Building Model-Based Code Generators for Lower Development Costs and Higher Reuse

Paper Presentation, 10/01/2024

19th Int. Conf. on Software Engineering Advances

ICSEA 2024, Venice Italy

Hans-Werner Sehring

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

Hans-Werner Sehring

Professor for Software Engineering
Head of the Business Informatics / IT Management (M.Sc.) degree program

Software Engineering

Model-Driven Software Engineering

Evolution-friendly software architecture

Software engineering education

Metamodellierung

Domain Modeling

Software Modeling

M³L

Content Management

Digital communication

Media-based knowledge representation

Personalization

Contact

hans-werner.sehring@**nordakademie**.de
https://www.nordakademie.de/die-hochschule/team/hans-werner-sehring

http://dr.sehring.name

https://**orcid**.org/0009-0008-3016-6868

https://www.**researchgate**.net/profile/Hans-Werner-Sehring

https://scholar.**google**.de/citations?user=hsSrVL8AAAAJ

https://www.**linkedin**.com/in/hwsehring/

Software Engineering

Agenda

01 **MDSE**
The context of model-driven software engineering

02 **Code Generation**
Typical code generation approaches

03 **Abstract Code Models**
Models of code at different levels of abstraction

04 **The M^3L**
The Minimalistic Meta Modeling Language

05 **Code Models in M^3L**
Some samples of abstract code models

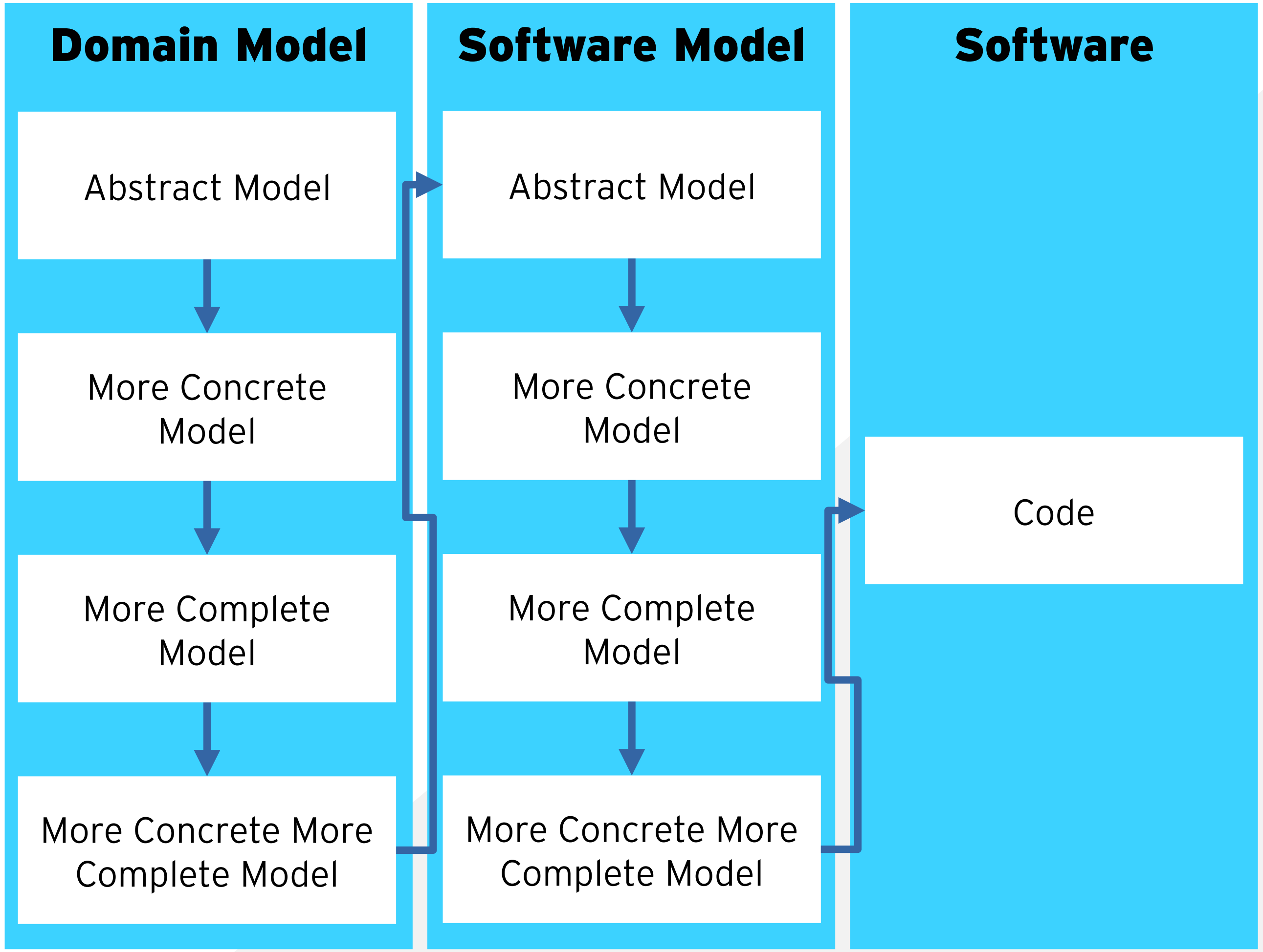
06 **Conclusion**
Summary and outlook

Section 01

Model-Driven Software Engineering (MDSE)

Model-Driven Software Engineering Approaches

Modeling often concentrates on the early development stages



Claim:

MDSE approaches typically concentrate on

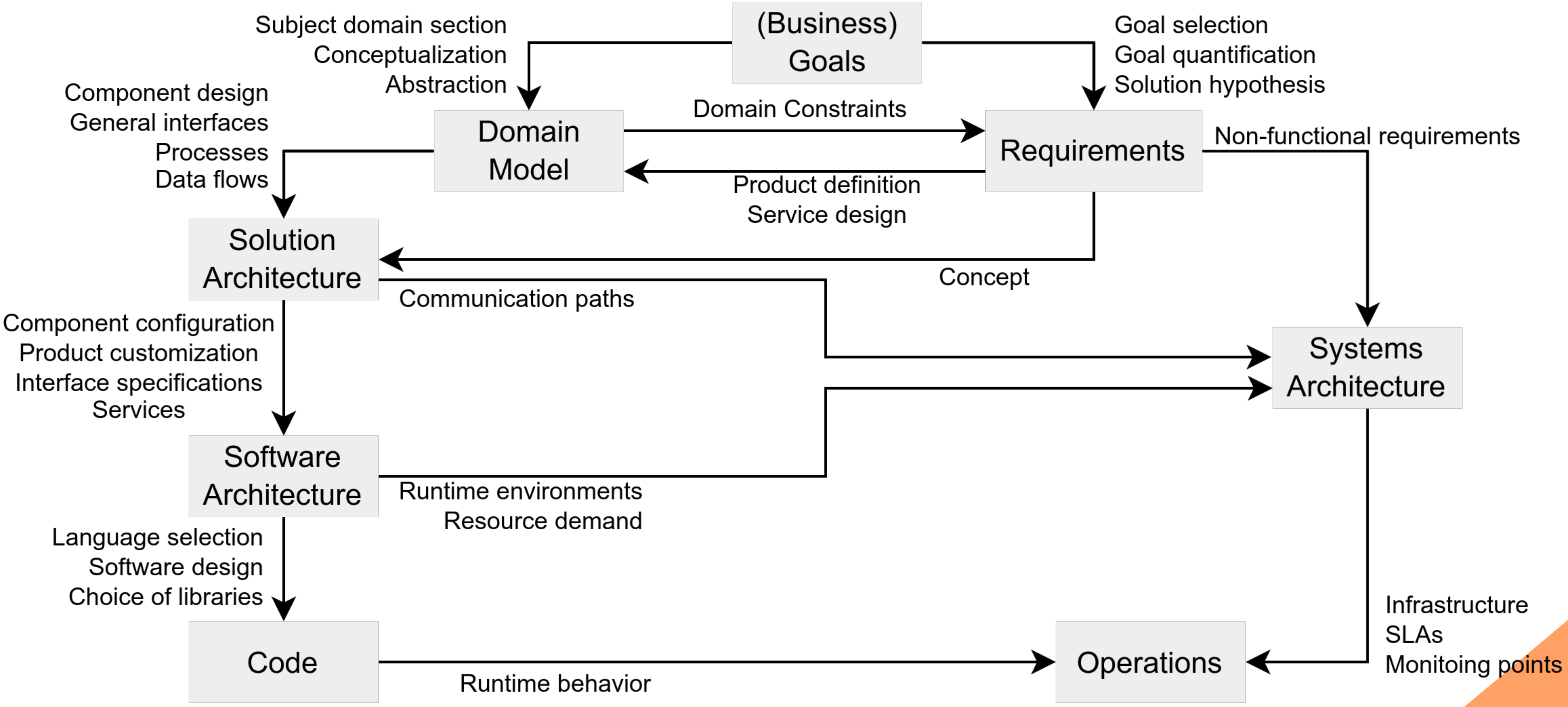
- subject **domain models** and
- high-level (abstract) **solution descriptions**.

