# Towards Unified Formal and Creative Software Development

Keynote, Computation World 2024, April 17, Venice

Hans-Werner Sehring, NORDAKADEMIE, Germany
sehring@nordakademie.de

IARIA

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

# Hans-Werner Sehring

## Software Engineering

Working as solution architect, software architect, product owner, and many other roles in commercial and scientific projects

## (Meta) Modeling

Working in the directions of domain modeling, software architecture, and model-driven software engineering

Interested in modeling layers

## Content Management

In both science and industry working on digital communication that is based on digital media

One focus: content management systems

## Teaching

Professor for Software Engineering

Currently teaching courses on

- Software engineering
- Programming (languages)
- Theoretical computer science

keynote speaker

# Introduction (1)

**Model-driven Software Engineering** (**MDSE**) has obvious advantages, most notably

- traceability
- increase of software quality (w.r.t. modeled properties)
- automation (assuming that programming cost exceeds modeling cost)

In practice, an important class of systems is built in **creative software development** processes
These do not integrate well with MDSE

- **subjective decisions:** from goals to requirements to software specification
- **outside-in perspective:** user-centric research, feeback in incremental development, user acceptance tests
- **media mismatch:** from user interface design to frontend code

Current research interest: how to integrate these two contradictory kinds of software construction

# Introduction (2)

**Recent interest? No!** Problem exists since the 1990s (personal view)

- advent of the WWW
- professions like communication designers, ..., UX designers

Perhaps much longer

Creative process made some progress since then
user-centric design, UX design, design sprint

Software development also, but: mainly optimization, scaling, and increase in quality

Claim: software development does not adapt to that situation, in particular MDSE

Additionally, there seem to be other points were simple MDSE falls short,
leading to the idea of **holistic Model-Supported Software Creation**

# Agenda

01 Previous and Ongoing Work

02 Model-Driven Software Engineering

03 Creative Software Development

04 Towards Holistic Model-Supported Software Creation

05 Conclusion

# Section 1

# Previous and Ongoing Work

# Starting Point: Previous Research

## Content Personalization

To express scientific work that is based on subjective views, e.g., in the humanities, a system for knowledge representation using (digitial) media was studied: **Concept-Oriented Content Management** (**COCoMa**)

Software generation and evolution-friendly software architecture where studied in support of the vast personalization requirements of COCoMa

## Meta (Meta) Modeling

The **Minimalistic Meta Modeling Language** (**M³L**) was originally intended as a textual language for MDSE

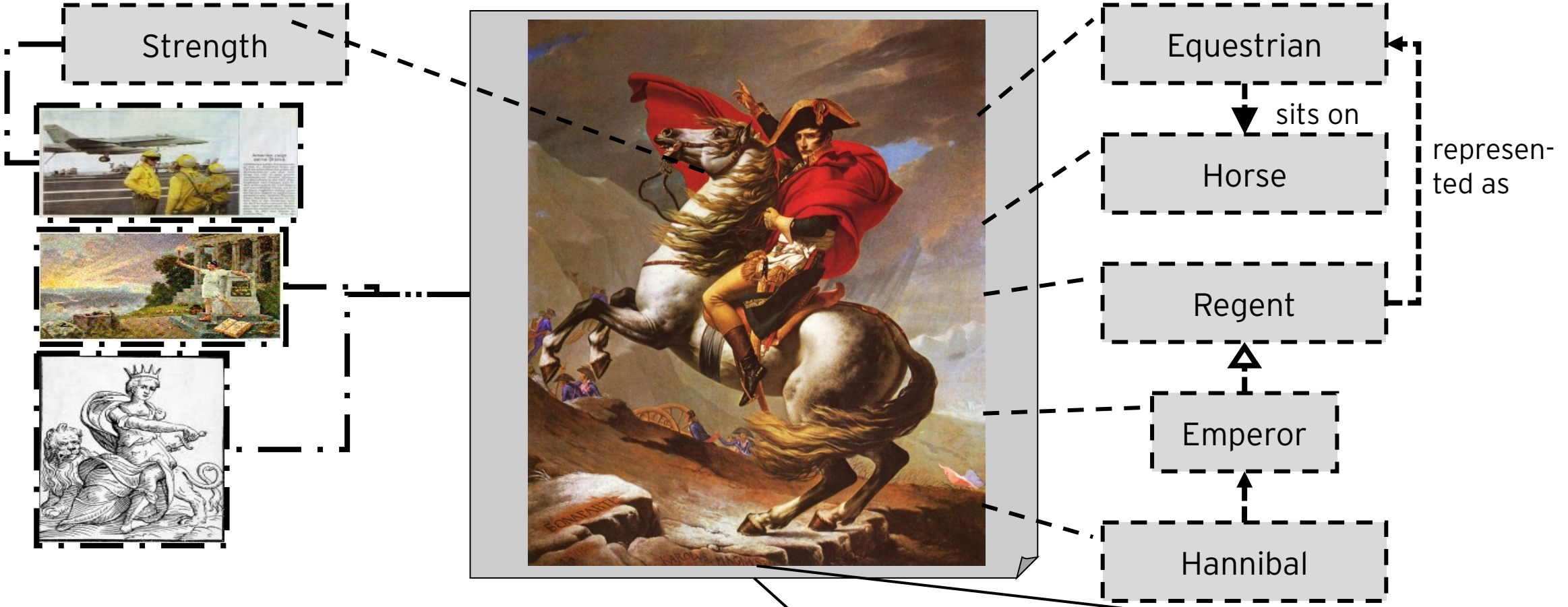
It turned out to be applicable in other areas as well, for instance, for content management tasks

## UI Specification and Generation

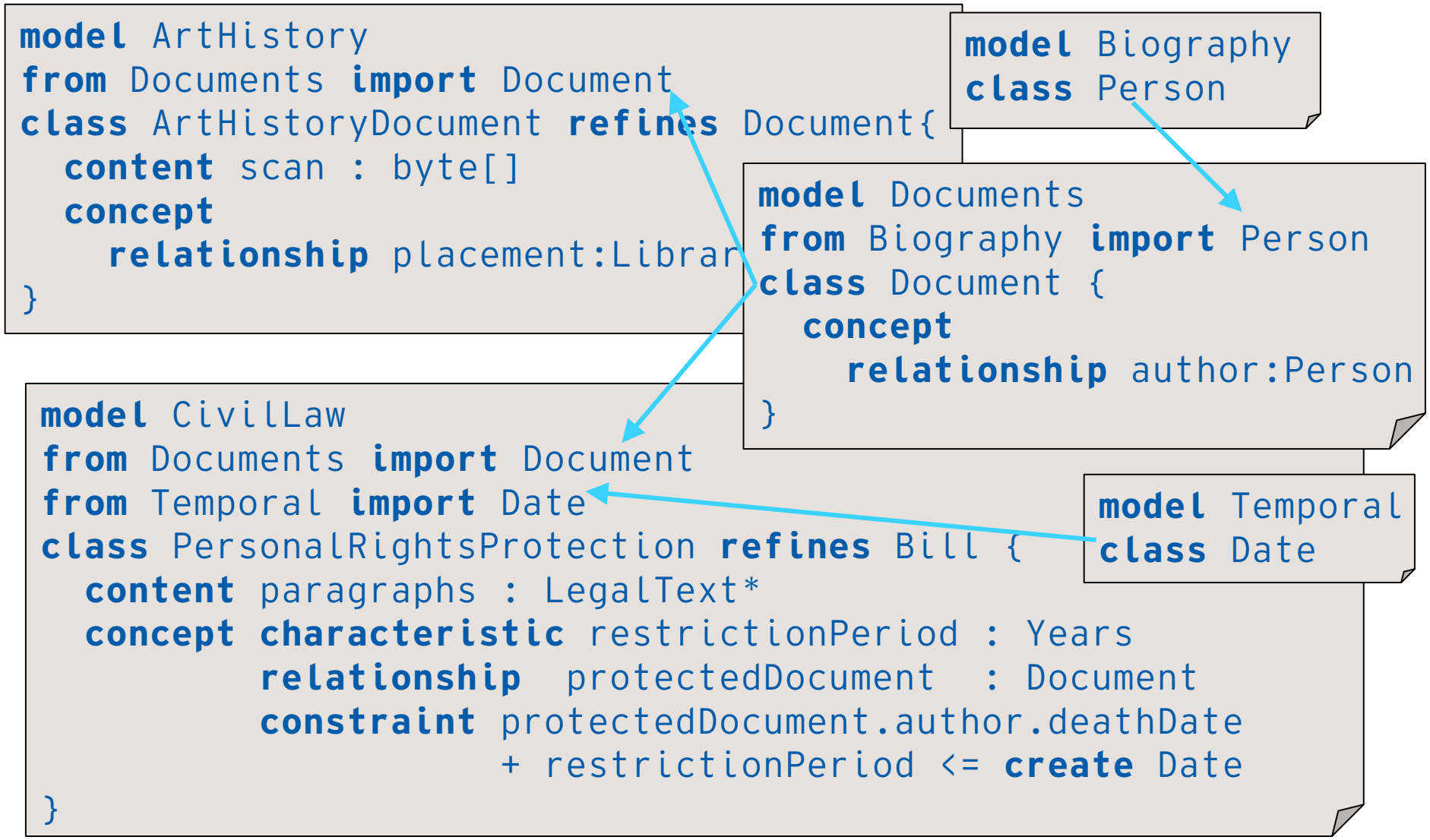
Work in generic UIs and in UI generation provides insights into working with visual prototypes

Additionally, there is experience from a variety of commercial software projects in the **digital omnichannel** domain

