Compositional Engineering of DSLs for Assistive Systems

LangDev Meetup 2023

Judith Michael, Bernhard Rumpe
Software Engineering
RWTH Aachen

http://www.se-rwth.de

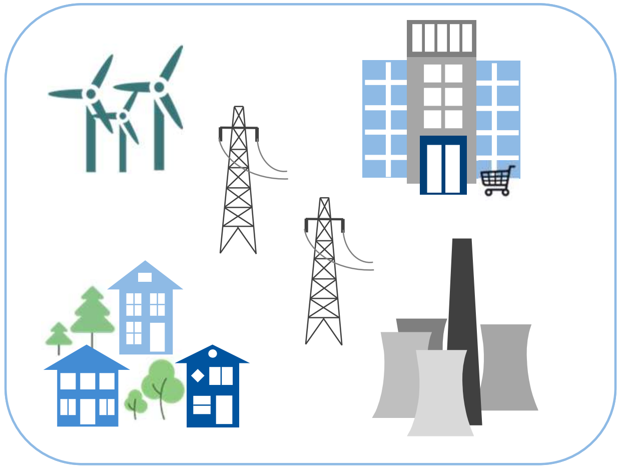
Based on: [HJK+23] M. Heithoff, N. Jansen, J. C. Kirchhof, J. Michael, F. Rademacher, B. Rumpe: Deriving Integrated Multi-Viewpoint Modeling Languages from Heterogeneous Modeling Languages: An Experience Report. In: 16th ACM SIGPLAN Int. Conf. on Software Language Engineering (SLE 2023), ACM, 2023. https://doi.org/10.1145/3623476.3623527

14.11.2023, Utrecht, NL

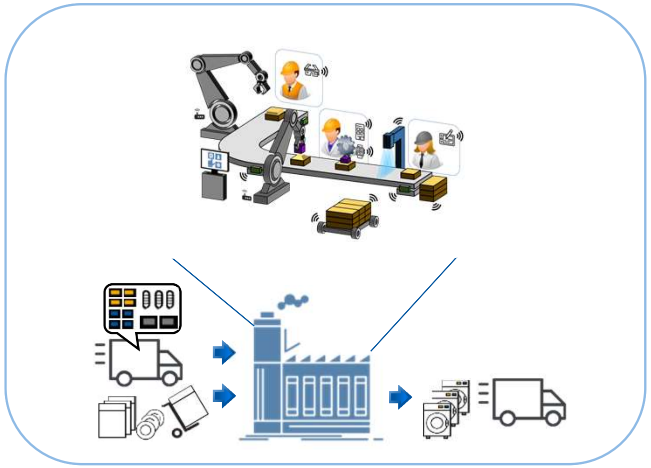
Why is language composition an interesting topic?

- Increasing *complexity* of the world
- Use *DSLs* to handle complexity as software engineers
  - Different perspectives and viewpoints

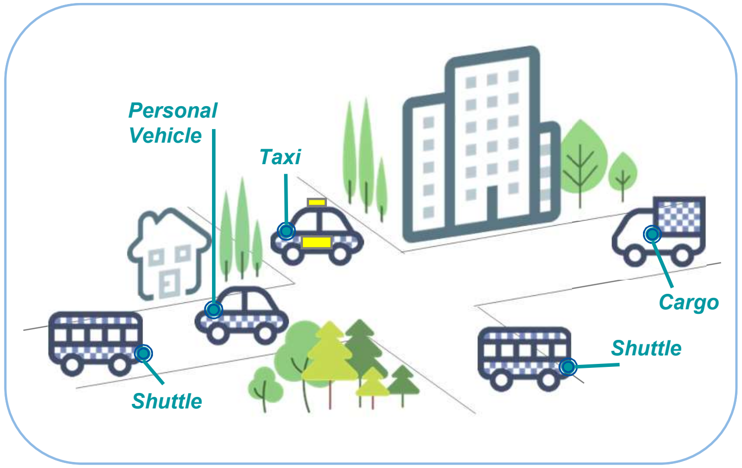
- *Increasing* number of DSLs
- Research perspective
  - foster *reuse* to increase quality and productivity