The final step of **code generation** relies on

- a predefined solution strategy (for example, for information systems) or
- a specification formalism (custom functionality)

(Software Engineering) Project Lifecycle

Actual (software engineering) projects span a larger lifecycle



Section 02

Code Generation

Approaches to Code Generation

Claim: current approaches are either limited or costly

Typical approaches to bridge the (rather large) gap between specification and code

- **Templates**
- **Meta programs**
- **Generative AI**

Hyprid approaches, for example,

- Templates and meta programming
  - Templates as a domain specific language for
  - Meta programming for application-specific idioms
- Generative AI and meta programming

  Software generators created by generative AI

Section 03

Abstract Code Models

Basic Idea

Break down the large step to code into smaller steps by means of model transformations

After finishing work on a **model of the solution** (architecture), transformation step into stage of coding

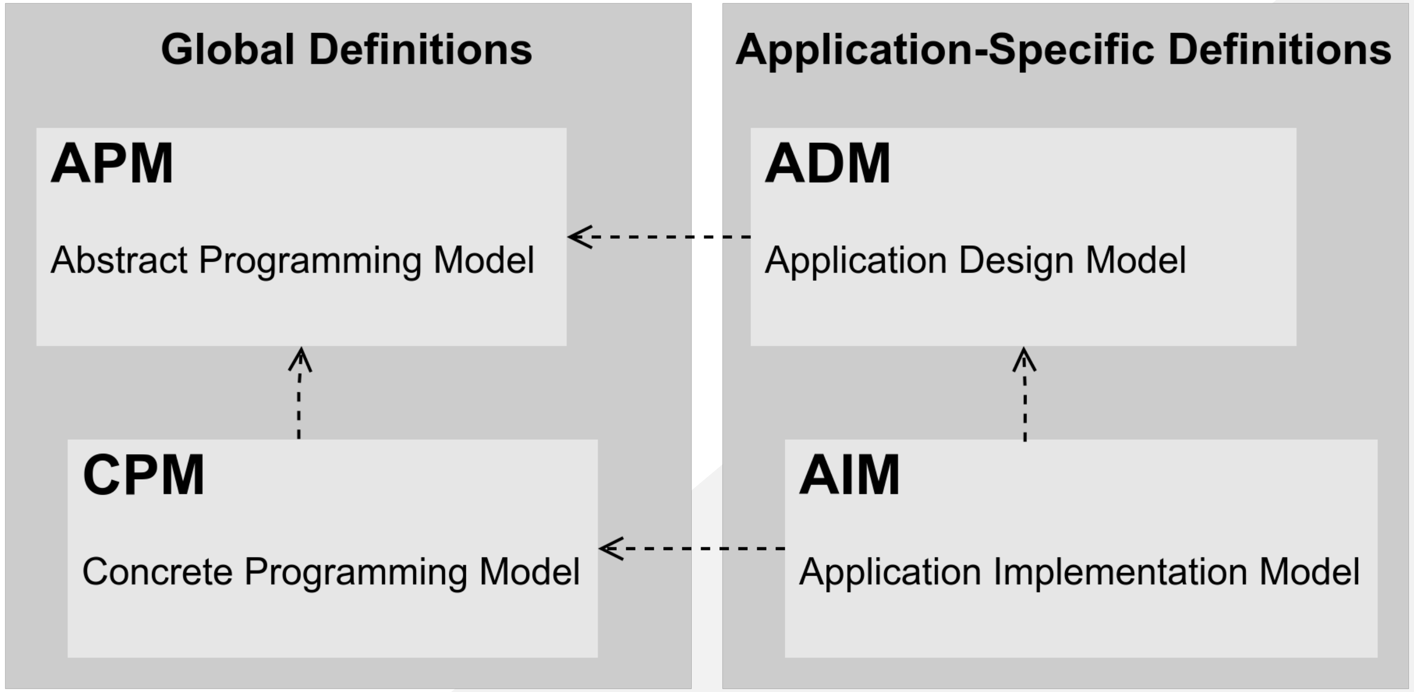
1) Choice of a basic **implementation strategy** (e.g., programming language of a certain paradigm)
2) Creation of a **model of implementation** (code)