# (Personalized) Content for Entity Description: COCoMa



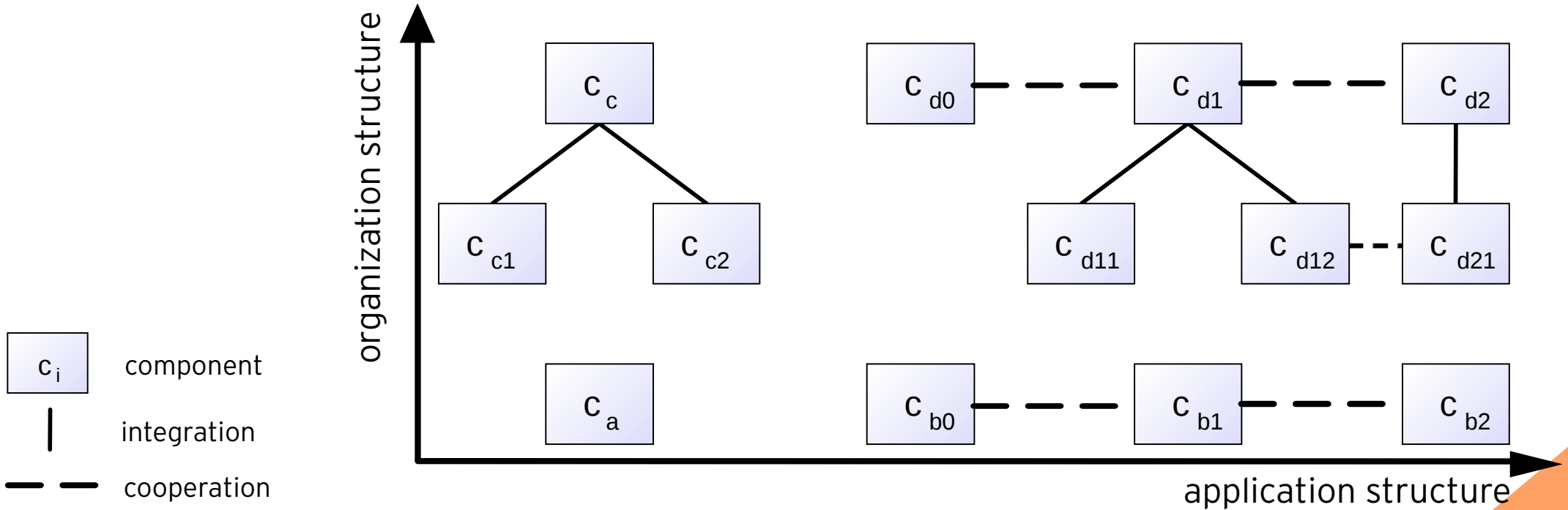
# A Multi-Domain Model in COCoMa



# Logical COCoMa Architecture: Components

COCoMa systems consist of components

- one component for each domain model
- cooperation for domain combinations
- integration to model revisions and to achieve personalization

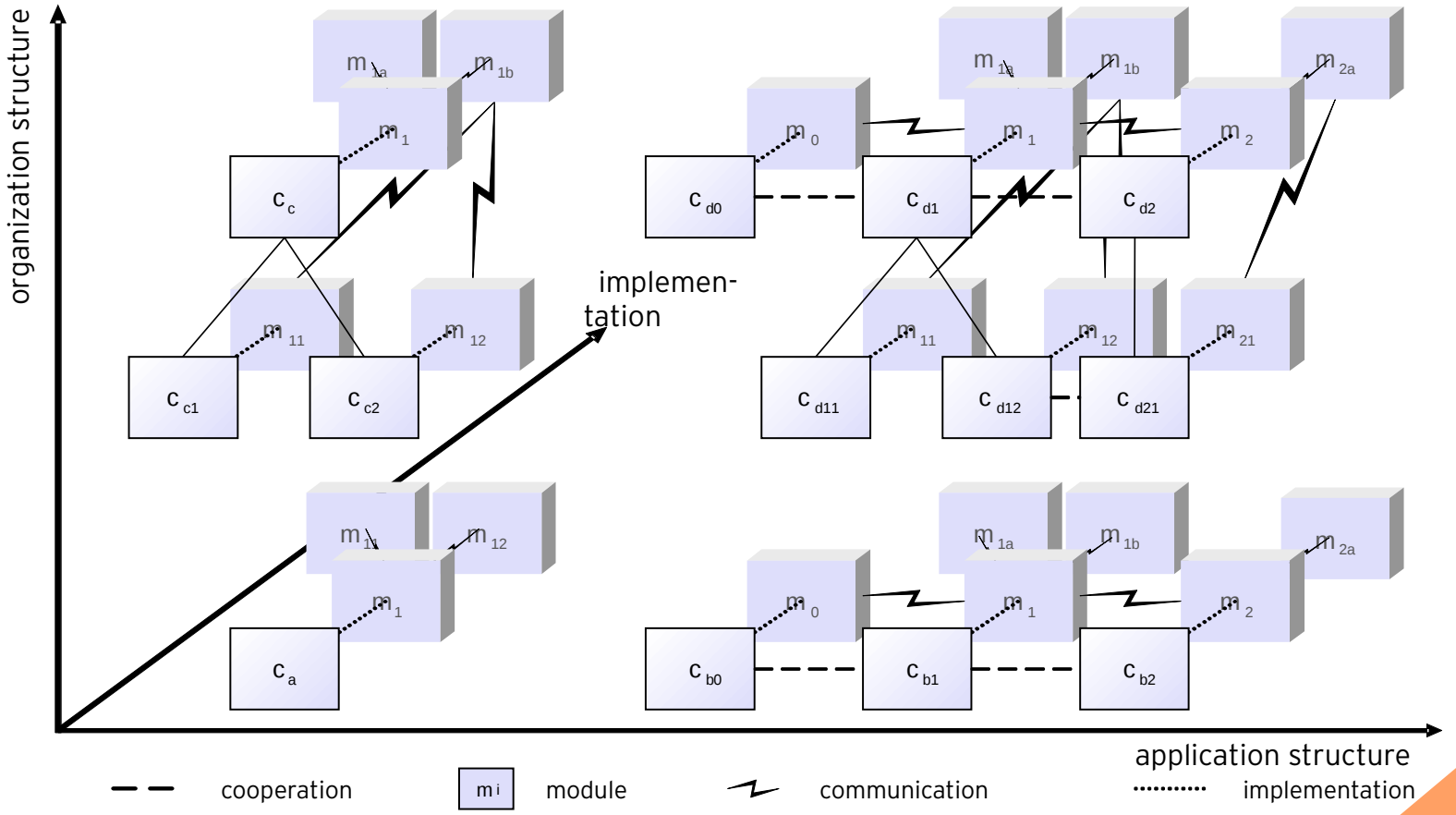


# COCoMa Systems Implementation: Modules

Modules implement components

Reconfigurability at runtime for dynamics through:

- Separation of Concerns by module kinds with distinguished functionality
- uniform module API
- statelessness of modules



# The Minimalistic Meta Modeling Language (M3L)

The **Minimalistic Meta Modeling Language** (**M³L**) will be presented in more detail later

Key features:

- **Meta** is relative; one language spanning from meta (meta) modeling to instance descriptions
- Puristic **syntax**, small (minimal?) set of built-in features, simple **semantics**
- **Variants** and **contexts** as the primary idioms
- Dynamic **evaluation**
- Wide range of **application scenarios**; particularly interesting
  - Software modeling
  - Content modeling

# 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
    }
}
```

# Syntactic Rules for External Representations

The **syntactic rules** for external representations serve as

- templates to print out concepts
- grammars to parse in concept representations

Example:

```
IfThenElse
|- if Condition
   then IfStatement
   else ElseStatement
```

Note: concepts are represented by their name by default

By means of contextualization, different representations can be defined on concepts

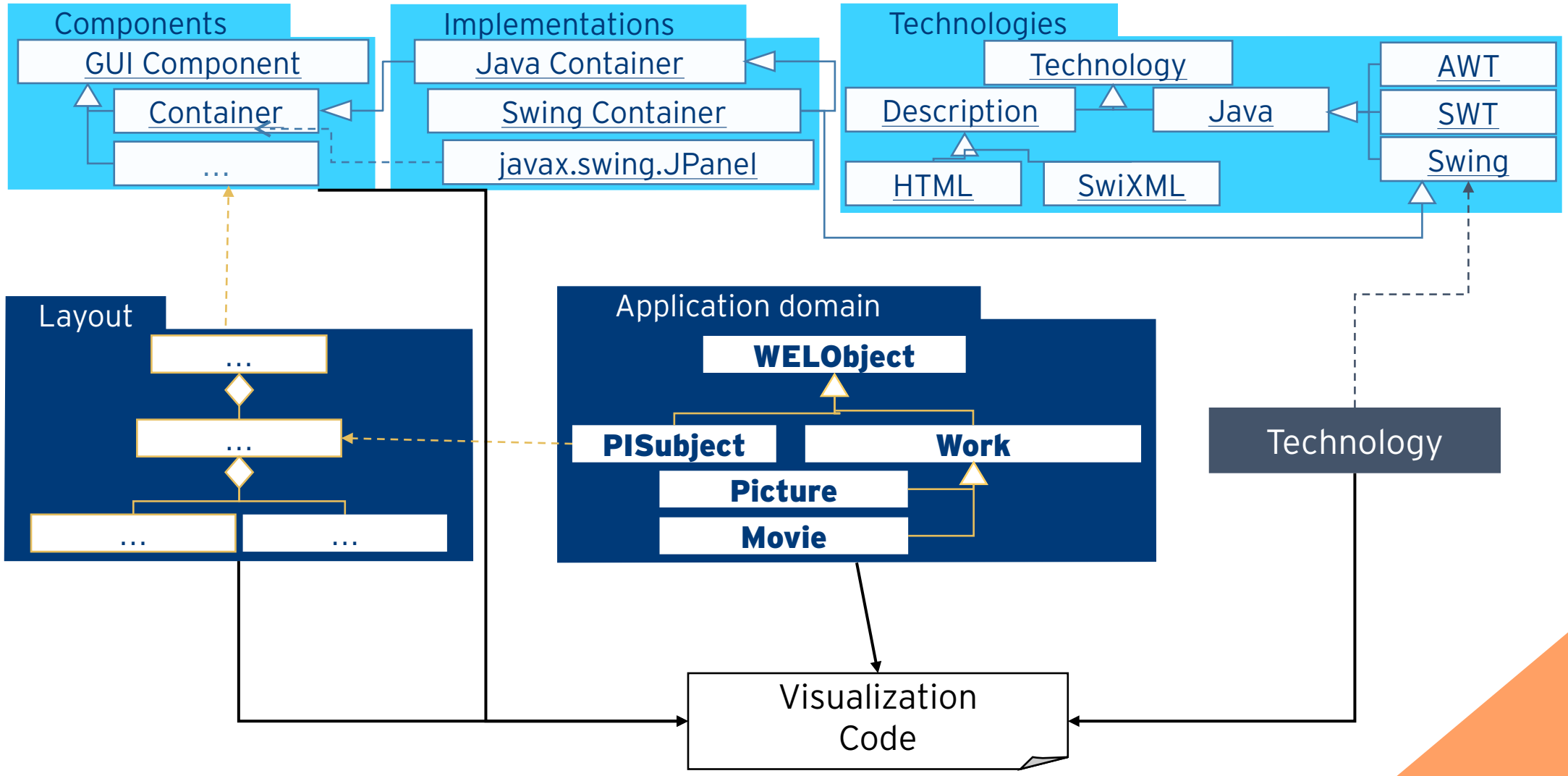
```
Java is a ProgrammingLanguage {
  IfThenElse |- if ( Condition )
                IfStatement
                ElseStatement

