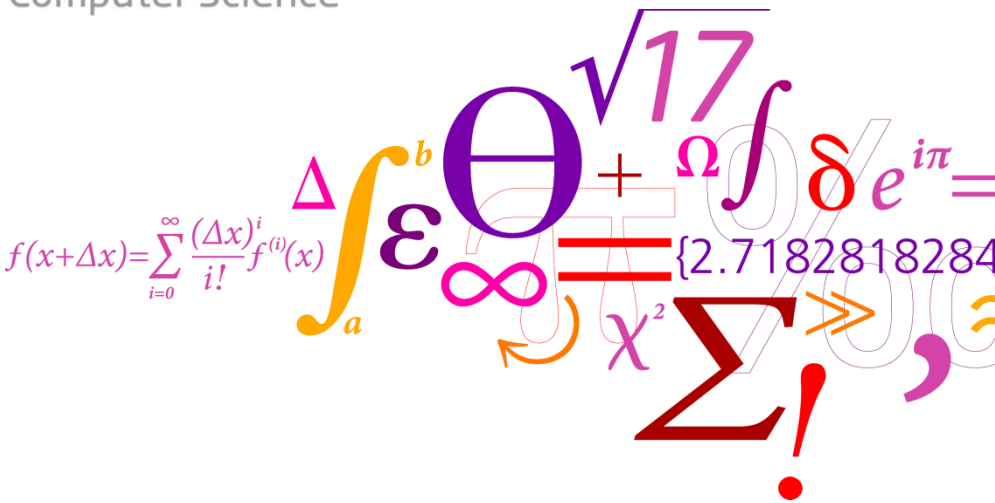
Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

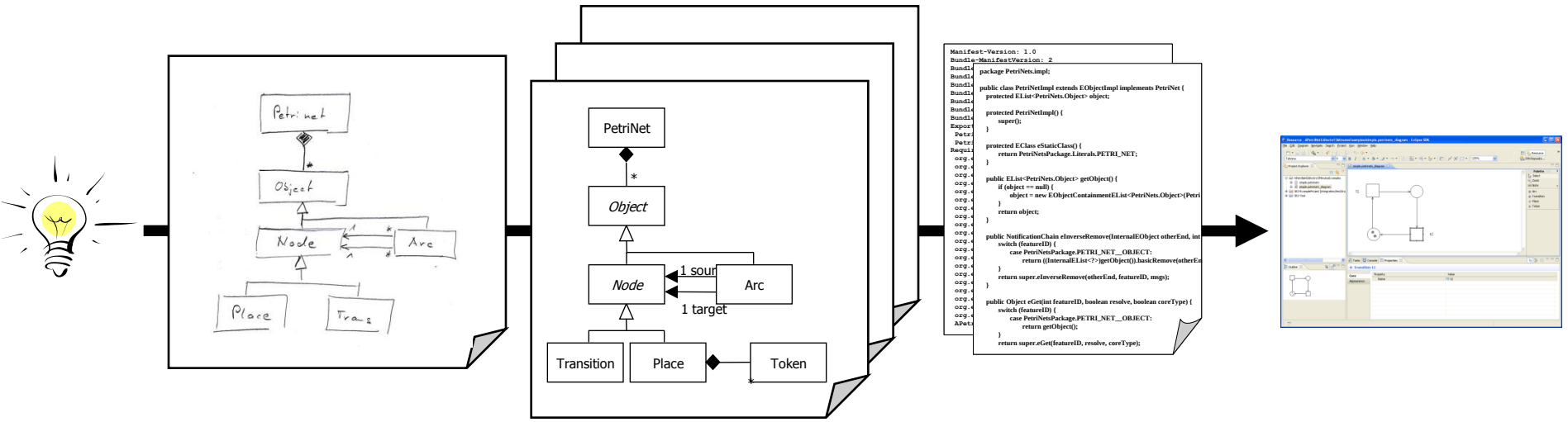


V. Transformations

Focus today: Model to Text (M2T) transformations
→ Code generation

Motivation (cf. Vision)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Analysis

Design

Implementation

~~Coding~~

Code is generated

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- If we want to get software automatically from models, we need to have a technology for transforming models into code
- Programming this transformation manually is very error prone (and not in the spirit of our endavour)
- In esssence, we need a technology for transforming Models to Text: M2T-transformation

Later, we will see that transformations between models are equally important: This is then called M2M

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Main idea: Template

Dear <Name>,

we are pleased to inform you that you you will be refunded <amount> in income tax.

