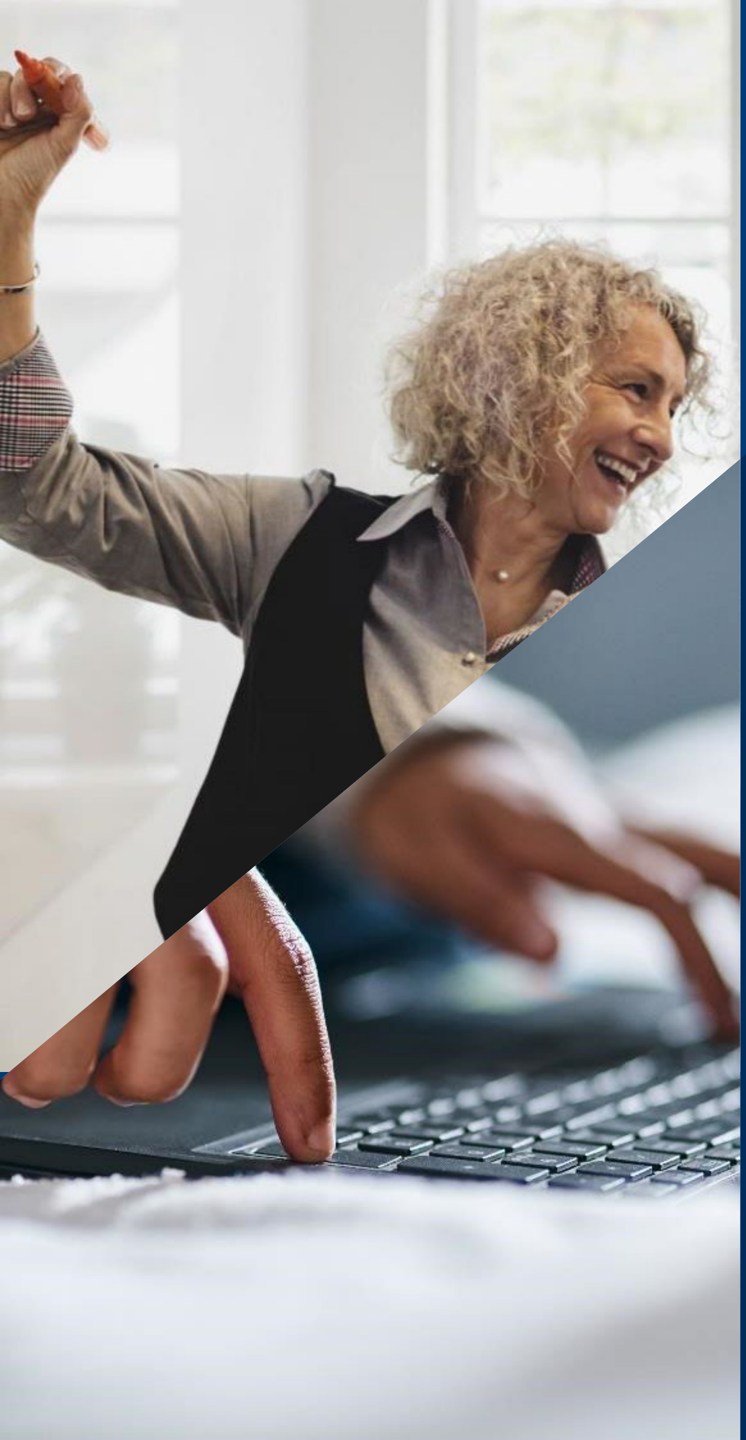
Model-supported Software Creation: Towards Holistic Model-driven Software Engineering

The 2023 IARIA Annual Congress on Frontiers in Science, Technology, Services, and Applications

