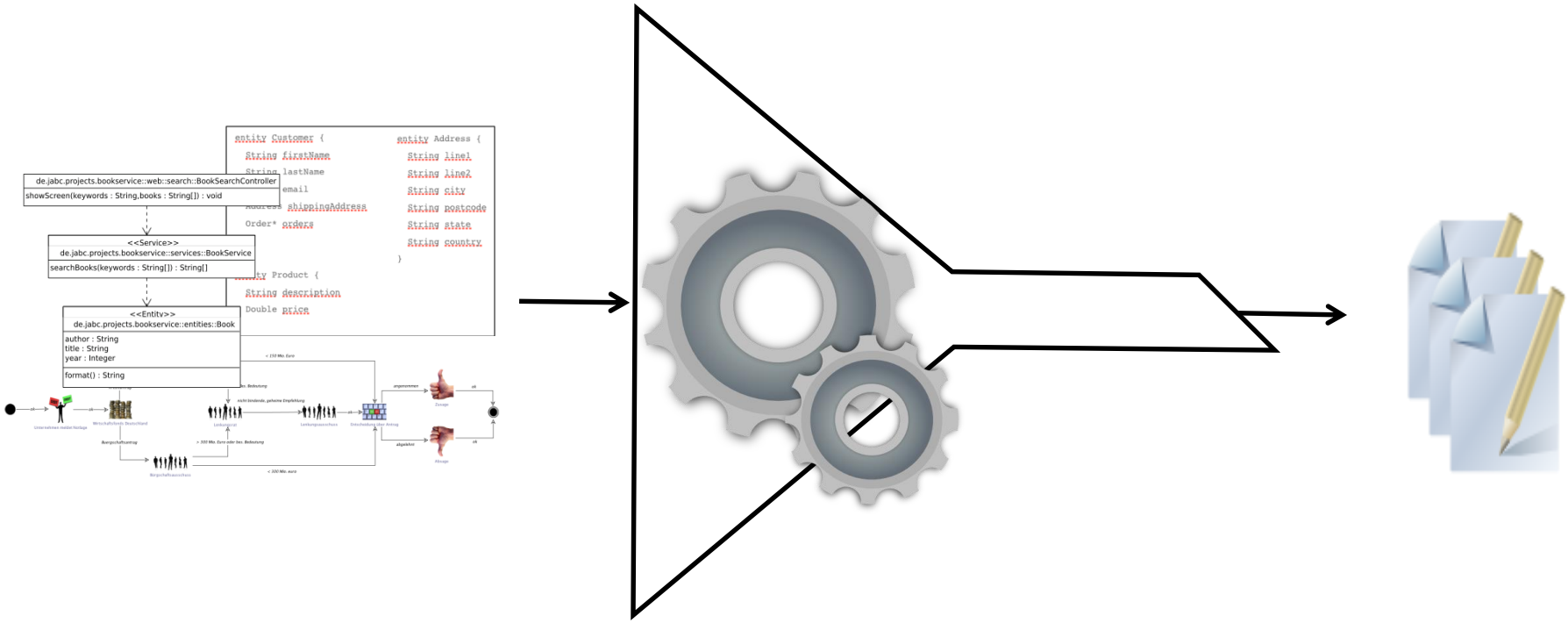
Fatal consequences of software errors



Billions of dollars lost, caused by software errors.