The reason is that <reason>.

Best regards,
<clerkname>

Today, this is much better known from web programming: PHP or JSP, ASP, …

Standard text in which some "specifics" will be filled in (attributes/parameters/fields).

Java Emitter Templates (JET)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

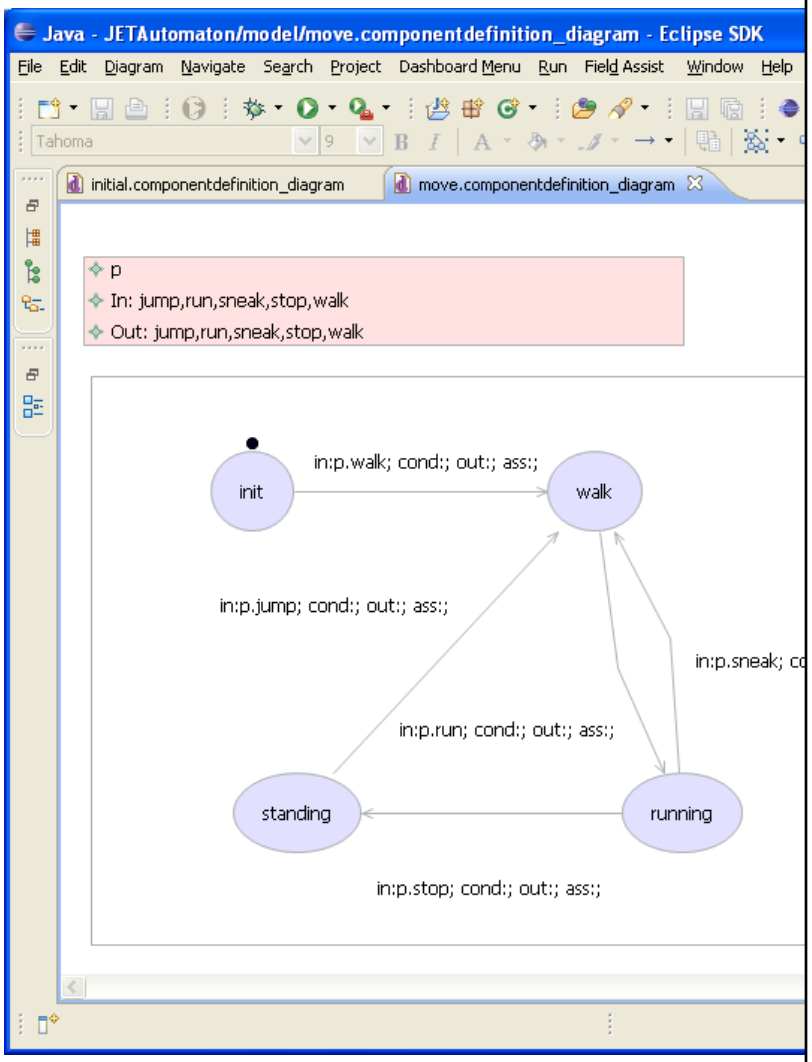
- There are different concrete tempalte technologies (JET, Xpand, Acceleo) that are made for transformin models into some form of text

- The ideas will be presented based on the Java Emitter Templates (JET)

JET is very similar to JSP and the control language is Java: we do not need to learn much new stuff!

