

Code Generation



www.mdsd-buch.de



www.mdsd-book.org

Markus Völter

voelter@acm.org

www.voelter.de



Ingenieurbüro für softwaretechnologie

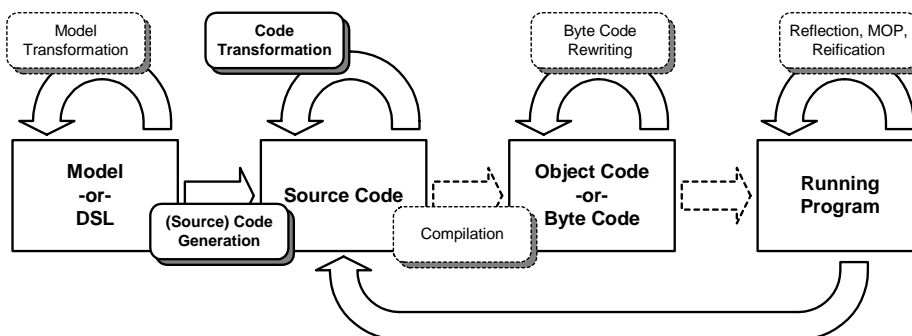
www.voelter.de

- 1 -

© 2003 - 2006 Markus Völter

Overview

- The following illustration shows where **program generation** can be applied.



- We will **focus on the highlighted steps** and briefly introduce the others.

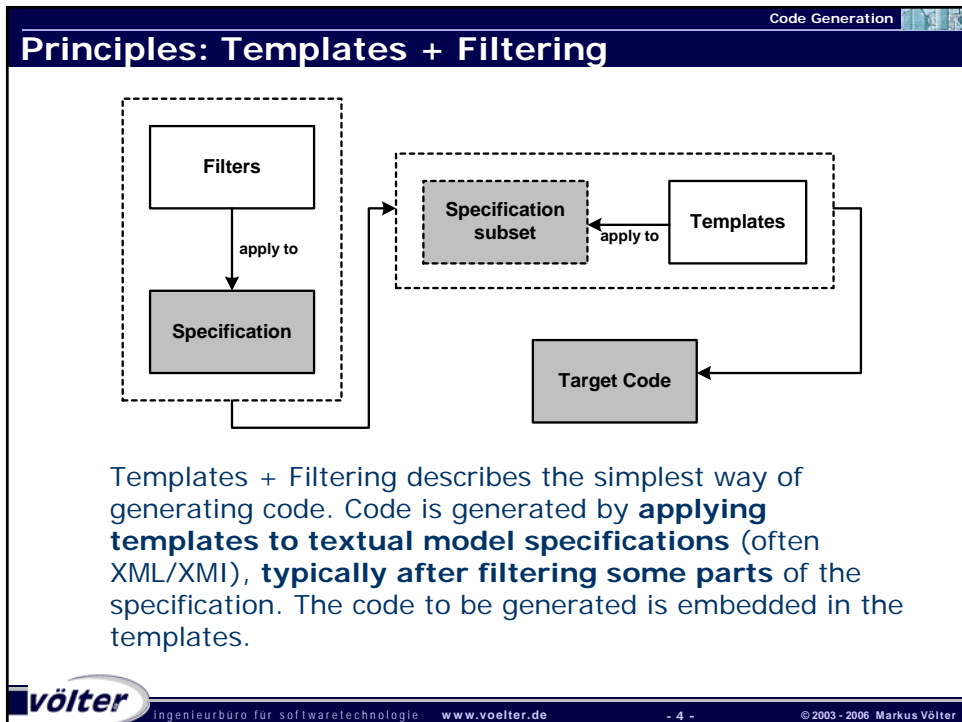
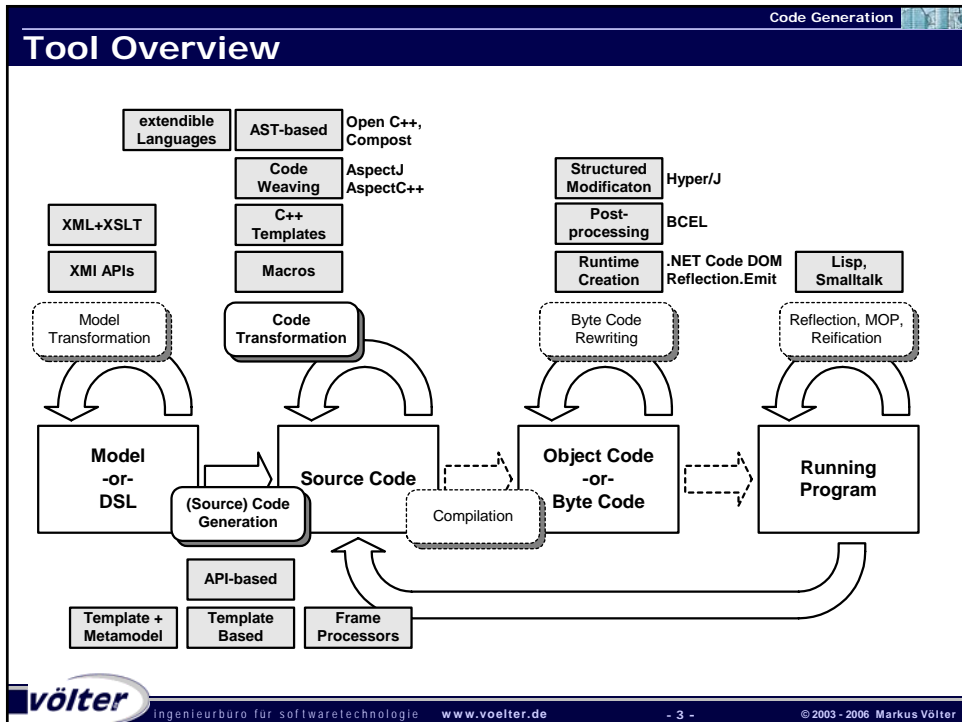


