Model Transformation and Code Generation

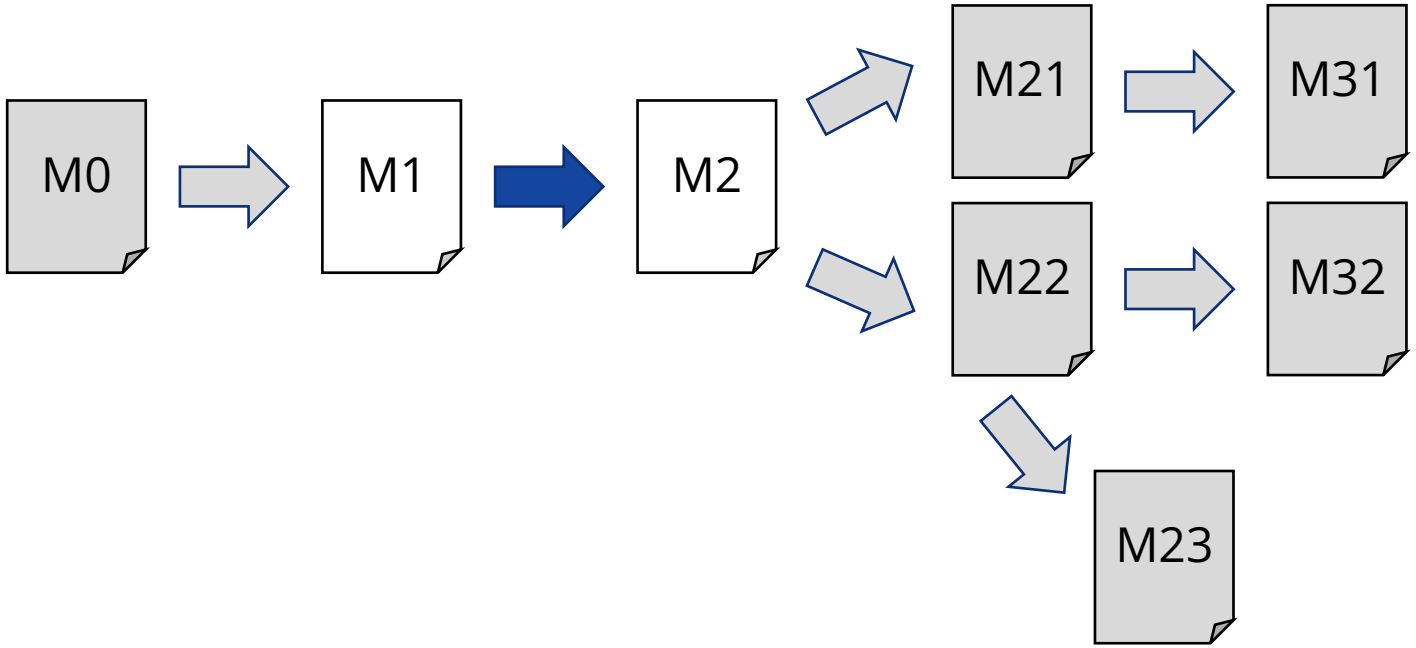
MŰEGYETEM 1782

Budapest University of Technology and Economics
Department of Measurement and Information Systems
ftsrg Research Group

ftsrg

Motivation: Transformation of models

- **Model-based development:** Models as primary documents
- Developing models, automating model processing



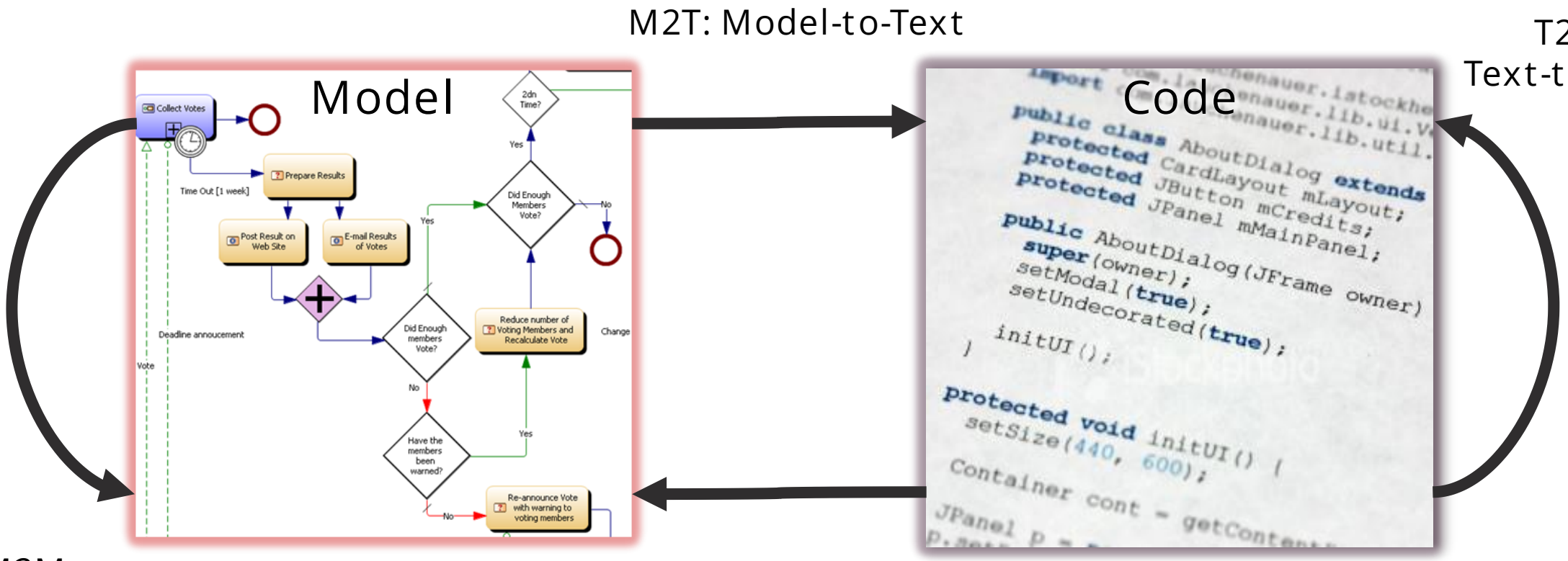
- **Goal:** to efficiently formulate and implement transformations