Energy Systems



Production Systems

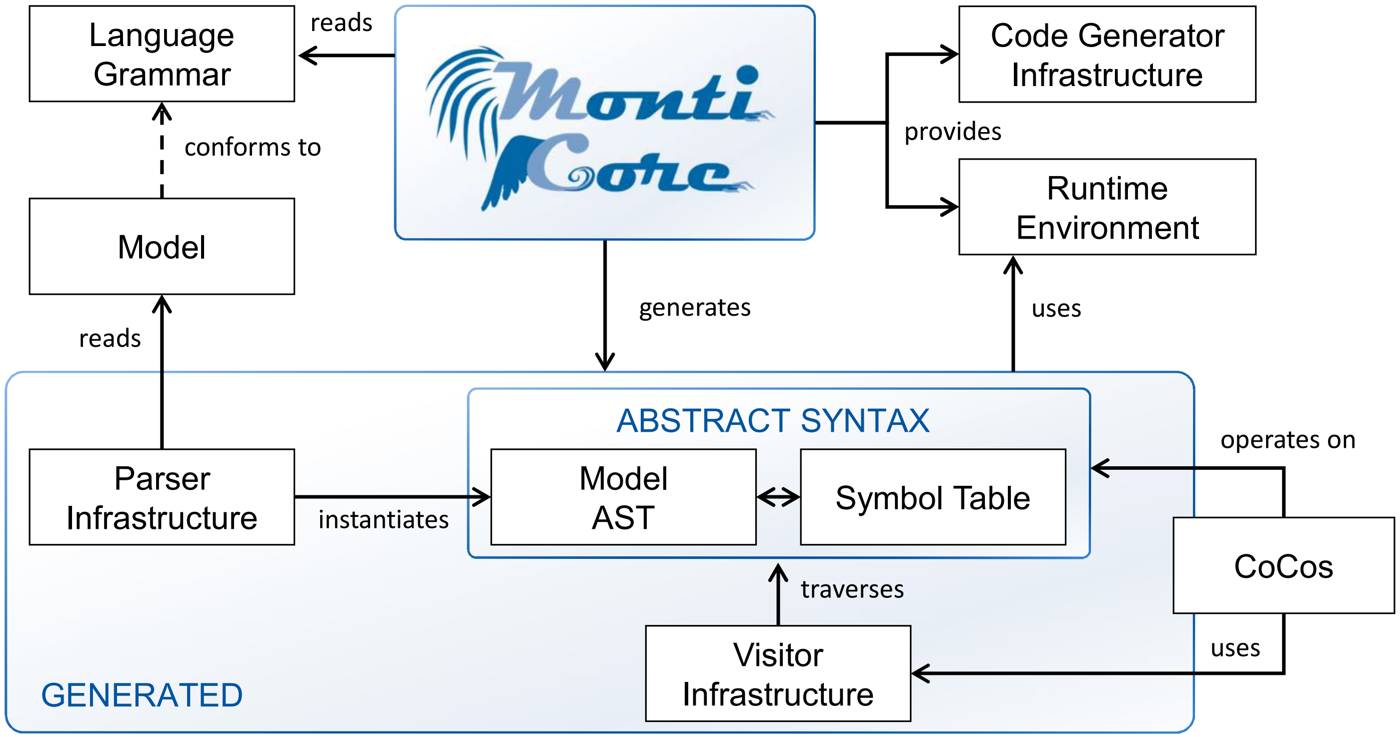


Transport Systems

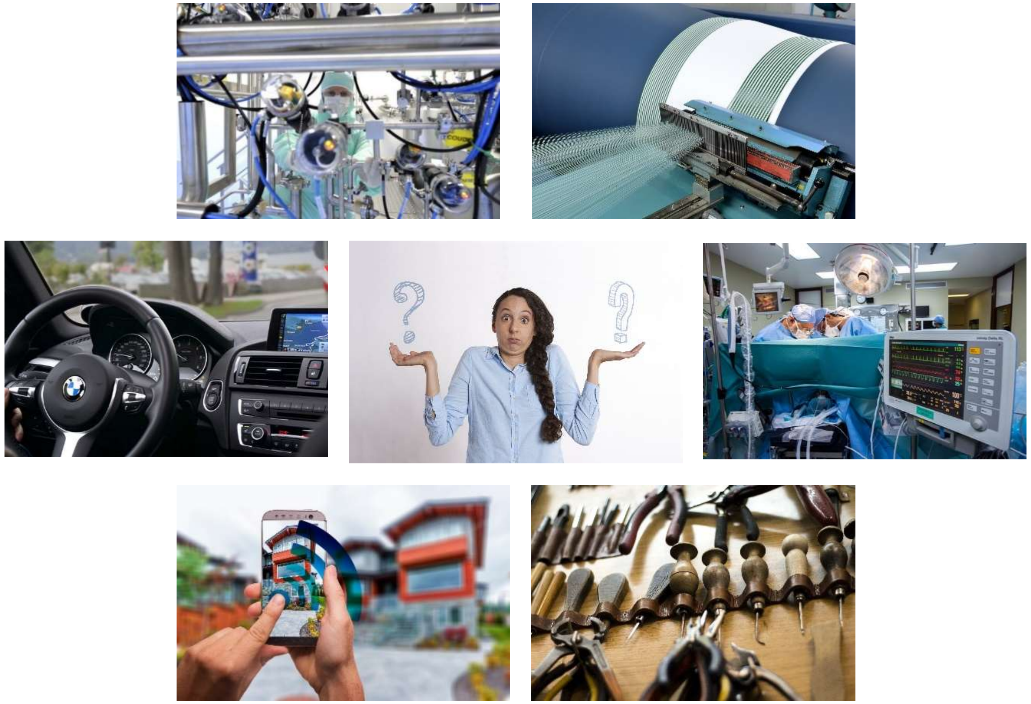
Research Question: How to *integrate* different modeling languages via *established language composition techniques*?
Special Focus: Assistive Systems

MontiCore Language Workbench

- Easy and fast engineering of DSLs
  - *define context-free grammars*
  - *supports language composition and reuse*
  - variability in syntax, context conditions, generation, semantics
- Definition of *modular language fragments*
  - interfaces between models and language fragments
- Support for analysis
- Support for transformations
- Pretty printing, editors



Assist Language Family | Overview



- Aim
  - *simplify* the development of assistive systems
- *Assistive Systems*
  - provide human behavior support, e.g., manual assembly in production, driving, activities of daily life
  - stress, new situations, age-related
- MDE of assistive systems
  - use MontiGem generator to develop web-applications
- *Language Family*
  - domain model, GUI-DSL, OCL for generation
  - context language for objects
  - task language for processes