MDSD Concept

Code Generation: Big Picture



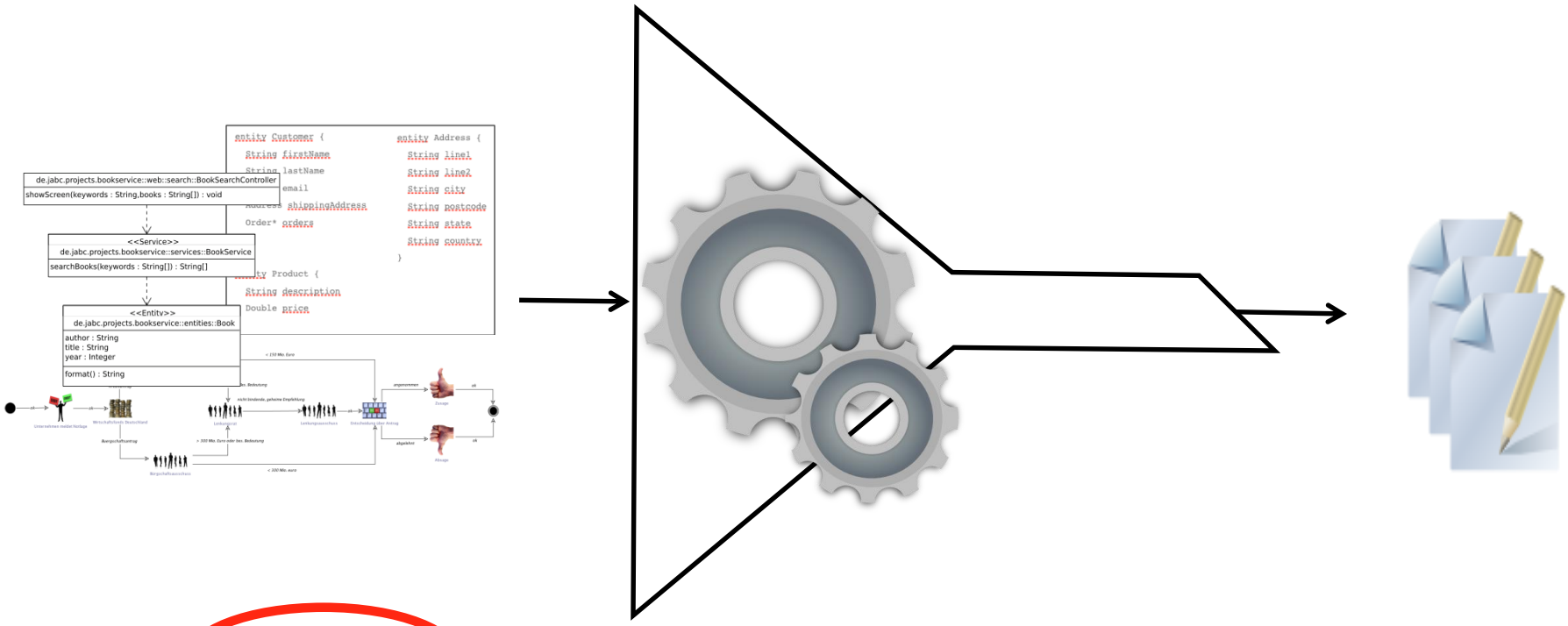
Models

Code Generator

Text

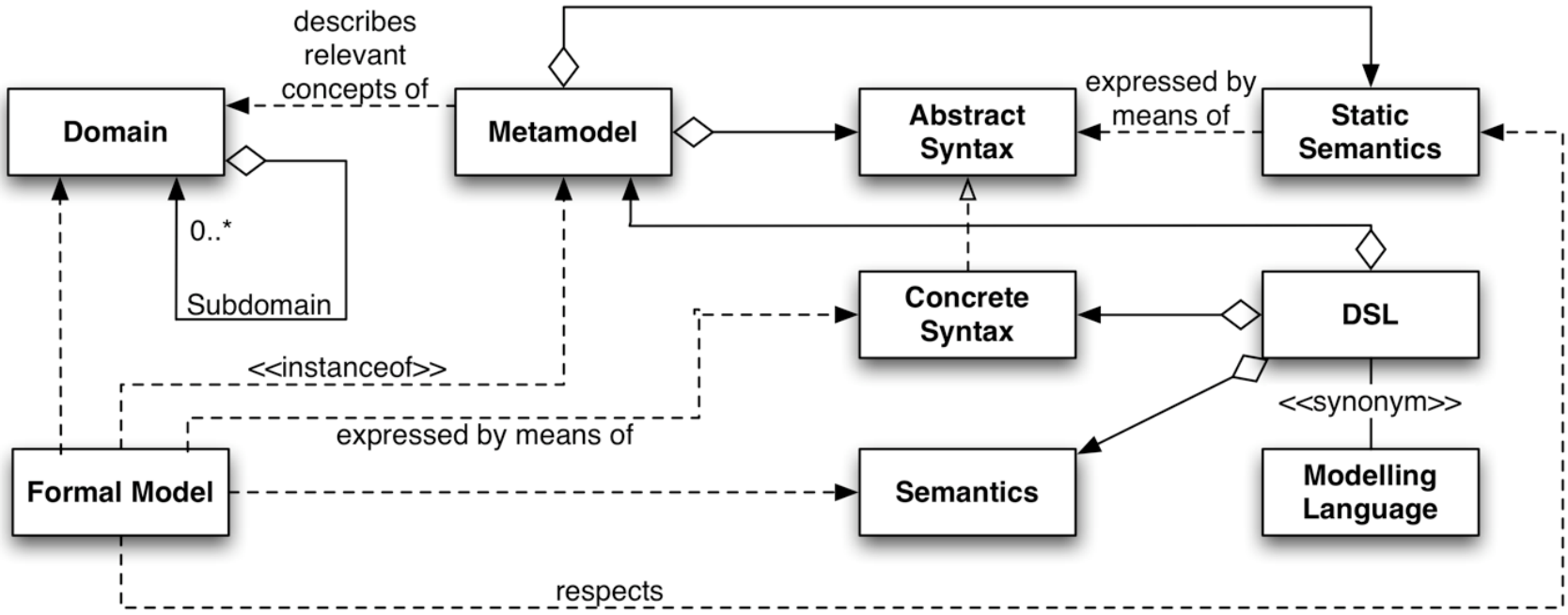
MDSD Concept

Code Generation: Big Picture



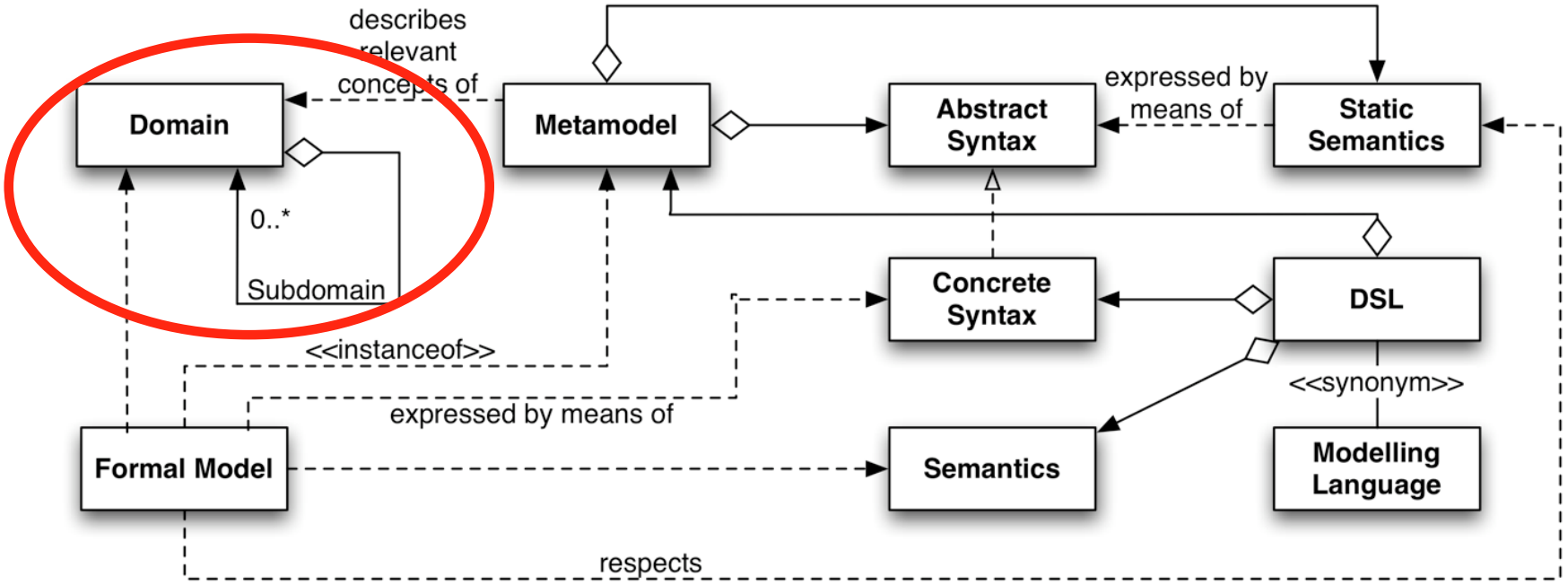
Domain Specific Languages

Definition of Domain Specific Languages - Concepts & Notions



Source: [MDSD], S. 28

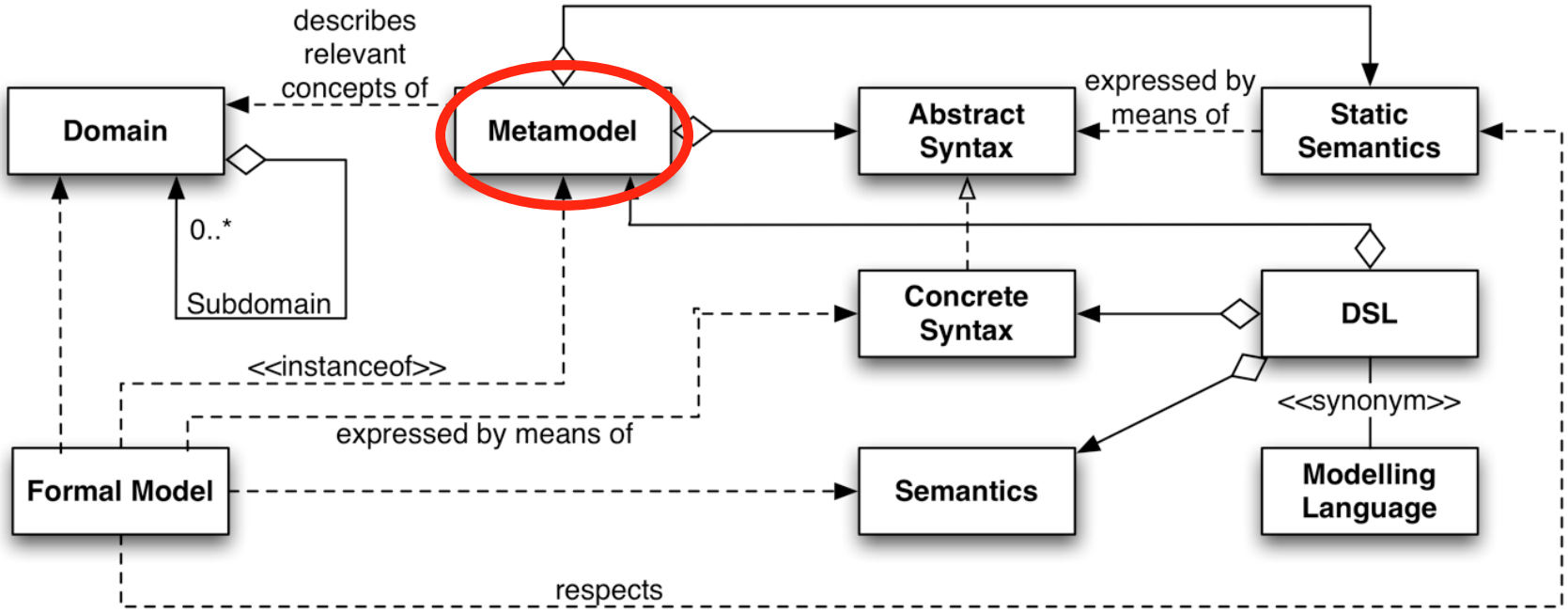
Domain Specific Languages



domain:

- delimited field of interest or knowledge
- may be divided into subdomains
- contains „real things“
- e.g. domain „hospital“, with subdomains „intensive care unit“, „coronary care unit“, ...

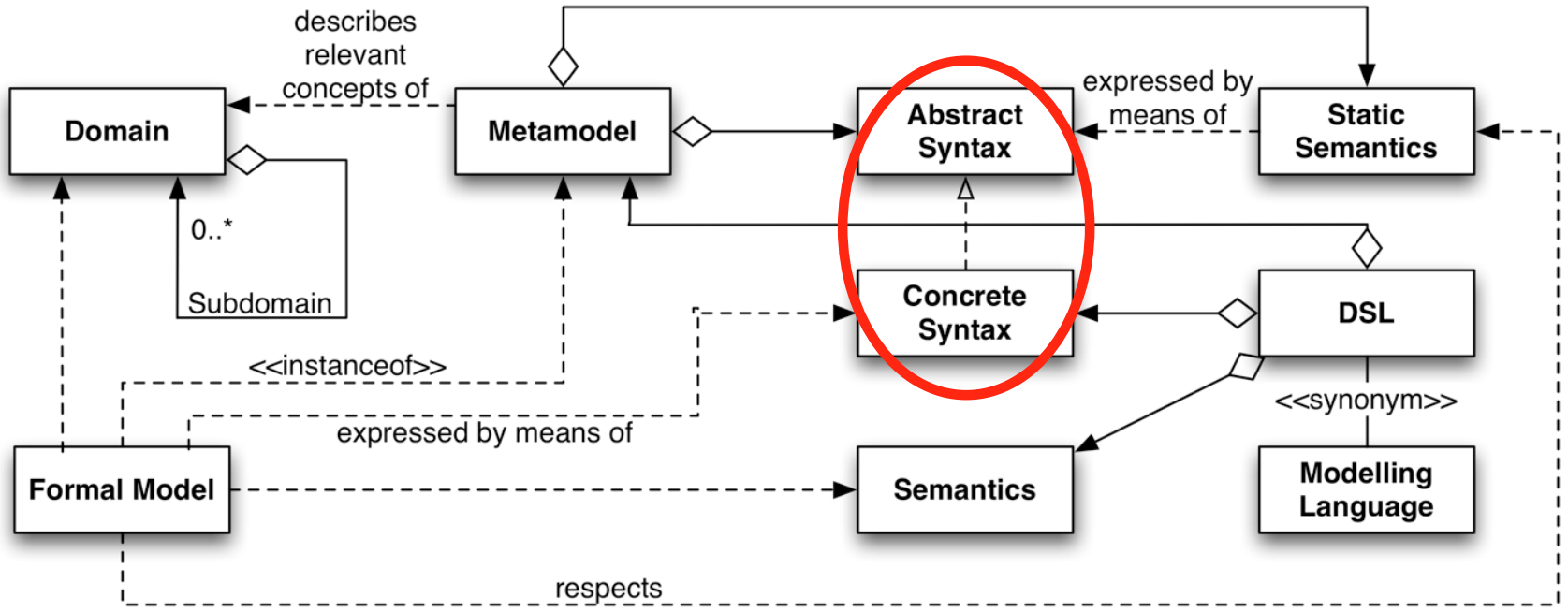
Domain Specific Languages



metamodel:

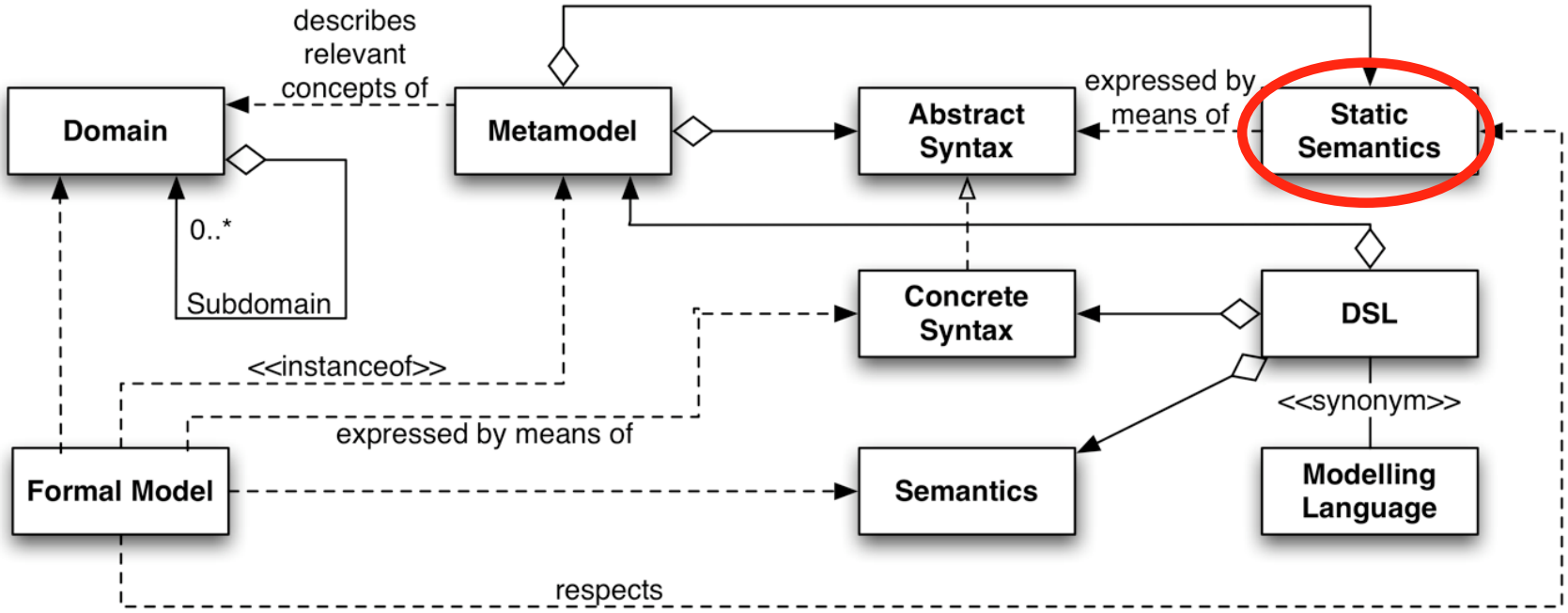
- formal description of a domain`s relevant concepts
- describes how formal models for a domain are composed (modeling constructs, their relations etc.)
- comprises the abstract syntax & static semantics of a language

Domain Specific Languages



- *abstract syntax* denotes the metamodel elements & their relations
- *concrete syntax*: actual syntax of the formal model like text, flow diagram etc.
- e.g.:
- abstract syntax of OOP: concept „*class* with property *name* may *inherit* from another *class*“
- concrete syntax of Java: `class`, `extends`, ...
- multiple concrete syntaxes for one abstract syntax may be defined