15 Nov 2023, Valencia, Spain, Hans-Werner Sehring

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT



Agenda

1. Model-driven Software Engineering

Various approaches to model-driven software engineering exist, for example,

- **Model-driven Architecture (MDA)**
  - Early MDSE approach
  - Models are created on (originally) three levels of abstraction
  - A *Computation-Independent Model* (*CIM*) from the perspective of the subject domain.
  - A *Platform-Independent Model* (*PIM*) as a first formal model.
  - Transformed into a *Platform-Specific Model* (*PSM*) used to generate a working implementation.
- **Software Generation**
  - Model contained in code
  - Different approaches, e.g., metaprogramming, templates, generative AI
- **Domain-specific Languages (DSLs)**
  - Languages ≙ Metamodels
  - Defined for a specific domain
- **Generic Software**
  - A domain model was used during the development of the software
  - If the model is parameterized, then software is configurable (low code / no code development)

2. MDSE in Practice

Approaches based on **formal** models and model transformations

Software engineering **reality** (at least in some domains):

Depending on the kind of software and the kind of project, we find

- Heterogeneous modeling artifacts: varying **degrees of formalism, ambiguity, detail**, etc.
- Artifacts often part of a **methodology or a tool**: notation and representations matter
- Several project stages, **not only software** (engineering) related; from inception to operations stages

Sample Development Artifacts and Formalizability

Phase	Order	Discipline	Artifact	Formal(izable) model
Inception / Research		Management	Goals	–
	↪	Management	Inception	–
	↪	Concept	Requirements (inside-out)	X
		Concept	Research (outside-in)	–
Analysis	↪	Concept	Personas	–
	↪	Concept	Customer journeys	–
	↪	Technology	Existing tools	X
		Technology	Information demand / data flows	X
Design		Concept	Information architectures (stationary web, mobile web, mobile app)	–
		Graphics	Wireframes (stationary web, mobile web, mobile app)	–
	↪	Technology	Solution architecture	–
		Graphics	UI design / style guide	(X)
Implementation	↪	Technology	SW arch (if not agile)	X
		Technology	System arch (if not agile)	X
	↪	Technology	Code design	X
		Technology	Code	X
		Concept	Test cases	X
		Technology	Test scripts	X
		Concept	Documentation	(X)
Operations	↪	Technology	Infrastructure	X
		Technology	Build and deploy scripts	X
		Concept	Training	–

Support for Informal Processes and Artifacts

Given that the various process steps and artifacts that are

- not formal
- ambiguous
- not producible by model transformations
- etc.

we cannot have MDSE.

Still, we want …

- Support in managing (modeling) artifacts
- Checks on models
- Deriving software from specifications
- Traceability
- Etc.

We want the benefits of MDSE.

3. Holistic Model-Supported Software Creation

For those software projects with imprecise, creative development steps, we need …

Holistic MDSE that covers all project stages

For example: project success is measured based on business goals, not requirements

Need to **model activities and artifacts outside SW production**

Model-**supported** SE acknowledges the fact that we cannot purely rely on formal models and model transformations

In the absence of formal models, these cannot be the overarching communication base

Models can describe the (final) informal **artifacts**

Model-supported Software **Creation** acknowledges the creative work that is part of the process

There is creative work on artifacts that cannot adequately be formalized by model transformations