}
Python is a ProgrammingLanguage {
  IfThenElse |- if Condition :
    Newline Indentation IfStatement
    else:
    NewLine Indentation ElseStatement
}
```

# Models for Visualizations

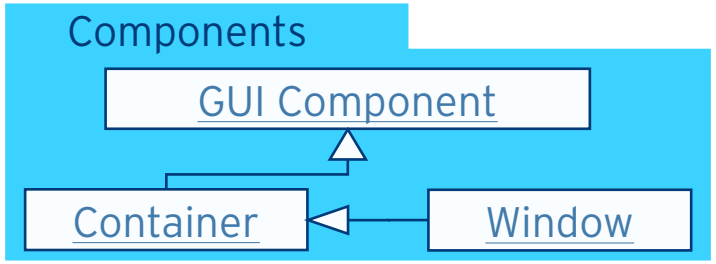
Visualization environment

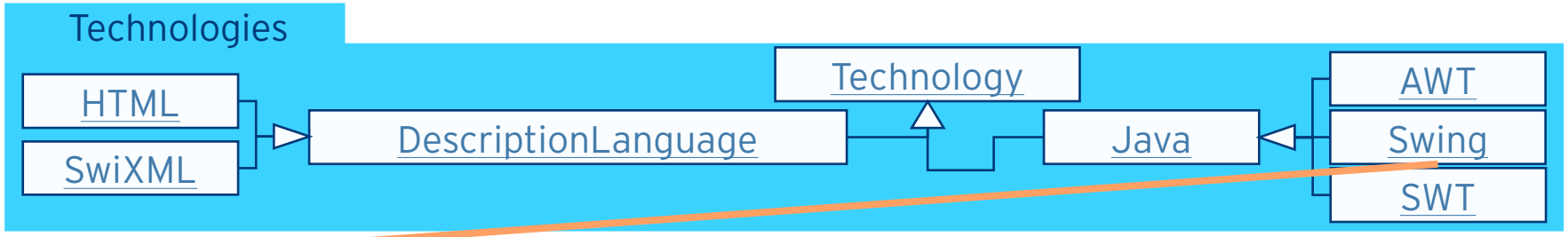
Domain expert

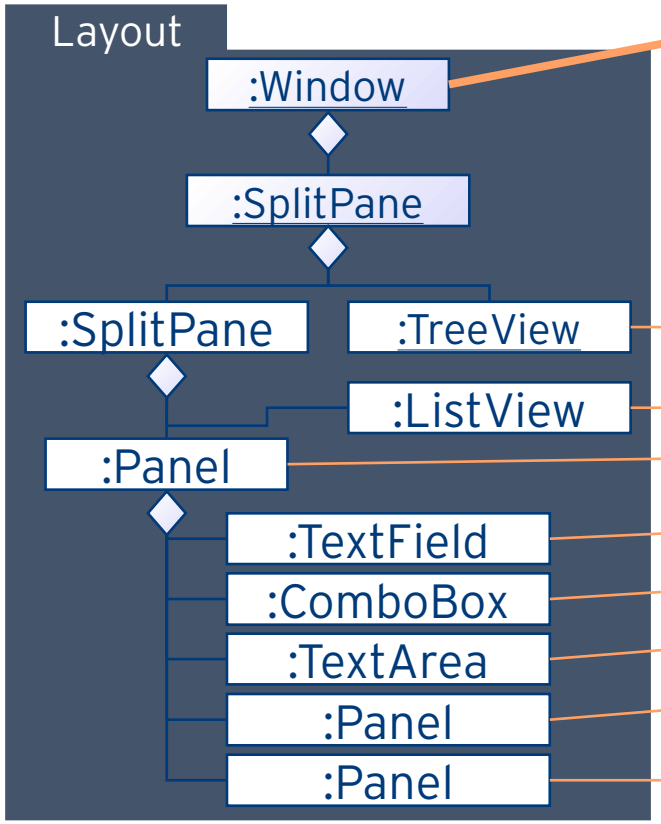
(CCM) Compiler

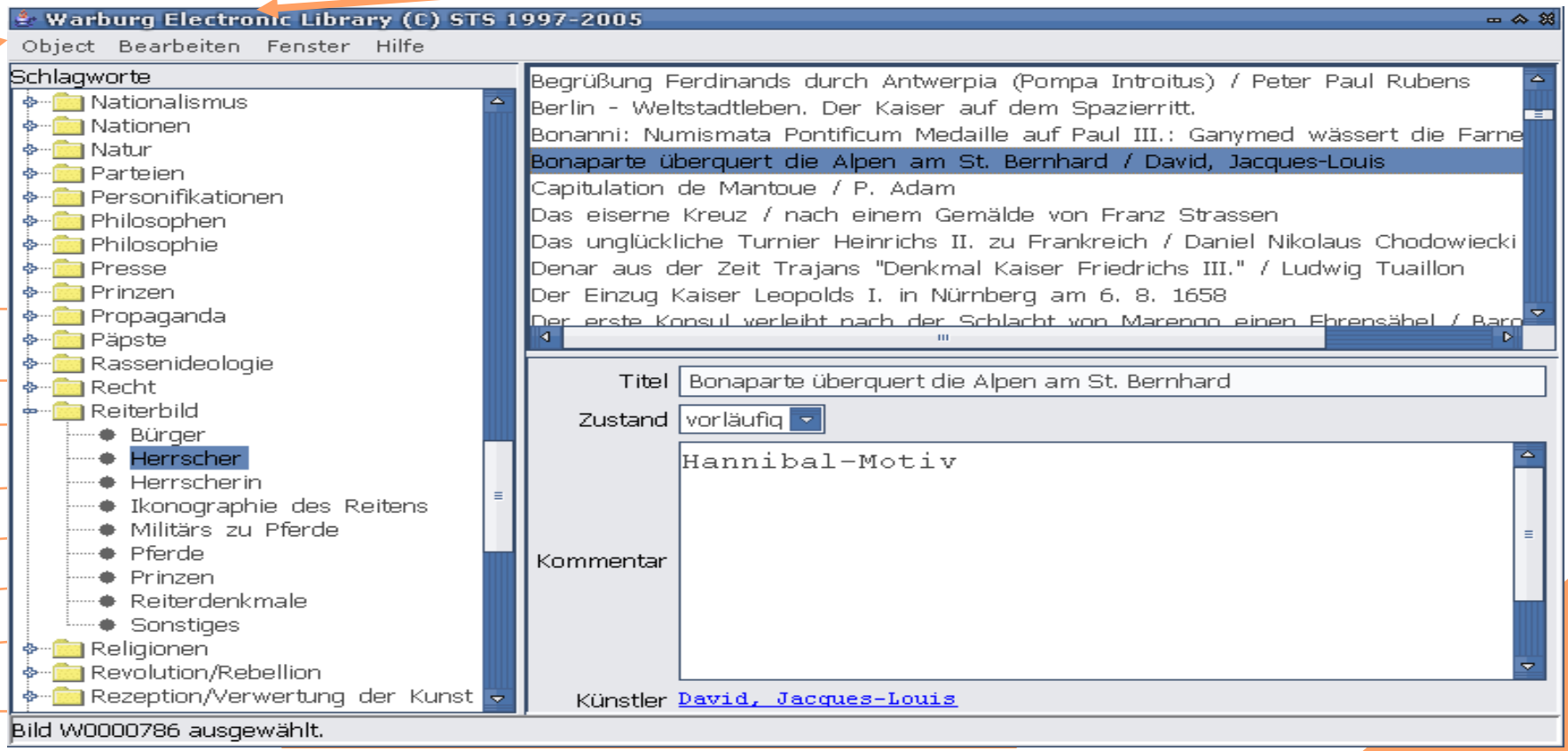


# Example: Rich Client for a Digital Library

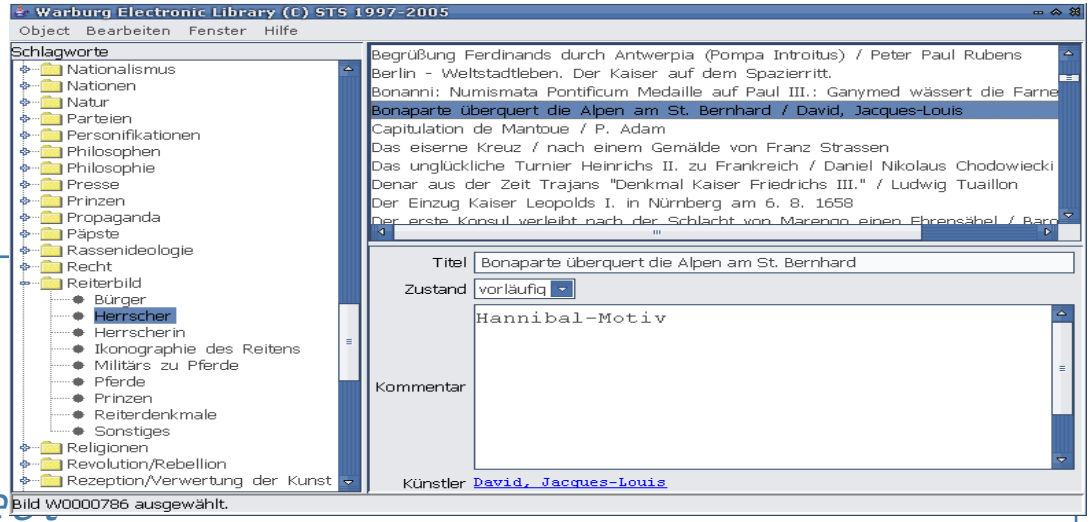


# Declarative Interaction Description