Make models of code evolve like models of other domains

3) Formulation of first **hypothetical code** (program in no particular programming language)
4) Stepwise optimization of the hypothetical program
5) Transformation into a model for the code in a **concrete programming language**
6) Application of idioms, patterns, best practices, … of that programming language
7) Application of local style guides
8) Transformation into a model for the utillization of specific software libraries, using specific APIs, etc.

Interplay of Software Models

Models of the software solution evolve like application domain models do



Examples:

APM:

- Object-oriented programming or
- Domain-Driven Design

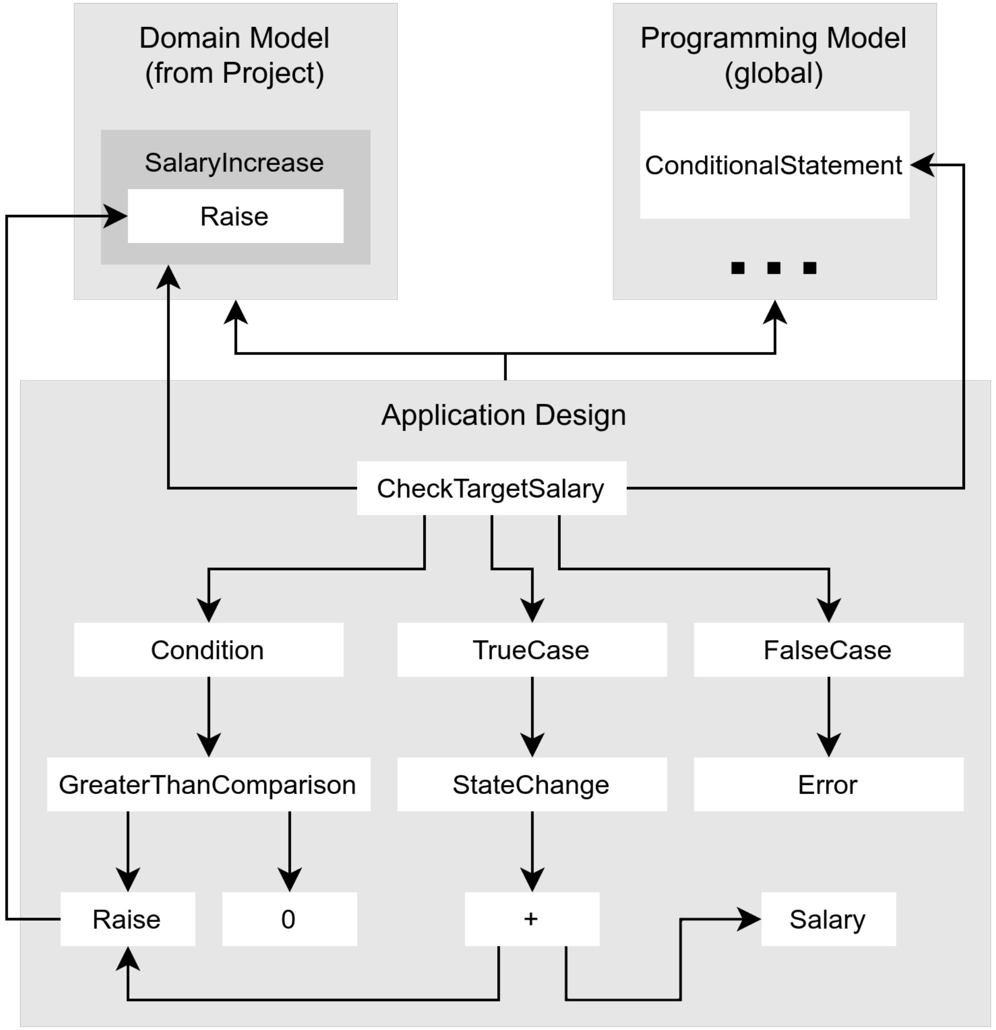
CPM:

- Java or
- Java according to some style guide

ADM:
solution expressed in abstract notation

AIM:
solution adopting best practices of some technology

Example of Software Model Relationships



Section 04

The Minimalistic Meta Modeling Language (M^3L)

Eine Folie für alle Inhalte

The Minimalistic Meta Modeling Language has been reported on in other talks.