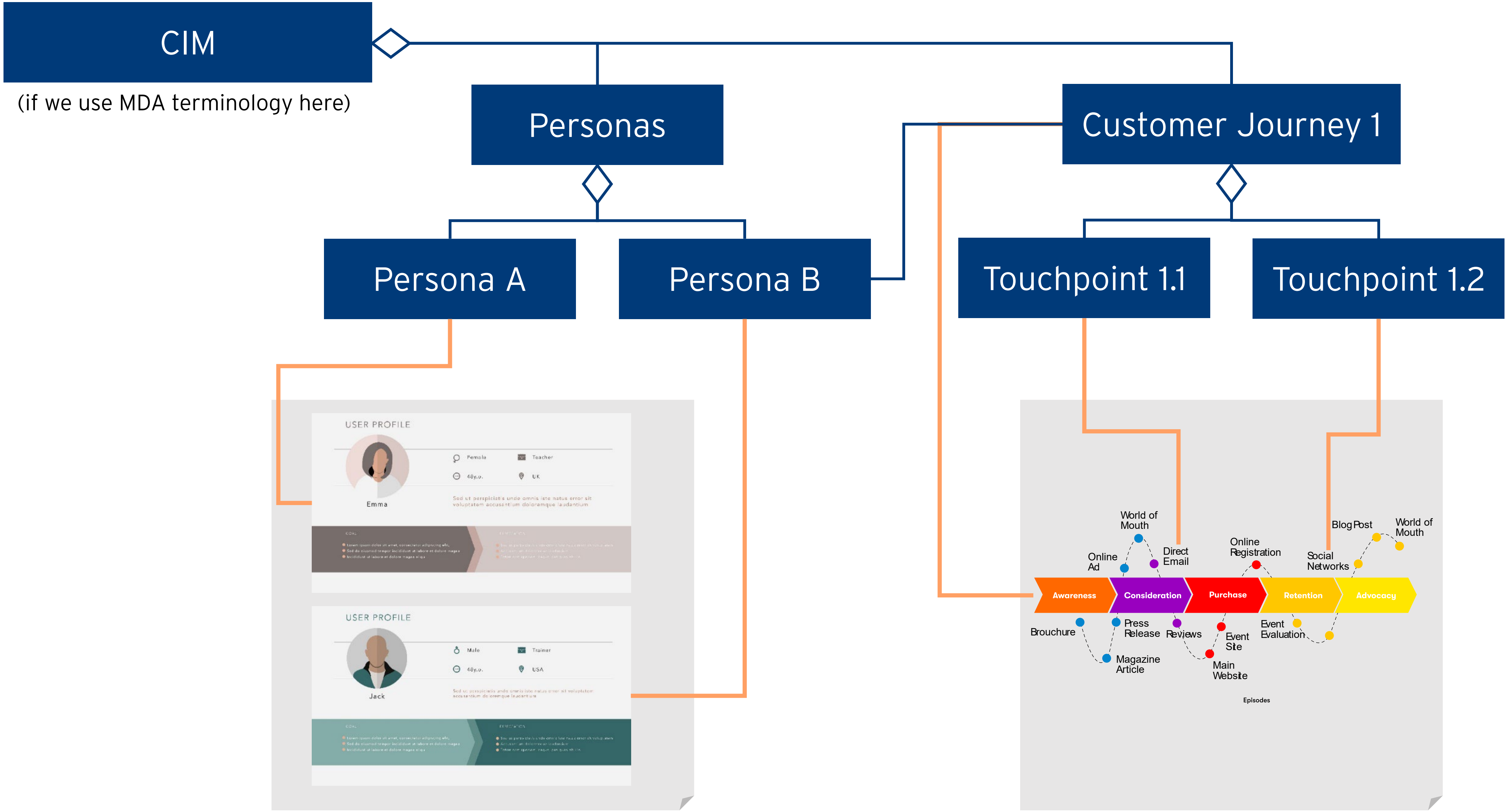
But: **model transformations to describe development steps**

Modeling Stages and Artifacts

Creation stage	Sample model entities on the stage
(Business) Goals	KPIs
	OKRs
Subject domain model	Information architecture
	Interaction design
	Wireframes
	Processes, data flows
Requirements, Conceptualization	Solution hypothesis
	Functional ~
	Non-functional ~
	Customer journeys
	Touch points
Solution architecture	Interfaces
	High-level architecture
	Functional mapping