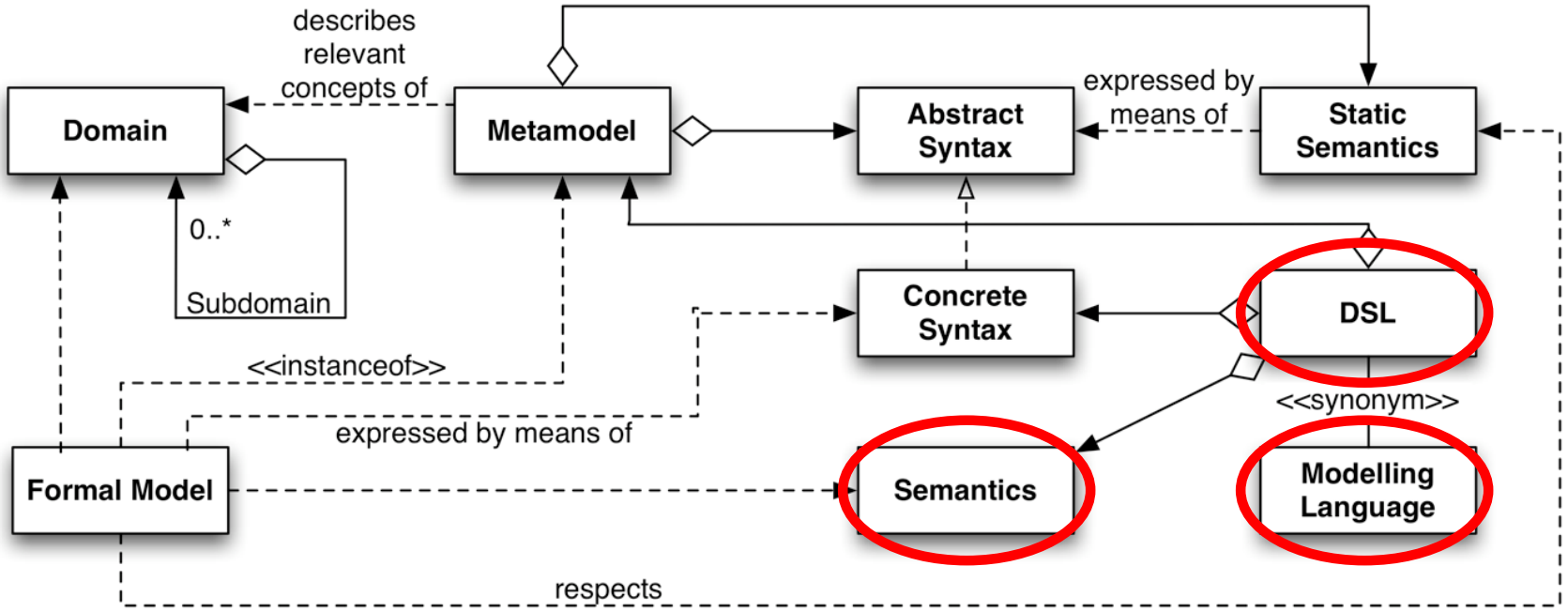
Domain Specific Languages



static semantics:

- specify constraints for well-formedness of a model
- defined for the abstract syntax
- e.g.: „No cycles in the inheritence relation allowed.“

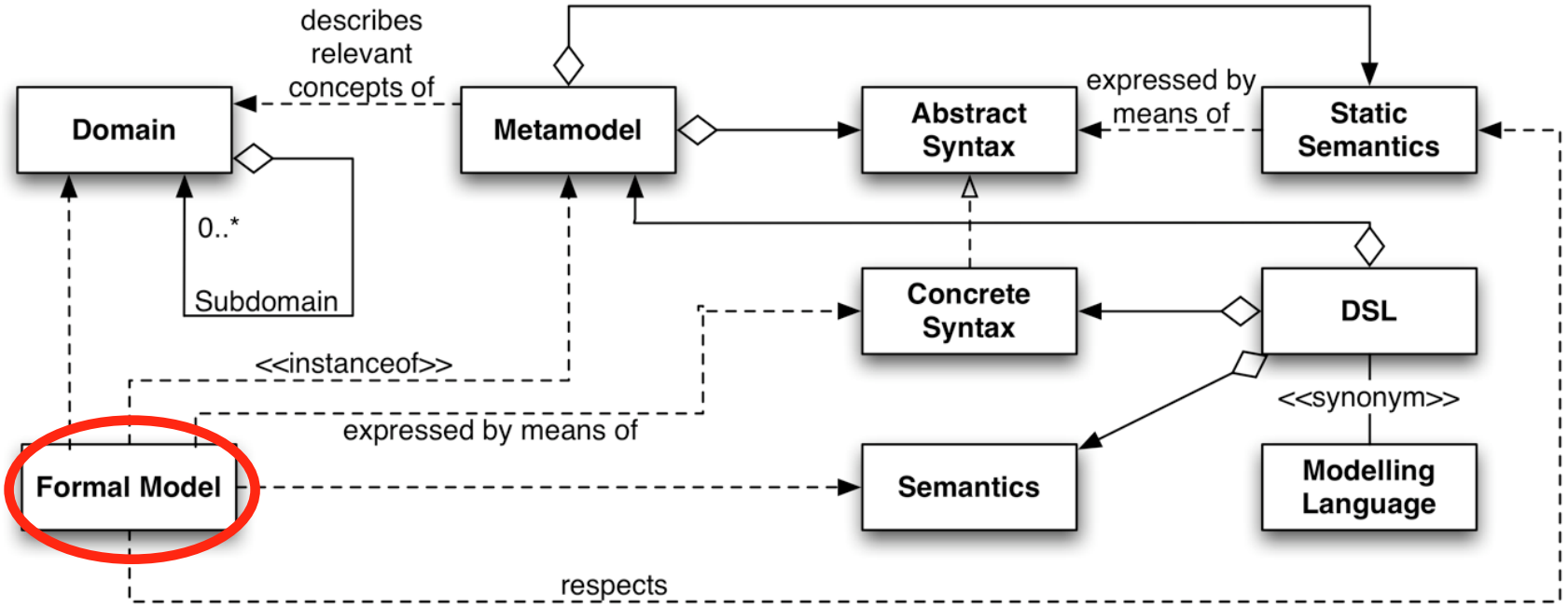
Domain Specific Languages



domain specific language:

- modelling/programming language for a domain
- consists of: metamodel & concrete syntax
- plus: (dynamic) *semantics*, describing the meaning of all language elements

Domain Specific Languages

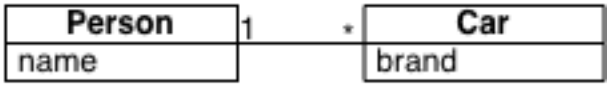


formal model:

- model/program written in a particular DSL
- can be automatically translated to code (interpretation/code generation)

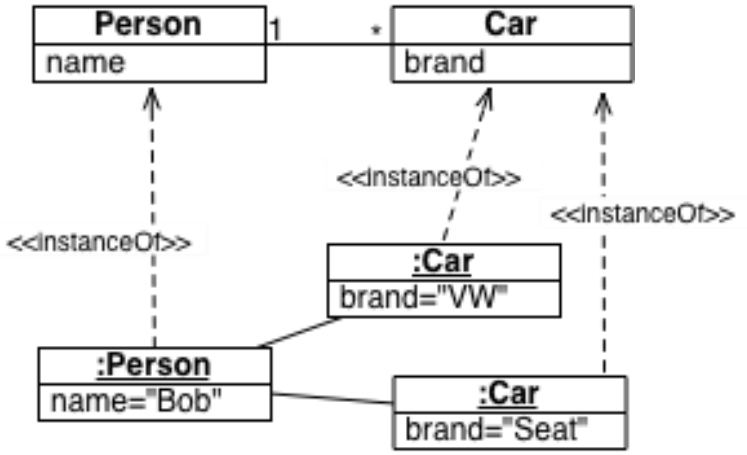
Domain Specific Languages

Metamodeling Example 1: UML (1.x)



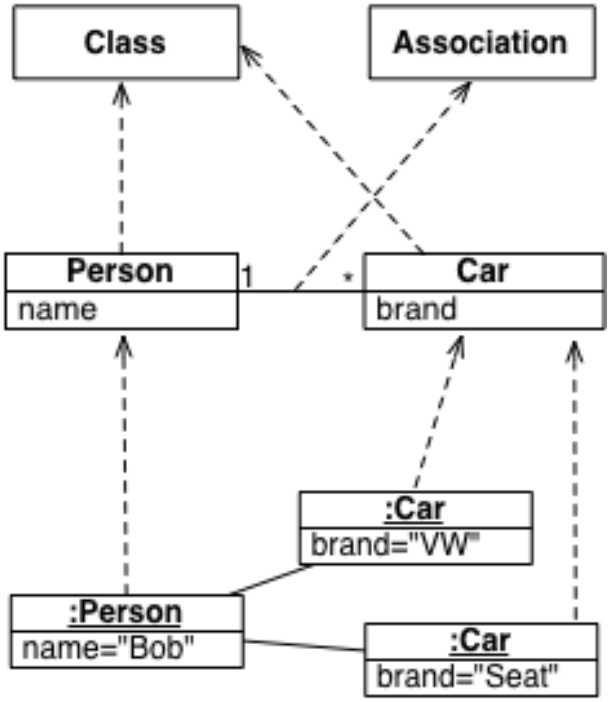
Domain Specific Languages

Metamodeling Example 1: UML (1.x)



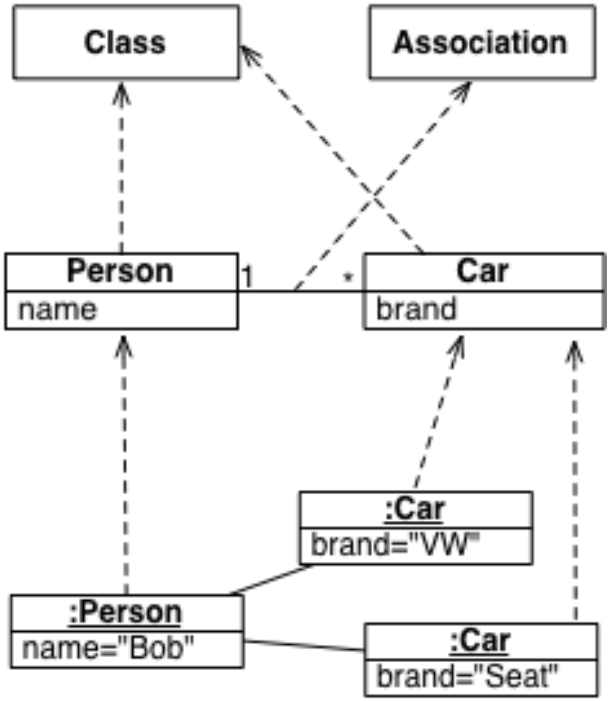
Domain Specific Languages

Metamodeling Example 1: UML (1.x)



Domain Specific Languages

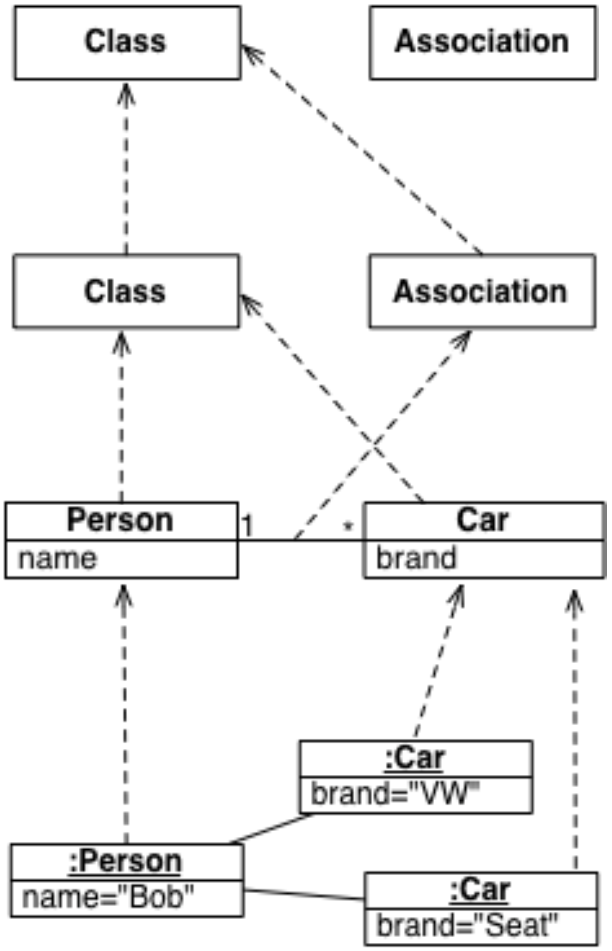
Metamodeling Example 1: UML (1.x)



More levels?