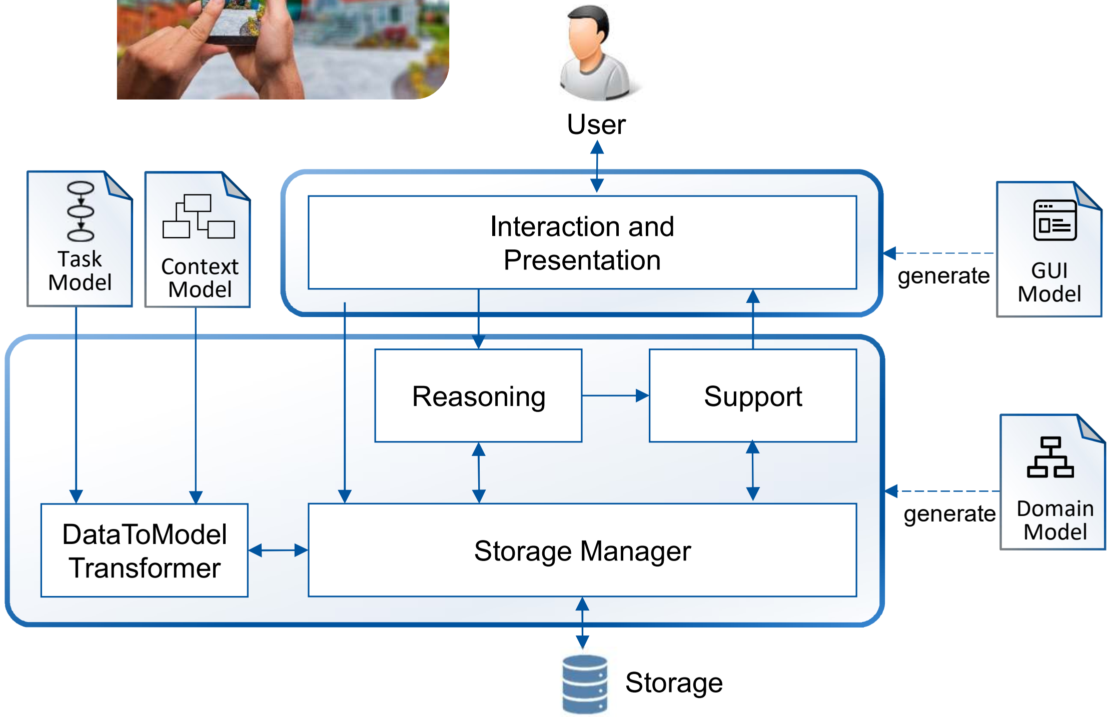
natural language based

System Architecture

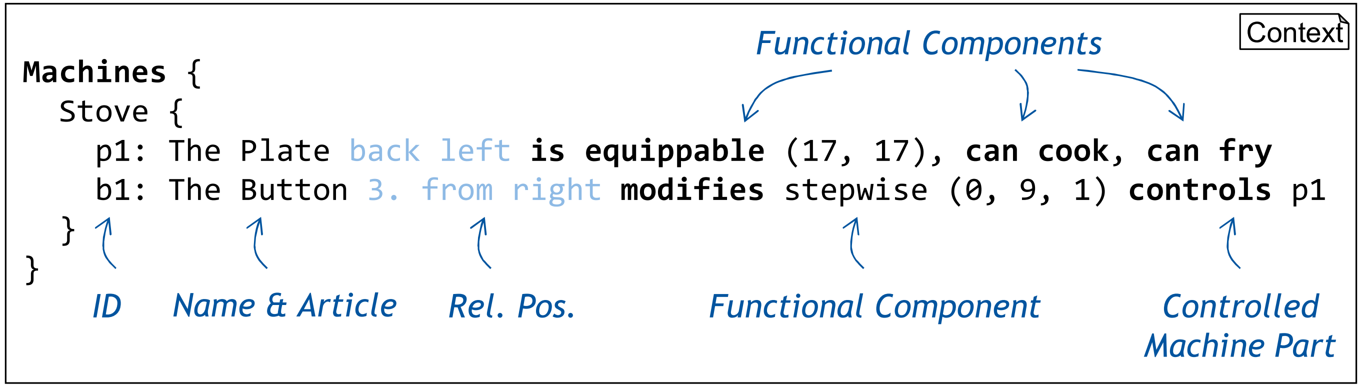
MontiGem
Generator Framework

- Input: *Domain* and *GUI models,* opt. *OCL* and *tagging models*
- Generates DB, Backend, Frontend, Communication Infrastructure
- Allow for *adding hand-written code* & continuous *re-generation*

- *Models at runtime*
  - Context Model
    - define concrete objects to be used in processes
  - Task Model
    - describe the processes to be supported



Context Modeling Language | Example Machines



Context

Functional Components

```
Machines {
  Stove {
    p1: The Plate back left is equippable (17, 17), can cook, can fry
    b1: The Button 3. from right modifies stepwise (0, 9, 1) controls p1
  }
}
```