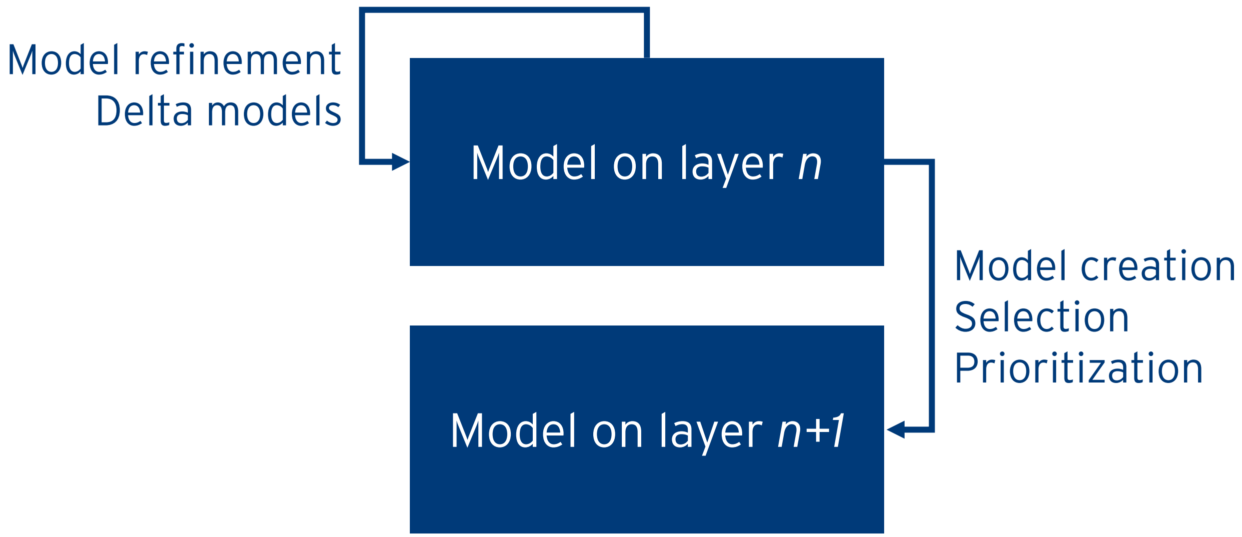
Creation stage	Sample model entities on the stage
Software architecture(s)	Components
	Communication between those components
	Interfaces to the environment
	Constraints of the resulting software system
	Requirements met by the architecture
	Rationale behind architecture decisions
Code	Metaprograms
	Input for software generators
	Domain-specific language expressions
Systems architecture	Infrastructure definition (IaC)
	Automated deployments (CI/CD)
Operations	Service level agreement
	Monitoring

Examples of Description Models for Informal Artifacts



Model Refinement and Transformations

The general theme of model transformations we consider



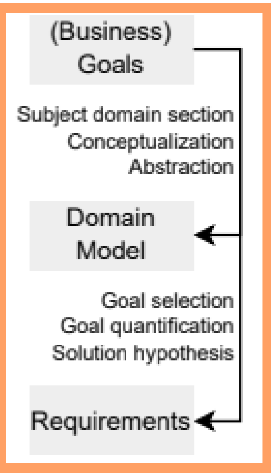
- **Models on one layer are refined** until the result of the corresponding phase
- **Models on a subsequent layer are created** from models of previous stages

Model Refinement and Transformations

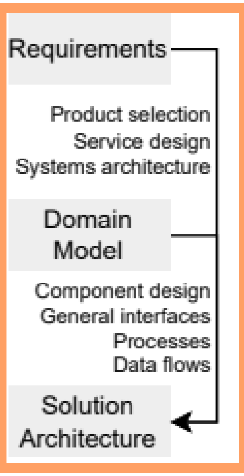
Typical phases of a software creation process and model transformations connecting them.