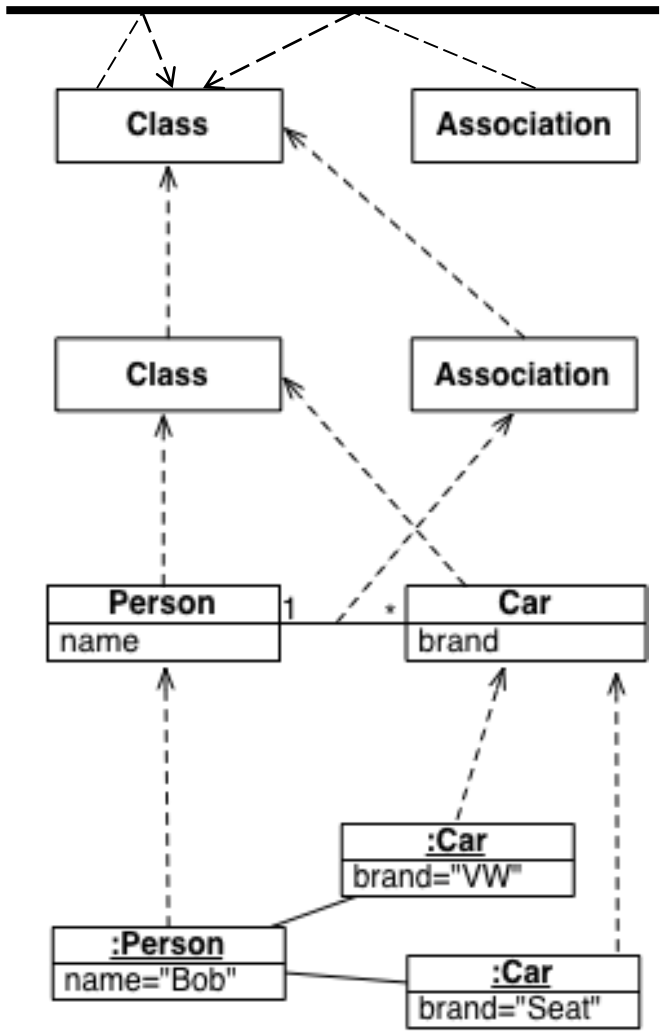
Domain Specific Languages

Metamodeling Example 1: UML (1.x)



Domain Specific Languages

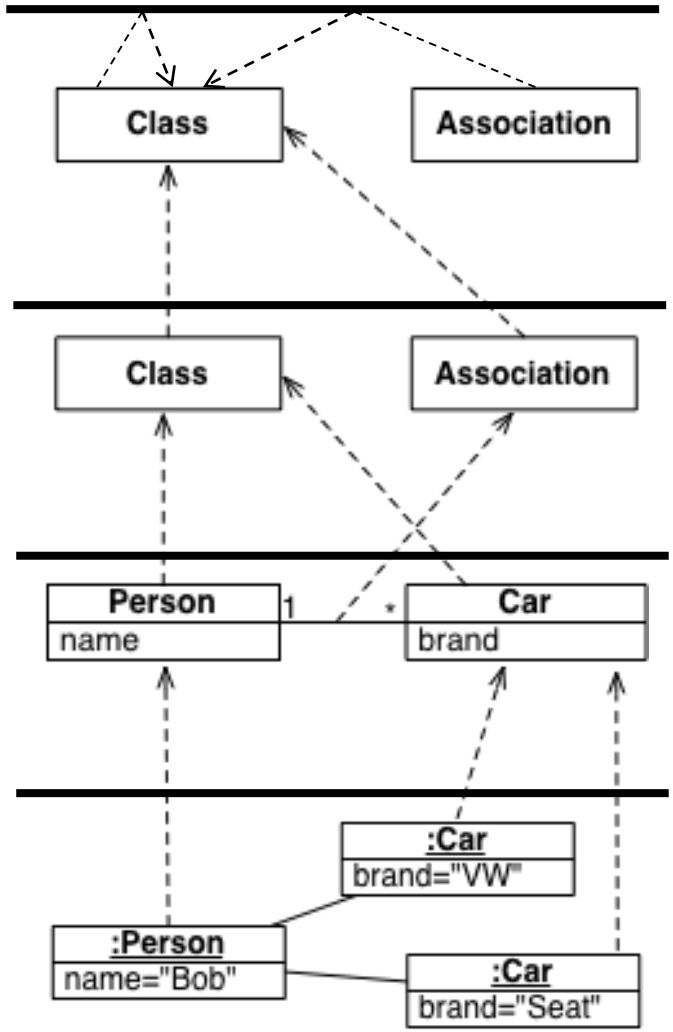
Metamodeling Example 1: UML (1.x)



„reflection“

Domain Specific Languages

Metamodeling Example 1: UML (1.x)



„reflection“

Meta-Object Facility (MOF)

M3: Metametamodel

UML 1.x

M2: Metamodel

Concrete Diagram

M1: Model

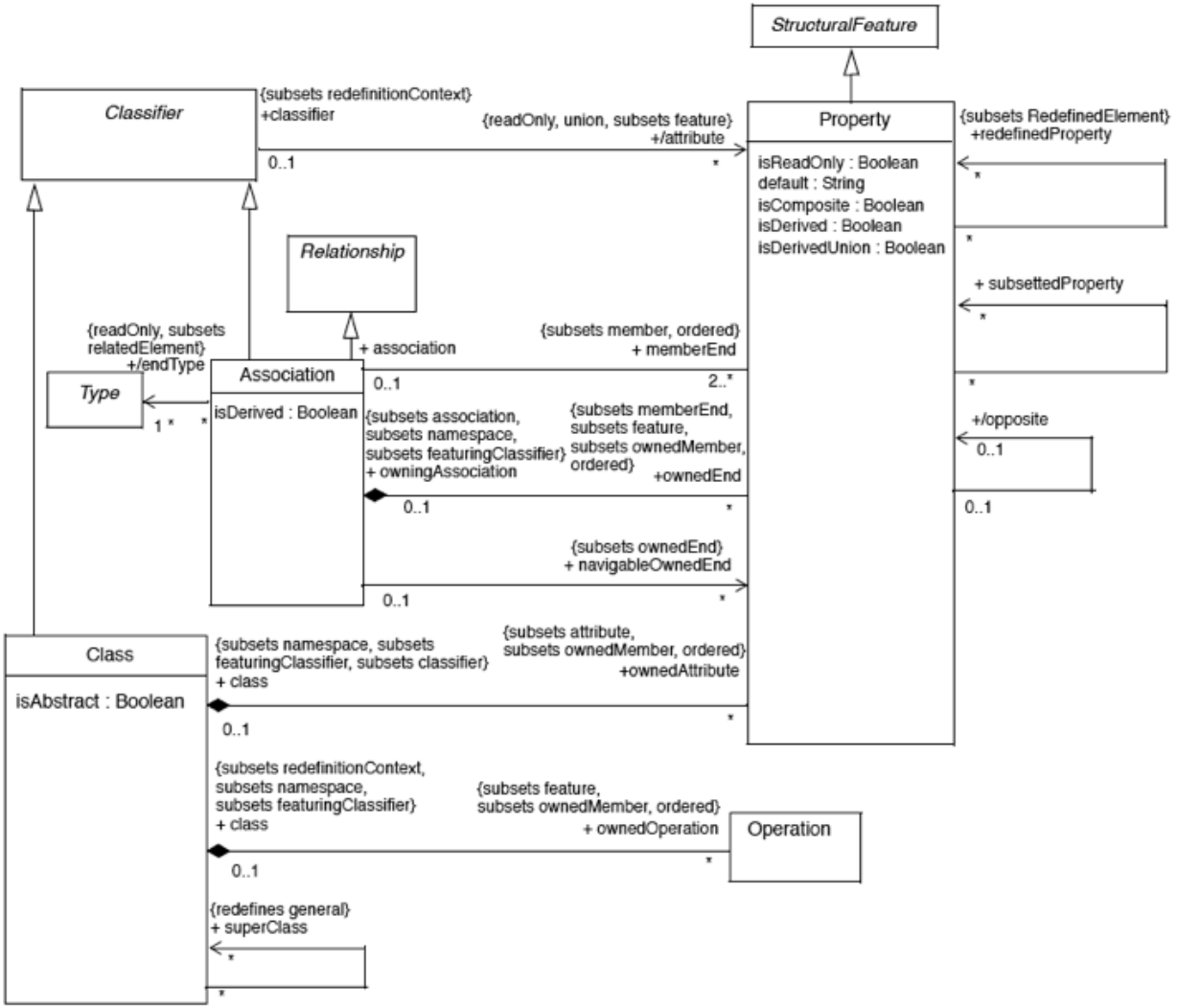
Concrete Objects

M0: Model Instance

Domain Specific Languages

Part of the UML specification [UMLSpec] :

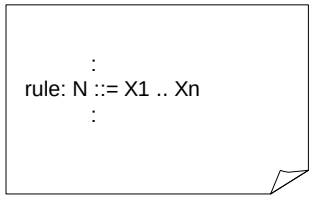
Abstract Syntax



Domain Specific Languages

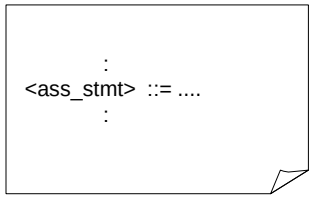
Metamodeling Example 2: Java

(E)BNF



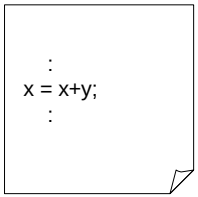
M3: Metametamodel

Java grammar



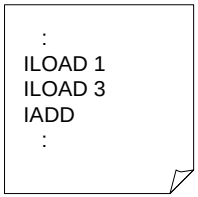
M2: Metamodel

Java Program



M1: Model

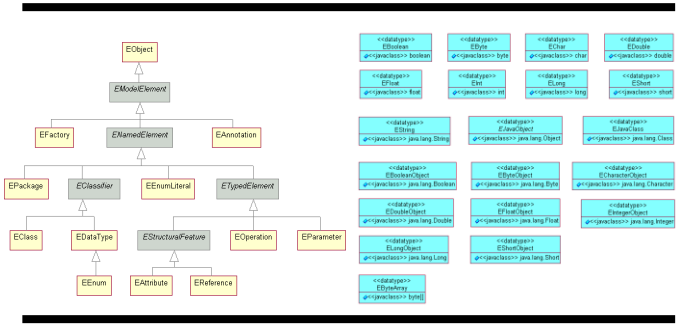
JVM Class File



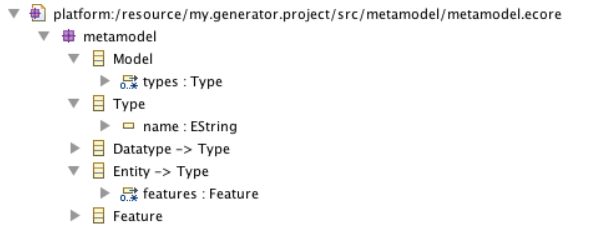
M0: Model Instance

Domain Specific Languages

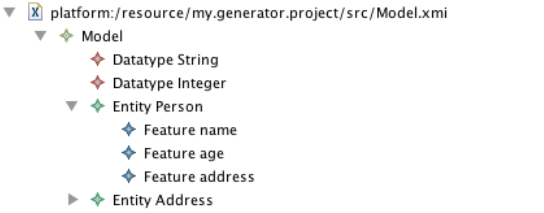
Metamodeling Example 3: Eclipse Modeling Framework [EMF]