```
class WorkList refines ListView {
 concept ; "visualizedInstance" inherited
         relationship selected :Work
         relationship subjTree :PISubjectTree
         ; always show extent of selected PISubject
         constraint visualizedInstance=subjTree.selected.extent
           onviolation modify self {
             visualizedInstance := subjTree.selected.extent
           }
         ; clear selection if model changed
         constraint { selected } <= subjTree.selected.extent
           onviolation modify self { selected := na }
}
class PISubjectTree refines TreeView {
 concept relationship selected :PISubject
         …
}
```

# Section 2

# Model-Driven Software Engineering

# Outline

**Model-driven Software Engineering** (**MDSE**) and

**Model-driven Software Development** (**MDSD**)

(we use these terms synonymously – some make a distinction)

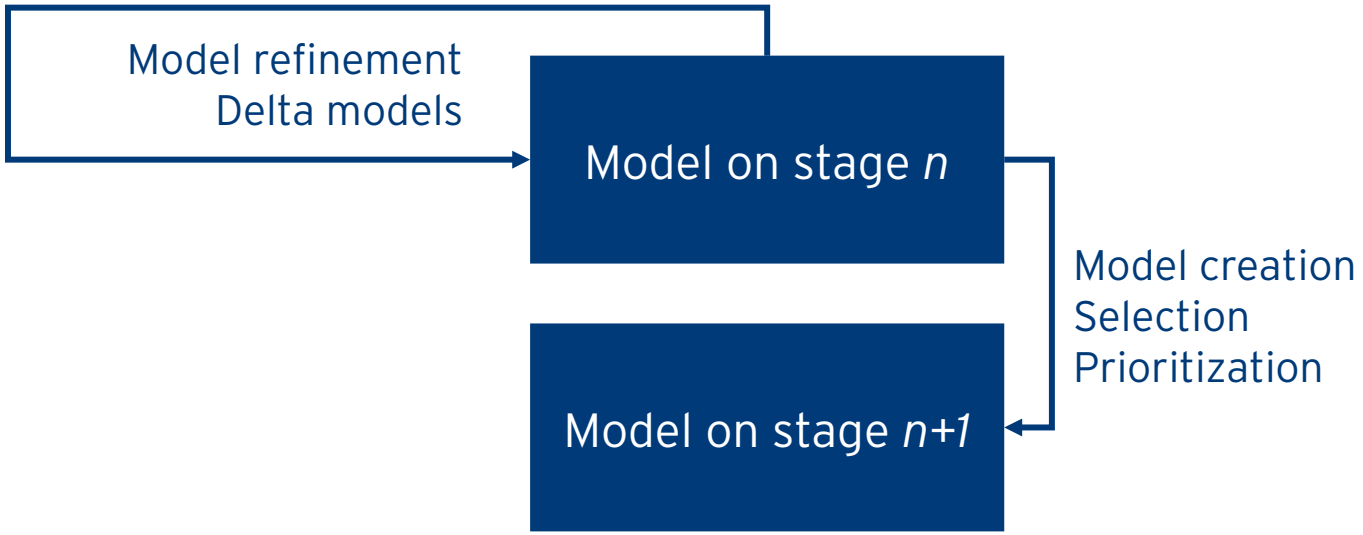
aim at building software from a series of (formal) models, where these models are related to each other by model transformations.

Goals are, among others,

- **validity** / possibility of validation by some degree of **formalism**
- **automation** by formal **model transformation** and **code generation** steps
- **traceability** (of artifacts) through comprehensible **modeling** steps

# Model Refinement and Transformations

The general theme of model transformations we consider




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

# Model-driven Software Engineering Techniques

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

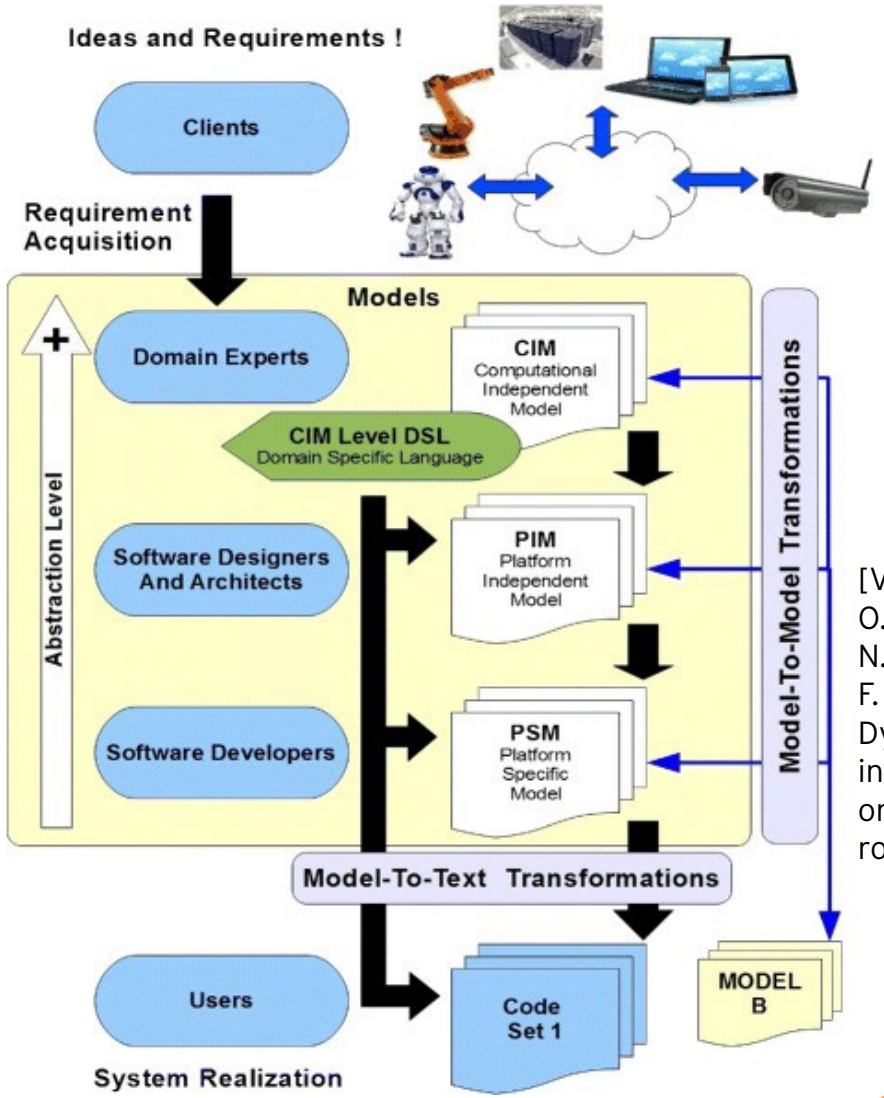
- **Model-driven Architecture (MDA)**
- **Software Generation**
- **Domain-specific Languages (DSLs)**
- **Generic Software**
- **Theoretical/scientific Approaches**

# Model-driven Architecture (MDA)

Early MDSE approach proposed by the Object Management Group (OMG)

Models are created on (originally) three levels of abstraction

1. A *Computation-Independent Model* (*CIM*) from the perspective of the subject domain
2. A *Platform-Independent Model* (*PIM*) as a first formal model
3. Transformed into a *Platform-Specific Model* (*PSM*) used to generate a working implementation




[V. Nagrath, O. Morel, A. Malik, N. Saad, and F. Meriaudeau: Dynamic electronic institutions in agent oriented cloud robotic systems]

# Software Generation

The solution model is contained in code

Different approaches, for example,

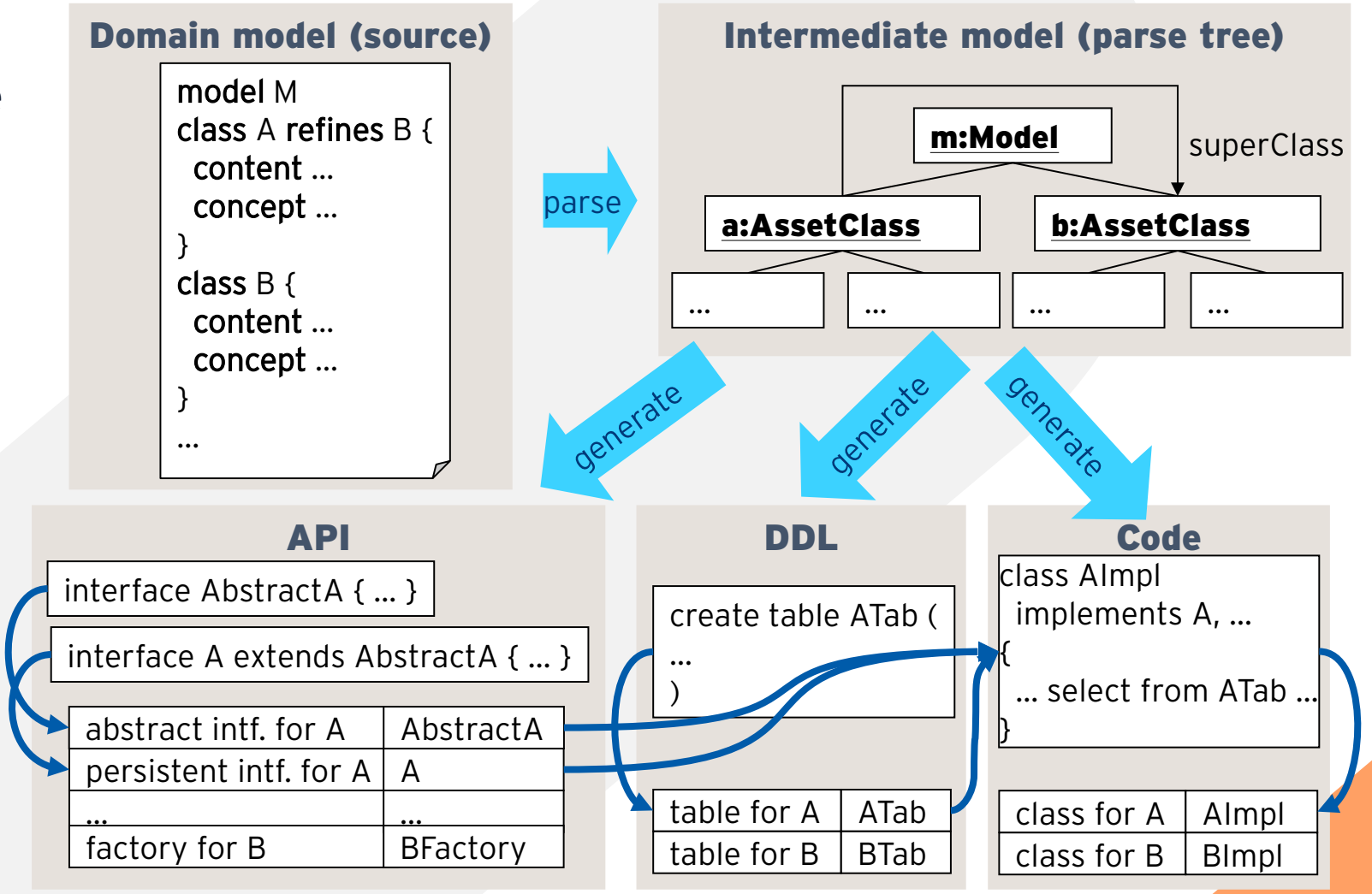
- metaprogramming
- templates
- generative AI

# Software Generation – Metaprogramming

The solution model is contained in code

Different approaches, for example,

- **metaprogramming**
  - Programs that write programs
  - Previous work: software generators for COCoMa
- templates
- generative AI



# Software Generation – Templates

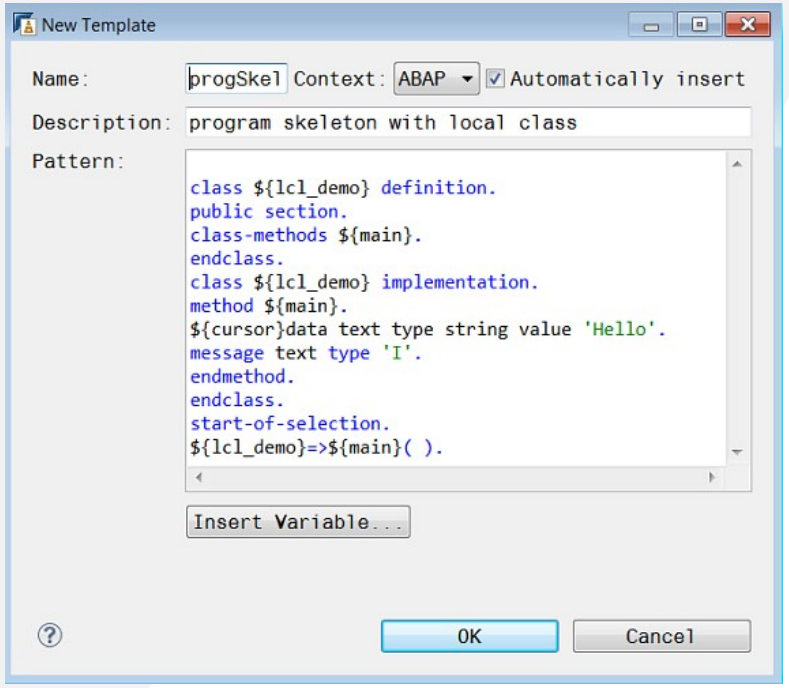
The solution model is contained in code

Different approaches, for example,

- metaprogramming
- **templates**
  - template provide a code skeleton that is filled with actual statements or values
  - a template is applied multiple times with different parameters
- generative AI