Ingenieurbüro für softwaretechnologie

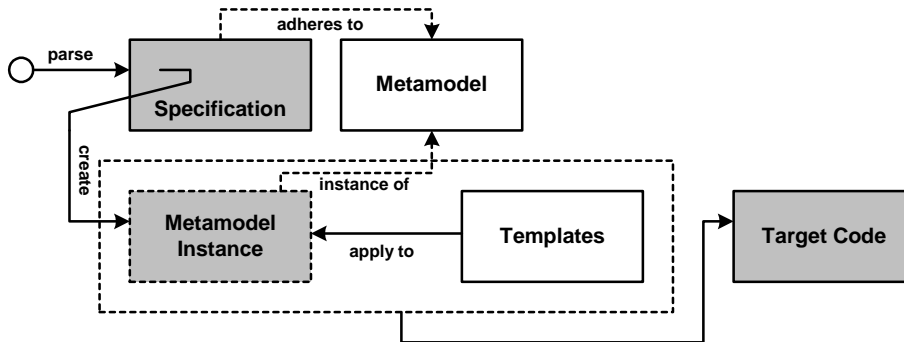
www.voelter.de

- 2 -

© 2003 - 2006 Markus Völter



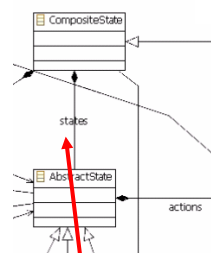
Principles: Templates + Metamodel



Templates + Metamodel is an extension of the templates + filtering pattern. Instead of applying patterns directly to the model, we **first instantiate a metamodel from the specification**. The templates are specified in terms of the metamodel. The metamodel can be extended to include domain or architecture specific aspects.