Agenda

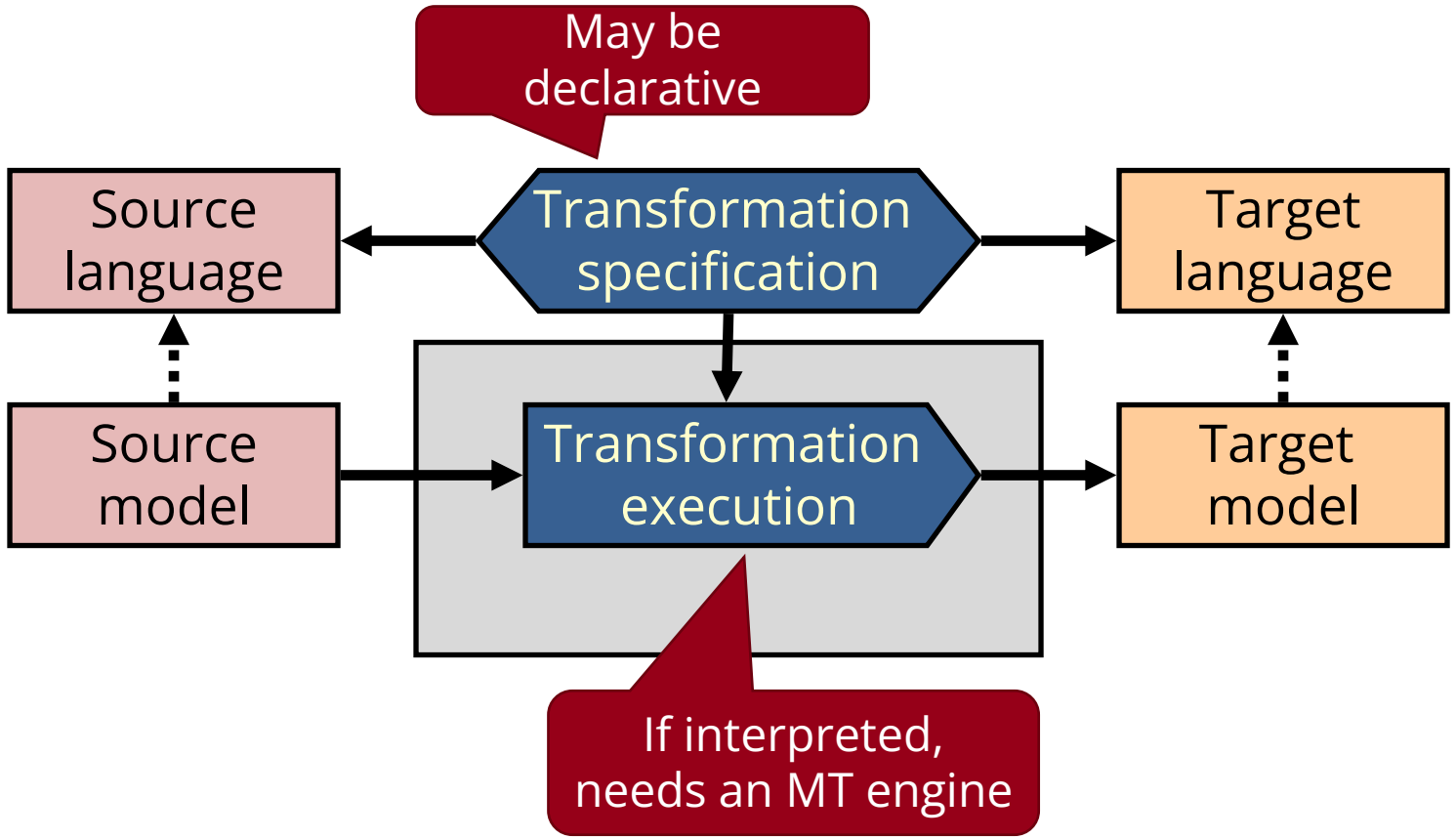
- Model Transformation overview
- Code Generation in general
- Approaches
- Advanced Text Generation Issues
- Example template languages
- JET, Velocity, Xpand and XTend