Creation stage	Sample model entities on the stage
(Business) Goals	KPIs
	OKRs
Subject domain model	Information architecture
	Interaction design
	Wireframes
	Processes, data flows
Requirements, Conceptualization	Solution hypothesis
	Functional ~
	Non-functional ~
	Customer journeys
	Touch points
Solution architecture	Interfaces
	High-level architecture
	Functional mapping
Software architecture(s)	Components
	Communication between those components
	Interfaces to the environment
	Constraints of the resulting software system
	Requirements met by the architecture
	Rationale behind architecture decisions
Code	Metaprograms
	Input for software generators
	Domain-specific language expressions
Systems architecture	Infrastructure definition (IaC)
	Automated deployments (CI/CD)
Operations	Service level agreement
	Monitoring

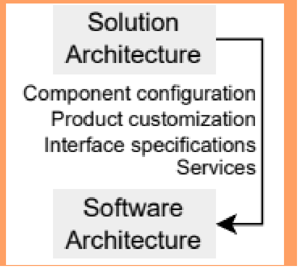
from
Business Goals
to
Domain Model
and
Requirements



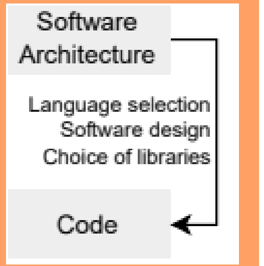
from
Requirements
and
Domain Model
to
Solution Architecture



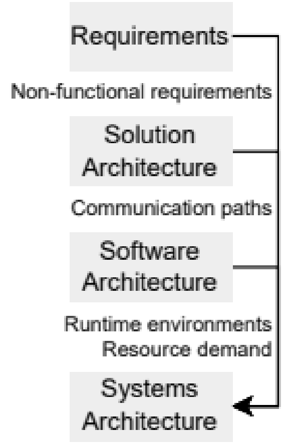
from
Solution Architecture
to
Software Architecture



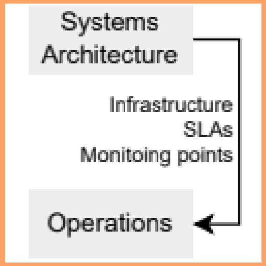
from
Software Architecture
to
Code



from
Requirements
and
Solution Architecture
and
Software Architecture
to
Systems Architecture



from
Systems Architecture
to
Operations



4. A Brief Introduction to the M³L

Basic language constructs. More complete descriptions can be found in the literature.

`A`	The declaration of or reference to a **concept** named `A`
`A is a B` `A is the B`	The **refinement** of a concept `B` to a concept `A`; `A` is a specialization of `B`, `B` is a generalization of `A` (`the`: `A` is the only specialization of `B`)
`A is a B { C }`	Containment of concepts; `C` belongs to the **content** of `A`, `A` is the **context** of `C`
`A |= D`	The **semantic rule** of a concept of a concept `A`; whenever `A` is referenced, `D` is bound; if `D` does not exist, it is created in the same context as `A`
`A |- E F G.`	The **syntactic rule** of a concept `A`; `A` is printed out as or recognized from the concatenation of the syntactic forms of concepts `E`, `F`, and `G`; if not defined, a concept evaluates to / is recognized from its name

M³L Expression Evaluation

```
Person {
  Name is a String }
PersonMary is a Person {
  Mary is the Name }
PersonPeter is a Person {
  Peter is the Name
  42 is the Age }
```

```
Person {
  Peter is the Name
  42 is the Age }
⇒ PersonPeter
```

```
Person {
  Mary is the Name
  42 is the Age }
⇒ Person {
    Mary is the Name
    42 is the Age }
```

The M³L has an operational semantics for expression evaluation

It is based on (any combinations of)

- Refinement
- Semantic rules
- **Visibility** rules
  - All concepts in the content of a concept are also visible in the content of refinements: `A { B }, C is an A` ⇒ `C { B }`
  - All concepts in the content of a concept are also visible in the contents of concepts in the context of that concept: `D E { F }` ⇒ `E { F { D } }`
- **Narrowing**

  If a concept **A** has a subconcept **B**, and if all concepts defined in the context of **B** are equally defined in the context of **A**, then each occurrence of **A** is narrowed down to **B**.