Templates + Metamodel

- Code Generation is used to **generate executable code** from models.
- Code Generation is **based on the metamodel** and uses **templates** to attach to-be-generated source code.
- In openArchitectureWare, we use a **template language** called **xPand**.
- It provides a number of **advanced features** such as polymorphism, AO support and a powerful integrated expression language.
- Templates can access **metamodel properties** seamlessly



```

«DEFINE SwitchBasedImpl FOR StateMachine»
«FOREACH states.typeSelect(State) AS s
public static final int «s.constantName»
«ENDFOREACH»
  
```

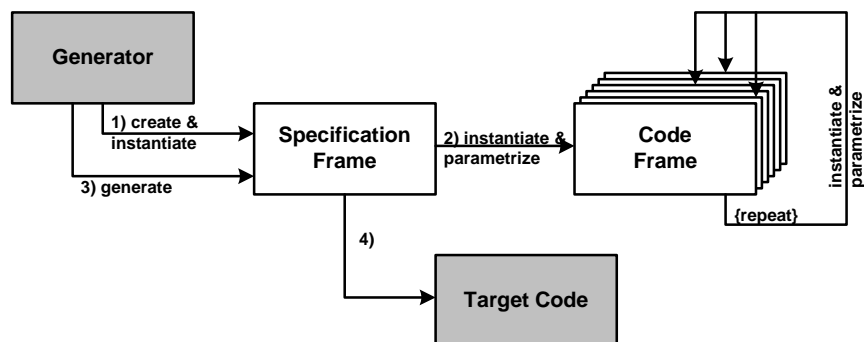
Templates + Metamodel

The screenshot shows a template file for generating Java code for a State Machine. Callouts point to specific parts of the code:

- Opens a File:** Points to the `DEFINE file FOR StateMachine` block.
- Namespace and Extension Import:** Points to the `IMPORT simpleM` and `EXTENSION templates::GeneratorUtils` lines.
- Name is a property of the State-Machine class:** Points to the `name, toFirstUpper()` property access in the file path.
- Iterates over all the states of the State-Machine:** Points to the `FOR EACH states AS s` loop.
- Calls another template:** Points to the `EXPAND executeTransition(this)` call.
- Extension Call:** Points to the `EXPAND handleIllegalTransition` call.
- Template name:** Points to the `handleIllegalTransition FOR StateMachine` definition.
- Like methods in OO, templates are associated with a (meta)class:** Points to the `FOR Transition` loop.

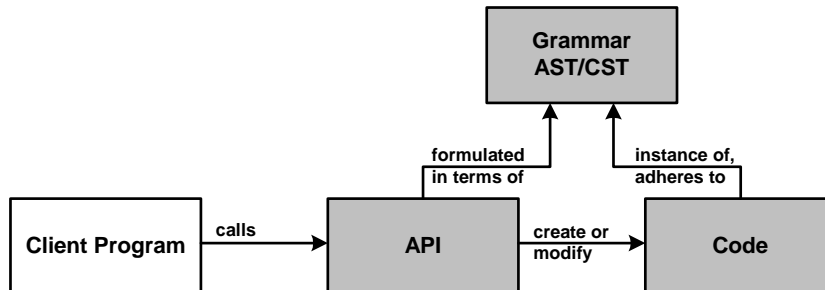
- The **blue text** is generated into the target file.
- The **capitalized words** are xPand keywords
- **Black text** are metamodel properties
- **DEFINE...END-DEFINE** blocks are called **templates**.
- The whole thing is called a **template file**.

Principles: Frame Processing



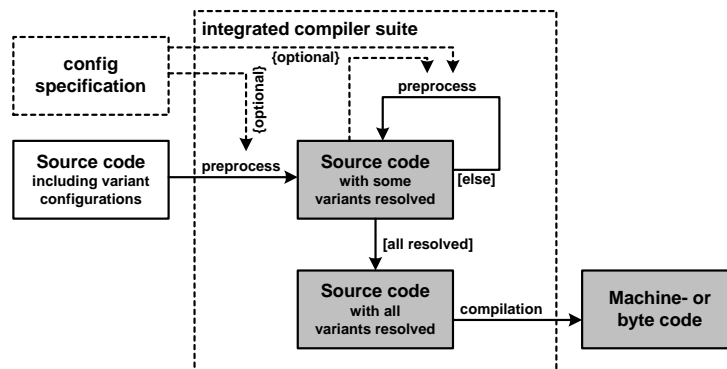
Frame Processing describes a way of generating code by means of so-called frames. **Frames can be seen as programs (functions) that generate code as the result of their evaluation.** Frames can be parametrized by number and string literals as well as other frame instances.