Model Transformation Overview

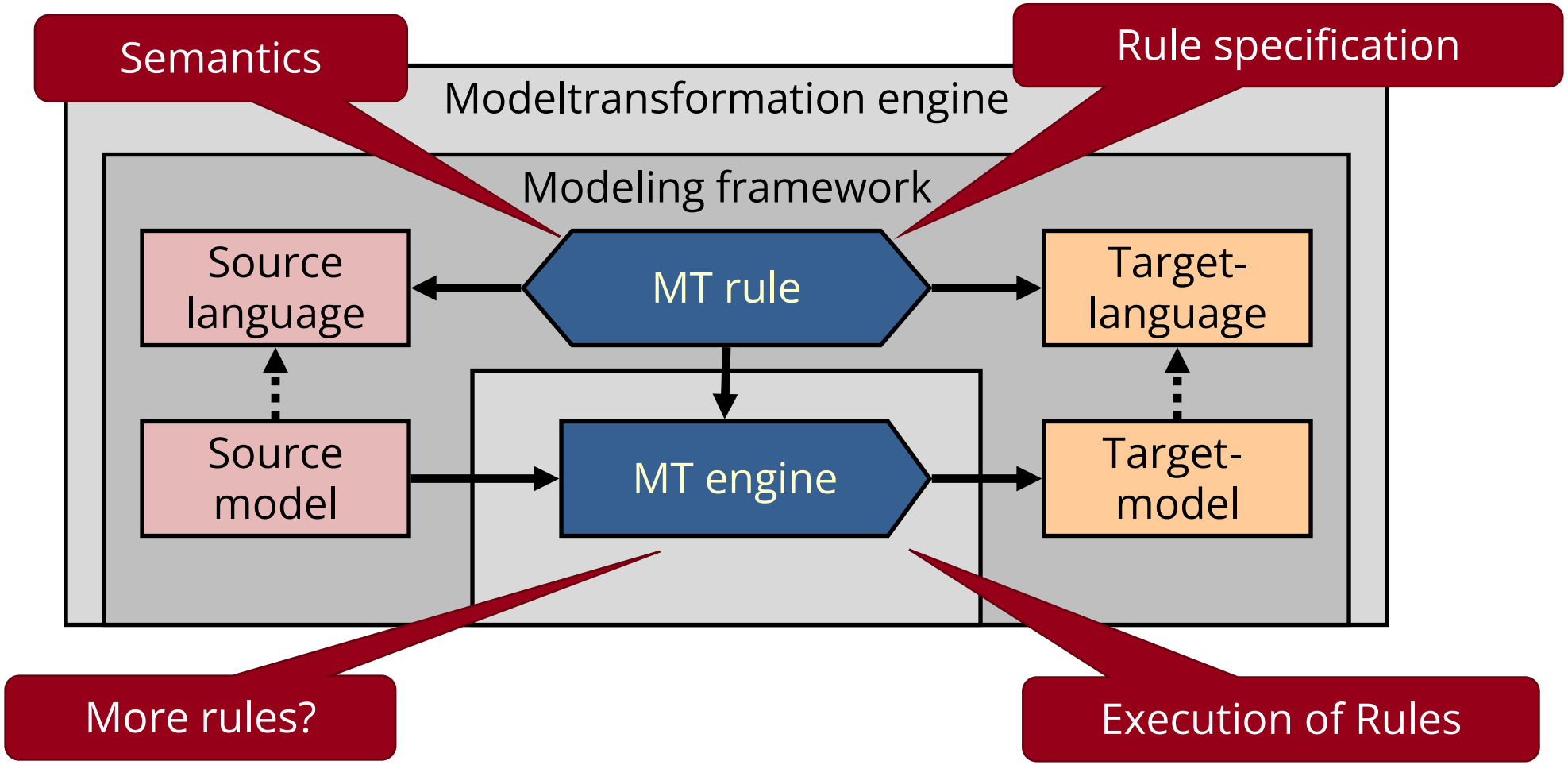
Types of transformation



Definition of Model Transformation

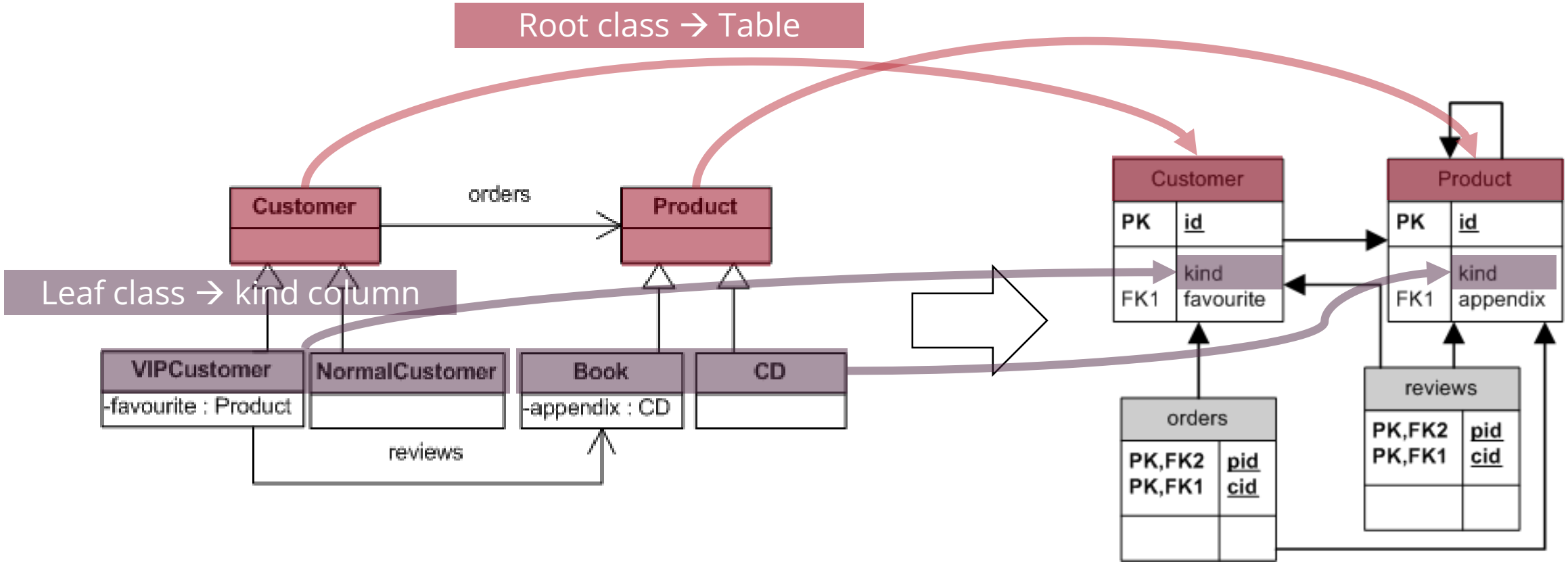


Defintions and Questions



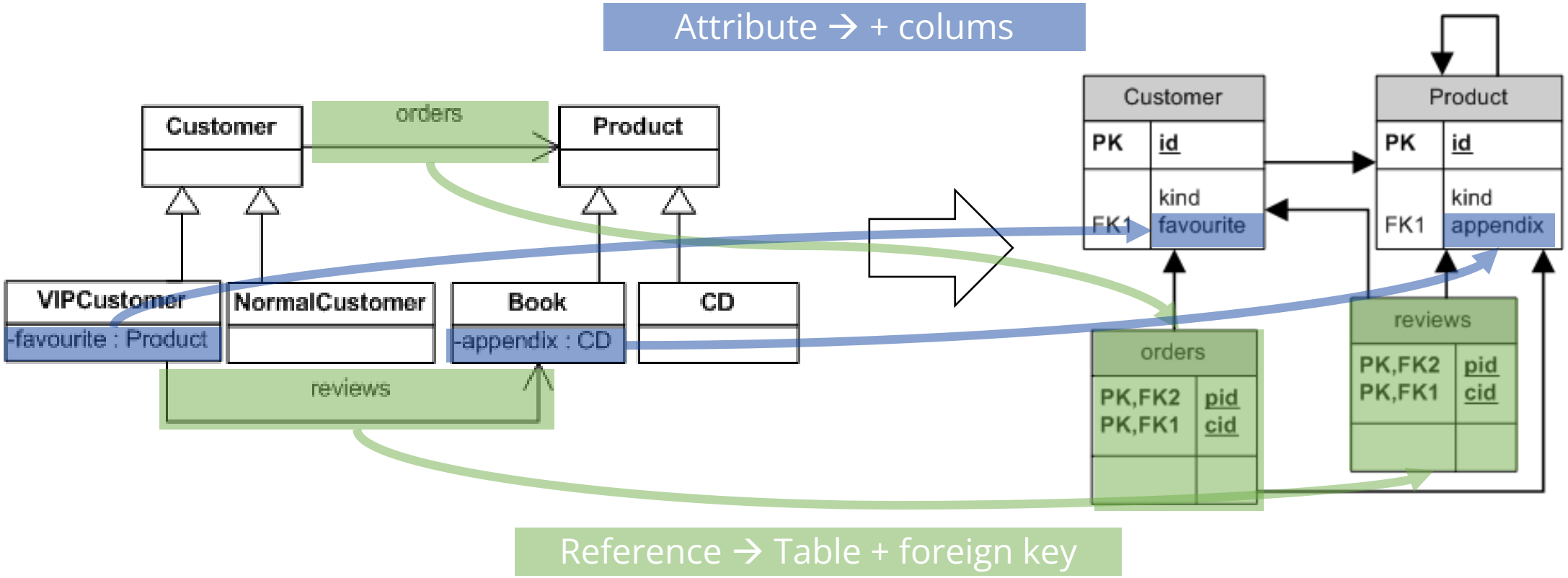
Example: Object-relational mapping

- Typical example: map a class diagram to database tables!