Metametamodel



Metamodel



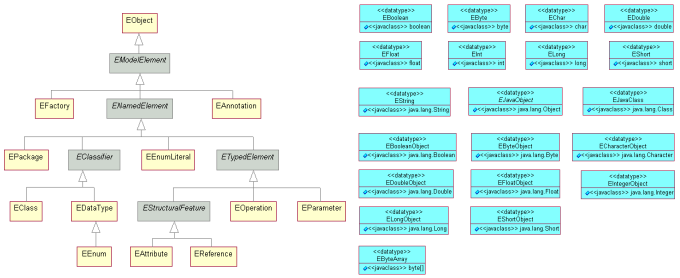
Model

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

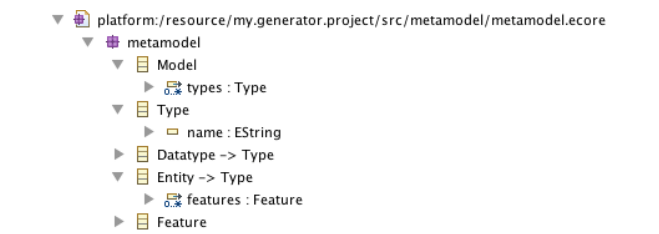
Domain Specific Languages

Metamodeling Example 3: Eclipse Modeling Framework

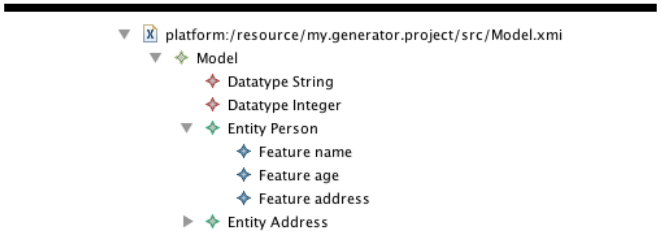


Metametamodel

„My Data Models“



Metamodel



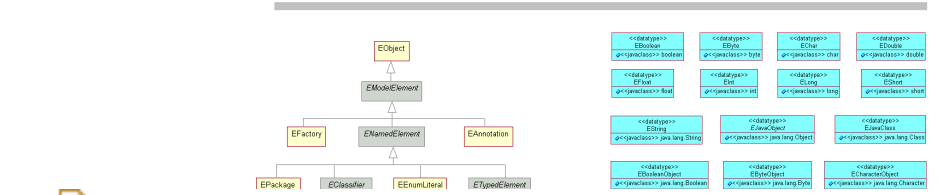
Model

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

Domain Specific Languages

Metamodeling Example 3: Eclipse Modeling Framework



Metametamodel

platform:/resource/my.generator.project/src/metamodel/metamodel.ecore

- metamodel
  - Model
    - types : Type
  - Type
    - name : EString
  - Datatype -> Type
  - Entity -> Type
    - features : Feature
  - Feature

lel

Model

- Entity Person
  - Feature name
  - Feature age
  - Feature address
- Entity Address

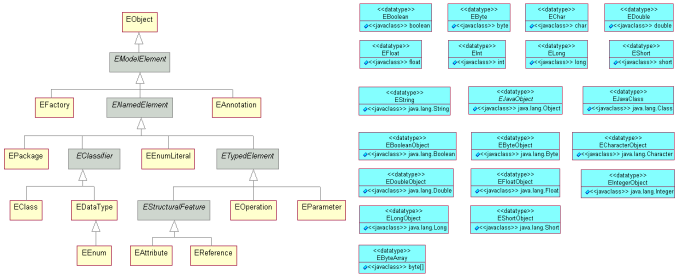
```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

Domain Specific Languages

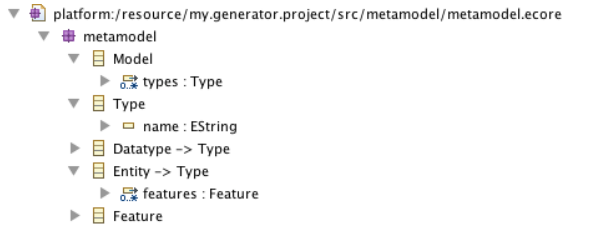
Metamodeling Example 3: Eclipse Modeling Framework

Ecore

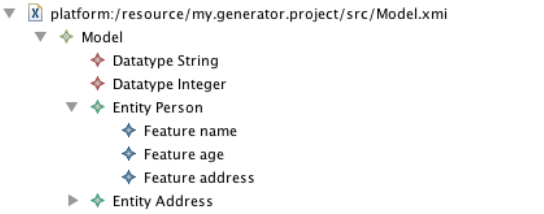


Metametamodel

„My Data Models“



Metamodel



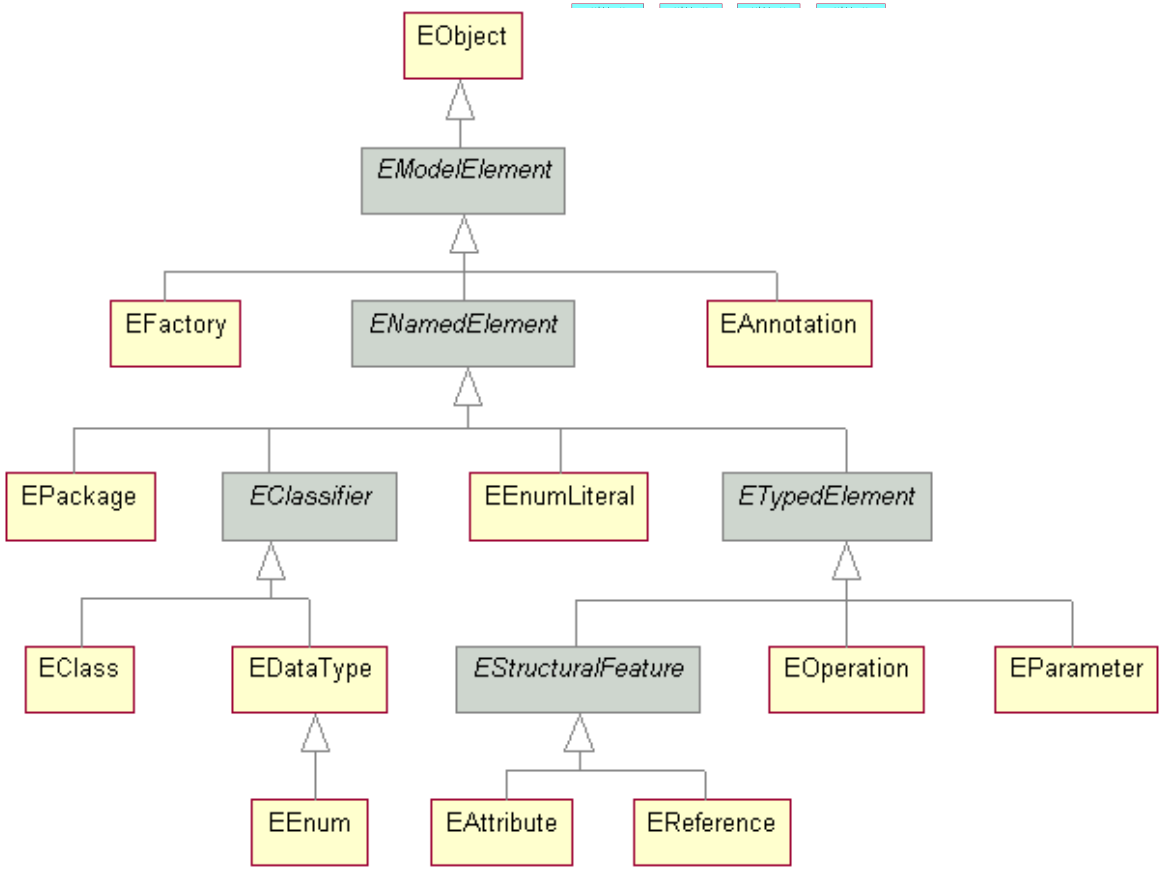
Model

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

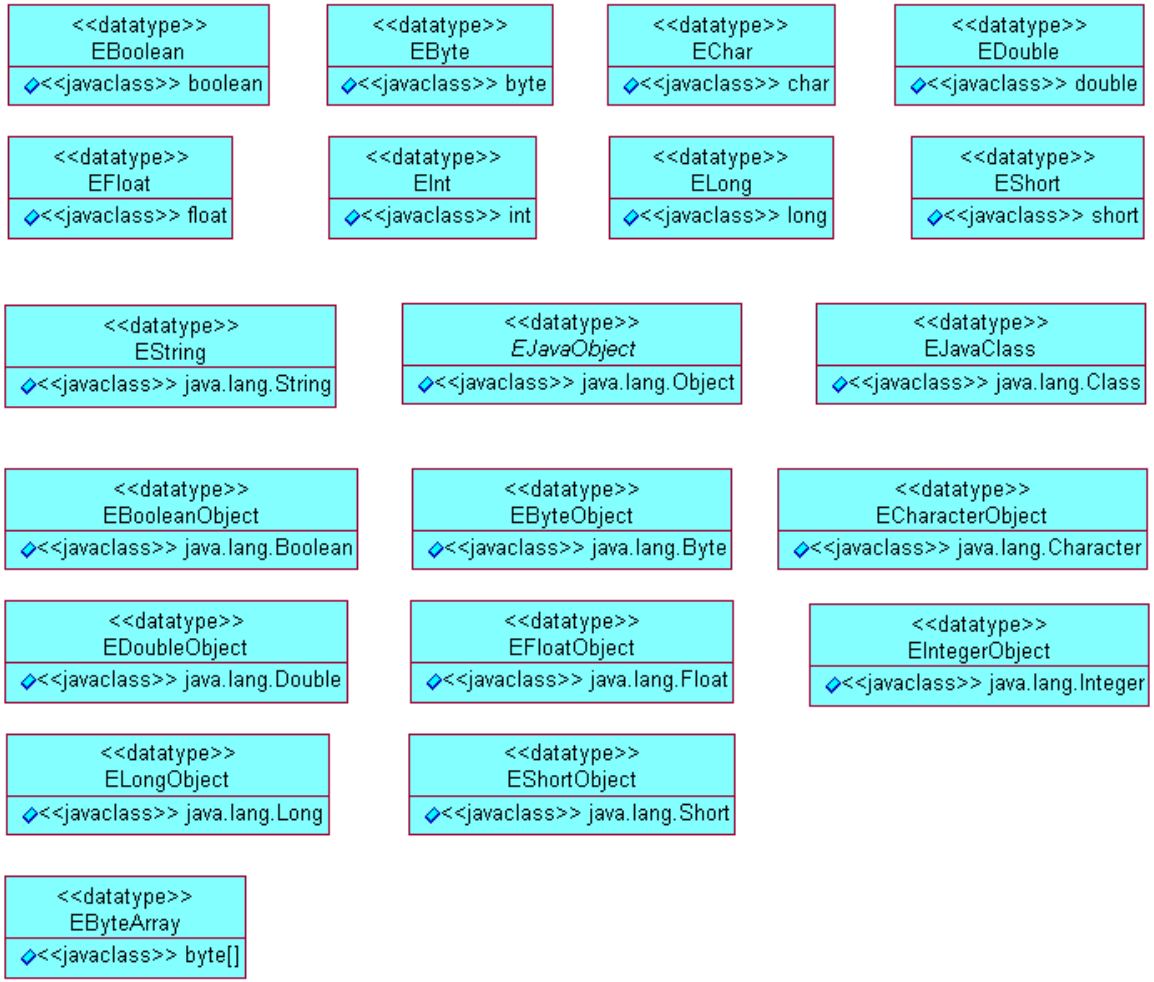
Domain Specific Languages

Metamodeling Example 3: Eclipse Modeling Framework



Domain Specific Languages

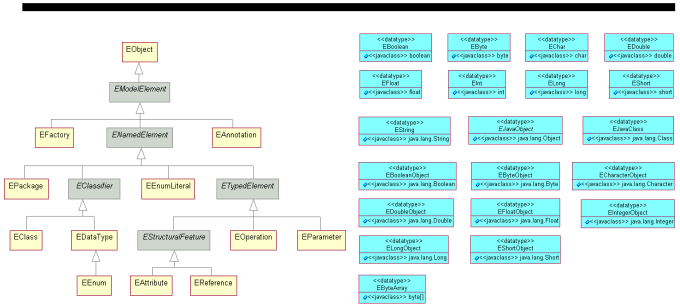
Metamodeling Example 3: Eclipse Modeling Framework



Domain Specific Languages

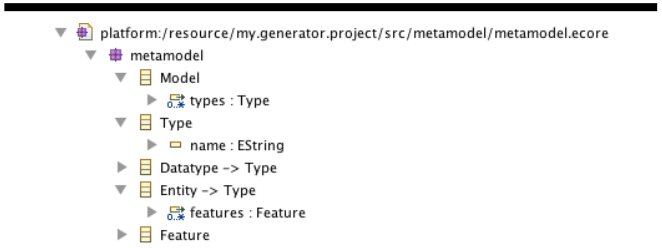
Metamodeling Example 3: Eclipse Modeling Framework

Ecore



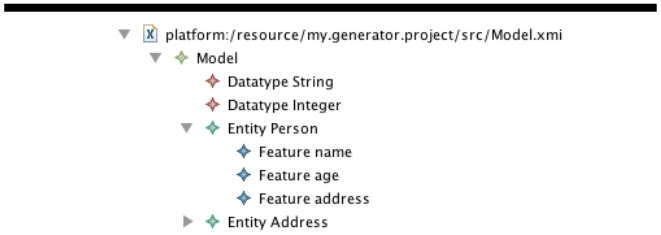
Metametamodel

„My Data Models“



Metamodel

Concrete “Data Model“



Model

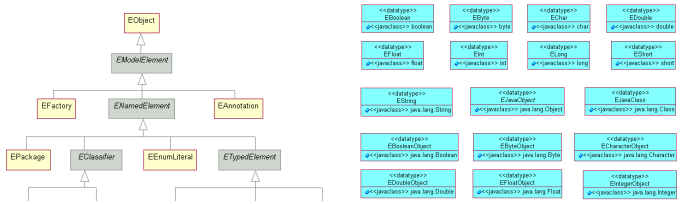
```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