M³L Example: Definition of a Programming Language

Definition of a conditional statement

```
Boolean
True is a Boolean
False is a Boolean

Statement
PrintStatement { Text is a String }

IfThenElse is a Statement {
    Condition is a Boolean
    IfStatement is a Statement
    ElseStatement is a Statement
}
IfTrue is an IfThenElseStatement {
    True is the Condition
} |= TrueStatement
IfFalse is an IfThenElseStatement {
    False is the Condition
} |= ElseStatement
```

Application in a program

```
SomeCondition is a ComputeSomeBoolean { … }

Conditional1 is an IfThenElse {
    SomeCondition is the Condition
    PrintStatement is the IfStatement {
        "It's true" is the Text
    }
    PrintStatement is the ElseStatement {
        "It's false" is the Text
    }
}
```

5. An MSSC Approach with the M³L

M³L concepts represent **different modeling components**

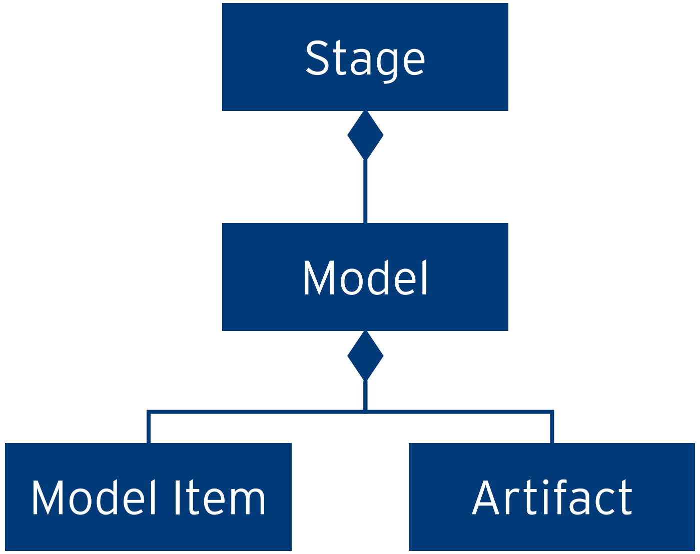
- Topmost concepts represent modeling stages and models
- They contain concepts that represent domain entities
- They relate models and model items to each other

These contained concepts

- May be **stand-alone concepts as model items** for domain entities or
- May represent **artifacts that represent such domain entities**

Model transformations trace the evolution of artifacts created during the course of software creation

Model transformations as considered here can be expressing by the M³L



Dimensions of Model Relationships: Combining Models

In the M³L, concepts are defined in context. Base definitions can be “imported” from foreign contexts.

This way, models on one layer can be defined by selecting model components of a previous layer as a basis.

For example, in an e-commerce application `DomainModel` there may be a definition based on commerce base models

```
ProductDescriptions is a DomainModel {
  ProductData
  PaymentMethods          from Commerce
  PackagingInformation from Logistics
}
```

As an example from another layer, there may be an abstract model of an information system defined as

```
OurInformationSystem
  is a PlatformIndependentModel
{
  AppServer   from SWComponents
  DBMS        from SWComponents
  DataSchema  from DBModeling
  WebServer   from SWComponents
  WebPage     from WebDesign
}
```

Dimensions of Model Relationships: Refining Models

On one layer, models are refined.