Example: Object-relational mapping

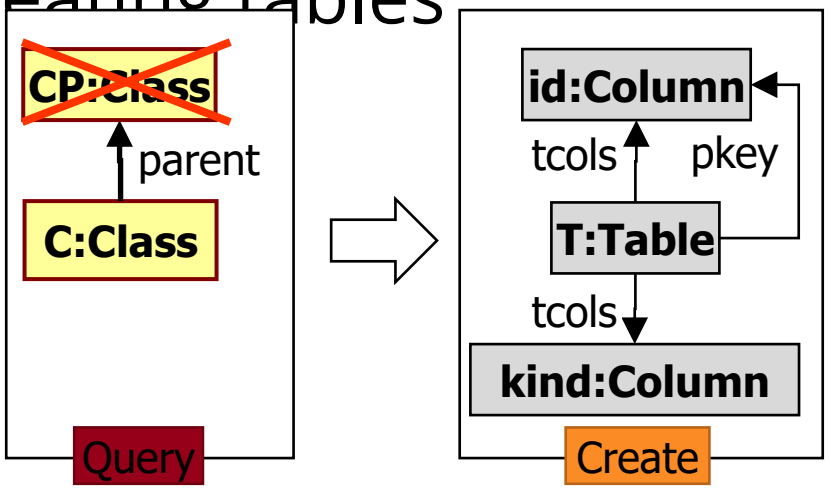
- Typical example: map a class diagram to database tables!



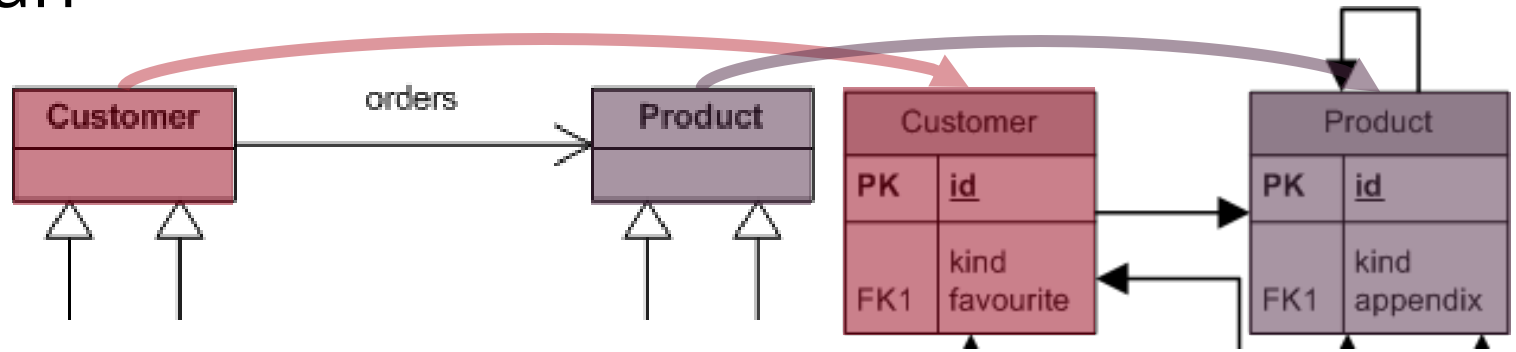
Example Transformation

- How would we solve the problem of creating tables representing root classes?

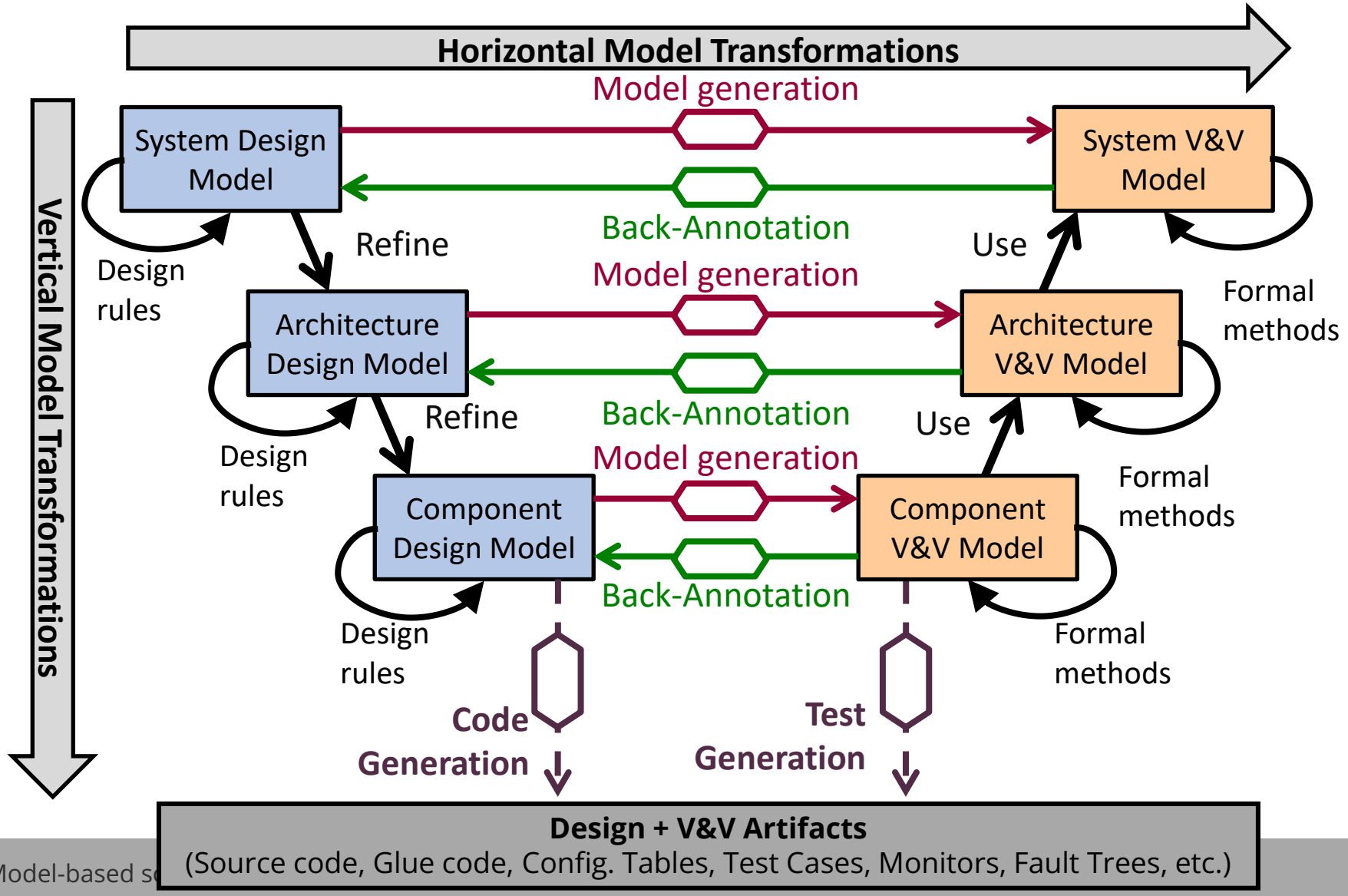
1. Query the root classes (class that has no ancestor)
2. Create the tables and with them the necessary columns
3. Repeat as long as we can



- Goal: To formulate the whole transformatio with similar rules



Models and Transformations in Critical Systems



Model Transformations:

- systematic foundation of knowledge transfer:

Theoretical results➔tools

- bridge / integrate existing languages&tools

Textual Model Transformation

Motivation for Code Generation

Why?