Domain Specific Languages

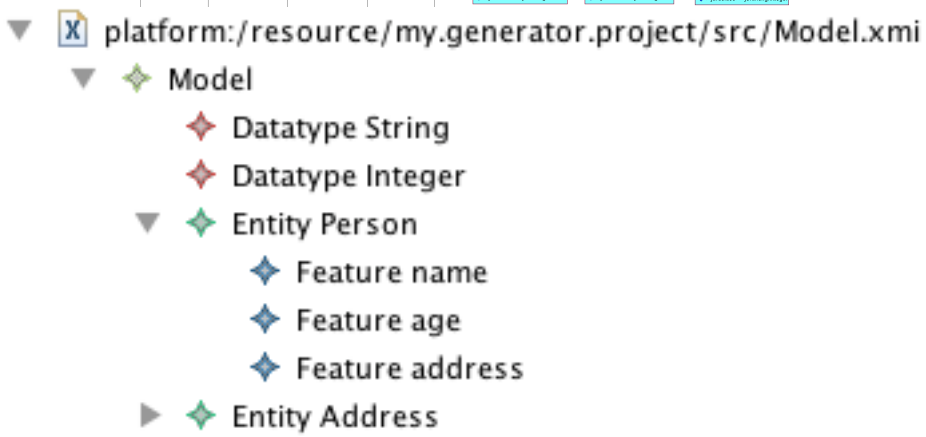
Metamodeling Example 3: Eclipse Modeling Framework

Ecore



Metametamodel

„My Data Models“



tamodel

Concrete „Data Model“

Model

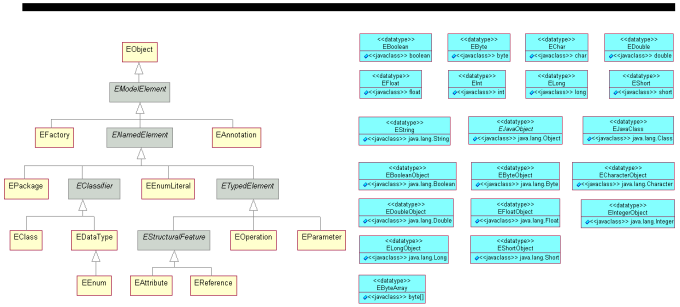
```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Model Instance

Domain Specific Languages

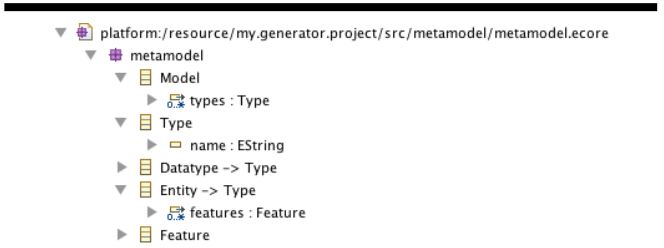
Metamodeling Example 3: Eclipse Modeling Framework

Ecore



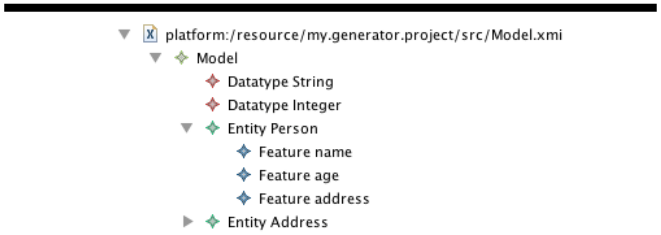
Metametamodel

„My Data Models“



Metamodel

Concrete „Data Model“



Model

Concrete Objects

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

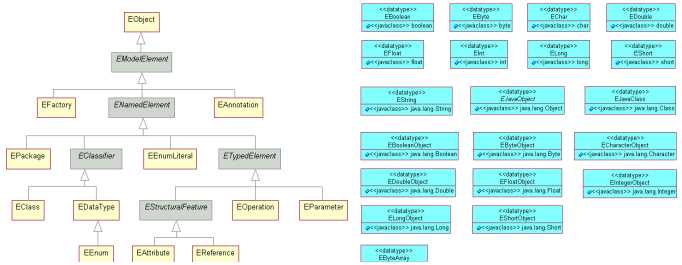
Model Instance

Domain Specific Languages

Metamodeling Example 3: Eclipse Modeling Framework

Ecore

Metametamodel



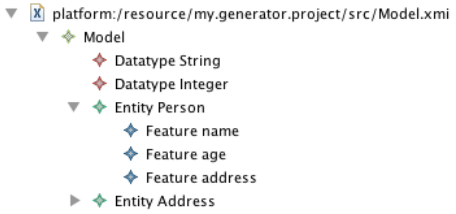
„My Data Models“

Metamodel

```
Person bob = new Person();
bob.setName("Bob");
bob.setAge(35);
Address address = new Address();
address.setStreet("Mulholland Drive");
address.setZip("8079");
address.setCity("Los Angeles");
```

Concrete „Data Model“

Model

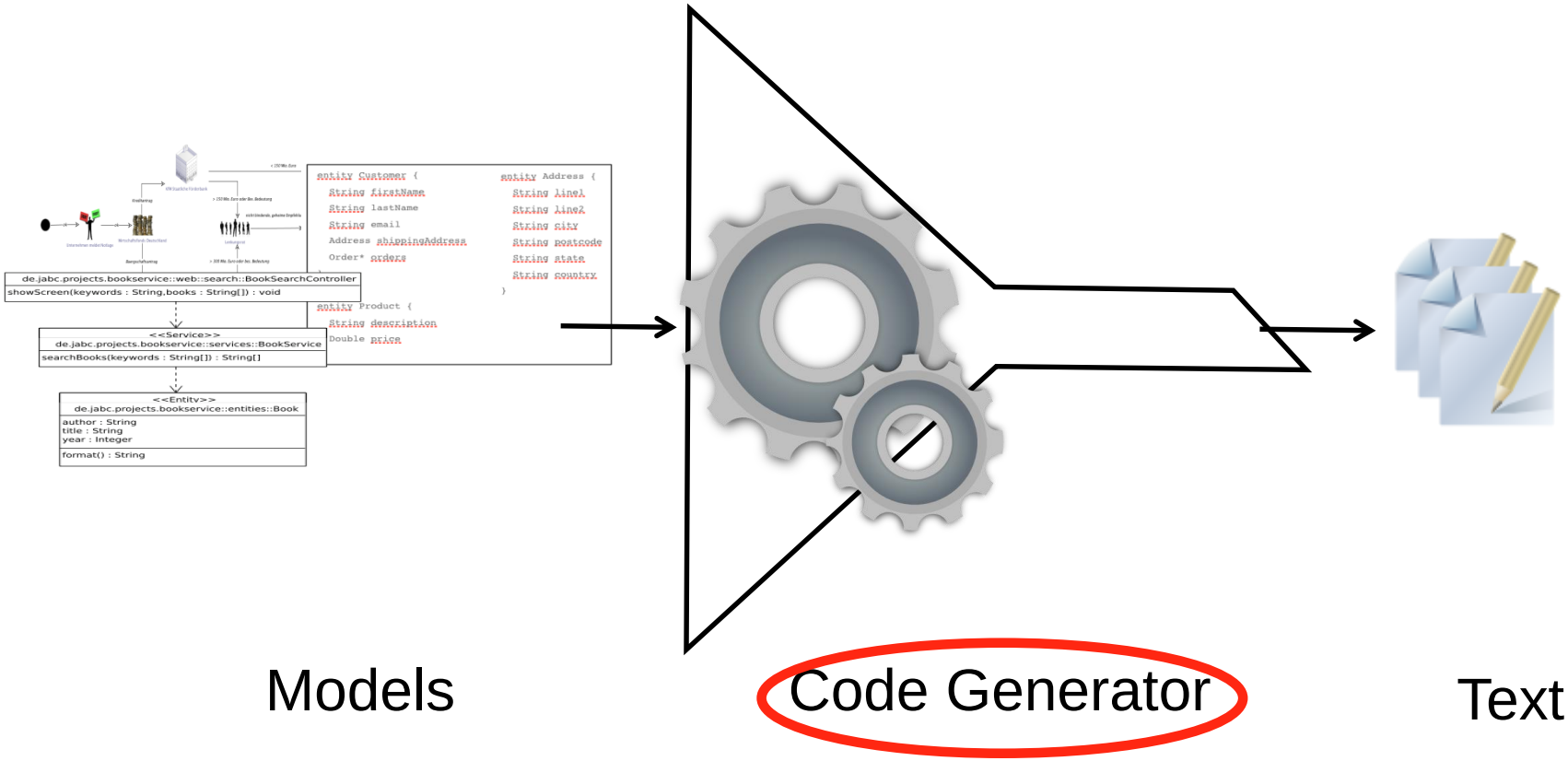


Concrete Objects

Model Instance

Code Generation

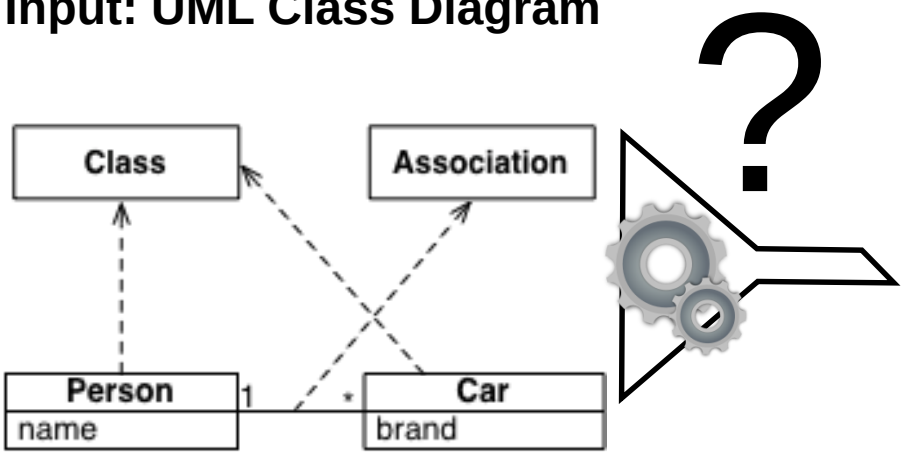
Code Generation: Big Picture



Code Generation

Code Generation Example

Input: UML Class Diagram



?

Desired Output: POJOs*