1.1 Example: Component Definition

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler




```
%--------------------------------
Overview of component definition
%--------------------------------

The components name is "Move".

The automaton has 4 states:
    init (initial)
    walk
    running
    standing

%--------------------------------
```

Example: ”template” it

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
%-----------------------------------
Overview of component definition
%-----------------------------------

The components name is "Move".

The automaton has 4 states:
    init (initial)
    walk
    running
    standing

%-----------------------------------
```

Example: ”template” it more

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
%--------------------------------
Overview of component definition
%--------------------------------

The components name is "<name>".

The automaton has <no-states>
states:
<foreach state>
   <state.name> <initial>
<hcaerof>

%--------------------------------
```

```
---------------------------
ew of component definition
---------------------------

mponents name is "Move".

tomaton has 4 states:
t (initial)
k
ning
nding

---------------------------
```

Example: Java it

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
%--------------------------------
Overview of component definition
%--------------------------------

The components name is "< c.getName() >".

The automaton has
< c.getAutomaton().getState().size() > states:
< for (State s:c.getAutomaton().getState() ) { >
   < s.getName() > < s.isIntial() ? "(initial)" : "" >
< } >

%--------------------------------
```

Example: JET it

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
<%@ jet package="translated" class="SimpleAutomaton"
    imports="dk.dtu.imm.se2e09.casetool.componentdefinition.*"
%>
%--------------------------------
Overview of component definition
%--------------------------------
<%
   ComponentDefinition c = (ComponentDefinition) argument;
   Automaton a = c.getAutomaton();
%>

The components name is "<%= c.getName() %>".

The automaton has <%= a.getState().size() %> states:
<% for (State s:a.getState()) { %>
   <%= s.getName() %> <%= s.isIntial() ? "(initial)" : "" %>
<% } %>

%--------------------------------
```

Not very readable, but does what we want!

Example: Lessons learned

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Principle behind JET is simple
- The final JET-template is not very readable
- But it is not difficult to ”work through” it
  (see later: class that does transformation)

- Always start from a concrete example, that you turn into a template

As long as you do not know where you are heading, DON’T even try to make a template.

1.2 Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
<%@ jet package="translated" class="SimpleAutomaton"
    imports="dk.dtu.imm.se2e09.casetool.componentdefinition.*"
%>
%-------------------------------
Overvie          ponent definition
%---
<%
%>
Th                                    >".
Th                                    %> states:
<%
                                      "(initial)" : "" %>
<% } %>
%-------------------------------
```

JET directive:

Actually from this JET-Template, a Java class will be generated that does the actual transformation.

These directives tell the name, package and imports for the generated class that generates the actual output.

If you want to see this class, go to your runtime workbench and have a look into the hidden JET project (.JET-Automaton in our case) in the navigator view.

We also need to configure the class-path of the generated class! This is done when the template is called (see slide 19).

Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
<%@ jet package="translated" class="SimpleAutomaton"
    imports="dk.dtu.imm.se2e09.casetool.componentdefinition.*"
%>
%-------------------------------
Overview of component definition
%-------------------------------
<%
   ComponentDefinition c = (Componen
   Automaton a = c.getAutomaton();
%>

The components name is "<%= c.getName
The automaton has <%= a.getState().size() %> states:
<% for (State s:a.getState()) { %>
   <%= s.getName() %> <%= s.isIntial() ? "(initial)" : "" %>
<% } %>

%-------------------------------
```

Text snippet directly going to the text-output (including all spaces, tabs, and line feeds!)

Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

JET Scriptlet: Must altogether give a legal Java method body!

```
<%@ jet packag                                    mpleAutomaton"
    imports="d                                    l.componentdefinition.*"
%>
%-------------------------------
Overview    component definition
%-------------------------------
<%
   ComponentDefinition c = (ComponentDefinition) argument;
   Automaton a = c.getAut maton();
%>

T                                  %= c.getName

T                                  etState().si
<                                  e()) { %>
                                     s.isIntial()
<% } %>

%-------------------------------
```

In principle, we can use all Java commands and all classes we want; don't forget to import them!