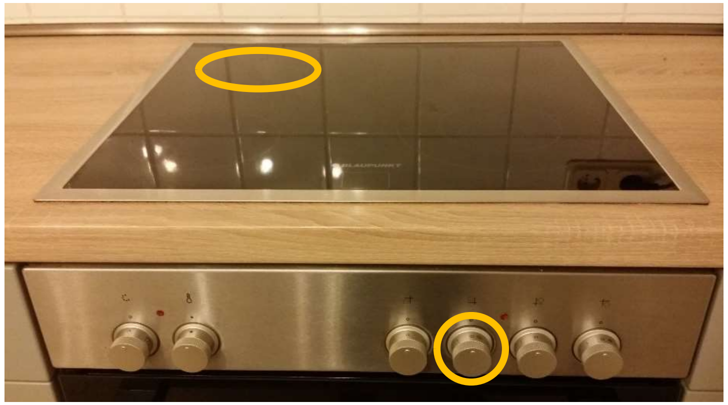
ID

Name & Article

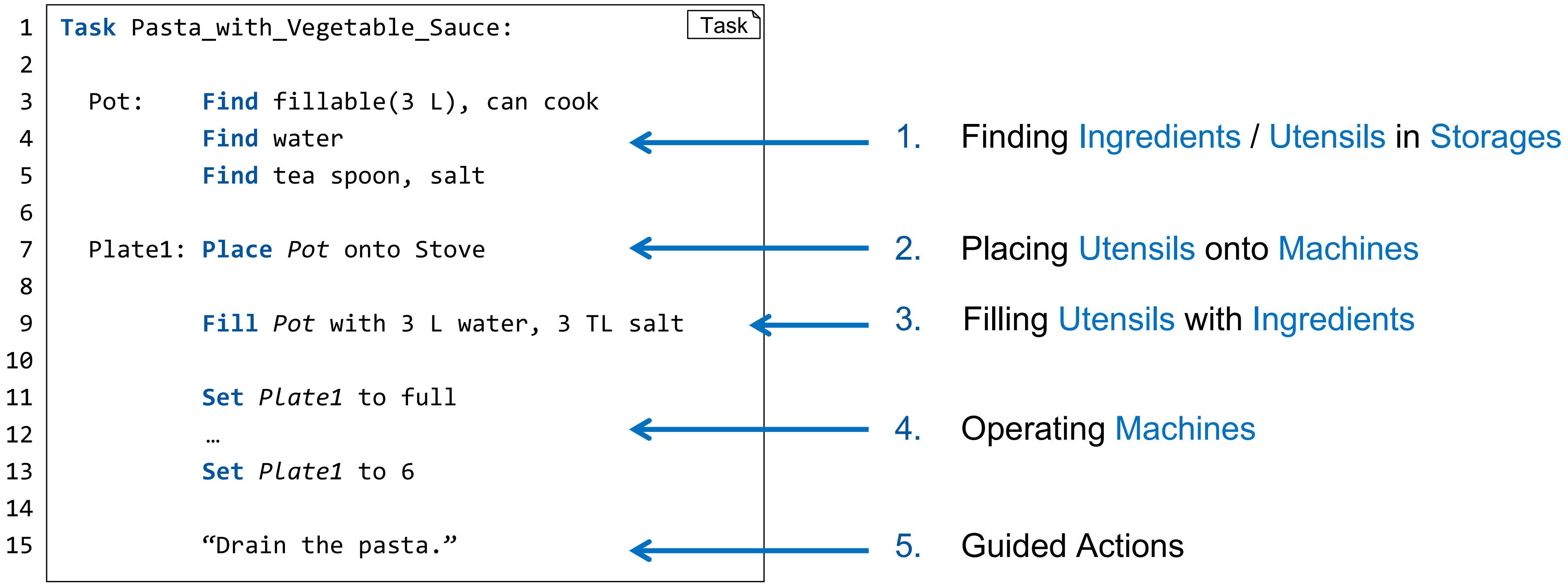
Rel. Pos.

Functional Component

Controlled Machine Part



Task Modeling Language | Describing human behavior

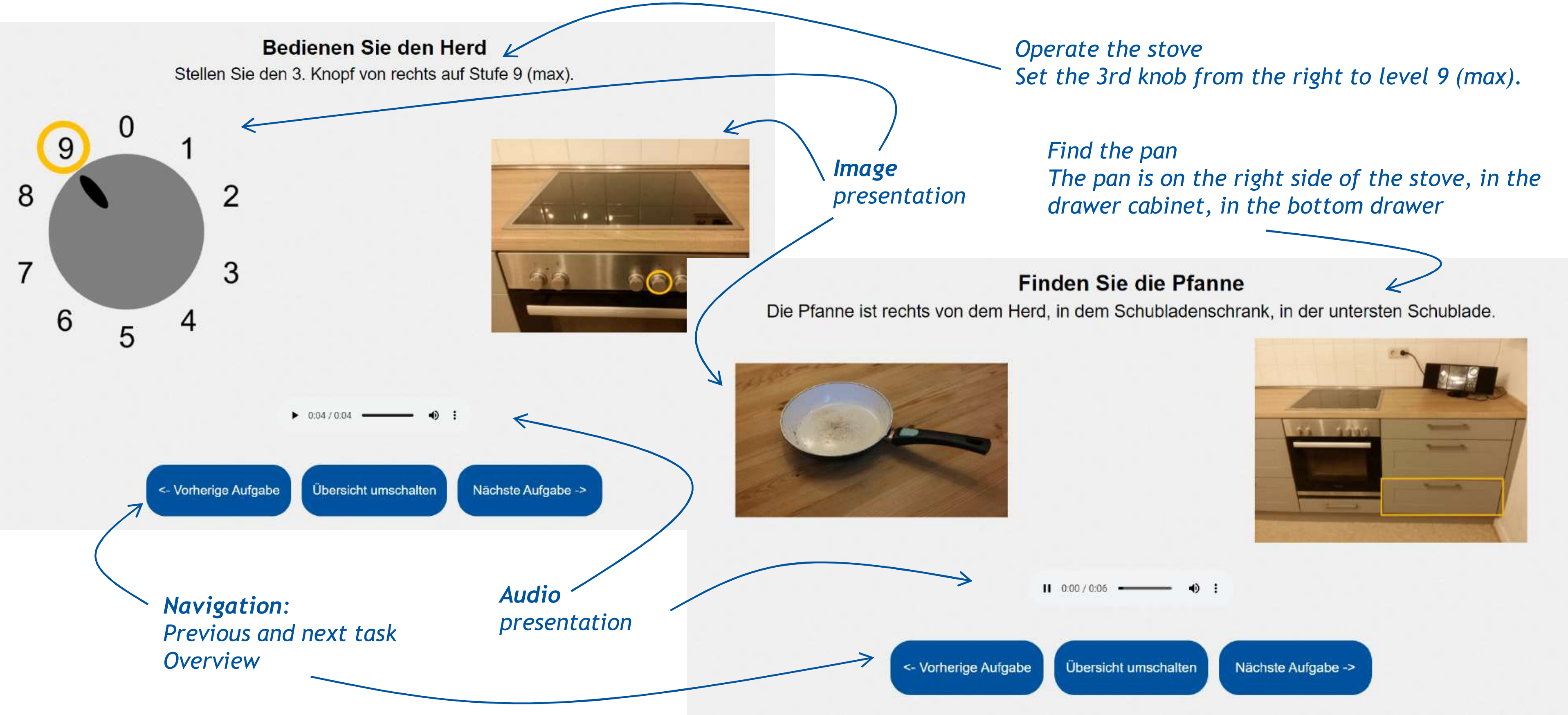


Task

```
Task Pasta_with_Vegetable_Sauce:

  Pot:    Find fillable(3 L), can cook
          Find water
          Find tea spoon, salt

  Plate1: Place Pot onto Stove

          Fill Pot with 3 L water, 3 TL salt

          Set Plate1 to full
          …
          Set Plate1 to 6

          “Drain the pasta.”
```

1. Finding Ingredients / Utensils in Storages
2. Placing Utensils onto Machines
3. Filling Utensils with Ingredients
4. Operating Machines
5. Guided Actions

Generated Information Presentation | Operate, Find, Place, Fill, Guided



Bedienen Sie den Herd

Stellen Sie den 3. Knopf von rechts auf Stufe 9 (max).