```
public class Person {
   private String name;

   public String getName() {
      return name;
   }

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

```
public class Car {
   private String brand;

   public String getBrand() {
      return brand;
   }

   public void setBrand(String brand) {
      this.brand = brand;
   }
}
```

*Plain Old Java Objects

Code Generation – Naive: String Concatenation

```
public static void generatePojo(UmlModel classDiagram) {
 // iterate all classes in diagram
 for (UmlClass aClass: classDiagram.getClasses()) {
  StringBuilder builder = new StringBuilder();
  builder.append("public class ").append(aClass.getName()).append(" {\n");

  // iterate all attributes of the class
  for (UmlAttribute anAttribute: aCl
   String attributeName = anAttrib
   String attributeType = anAttribute.getType();

   // write member variable
   builder.append("\tprivate ").append(attributeType).append(" ").append(attributeName).append(";\n");

   // write getter method
   builder.append("\tpublic ").append(attributeType).append(" get")
                              .append(capitalizeFirstLetter(attributeName)).append("() {\n");
   builder.append("\t\treturn ").append(attributeName).append(";\n");
   builder.append("}");

   // write setter method
   builder.append("\tpublic void set").append(capitalizeFirstLetter(attributeName)).append("(")
                              .append(attributeType).append(" ").append(attributeName).append(") {\n");
   builder.append("\t\tthis.").append(attributeName).append(" = ").append(attributeName).append(";\n");
   builder.append("}");
 }

 builder.append("}");

 // write content to file
 [...]
}
}
```

Code Generator works on elements of the *metamodel*!

Code Generation – Naive: String Concatenation

```
public static void generatePojo(UmlModel classDiagram) {
 // iterate all classes in diagram
 for (UmlClass aClass: classDiagram.getClasses()) {
  StringBuilder builder = new StringBuilder();
  builder.append("public class ").append(aClass.getName()).append(" {\n");

  // iterate all attributes of the class
  for (UmlAttribute anAttribute: aClass.getAttributes()) {
   String attributeName = anAttribute.getName();
   String attributeType = anAttribute.getType();

   // write member variable
   builder.append("\tprivate ").append(attributeType).append(" ").appen

   // write getter method
   builder.append("\tpublic ").append(attributeType).append(" get")
                              .append(capitalizeFirstLetter(attributeName)).ap
   builder.append("\t\treturn ").append(attributeName).append(";\n");
   builder.append("}");

   // write setter method
   builder.append("\tpublic void set").append(capitalizeFirstLetter(attributeName)).append("(")
                              .append(attributeType).append(" ").append(attributeName).append(") {\n");
   builder.append("\t\tthis.").append(attributeName).append(" = ").append(attributeName).append(";\n");
   builder.append("}");
 }

 builder.append("}");

 // write content to file
 [...]
}
}
```

Code Generator consists of:

Code Generation – Naive: String Concatenation

```
public static void generatePojo(UmlModel classDiagram) {
  // iterate all classes in diagram
  for (UmlClass aClass: classDiagram.getClasses()) {
    StringBuilder builder = new StringBuilder();
    builder.append("public class ").append(aClass.getName()).append(" {\n");

    // iterate all attributes of the class
    for (UmlAttribute anAttribute: aClass.getAttributes()) {
      String attributeName = anAttribute.getName();
      String attributeType = anAttribute.getType();

      // write member variable
      builder.append("\tprivate ").append(attributeType).append(" ").appen

      // write getter method
      builder.append("\tpublic ").append(attributeType).append(" get")
                                .append(capitalizeFirstLetter(attributeName)).ap
      builder.append("\t\treturn ").append(attributeName).append(";\n");
      builder.append("}");

      // write setter method
      builder.append("\tpublic void set").append(capitalizeFirstLetter(attributeName)).append("(")
                                .append(attributeType).append(" ").append(attributeName).append(") {\n");
      builder.append("\t\tthis.").append(attributeName).append(" = ").append(attributeName).append(";\n");
      builder.append("}");
    }

    builder.append("}");

    // write content to file
    [...]
  }
}
```

Code Generator consists of:

1. *generation logic*

Code Generation – Naive: String Concatenation

```
public static void generatePojo(UmlModel classDiagram) {
 // iterate all classes in diagram
 for (UmlClass aClass: classDiagram.getClasses()) {
  StringBuilder builder = new StringBuilder();
  builder.append("public class ").append(aClass.getName()).append(" {\n");

  // iterate all attributes of the class
  for (UmlAttribute anAttribute: aClass.getAttributes()) {
   String attributeName = anAttribute.getName();
   String attributeType = anAttribute.getType();

   // write member variable
   builder.append("\tprivate ").append(attributeType).append(" ").appen

   // write getter method
   builder.append("\tpublic ").append(attributeType).append(" get")
                              .append(capitalizeFirstLetter(attributeName)).ap
   builder.append("\t\treturn ").append(attributeName).append(";\n");
   builder.append("}");

   // write setter method
   builder.append("\tpublic void set").append(capitalizeFirstLetter(attributeName)).append("(")
                              .append(attributeType).append(" ").append(attributeName).append(") {\n");
   builder.append("\t\tthis.").append(attributeName).append(" = ").append(attributeName).append(";\n");
   builder.append("}");
 }

 builder.append("}");

 // write content to file
 [...]
}
}
```

Code Generator consists of:

1. generation logic
2. *output description*

Code Generation

Question: What is the Problem with the Naive Solution?

Answer: Mixing of generation logic & output description.

- ➞ final output is hard to recognize (output is “embedded“ in generation logic)
- ➞ bad maintainability
- ➞ contains uninteresting details (string buffer handling, escape sequences, ...)

Goal: Separation of Concerns

- ➞ separate generation logic from output description as much as possible

Code Generation

Template Engines

- output description is performed in ***templates***
- template:
  - static text + dynamic parts
  - e.g. form letter

To:¶
<Title> <First Name> <Last Name>¶
<Address>¶
<State/County>¶
<Post Code>¶
<Country>¶

5 November 2007¶

Dear <Title> <Last Name>,¶

Thank you very much for your participation in our "Points" promotion. We are pleased to inform you that you have earned <Points> this year.¶

Your loyalty to our company is greatly appreciated and we hope to be of continuing service in the future.¶

Yours sincerely¶

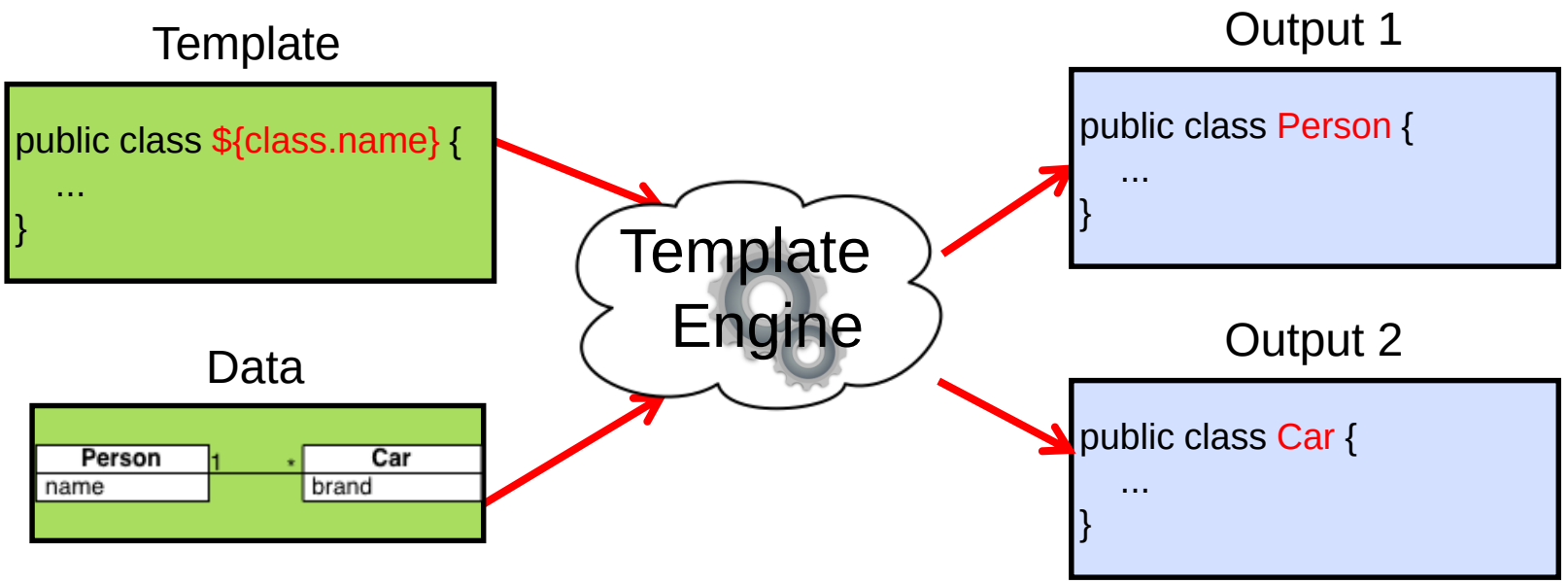
Code Generation

Template Engines

- output description is performed in ***templates***
- template:
  - static text + dynamic parts
  - e.g. form letter
- often - but by far not only! - used for web development
- code generators which use this mechanism are called ***template-based***

Code Generation

Template Engines (2) - Basic Approach



Code Generation

Template Engines (3) - Some Implementations

Template Engine	Implemented in	Remarks
ASP.net	C#, VB.net	
Apache Velocity	Java, C#	
CheetahTempate	Python	
FreeMarker	Java	
JavaServer Pages (JSP)	Java	
Smarty	PHP	
StringTemplate	Java, C#, Phyton	
Xpand	Java	used by openArchitectureWare -
XSL Transformation (XSLT)	e.g. Java (Saxon, Xalan), .NET (Saxon), C++ (Xalan), ...	language-independent *specification*

Code Generation

Template Engines (4) - Example Template [Velocity]

```
#if (!$class.packageName.equals(""))
package $class.packageName;
#end

public class $class.name
#if ($class.generalization)
extends $class.generalization.fullyQualifiedName
#end
{

#foreach ($attribute in $class.attributes)
private $attribute.getterSetterTypeName $attribute.name;

$attribute.visibility $attribute.getterSetterTypeName $attribute.getterName()
{
     return this.${attribute.name};
}

[...]
#end
}
```