The object passed to the template as an argument when the template is started.

Concepts

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
<%@ jet package="translated" class="SimpleAutomaton"
    imports="dk.dtu.imm.se2e09.casetool.componentdefinition.*"
%>
%-------------------------------
Over                                          n
%---                                          --
<%
    C                                    ponentDe:
    A                                    <
%>

The components name is "<%= c.getName() %>".

The automaton has <%= a.getState().size() %> states:
<% for (State s:a.getState()) { %>
   <%= s.getName() %> <%= s.isIntial() ? "(initial)" : "" %>
<% } %>

%-------------------------------
```

JET/Java expression: must return a String (or something that can be “used” as a String).

More technically, it must be possible to append the return value to a Java StringBuffer (see later).

More details

Note: Scriptlets, and expressions in this template will also be resolved!

Other JET directives:

- inlcude:
  ```
  <%@ include file="anotherTemplate.jet" %>
  ```

- skeleton:
  Defines the skeleton class to be used (later)

- startTag, endTag:
  used to replace `<%` and `%>` as start and end tags.

- nlString:
  defines the String serving as a newline

1.3 Using a template

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Up to now:

- What is a template
- What does it mean

For more details, see the tutorial and have a look into the code.

In particular, have a look into the automatically generated project .JETEmitters in your runtime workbench.

See also material on the course's web pages.

Question now:

- How do we start (and configure) a transformation for such a template?
- What happens behind the scenes?

See ECNO projects for another (more complex) example: See package dk.dtu.imm.se.ecno.model.generator in project with the same name and folder templates.

Some code snippets

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
import org.eclipse.emf.codegen.jet.JETEmitter;
import org.eclipse.emf.codegen.jet.JETException;
...

ComponentDefinition c = ...

JETEmitter emitter = new JETEmitter(uriOfTemplate,
   getClass().getClassLoader());

emitter.addVariable(
   "CASETOOL", "dk.dtu.imm.se2e09.casetool");
emitter.addVariable(
   "EMF_COMMON", "org.eclipse.emf.common");

String result = emitter.generate(
   monitor, new Object[] { c });
```

Generate and initialize generator class from JET-template.

Configures the classpath for the automatically generated JET emitter project!

Generated Class (snippets)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
package translated;

import dk.dtu.imm.se2e09.casetool.componentdefinition.*;

public class SimpleAutomaton {
...

  protected final String TEXT_1 =
    "%-----…" + NL +
    "  Overview of component  definition" + NL + "%-----…";
  protected final String TEXT_2 =
    NL + NL + "The components name is \"";
  protected final String TEXT_3 =
    "\"." + NL + "" + NL + "The automaton has ";
  protected final String TEXT_4 = " state(s):";
  protected final String TEXT_5 = NL + "   ";
  protected final String TEXT_6 = " ";
  protected final String TEXT_7 = NL + " …" + NL + "%-----…";
```

Generated Class (ctnd.)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public String generate(Object argument)
 {
   final StringBuffer stringBuffer = new StringBuffer();
   stringBuffer.append(TEXT_1);

  ComponentDefinition component = (ComponentDefinition) argument;
  Automaton automaton = component.getAutomaton();

   stringBuffer.append(TEXT_2);
   stringBuffer.append( component.getName() );
   stringBuffer.append(TEXT_3);
   stringBuffer.append( automaton.getState().size() );
   stringBuffer.append(TEXT_4);
    for (State state : automaton.getState()) {
   stringBuffer.append(TEXT_5);
   stringBuffer.append( state.getName() );
   stringBuffer.append(TEXT_6);
   stringBuffer.append( state.isInitial() ? "(initial)" : "" );
    }
   stringBuffer.append(TEXT_7);
   return stringBuffer.toString();
 }
```

1.4 Skeleton

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Now:

- We know how to use the generator
- We know how the generated generator class looks

Question now:

- Can we adapt or change the generation of the generator class (might be interesting, once we call generators from other templates)

The answer: Skeletons

Recommendation: Do not do complex computations in the JET-template; do it in methods of the skeleton (or helper classes referred to from there).