- Automate development steps!
- Use our **models/requirements/plans** to derive…
  - Documentation
  - Source code
  - Configuration descriptors
  - Communication messages
- **But:** don't generate for generation's sake
  - Some things are better left to the target language
  - Lifting knowledge to a model → does it add value?

  Conciseness, usability, analysis/validation, multiple targets…

Example setup

Domain-specific Model

Code generation

High-level language

Compile

Assembly

Code generation vs Compilers

- Mapping between abstraction levels
  - e.g., From C to assembly
- Usage of design patterns
  - e.g., function calls in C
- Many similarities, NOT a strict separation
  - pl. C++ templates, automatically generated ctor+dtor
- Prediction:

yesterday's design pattern → today's code generation feature → tomorrow's language element

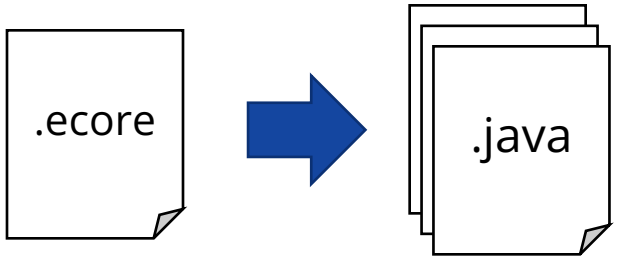
- Domain-specific instead of universal languages

Code generation vs Interpreters

Code generator

- Loads the model
- Explores the model
- Outputs an artifact that describes the behavior of the model.

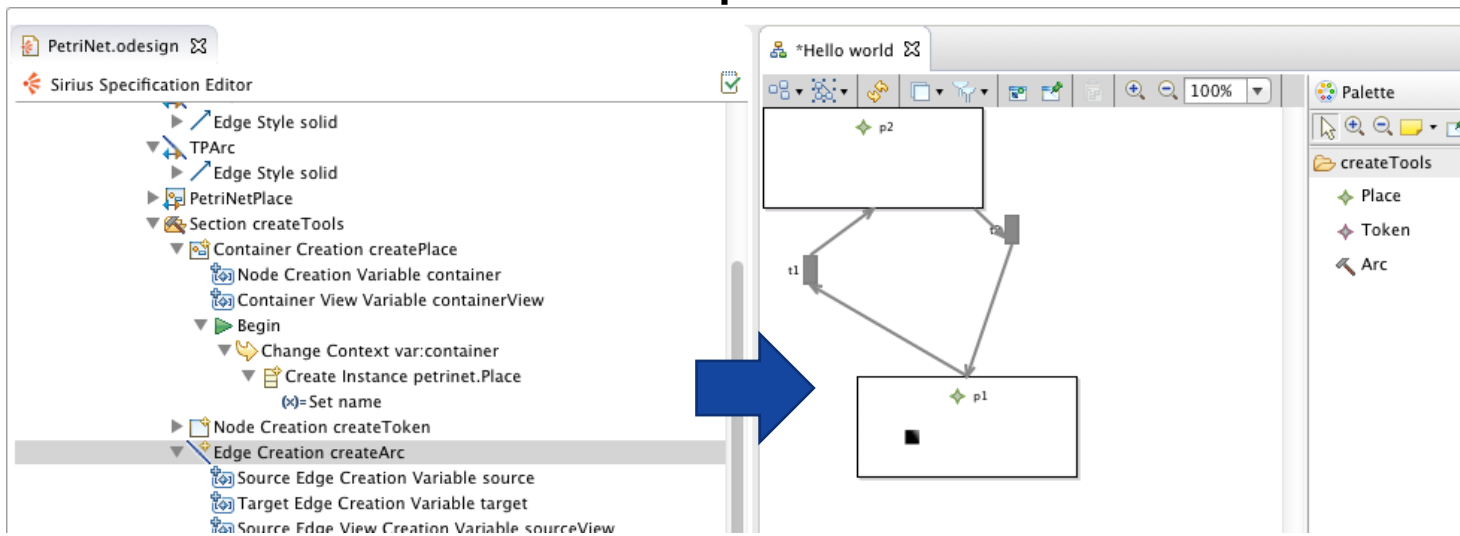
Example: EMF



Dynamic Interpreter

- Loads the model
- Behavior is based on the content of the model

Example: Sirius



Code generation vs Interpreters

Code generator

- Start after generation & compilation only
- May pre-optimize for efficient runtime code
- With or without runtime dependencies
- Model change requires re-generation
- Code may be manually modified (is this good?)

Dynamic Interpreter

- Quick startup, short feedback
- Typically introduces overhead (time, mem)
- Runtime dependency on interpreter
- May support changing model during execution
- Always adheres exactly to model

Conceptual approaches

Approaches

- Dedicated
  - Specific, ad-hoc
  - Using a dedicated code generator
- Template-based
- Serializer-based

Specific, ad-hoc

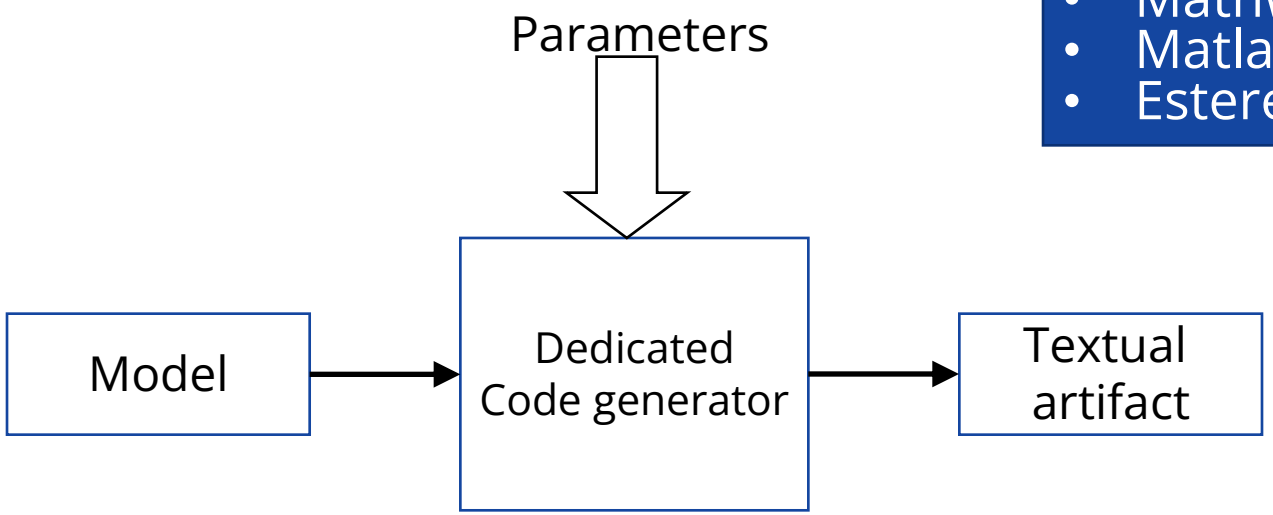
Designed for the specific problem domain