Code templates can be found in various software tools.

Example:




[SAP Documentation, ABAP Development User Guide]

# Software Generation – Generative AI

The solution model is contained in code

Different approaches, for example,

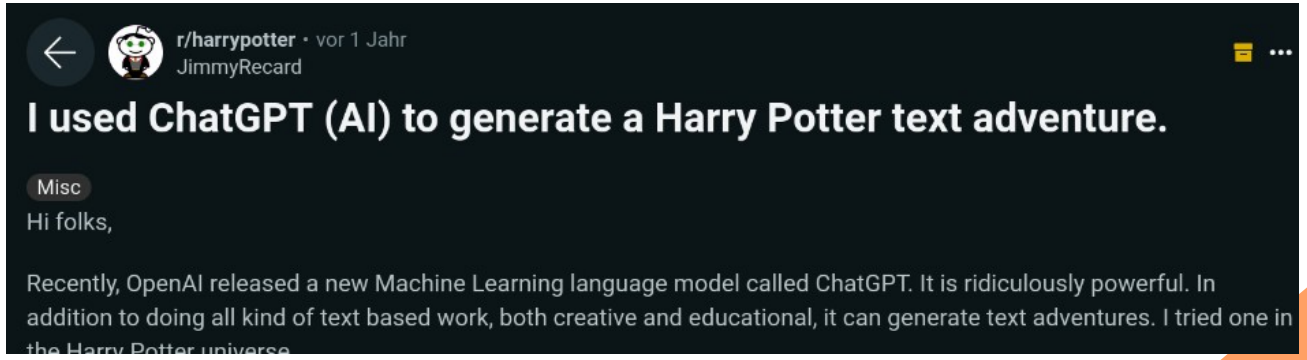
- metaprogramming
- templates
- **generative AI**
  - Fully automated generation, where the functionality is described using natural language
  - Assisted programming, where coding is done manually, an AI completes and improves the code

Modeling is twofold

- Training AI: model of software out of existing solutions
- Generating software: the actual solution is described by the expected behavior, formulated in natural language

Current state:

- Models cannot (fully) be checked
- Traceability is limited




[https://www.reddit.com/r/harrypotter/comments/zdcdom/i_used_chatgpt_ai_to_generate_a_harry_potter_text/]

# Domain-Specific Languages (DSLs)

**Domain-Specific Languages** (**DSLs**) built with language construction tools

Defined for a specific domain to provide linguistic means for typical problems

More abstract (concerning implementation) than programming languages

Trade-off:

- generality vs. specificity
- granularity of language expressions; how much code generation is built in?

Languages ≙ Metamodels – DSLs are built from domain models (though not always visible)

# Generic Software

**Generic Software** is not generated at all

One-size-fits-all solution, possibly configurable

In a sense, also generic software is created in a model-driven way

- A **domain model** was used during the development of the software
- Functionality should also have been modeled; only the requirements
- If the model is parameterized, then software is configurable (low code / no code development)

In how far models are made explicit

Traceability is not given / not applicable

# Theoretical/scientific Approaches

There are scientific mathematical approaches to software construction, for example,

## Proofs

Basic idea:
If the correctness of a software with respect to the requirements can be proven,
and if the proof is constructive,
then the proof correctly creates the software

## Category Theory

Using category theory, model refinements are expressed as morphisms, and a pushout adds model properties to the next modeling stage

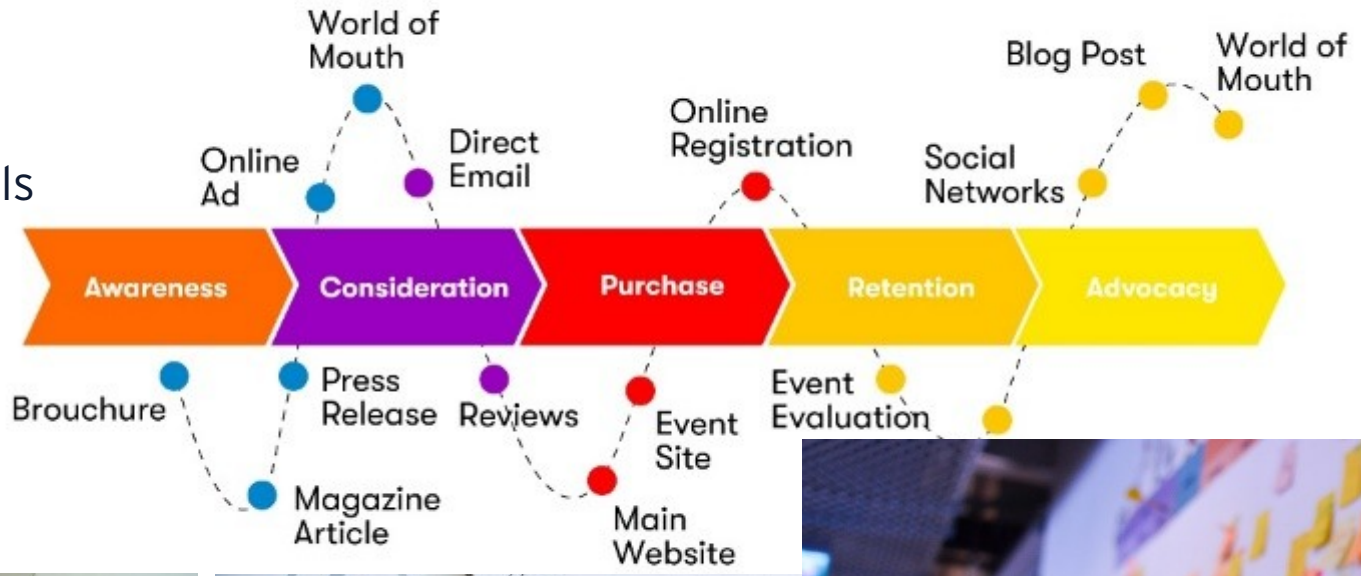
# Section 3

# Creative Software Development

# MDSE in Practice

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

Software engineering **reality** (at least in some domains), see modeling stages and artifacts:




# Cases of Creative Software Development

Class of solutions whose development includes **creative tasks** that lead to non-formal, often **visual artifacts**:

- carried out by **domain experts**
- … using heterogeneous modeling artifacts: varying **degrees of formalism, ambiguity, detail**, etc. …
- … with a **methodology** or a **tool**: notation and representations matter …
- … for **communication** with (non-technical) stakeholders

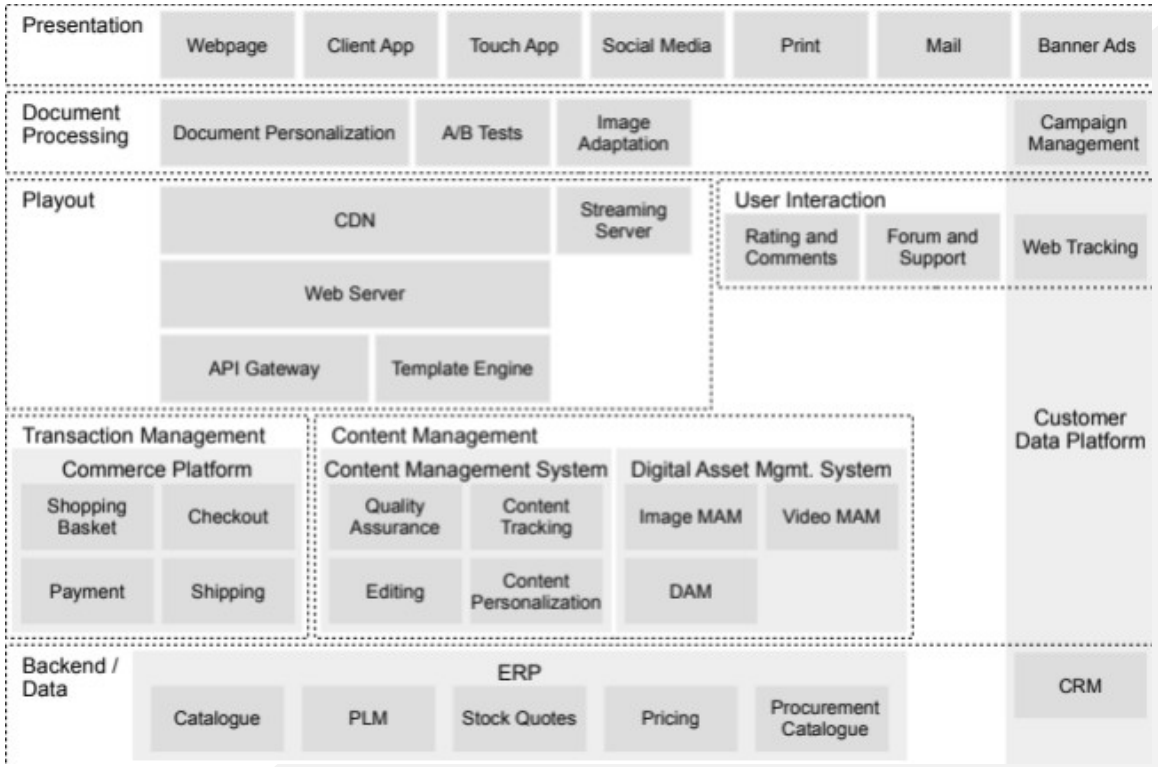
This happens in several project stages, **not only software** (engineering) related ones