Idea:

- Modeling language with very lean syntax and semantics
- Applicable on all (four) levels from instance to meta-meta
- A framework for seamless modeling of all aspects of a problem solution

Only construct: **concept definition** (or **reference**)

`SomeConcept is a BaseConcept {`	*concept, base concept, refinement*
`  Content is a ContextSpecificRefinement`	*content in context*
`} |= ProductionRule`	*semantic rule*
`  |- PartialGrammarForSyntax .`	*syntactic rule*

Plus: **inheritance** (from base concepts), **scopes**, **redefinitions** (in context), **pattern matching**, **evaluation**

Section 05

Code Models in M^3L

Programming Paradigms – Imperative PLs

Type system (any paradigm)

```
Type

Boolean is a Type
True  is a Boolean
False is a Boolean

Integer is a Type
0       is an Integer
PositiveInteger
  is an Integer {
  Pred is an Integer }
1 is a PositiveInteger {
  0 is the Pred }
```

Imperative Basics

```
Statement

Expression
  is a Statement

Variable {
  Name
  Type }

Procedure {
  FormalParameter
    is a Variable
  Statement }
```

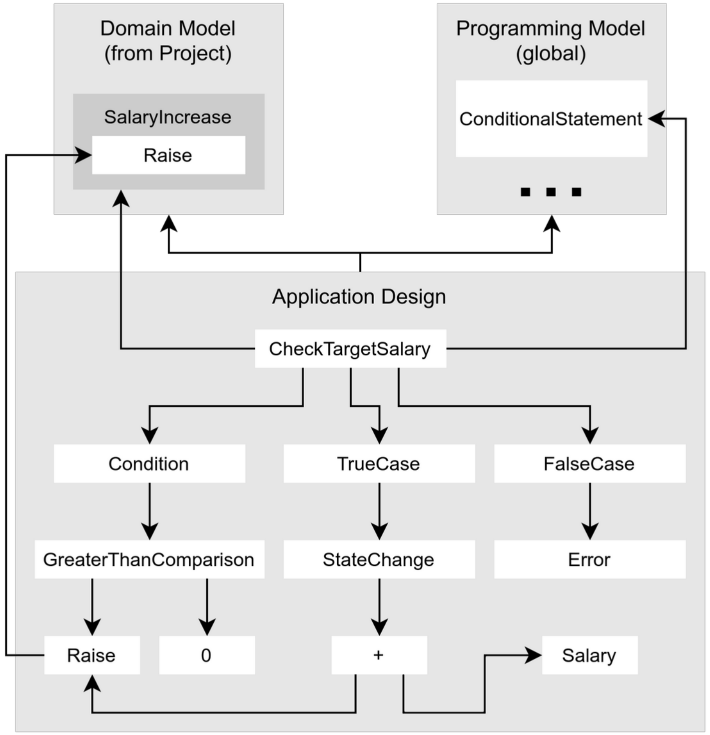
Some Statements

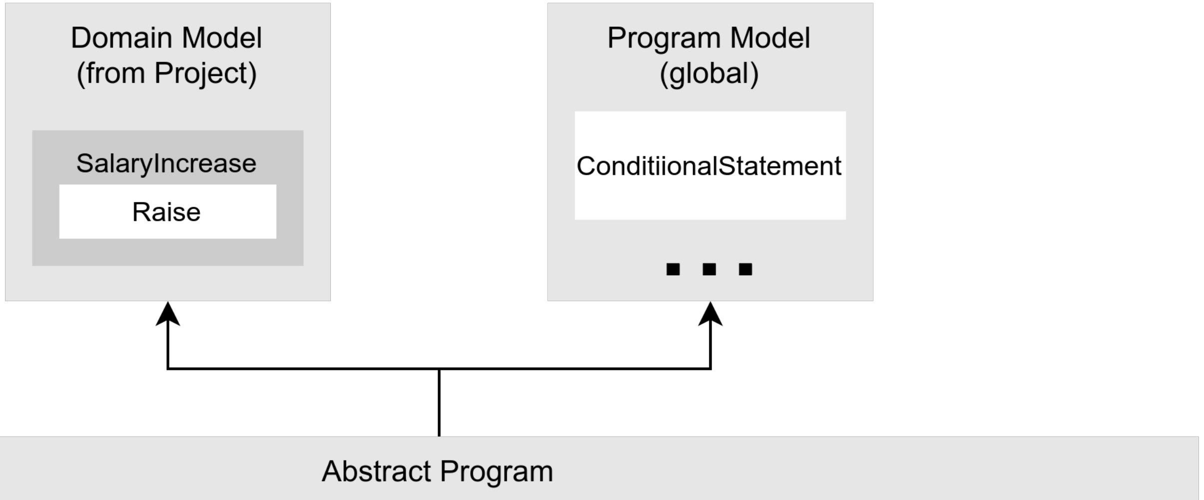
```
ConditionalStatement
  is a Statement {
  Condition is a Boolean
  ThenStatement
    is a Statement
  ElseStatement
    is a Statement }

Loop is a Statement {
  Body is a Statement }

HeadControlledLoop
  is a Loop {
  Condition is a Boolean }
```

Example of Software Model Relationships in M³L






```
CheckTargetSalary is the SalaryIncrease       from SomeSubjectDomainModel
                     a   ConditionalStatement from ImperativeProgramming {
  GreaterThanIntegerComparison from Programming {
    Raise is the Value1
    0     is the Value2 } is the Condition
  StateChangeStatement from OOProgramming {
    Salary is the Property
    IntegerSum {
      Salary   is the Summand1
      Increase is the Summand2
    } is the Expression
  }                                         is the ThenStatement
  ReturnStatement from Programming is the ElseStatement
}
```

Code Model Refinements

ADM refinements in order to optimize a program on the abstract level.

Example: company organization

```
Unit {
  Departments is a Department
}

Department {
  Teams is a Team
}

Team {
  TeamMembers is an Employee
}

Employee is a Person
```

```
OrgUnits is a CompositePattern {
  OrgUnit    is the CommonType