- Best performance
- Quick and dirty, Long development, Hard maintainability
- Zero reusability
- Dedicated problem domains:
  - Minimal changes during support cycle (safety critical embedded system, defense)
  - Certifiability
- **Example:** ARINC653 Multistatic configuration generator (python)

```
sourceFile.write("   temp = ((AIDA_PARTITION_TYPE*) selfModule.partitions.elements);\n" )
i = 0
for partition in partitions:
 numPorts = getNumberOfAllCommPorts_Partition(currModuleComm, interPartitionComm, partition.partitionName)
 sourceFile.write("   temp[" + str(i) + "].partition_id = " + str(partition.partitionID) + ";\n" )
 sourceFile.write("   strcpy( &temp[" + str(i) + "].partition_name[0], \"" + str(partition.partitionName) + "\");\n")
 sourceFile.write("   temp[" + str(i) + "].ports.type = CONST_AIDA_PORTS_TYPE;\n")
 sourceFile.write("   temp[" + str(i) + "].ports.elements = &mem_ports_" + str(partition.partitionName) + "[0];\n")
 sourceFile.write("   temp[" + str(i) + "].ports.numOfElements = " + str(numPorts) + ";\n")
 sourceFile.write("\n")
 i = i + 1
## end for
sourceFile.write("\n")
```

Dedicated code generator

Examples:
- IBM Rational Software Architect
- VASP (DO-178B Level A)
  Display graphics in avionics
- Mathworks
- Matlab Simulink
- Esterel Scade suite

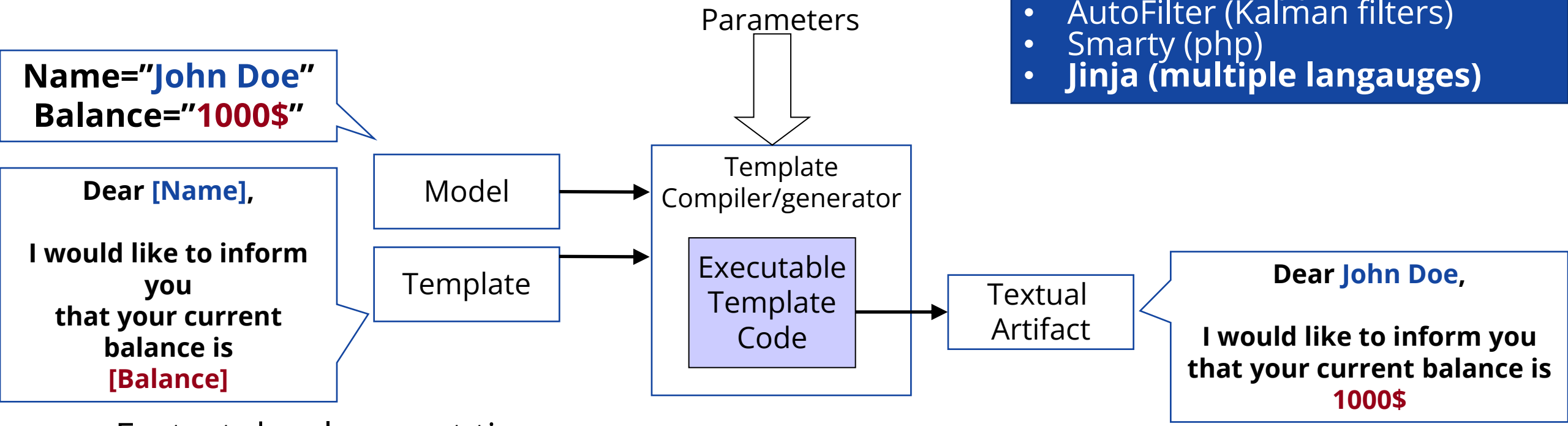


Based on a framework:

- Faster development time
- Slower performance, better reusability
- Embedded systems, moderate changes during project lifecycle

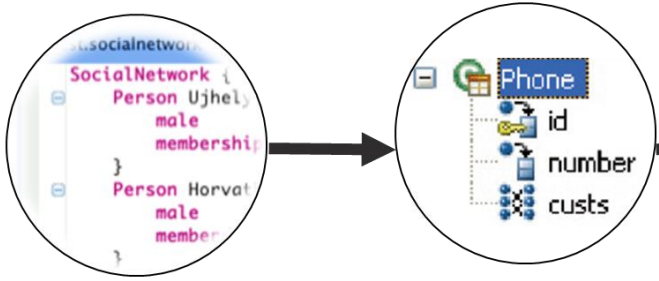
Template based generator

Examples:
- JET (for EMF models)
- Velocity (/JSP)
- Xtend (MDD approach)
- AutoFilter (Kalman filters)
- Smarty (php)
- **Jinja (multiple langauges)**



- Fastest development time
- „Slowest" performance, highest reusability
- Fast changing environments (e.g., web based technologies)
- Complex changes during project lifecycle
- Models and templates can be changed independently

Serializer-based generator



Textual syntax

Idea: do not generate code, but DOM/AST instead

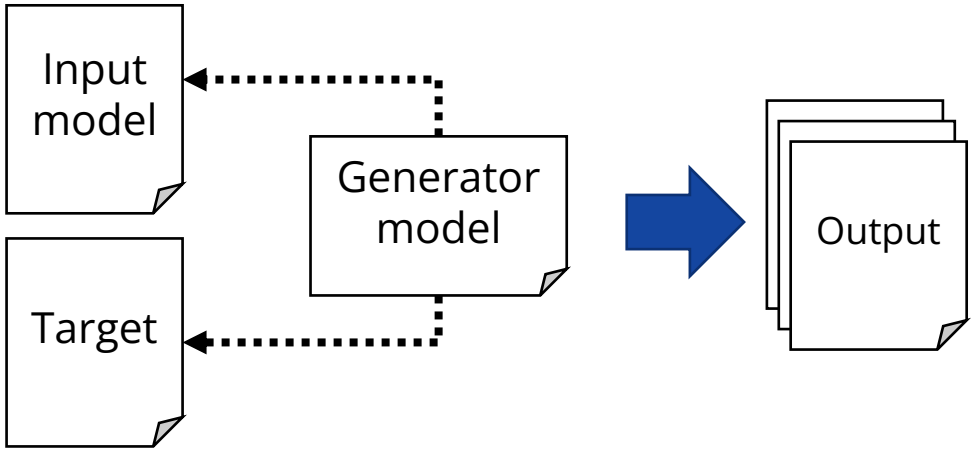
Code generator = Textual Syntax* + Model transformation*