**Creative software development** (no agreed name for this kind of software development)
One major creative activity is **UX design** (also no agreed term)

- requirements analysis, target group analysis
- domain modeling
- information architecture, conceptualization, …
- etc.

# Typical Architecture of Communication Infrastructure

Contemporary systems exhibit a fair amount of complexity




Examples:

- Content management functionality is provided by a CMS and a DAM
- User interactions consist of ratings, comments, forums, and a support database, and they are measured by web tracking
- CDP is built from campaign management, web tracking, and CRM components
- Commerce functionality drive transactions, provide data, etc.

A **system of systems**

# Fast-paced Change in Business Requirements

Digital business, in particular, has some particularly **challenging** implications, for example,

- **Outside-in view**: requirements dictated by customer expectations, competitors, technical possibilities
- **Time-to-market**

Time of relaunches is over (for quite a while)

- Base systems are introduced and **sustainable**
- **Continuous improvement** of interfaces, processes, …
- Instead: Deploy, Measure, Improve

**Incremental development:** traceability (as well as other features provided by MDSE) is key

# Modeling Phases and Artifacts

| Phase | Order | Discipline | Artifact |
|---|---|---|---|
| Inception / Research | | Management | Goals |
| |  | Management | Inception |
| |  | Concept | Requirements (inside-out) |
| | | Concept | Research (outside-in) |
| Analysis |  | Concept | Personas |
| |  | Concept | Customer journeys |
| |  | Technology | Existing tools |
| | | Technology | Information demand / data flows |
| 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 |
| Implementation |  | Technology | SW arch (if not agile) |
| | | Technology | System arch (if not agile) |
| |  | Technology | Code design |
| | | Technology | Code |
| | | Concept | Test cases |
| | | Technology | Test scripts |
| | | Concept | Documentation |
| Operations |  | Technology | Infrastructure |
| | | Technology | Build and deploy scripts |
| | | Concept | Training |

# Modeling Stages

On top of the classical, coarse-grained development phases (inception, analysis, design, implementation, operation), we see the following (modeling) **stages** that each have their own contributions

1. **(Business) Goals:** definition of a new state / an achievement / … that is aimed at
2. **Subject domain model:** an abstraction of the domain at hand, identification of a solution area
3. **Requirements, conceptualization:** a description of the (software) solution in application domain terms
4. **Solution architecture:** a first description of the software solution implementation
5. **Software architecture(s):** refined specifications of different components of the software solutions
6. **Code:** the actual implementation of the software solution
7. **Systems architecture:** a specification of the infrastructure for the operation of the software