Finden Sie die Pfanne

Die Pfanne ist rechts von dem Herd, in dem Schubladenschrank, in der untersten Schublade.

Operate the stove
Set the 3rd knob from the right to level 9 (max).

Image *presentation*

Find the pan
The pan is on the right side of the stove, in the drawer cabinet, in the bottom drawer

Navigation:
Previous and next task
Overview

Audio *presentation*

Modeling and Generated Information Presentation

- Reasoning in the backend to navigate user to find *fitting* objects
- Presentation for each task type (Find, Fill, Place, Operate, (Guided))
  - Text, Image and Audio

Task

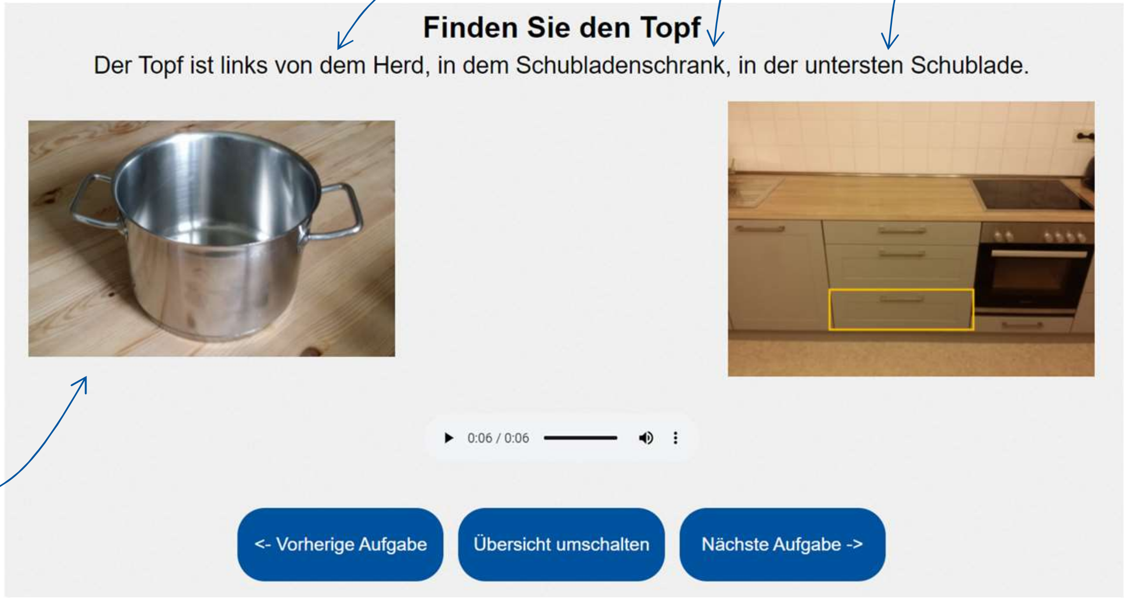
```
Find fillable(3 L), can cook
```

Context

```
Storages {
  The drawer cabinet left of the stove {
    …
    s3: The drawer 1. from below
  }
}
Utensils {
  Pot1: 1x the pot(15,10) in s3,
    is fillable(5 L, 15, 10), can cook
  Pot2: 1x the pot in s2,
    is fillable(2 L, 5, 5)
}
```

Picture

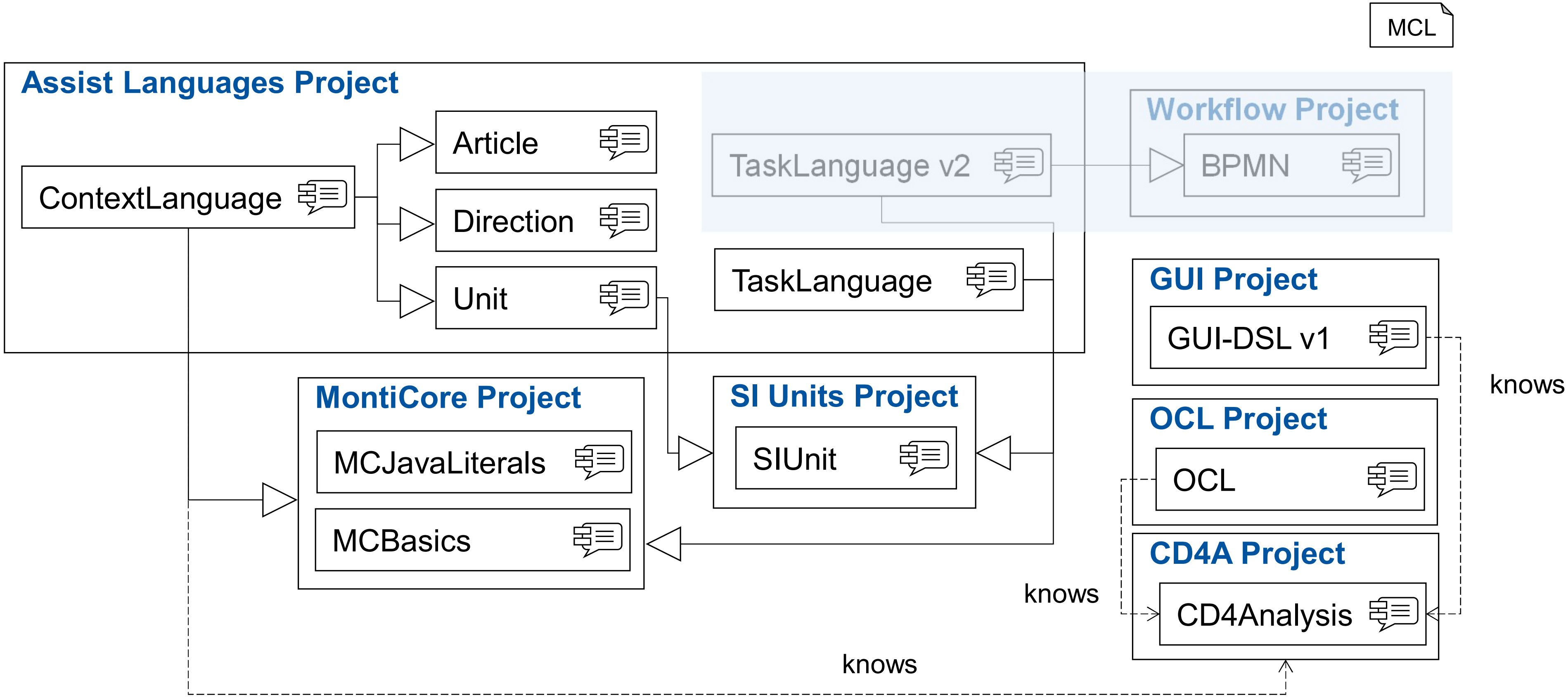
Nested Layout
Order: Reference, Composite, Leaf