Standard skeleton

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class CLASS
{
  public String generate(Object argument)
  {
    return "";
  }
}
```

Will be replaced by the class name from the JET directive

Will be replaced by code that produces the output for the JET template (see p. 20/21)

Change skeleton

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class CLASS extends MyClass
{
  private WhatEverType attribute;

  public OtherType myMethod()
  {... }

  // Some comment.
  public String generate(Object argument)
  {
    return "";
  }
}
```

Attention: This part must be last!

If you want your template to use another skeleton, use the JET directive:

```
<%@ jet package="translated" …
      skeleton="my.skeleton" %>
```

Skeleton → Extends

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

In JET2, the skeleton was replaced by the extends tag:

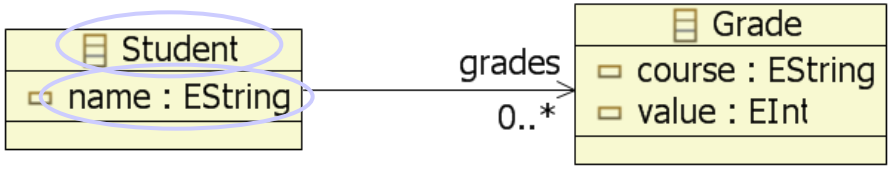
```
<%@ jet extends="dk.dtu.imm.se.ecno.generator.
    SomeBaseClass" %>
```

The generated class that generates the result extends the class (which is given by a fully quantified name). What we implemented in the skeleton can be implemented in this super class.

skeleton is deprecated

1.5 JET Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



how?

Idea: use templates!

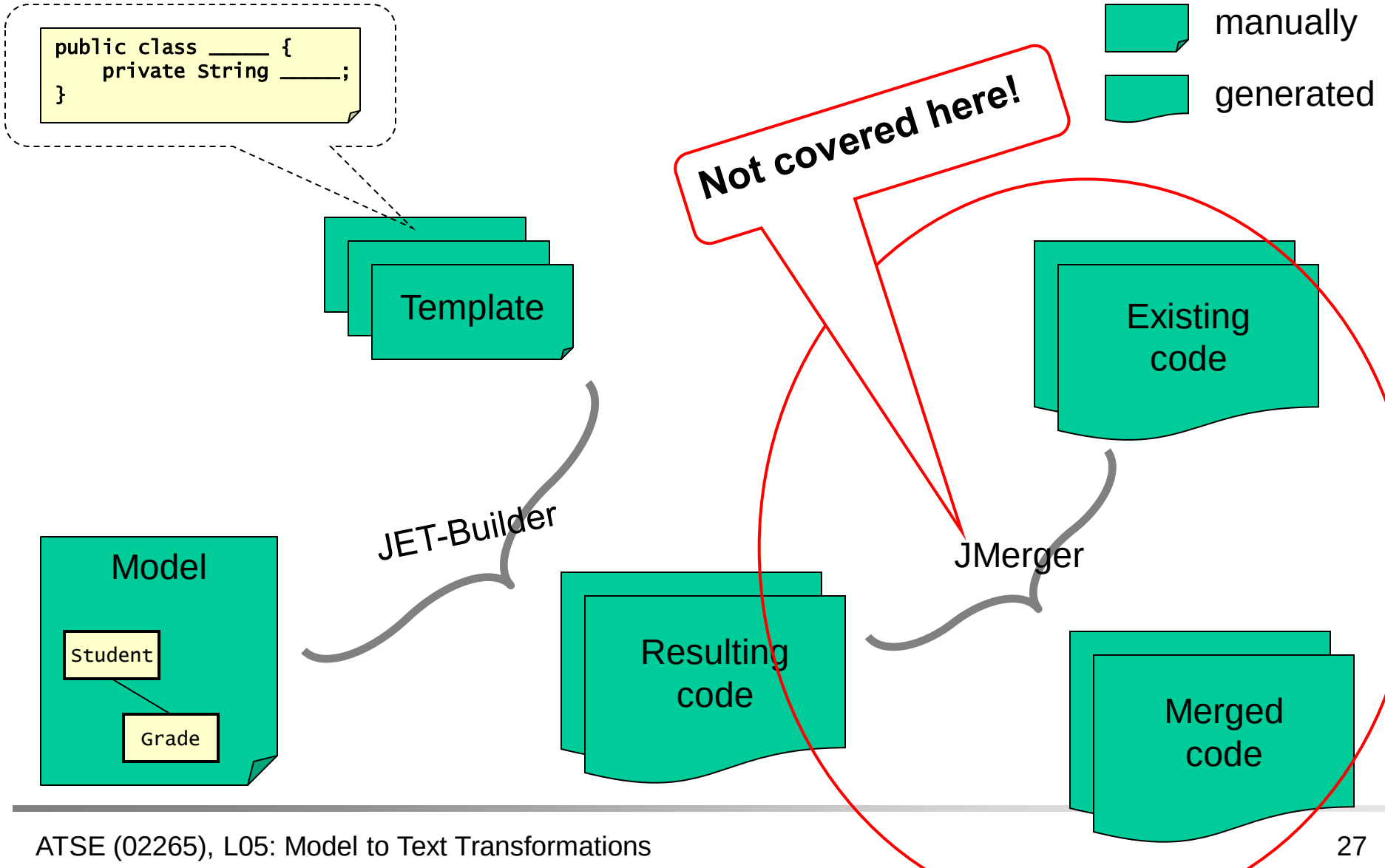
```
public class Student {

    private String name;

    private List<Grade> grades;

}
```

```
public class Grade {

    private String course;

    private int value;

}
```