  Team       is the LeafClass
  Unit       is a   BranchClass
  Department is a   BranchClass
}
```

Concrete Code Models

ADM to AIM transformations to accomodate for a specific target language (or other technology)

Model-to-Text Transformations are defined in the CPM – in our case, M³L again

For example, generic OO to Java:

```
PersonClass is a ConcreteClass {
  AgeOfMajority is an Integer
  18 is the AgeOfMajority
}

Person is a PersonClass {
  Name is a String
}

Peter is a Person {
  "Peter Smith" is the Name
}
```

```
Java {
  Person is a Class {
    AgeOfMajority is an int {
      static is a Modifier
      public is a Modifier }
    18   is the AgeOfMajority
    Name is a String … }
  PeterHandle is a Variable {
    peter is the Name String is the Type
    ConstructorCall {
      Person is the Class
      "Peter Smith" is a Parameter
    } is the InitialValue } }
```

Section 06

Conclusion

Summary

Code generation as the final step of Model-Driven Software Engineering processes is typically expressed as a **model-to-text transformation**.

This transformation has to **bridge a large gap** from an abstract description of the desired software solution to working code.

Furthermore, **code** to meet nonfunctional requirements and project constraints is **added in this step**.

As a result, the development of code generators is a **demanding and expensive task**.

By introducing models of the domain code, **model-to-model transformations** can be applied longer down the sequence of development steps. As a result, code generation becomes

- more **feasible**,
- **less costly**, and
- allows more **reuse** (on the level of models).

Outlook on Future Work

Currently work carried out on the basis of small code samples → **experiments with large scale applications**

Contemporary programming languages are of a multi-paradigm nature → study **degrees to which each paradigm is followed** varies, as well as the **interplay of** language constructs of **different paradigms**

Models of code may carry semantics – of abstract programs as well as of concrete code → **translation of domain semantics into program semantics** needs investigation

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

NORDAKADEMIE gAG Hochschule der Wirtschaft
Köllner Chaussee 11 · 25337 Elmshorn · Tel.: +49 (0) 4121 4090-0 · E-Mail: info@nordakademie.de · Web: www.nordakademie.de