Finden Sie den Topf

Der Topf ist links von dem Herd, in dem Schubladenschrank, in der untersten Schublade.

Assist Language Family | Languages and Components



MontiCore | Three “Layers” of Languages

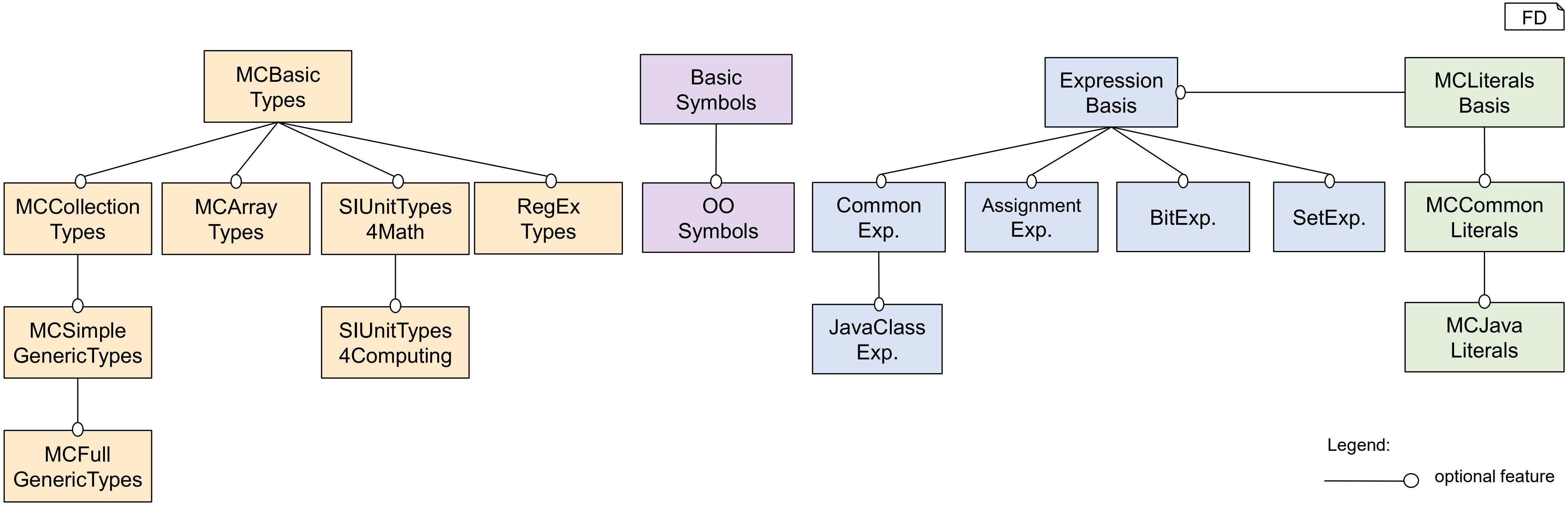
Base Layer:
Components

MCBasics Expressions Statements Cardinality SI Units

MCCommon Literals Types Completeness

Feature Diagram for MontiCore Language Components

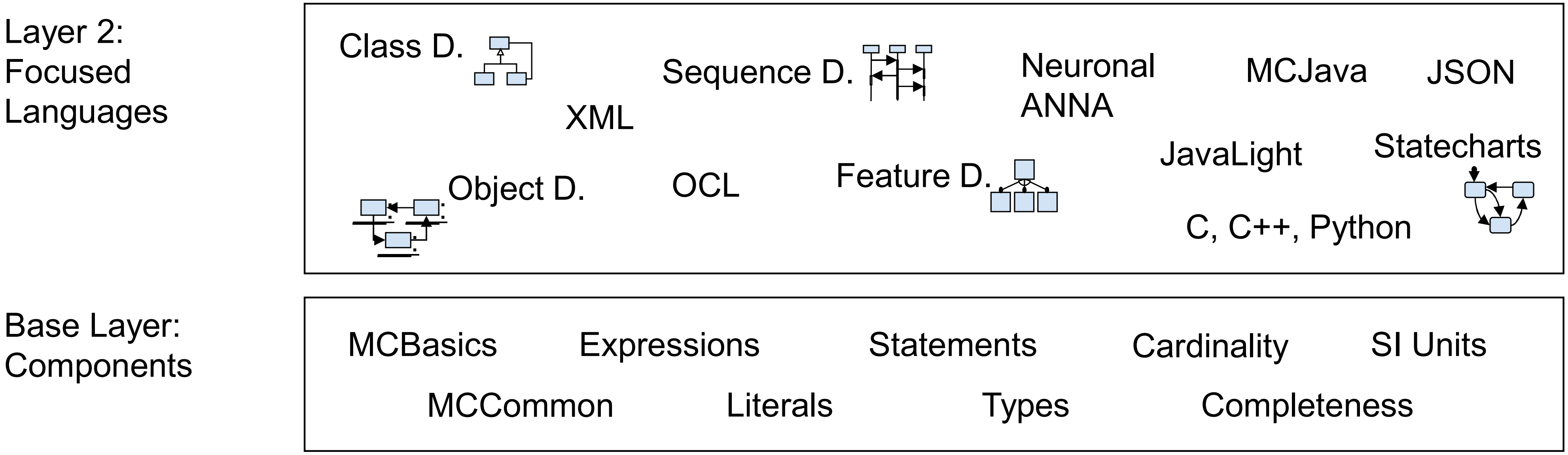
- MontiCore provides a set of language components that can be used as features
  - Some dependencies exist, e.g. certain expressions rely on appropriate literals



Grammars for these languages can be found at: https://monticore.github.io/monticore/monticore-grammar/src/main/grammars/de/monticore/Grammars/

[BEH+20] A. Butting, R. Eikermann, K. Hölldobler, N. Jansen, B. Rumpe, A. Wortmann: A Library of Literals, Expressions, Types, and Statements for Compositional Language Design. JOT 19 (3), 2020.