- Needs DOM+serializer for the target textual language
  - Typically available for textual modeling languages
  - Also for many mainstream programming languages
- Formatting + syntax difficulties handled by serializer
  - Especially (qualified) cross-references, e.g. Java imports
  - Escaping (special characters) as well
- Supports multiple iterations, non-linear logic, incrementality
- Not as practical for text-heavy target (e.g. documentation)

*see M2M + Textual Syntax lectures

Advanced Text Generation Issues

Generator model



Generator model: encapsulates all required information

- References external input models (possibly multiple models)
- Add non-domain parameters
  - Target namespace, target code style, generation alternatives…
  - “Outlet” / target folder
- (May reference target models, once generated)
- Traceability links for multipass / incremental maintenance
- **Example:** ecore generator model (.genmodel)

Manual vs generated parts

- Don’t overwrite manual extensions upon re-generation!
- Where to put manual code parts (to be preserved)?
  - Model: Allows better reusability ☺,
    Increases complexity ☹,
    Text editing within model?
  - Template: for simple cases
  - DOM/AST: not user-facing ☹
  - Generated code files: Difficult, ad-hoc separation ☹
  - Separate code file/folder: VCS-friendly (diff, ignore) ☺, Cleaning easy ☺

See EOperation body annotations

See EMF and @generated NOT

How to connect generated and manual?

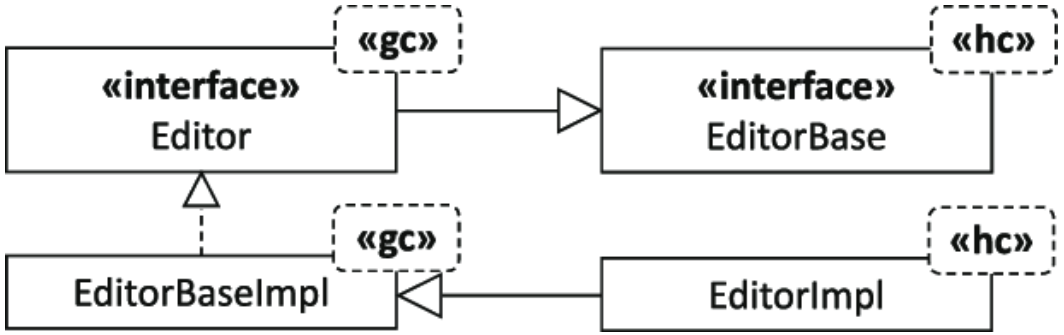
How to connect them?

- Interleave in same class
  - Java: no support ☹
  - C# partial classes ☺
- Manual invokes generated
  - Handling control difficult ☹
- „Generation Gap” pattern
  - Manual inherits from generated

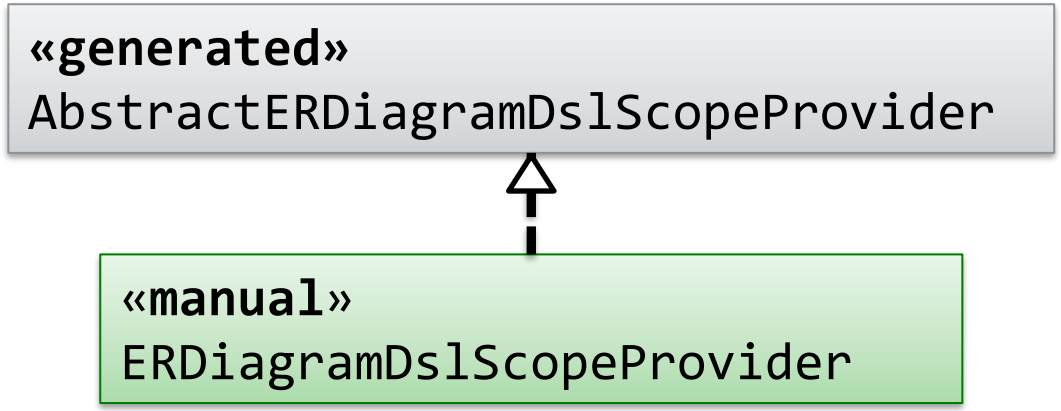
Generation gap pattern

- **Generation gap:** inherit from generated

Roth, Alexander & Greifenberg, Timo & Hölldobler, Katrin & Kolassa, Carsten & Look, Markus & Mir Seyed Nazari, Pedram & Müller, Klaus & Perez, Antonio & Plotnikov, Dimitri & Reiss, Dirk & Rumpe, Bernhard & Schindler, Martin & Wortmann, Andreas. (2015). Integration of Handwritten and Generated Object-Oriented Code. Model-Driven Engineering and Software Development. 580. 10.1007/978-3-319-27869-8_7.



- Generation gap example: Xtext custom scoping (Eclipse solution)



Regenerated after each grammar change

Initialized, then left alone

Code formatting

Where to include?

- **Model:** Does not follow typical MVC design paradigm
- **Templates:** Simple formatting element
- **DOM/AST/CST:**
  - Can store all relevant information
  - Makes it very complex
- **Best solution:** Code formatting as separate step
  - a new step in the generation workflow
  - Can be handled with 3rd party code formatters
  - E.g. Eclipse JDT formatter, XML DOM serializer

Keywords and special characters

Restricted keywords in the target language

- Java: **abstract**, **class**
- XML: < , >

Needs to validate the model before generation

- Can be very complex → separate step before code generation
- Example Java simple support: *isJavaIdentifierStart()* (in Character)

Escaping

- On the model (in separate generator model?)
- Only at code generation time

Technologies

Xtend

A **general-purpose** JVM-based language

- Imperative, statically typed, compiles to Java, good Java interop
- Incorporates functional programming constructs
- Original purpose: compile Xtext2 DSLs to Java

Advanced features:

- Type inference
- Properties
- Everything is an expression
- Operator overloading
- Power switch
- Lambda expressions
- **Templates**

Homework option!

https://www.eclipse.org/xtend/

Xtend example

```
import com.google.inject.Inject

class DomainmodelGenerator implements IGenerator {

 @Inject extension IQualifiedNameProvider nameProvider

 override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        for(e: resource.allContentsIterable.filter(typeof(Entity))) {
            fsa.generateFile(
                e.fullyQualifiedName.toString.replace(".", "/") + ".java",
                e.compile)
        }
 }
 def compile(Entity e) '''
     «IF e.eContainer != null»
      package «e.eContainer.fullyQualifiedName»;
     «ENDIF»
     public class «e.name»
      «IF e.superType != null»extends «e.superType.shortName» «ENDIF»
     {
     «FOR f:e.features»
      «f.compile»
     «ENDFOR»
     }
 '''
```

Full Java interop

Type inference

Syntactic sugar for first parameter

''' =template expression

String interpolation

In-template control structures

Acceleo

Acceleo by Obeo

- Template language
- Supported by advanced tooling
- Template + input EMF model → textual output

- Laboratory material!