one template?!

Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Template for result

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler


```
public class Student {

    private String name;

    private List<Grade> grades;

}
```

Directives (2 types): `<%@ jet … %>` `<%@ include file="…" %>`

Scriptlets: *<% (arbitrary java code) %>*
e.g.: `<% String s = "Hello, transformation world!"; %>`

```
<%@ jet package="codegen" ... ClassGenerator" imports="org.eclipse.emf.ecore.*" %>

<% EPackage pack = EmfHelper.loadModel(argument); %>

<% for (EClassifier eClassifier : pack.getEClassifiers()) { %>

public <%= eClassifier.abstract ? "abstract" : "" %> class <%= eClassifier.getName()%> {

  <% for (EAttribute eAttribute : ((EClass)eClassifier).getEAttributes()) { %>

  private <%= eAttribute.getEType().getInstanceTypeName()%> <%= eAttribute.getName()%>;

  <% } %>

}
<% } %>
```

Expressions: *<%= (java expression) %>*
e.g.: `<%= (s.length() > 0 ? "<null>" : s + "!") %>`

Discussion M2T/JET

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

M2T = “Model to Text”
We will come back to this discussion, when we talk about M2M!

Advantages

- Independent of generated language
- No need for a precise model of the languages syntax
- Flexible

- Easy to get started
- Works for any Java structure (not restricted to EMF)

Other technologies like XPand are more tightly integrated with EMF. Acceleo has a better IDE support.

Discussion M2T

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Disadvantages

- Templates become easily unreadable
  (mix of two languages)
  → hard to debug (error messages not too helpful, either)
- No guarantee that generated text is syntactically correct
- After generation no relation between model and text
  (unless you explicitly do something about it →tutorial)
- Regeneration overwrites manual changes
- JET technology is bound to Eclipse!
  (used in EMF)

Can be overcome in combination with other technologies: e.g. JMerge

Discussion: Other ways out?