MontiCore | Three “Layers” of Languages



Layer 2: Focused Languages

Class D.

XML

Sequence D.

Neuronal ANNA

MCJava

JSON

Object D.

OCL

Feature D.

JavaLight

Statecharts

C, C++, Python

Base Layer: Components

MCBasics

Expressions

Statements

Cardinality

SI Units

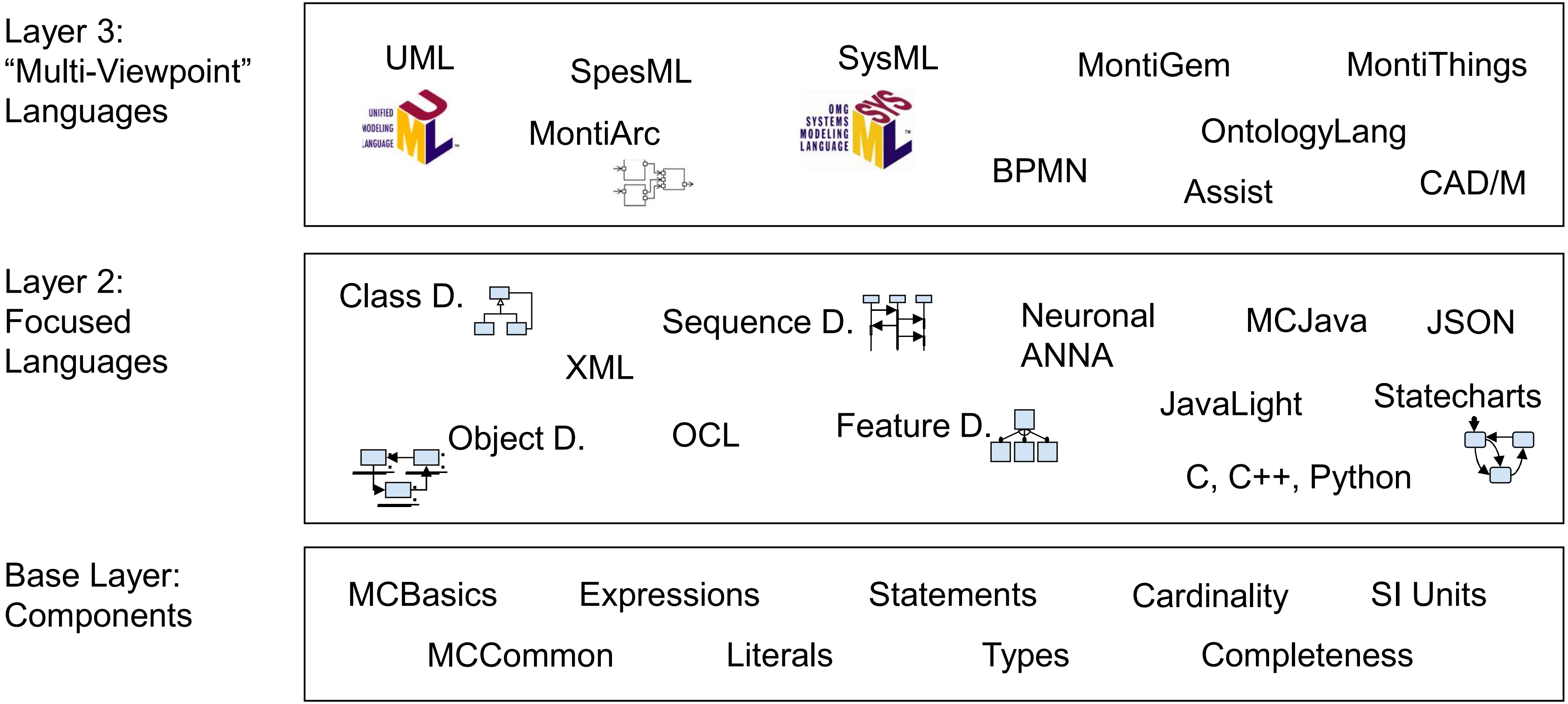
MCCommon

Literals

Types

Completeness

MontiCore | Three “Layers” of Languages



Layer 3:
“Multi-Viewpoint”
Languages

UML SpesML SysML MontiGem MontiThings

MontiArc OntologyLang BPMN Assist CAD/M

Layer 2:
Focused
Languages

Class D. Sequence D. Neuronal ANNA MCJava JSON

XML JavaLight Statecharts

Object D. OCL Feature D. C, C++, Python

Base Layer:
Components

MCBasics Expressions Statements Cardinality SI Units

MCCommon Literals Types Completeness

MontiCore | Three “Layers” of Languages

Layer 3: “Multi-Viewpoint” Languages

UML

SpesML

SysML

MontiGem

MontiThings

MontiArc

OntologyLang

BPMN

Assist

CAD/M

Layer 2: Focused Languages

Class D.

Sequence D.

Neuronal ANNA

MCJava

JSON

XML

JavaLight

Statecharts

Object D.

OCL

Feature D.