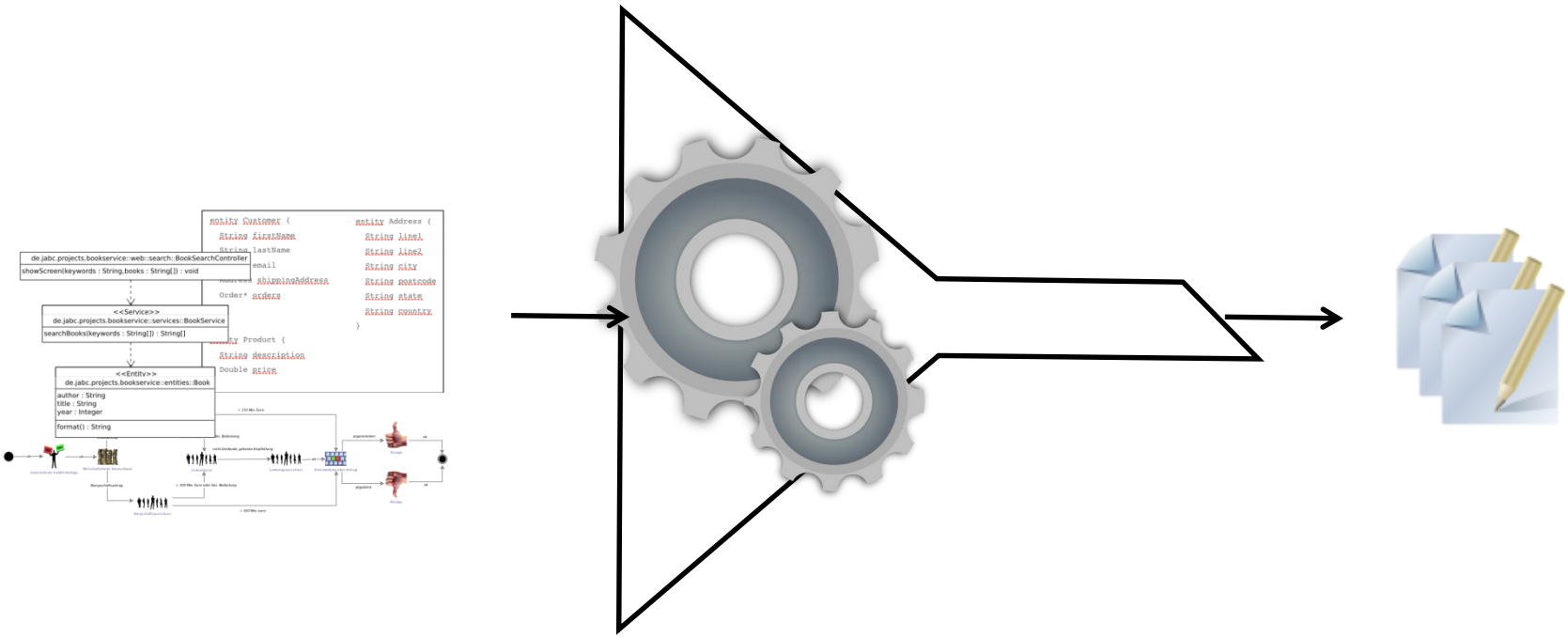
Code Generation

Template Engines (5) - Example Template [FreeMarker]

```
<#if class.packageName?length > 0>
package ${class.packageName};
</#if>

public class ${class.name}
<#if class.generalization?has_content>
extends ${class.generalization.fullyQualifiedName}
</#if>
{

<#list attribute as class.attributes)
private ${attribute.getterSetterTypeName} ${attribute.name};

${attribute.visibility}
            ${attribute.getterSetterTypeName} ${attribute.getterName}()
{
     return this.${attribute.name};
}

[...]
</#list>
}
```

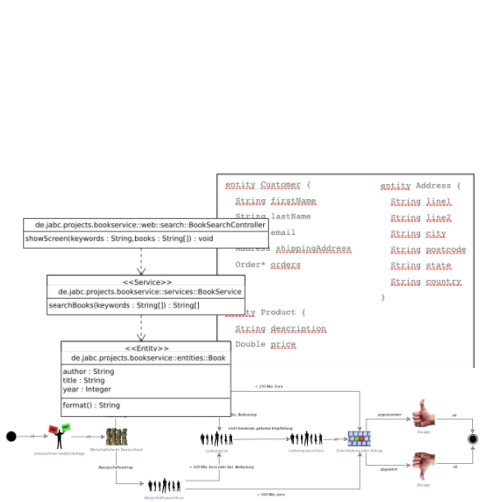
Code Generation

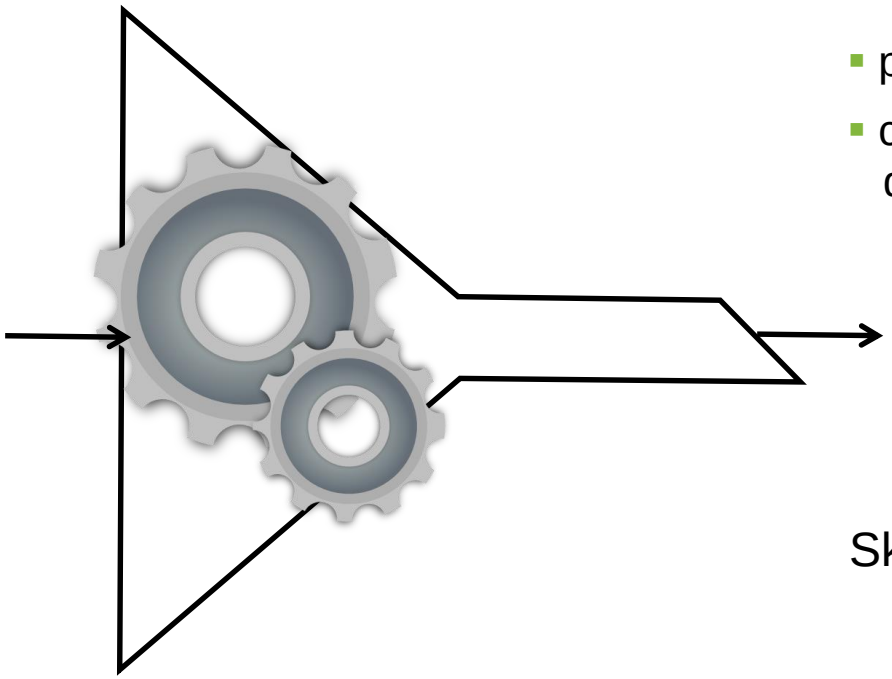
Code Generation: Big Picture



Code Generation

Skeleton/Stub Generation





- partial code
- completed by developer

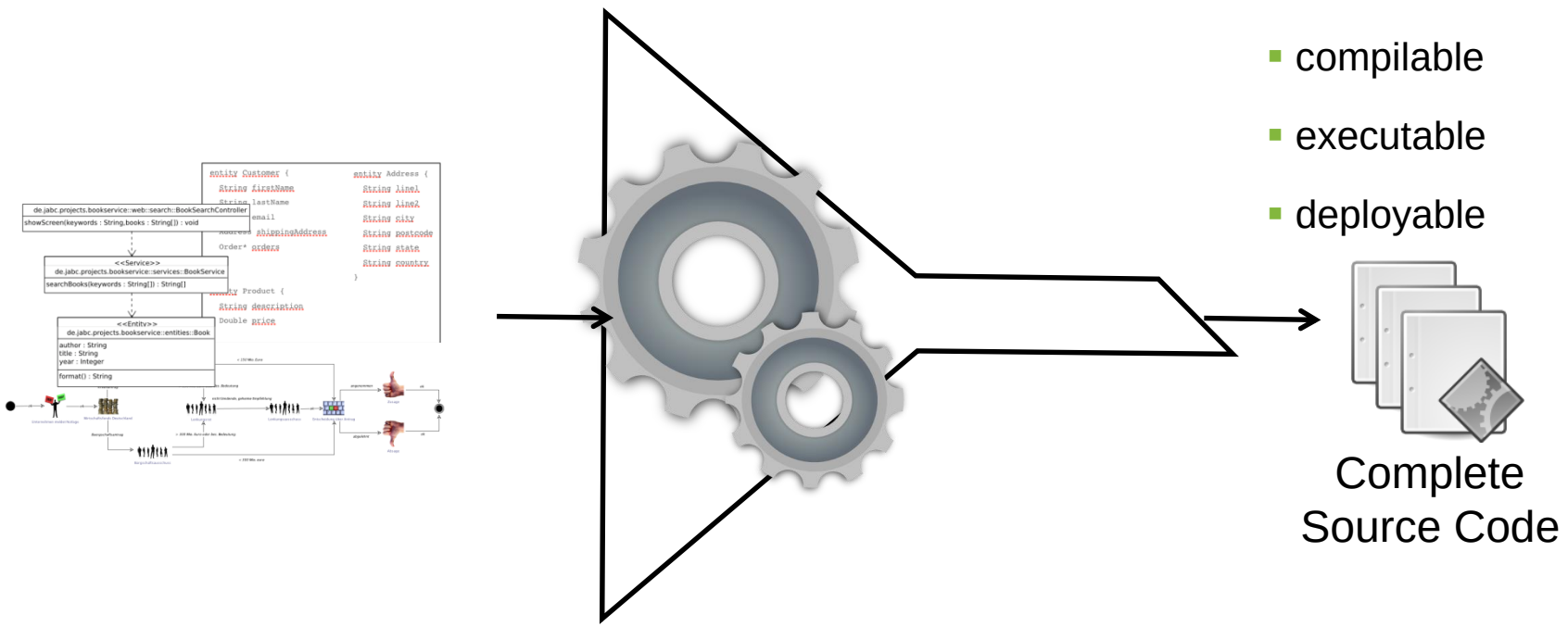
Code Skeletons/Stubs

„roundtrip-engineering“

- changes allowed on model & code ➞ synchronization required (difficult)
- code generation more challenging (Overwrite? Protected regions?)

Code Generation

Full Code Generation



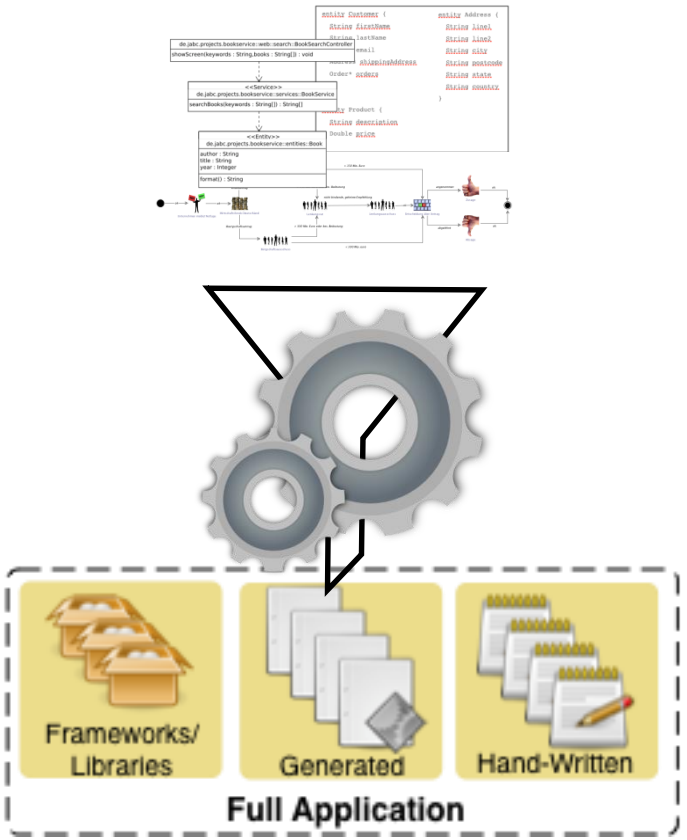
- changes only allowed on the model
- generated code is a by-product that must not be modified

Code Generation

Full Code Generation (2)

see also: [DSM]

- full code ≠ full application
- application consists of:
  - dependencies (frameworks, libraries), e.g. EJB, log4j, ...
  - generated code produced by code generator(s)
  - hand-written code (e.g. custom data objects, config files, ...)



References

- [MDSD] Thomas Stahl, Markus Völter, Sven Efftinge, Arno Haase: *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*, dpunkt.verlag, 2. Auflage, 2007
- [UMLSpec] Object Management Group: *Unified Modeling Language, Infrastructure, v2*.2: http://www.omg.org/cgi-bin/doc?formal/09-02-04
- [EMF] Eclipse Modeling Framework: http://www.eclipse.org/modeling/emf/
- [Velocity] Apache Velocity: http://velocity.apache.org/
- [FreeMarker] FreeMarker: http://freemarker.org/
- [DSM] Kelly, S., Tolvanen, J.P.: *Domain-Specific Modeling: Enabling Full Code Generation*, John Wiley & Sons, 2008.