In the M³L, model refinement happens along the different axes of M³L models

- by introducing a refined concept of an existing model concept: `OurInformationSystem` → `OurInformationSystemConcept`
- by refining base concepts of a concept: `WebServer` is a `ServletEngine`
- by refining the content of a model concept: `ProductDataSchema` is the `DataSchema`

Example:

```
OurInformationSystemConcept is an OurInformationSystem {
  RDBMS from SWComponents is the DBMS
  ProductDataSchema is an RDBSchema from DBModeling, the DataSchema
  WebServer is a ServletEngine from Java
}
```

Dimensions of Model Relationships: Creating Models

Semantic rules can be used to

- Evaluate concepts
- Assign (operational) semantics to concepts
- Create models in a subsequent stage

Example for the creation of a model on a subsequent stage:
From the software design model of the information system, `OurInformationSystemConcept`, we have a more concrete model of the data layer, `OurInformationSystemDataLayer`, derived by

```
OurInformationSystemConcept |= OurInformationSystemDataLayer {
  RDBMS
  ProductDataSchema {
    ProductsTable is a Table from DBModeling
  }
}
```

Taken over from source concept

Taken over from source concept and content refined

Model Transformation with the M³L 1

On one modeling stage, concept refinements are used to elaborate models

```
ModelOnStage1 {
  DomainConcept { Attribute }
  MoreElaboratedDomainConcept is the DomainConcept {
    Attribute is an AttributeClass }
  EvenMoreElaboratedDomainConcept is the MoreElaboratedDomainConcept {
    SpecificValue is the Attribute
    AnotherAttribute is an AnotherAttributeClass }
}
```

By M³L's contextual definitions and refinements

- All **intermediate modeling** steps are accessible (`DomainConcept`, `MoreElaboratedDomainConcept`)
- The **cumulated model** is available in the most specific context (`EvenMoreElaboratedDomainConcept`)

Model Transformation with the M³L 2

By equipping the more refined concepts of one stage with semantic rules, models in a subsequent stage are initially created

```
ModelOnStage1 {
  EvenMoreElaboratedDomainConcept |= ModelOnStage2 {
    Stage2Concept { … }
  }
} |= ModelOnStage2
```

Definition on newly created model stage 2

Additionally, concepts can refer to concepts of a preceding modeling stage

```
ModelOnStage2 {
  DomainConceptSpecification is the DomainConcept from ModelOnStage1 { … }
}
```

Software Generation from Models 1

In the case of source code generation, software is generated by the M³L using its syntactic rules

Example (assume that `ProductsTable` has content `Columns`):

```
OurInformationSystemDBImplementationSQLOutput is an SQL {
  OurInformationSystemDBImplementation is an OurInformationSystemDataLayer {
    ProductDataSchema {
      ProductsTable |- "PRODUCTS(" Columns ")" .
    } |- "CREATE TABLE " ProductsTable .
  }
}
```

Software Generation from Models 2

With contextual definition of syntactic rules, different output formats can be defined on one model.

Example: generate an external format matching the database schema

```
OurInformationSystemDBImplementationJSONOutput is a JSONSchema {
  OurInformationSystemDBImplementation is an OurInformationSystemDataLayer {
    ProductDataSchema {
      ProductsTable |- "  \"title\": \"Product\","
                       "  \"description\": \"product description\","
                       "  \"type\": \"object\","
                       "  \"properties\": {" Columns "}" .
    } |- "{"
         "  \"$schema\": \"https://json-schema.org/draft/2020-12/schema\","
         "  \"$id\": \"https://example.com/product.schema.json\""
         ProductsTable
         "}" .
} }
```

6. Summary and Outlook

Summary

- Software projects consist of more activities than the software production itself – we need **holistic** processes
- There is a class of software projects that includes activities that lead to the creation of unstructured/informal artifacts; there activities are more creative than they are engineering tasks
- For such projects, a model-driven approach that is based on formal models is not possible
- To benefit from the advantages of model-driven development, models shall **support** the process, though

Outlook

- The **references to artifacts** need to be elaborated; we can build on previous work at this point
- Investigate the utilization of generated models as **checklists** that describe the required artifacts
- Above the topic of this paper, the general modeling with the **M³L in MDSE** will be investigated further
  For example, can it additionally be used as a reasoner or combined with one?

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