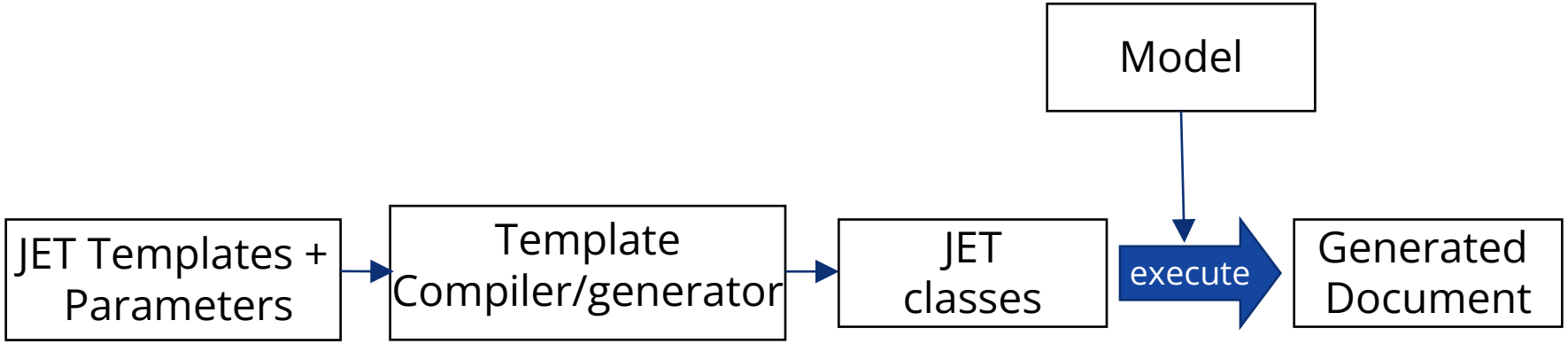
Acceleo example

generate.mtl

```
    [comment @main /]
    [file (c.fullFilePath(), false, 'UTF-8')]
package [packageName()/];

import java.util.List;

public class [javaName()/] {

    [for (att : Property | ownedAttribute) ]
    private [javaType()/] [javaName()/];

    public [javaType()/] get[javaName().toU
        return [javaName()/];
    }

    public void set[javaName().toUpperFirst
        this.[javaName()/] = [javaName()/];
    }

    [/for]
}
    [/file]
```

- * before ()
- * separator ()
- * after ()
- * ? ()
- * {}
- att:Property
- c:Class
- self
- aggregation:AggregationKind [1]
- association:Association [0..1]
- associationEnd:Property [0..1]
- class:Class [0..1]

Java Emitter Templates



- JSP-like template language using Java as its control sequence
- **Compiled** to Java
- Open output format (Text)
- Parameters as Java objects
- Part of EMF
- Eclipse uses JET as its own template language

JET example

Template

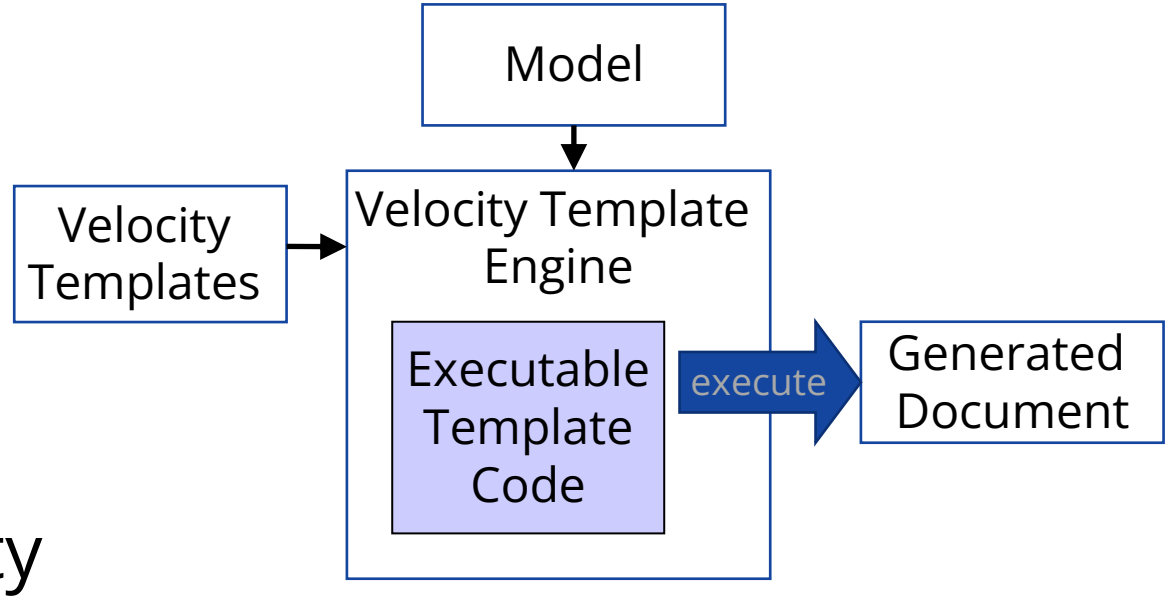
Output

Example output

```
<demo>
        <element>A</element>
        <element>B</element>
</demo>
```

```
<%@ jet package="hello"
imports="java.util.*" class="XMLDemoTemplate" %>
<% List elementList = (List) argument; %>
<?xml version="1.0" encoding="UTF-8"?>
<demo>
<% for (Iterator i = elementList.iterator();
i.hasNext(); ) { %>
<element><%=i.next().toString()%></element>
<% } %>
</demo>
```

Apache Velocity



Apache Velocity

- JSP-like template language with limited control sequence
- Interpreted
- Open output format (Text)
- Parameters as a Map

Apache Velocity

Template

Output

Example output

```
<demo>
        <element>A</element>
        <element>B</element>
</demo>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<demo>
#set( $tempString = "Element")
#foreach( $element in $elementList)
    <element> ${element.toString()} <element>
#end
</demo>
```

Summary

Code generation - Summary

- Started from source code generation
  - UML -> Java, C++, ….
- Used in many other text based artifacts
  - document generation (web)
  - report generation (XML, XLS, CSV, print)
  - Configuration (wsdl)
- Strong tool support
  - Xtend
  - (CodeDOM)
- There are some use cases outside of the MDE field