Principles: API Based



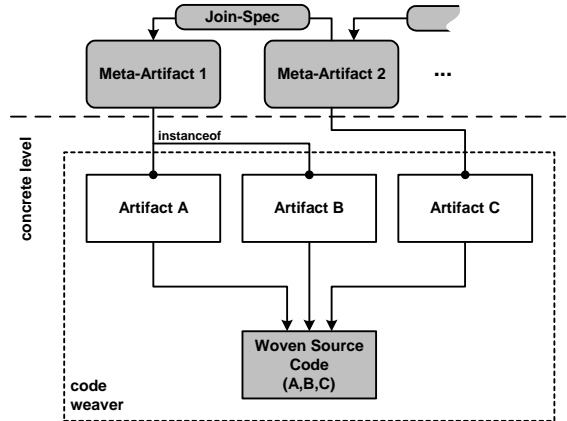
API-based generators **provide an API against which code-generating programs are written**. This API is typically based on the **metamodel/syntax** of the target language.

Principles: Inline



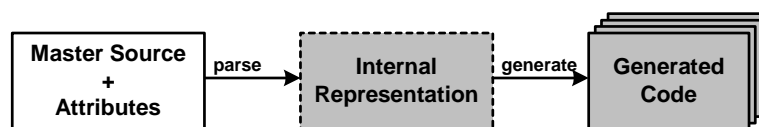
Inline code generation describes a technique where code generation is done **implicitly during interpretation or compilation of a regular, non-generated program**, or by means of a **precompiler**. This process typically modifies the program that is then subsequently compiled or interpreted.

Principles: Code Weaving



Code weaving is about **combining, or weaving, different parts of program text together**. These different parts typically specify **different independent aspects** which are then combined in the woven program.

Principles: Code Attributes



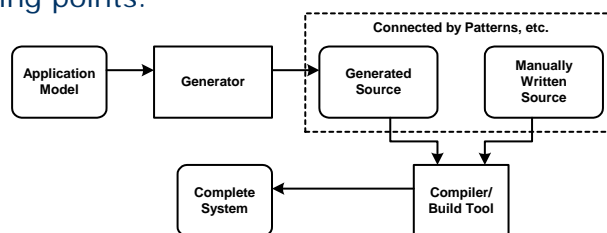
Code Attributes describe a means by which **normal, non-generated program code contains annotations, or attributes**, that specify things that are not contained in the code per se. Based on these attributes, additional code can be generated.

Principles: Overview

	generated/ ungenerated code	template / API	initial learning complexity	suitability for complex uses	flexibility
Template + Filtering	separate	template	simple	not very good	not very good
Template + Metamodel	separate	template (plus m- model API)	high	very good	very good
Frame Processors	separate	template	high	very good	very good
API-based	separate/ integrated	API	simple/high	depends on abstraction level of API	depends on abstraction level of API
Inline Code Generation	integrated	template/API	simple	not very good	good
Code Attributes	separate	template/API	simple	good	not very good
Code Weaving	integrated	API	high	good	good

Separate Generated and Non-Generated Code

- Keep generated and non-generated code in **separate files**.
- **Never modify generated code.**
- Design an architecture that clearly defines **which artifacts are generated, and which are not**.
- Use **suitable design approaches** to “join” generated and non-generated code. Interfaces as well as design patterns such as factory, strategy, bridge, or template method are good starting points.



Separate Generated and Non-Generated Code II

- A) Generated code can **call** non-generated code contained in libraries
- B) A non-generated framework can **call** generated parts.
- C) **Factories** can be used to „plug-in“ the generated building blocks
- D) Generated classes can also **subclass** non-generated classes.
- E) The base class can also contain abstract methods that it calls, they are implemented by the generated subclasses (*template method* pattern)