8. **Operations:** the implementation of the infrastructure, operation and maintenance of the software

# Artifacts in Creative Software Development

| 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 |

# Sample Development Artifacts and Formalisms

| Phase | Order | Discipline | Artifact | Formal(izable) model |
|---|---|---|---|---|
| Inception / Research | | Management | Goals | (X) |
| | 🔄 | 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 various process steps and artifacts that are

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

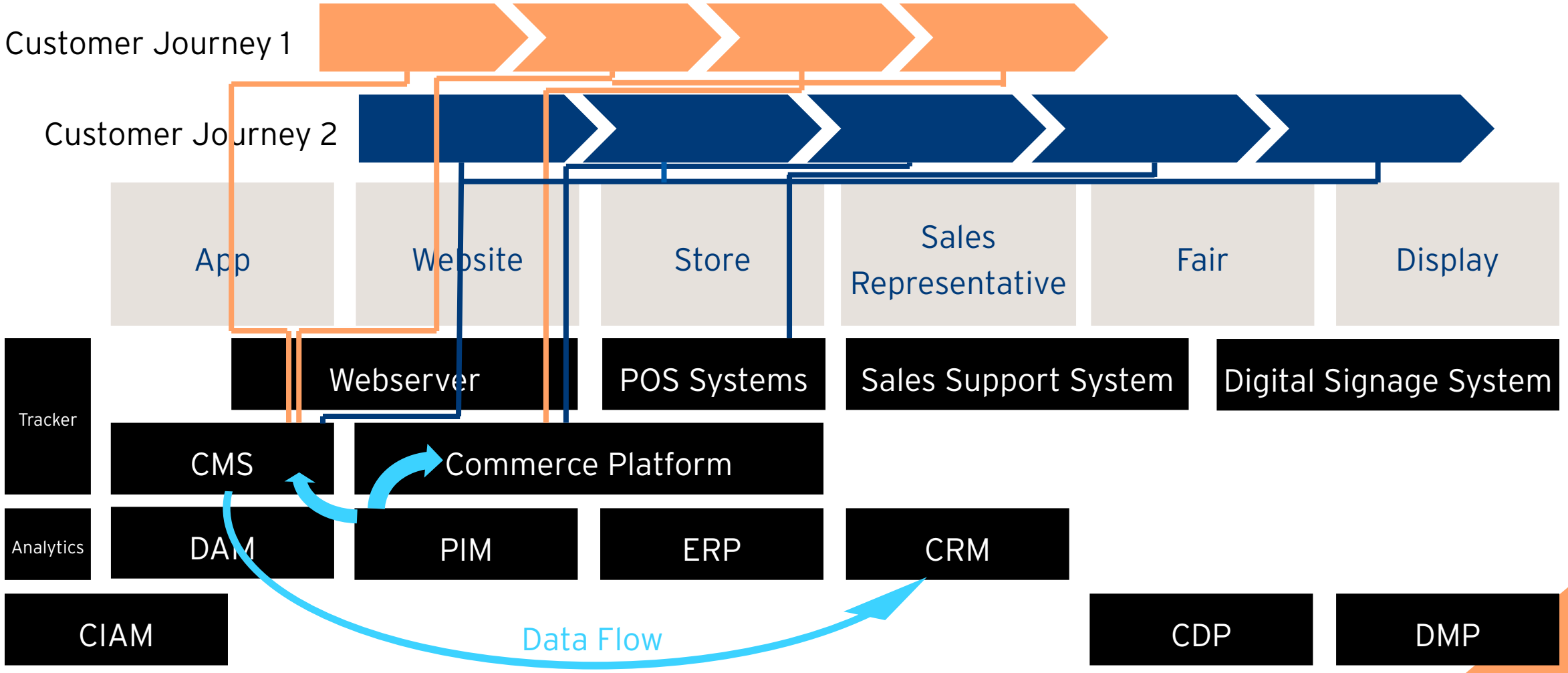
**We cannot have MDSE**.

Still, we want …

- support in managing system (of systems) complexity
  - support in managing (modeling) artifacts
  - checks on models
- quick reactions to changing requirements
  - deriving software from specifications
  - traceability
- etc.

**We want the benefits of MDSE**.

# Creative Output Leads to Formal Specifications



# Section 4

# Towards Holistic Model-Supported Software Creation

# Holistic Model-Supported Software Creation (MSSC)

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

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

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

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

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

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

# MSSC Investigations

We currently investigate two kinds of model support for software creation

1. **description of (informal) artifacts by (formal) models**
2. **generation of artifacts from models**
3. **generation of prototypes from models**

We also take first steps toward **software generation** from such models

# Section 4.1

# Descriptive Model Transformations

# First Experiments

We experiment with models that both represent

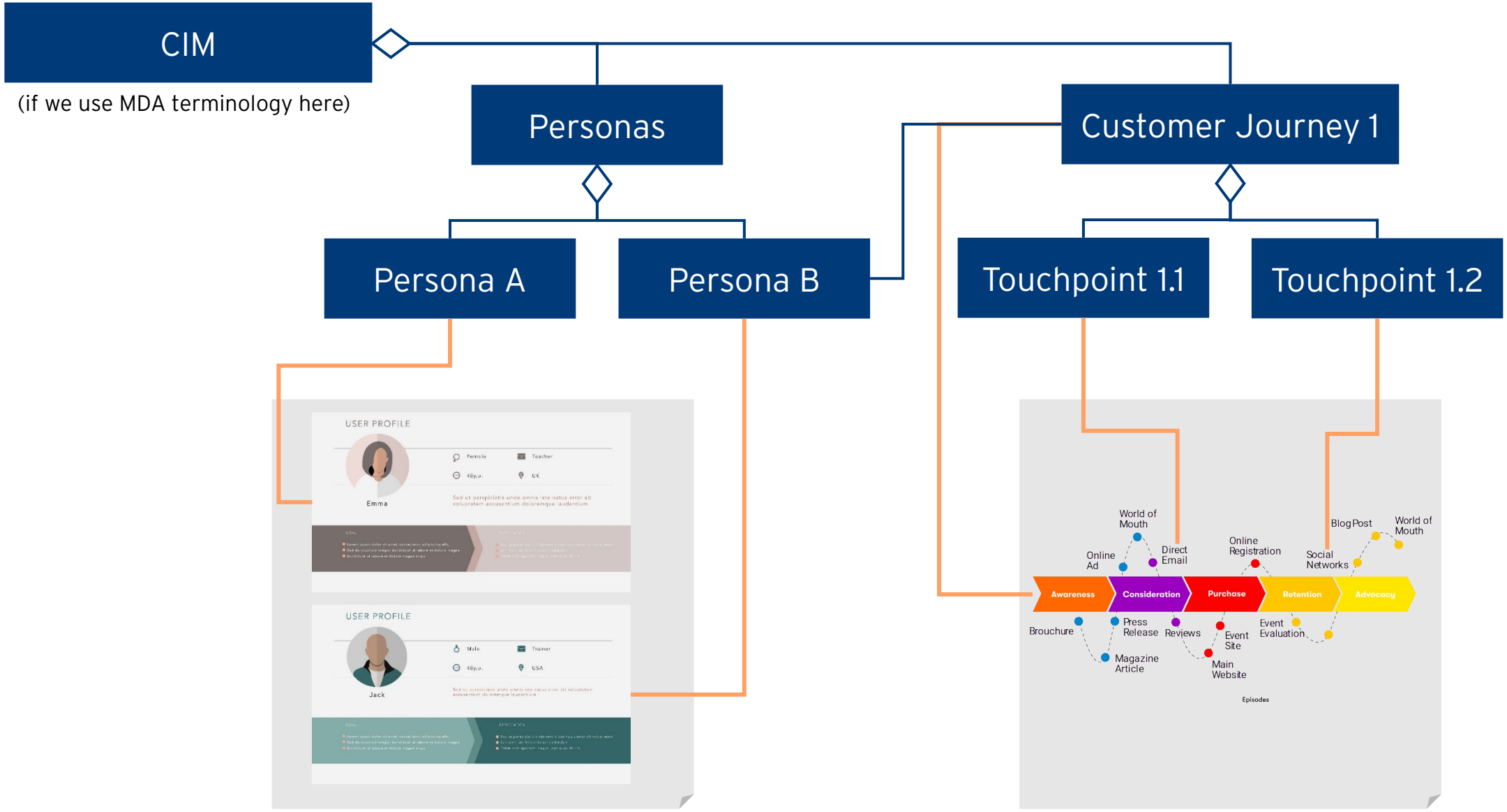
- **models that specify** the software to be built and
- **artifacts that describe** the software

and that allow expressing

- **model transformations within one stage** and
- model transformations to **enter a subsequent modeling stage**

Question: When is work on an artifact isomorphic to a model transformation?

# Examples of Description Models for Informal Artifacts

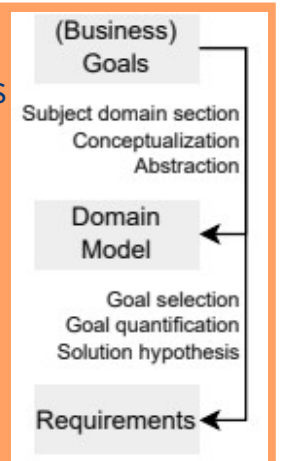


# Model Refinement and Transformations

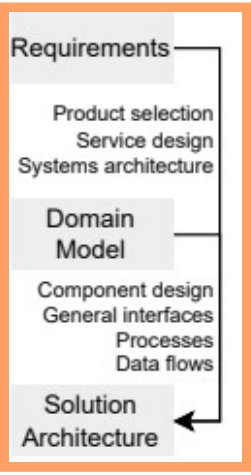
Typical phases of a software creation process and model transformations connecting them.

| 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 |

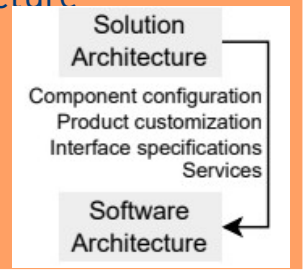
from Business Goals to Domain Model and Requirements



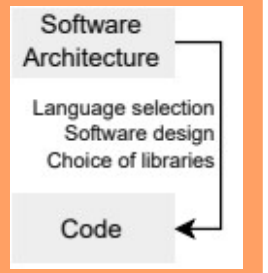

from Requirements and Domain Model to Solution Architecture



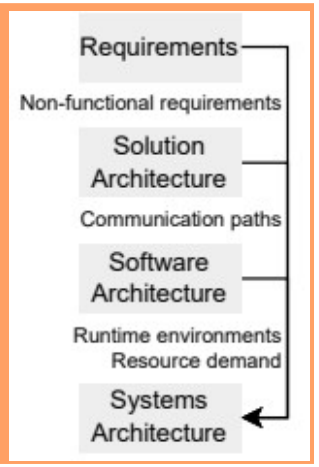

from Solution Architecture to Software Architecture



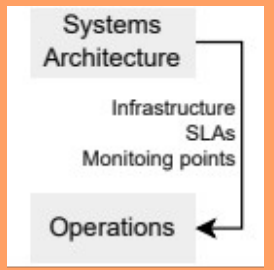

from Software Architecture to Code




from Requirements and Solution Architecture and Software Architecture to Systems Architecture




from Systems Architecture to Operations



# An MSSC Approach with the M³L

M³L concepts represent **different modeling components**

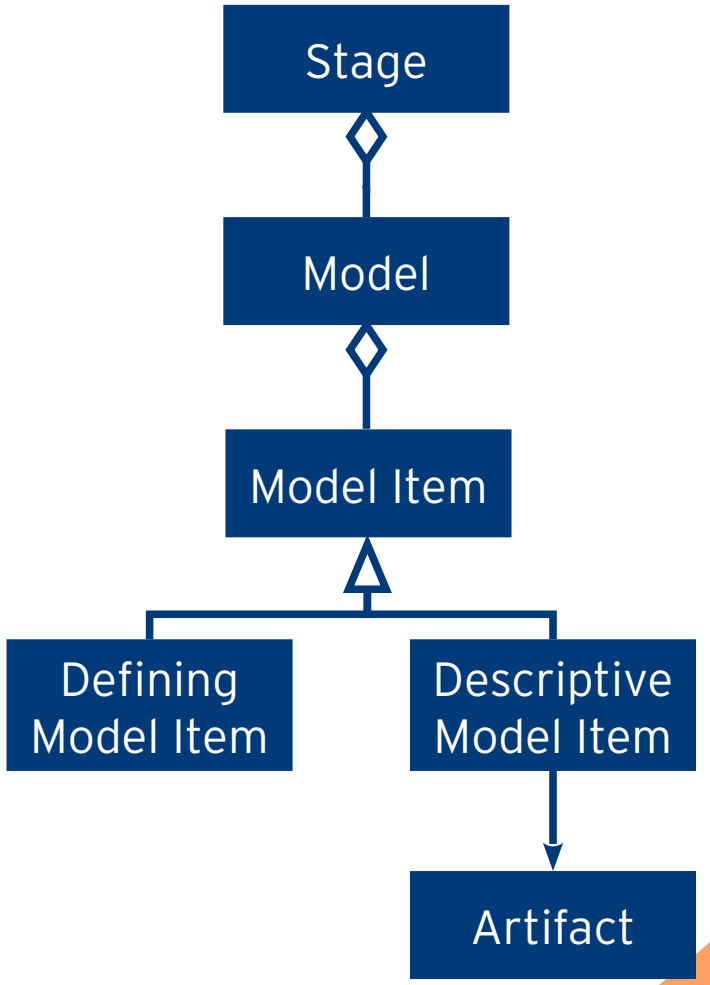
- Topmost concepts represent modeling stages and models
- They contain concepts that represent domain entities
- They relate models and model items to each other

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 expressed by the M³L
The aim is to design them in such a way that traceability is achieved



# Section 4.2

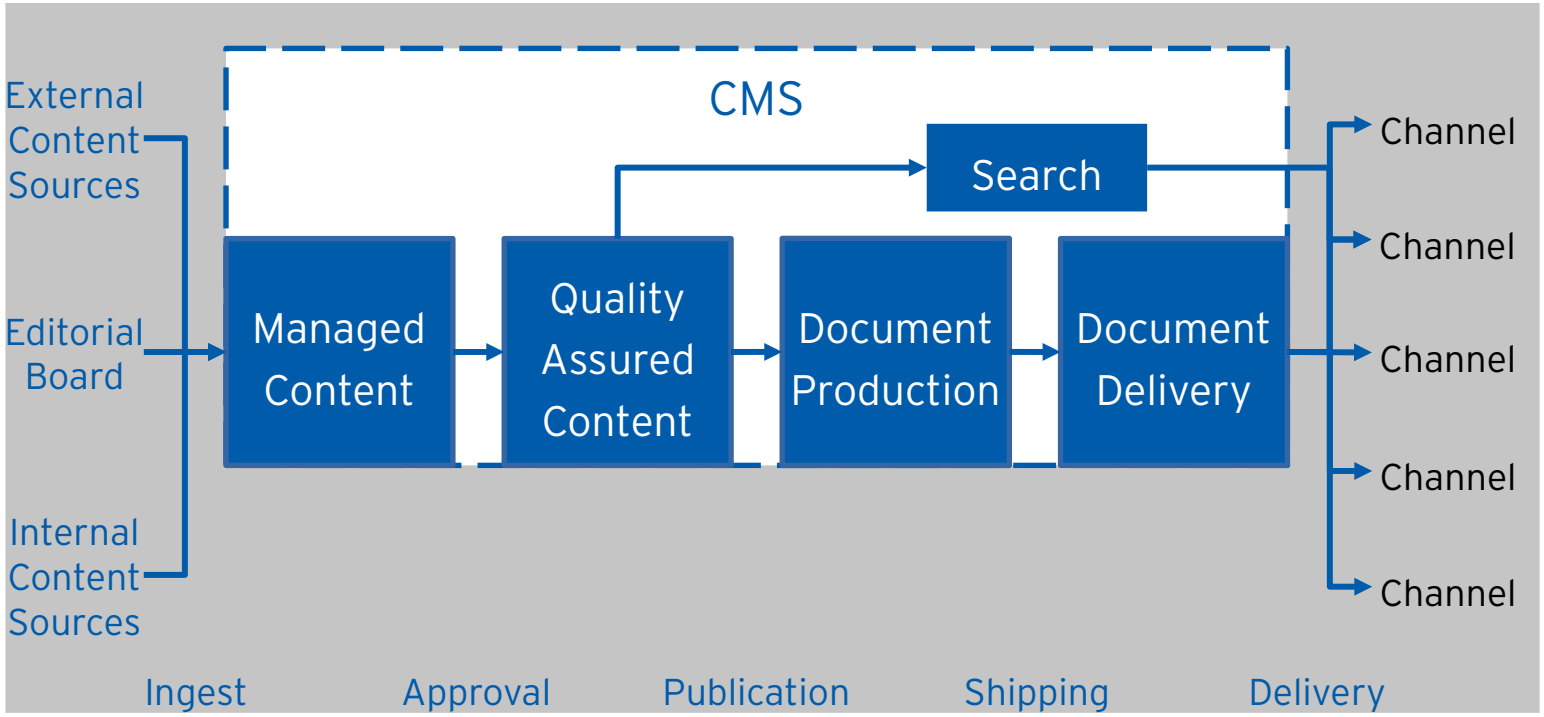
# Artifact Creation from Models

# Content Management (1)

**Content Management Systems** (**CMSs**) offer a range of functionality to incorporate creation/ingest and editing of content, quality assurance processes as well as the creation and distribution of digital documents

This means that content is created in a model-driven fashion

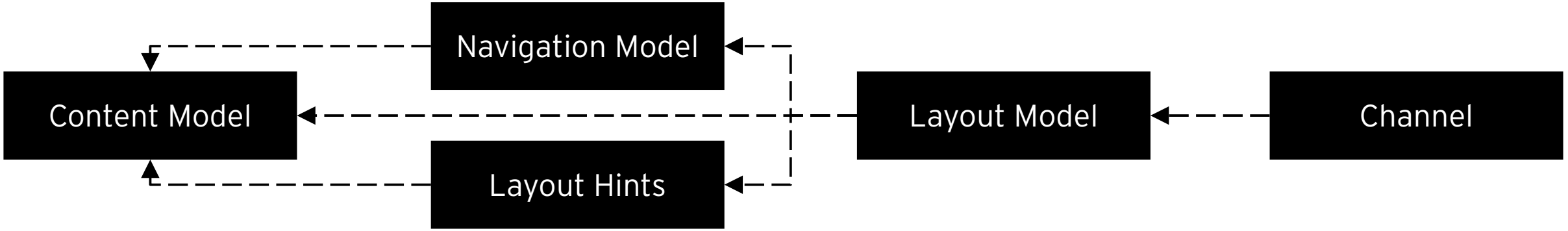
In the past, we already applied the M³L for such tasks



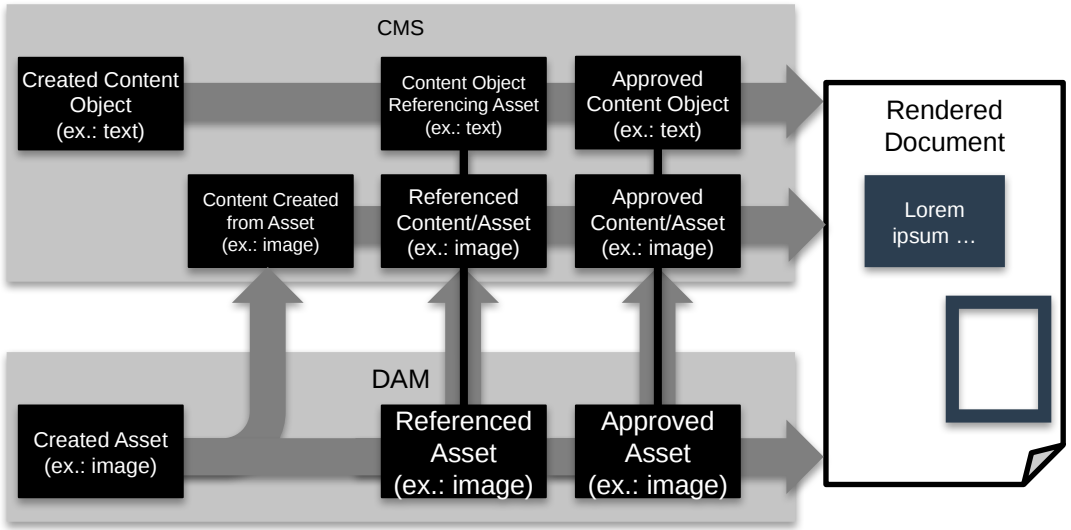

Now we can reuse that knowledge to create artifacts that describe software from models of the software

# Content Management (2)

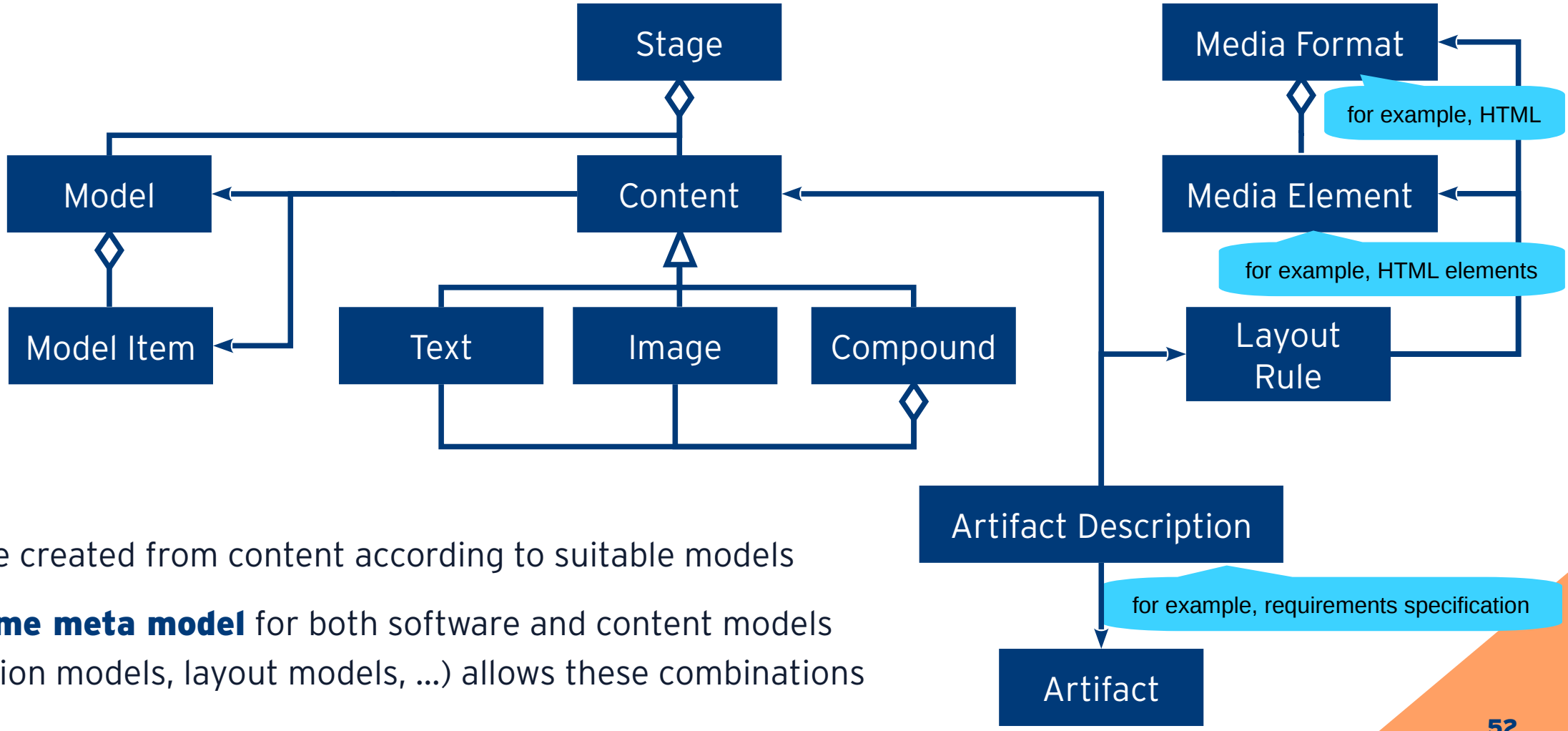
CMSs separate **content**, **structure**, and **layout** of documents




Content items have a **lifecycle**



# Software Engineering Artifacts Generation



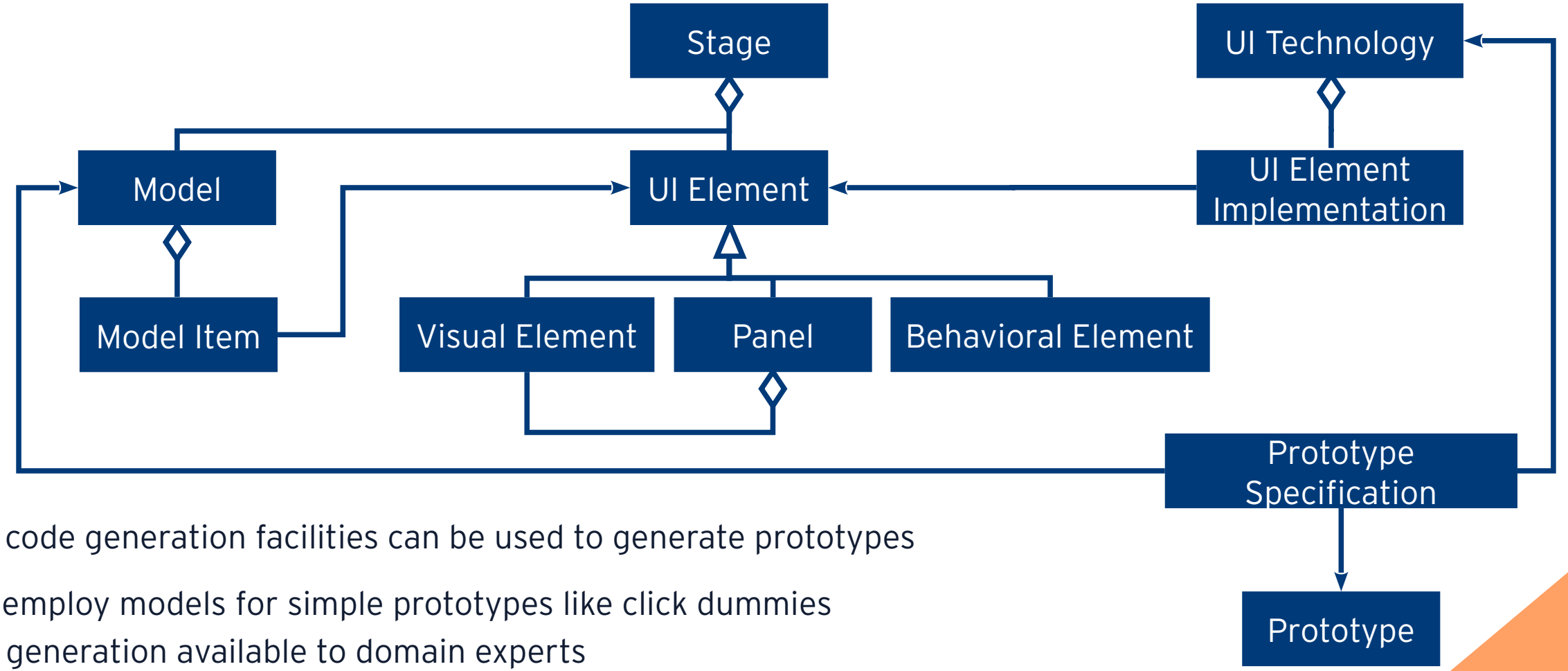

**Artifacts** are created from content according to suitable models

Using the **same meta model** for both software and content models (plus navigation models, layout models, ...) allows these combinations

# Section 4.3

# Prototype Generation from Models

# Interactive Software Engineering Artifacts




Existing code generation facilities can be used to generate prototypes

Still, we employ models for simple prototypes like click dummies to make generation available to domain experts

# Section 5

# Conclusion

# 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 **creative activities** that are carried out using adequate notations and specific tools that lead to the creation of unstructured/informal artifacts

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, like traceability, efficiency, and others,

models shall at least **support** the development process

We currently investigate two approaches

1) Models that have elements that **describe creative artifacts**

2) The combination of software and content models to **generate creative artifacts**

First **experiments using the M³L** for both approaches show that both approaches can be combined in the same (meta) framework

# Outlook

Many questions are still open; just to name a few

The **references to artifacts** need to be elaborated; we can build on previous work at this point

- fragments of artifacts
- related artifacts

**Parsing digital artifacts** to allow manual changes in the generation case (approach 2)

- round-trip modeling
- finally convergence of approach 1 and approach 2?

Investigate the utilization of generated models as **checklists** that describe the required artifacts

In general, modeling with the **M³L in MDSE** will be investigated further, for example, reasoning capabilities

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