C, C++, Python

Base Layer: Components

MCBasics

Expressions

Statements

Cardinality

SI Units

MCCommon

Literals

Types

Completeness

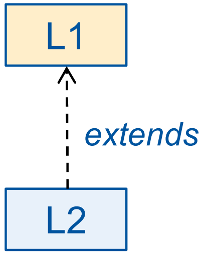
Language Composition Mechanisms

Language Inheritance

- Use original language
  - remains unchanged
- New DSL
  - adopt and extend or modify concepts
  - concrete and abstract syntax, generated tooling and hand-written extensions

```
grammar Entities extends MCBasics, MCBasicTypes {
  start CDCompilationUnit;

  @Override
  symbol scope CDClass implements CDElement =
 “entity" Name "{"
      CDAttribute*
    "}";
…
```

MG



Language Extension

- Use original language
  - remains unchanged
- New DSL
  - new elements can be added
  - existing elements can be modified only in an extending but non-restricting way
  - valid models of the original language still remain valid

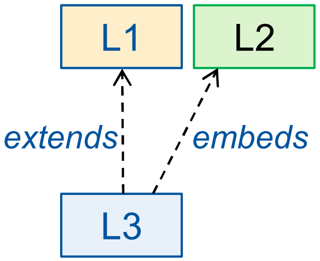
```
grammar CD4Code extends MCBasics, MCBasicTypes {
  start CDCompilationUnit;

  @Override
  symbol scope CDClass implements CDElement =
    "class" Name "{"
      (CDAttribute CDMethod)*
    "}" ;

  symbol CDMethod implements CDMember =
    MCType Name "(" Args ")" ";";
…
```

MG

Language Composition Mechanisms

Language Embedding

- integrate multiple DSLs
  - combining their production rules in a single grammar
  - enabling integrated modeling

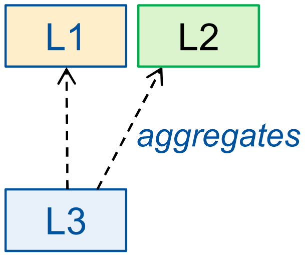


MG

```
grammar MealyAutomata extends Automata,
                              CommonExpressions {

  MealyAutomaton = MCImportStatement* Automaton;

  @Override
  Transition =
    from:Name "-" input:Expression "/"
    output:Expression ">" to:Name ";" ;
}
```

Language Aggregation

- integrate models of multiple DSLs
  - keep them as separate artifacts
- loose coupling of DSL definitions
  - symbol table infrastructure allows for cross-referencing



Aut

```
…
automaton PingPong {
  state NoGame <<initial>> <<final>>;
  state Ping;
  state Pong;
  …
  Pong - missBall / p1_points+=strokes > NoGame;
}
```

CD

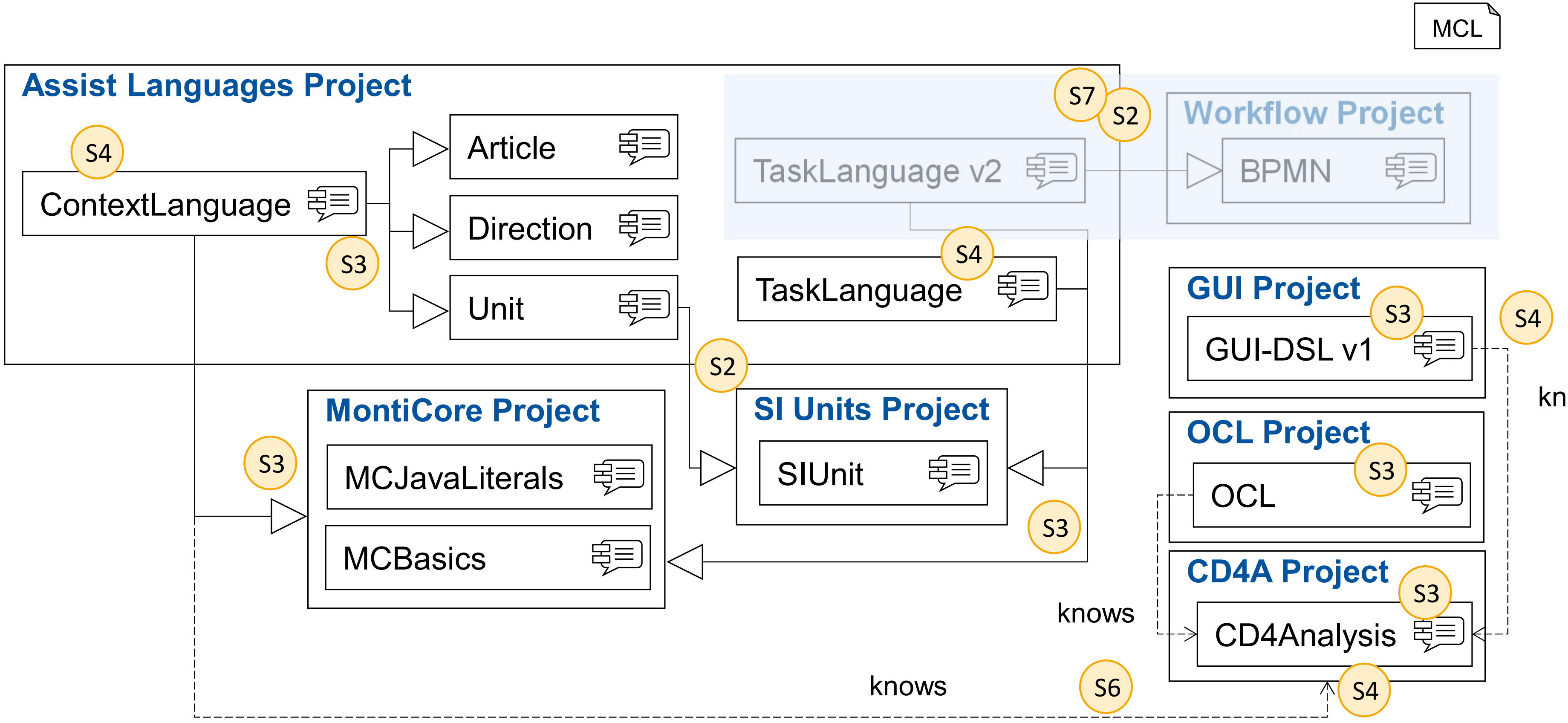
```
classdiagram games {
  class Tennis {
    int strokes;
    int p1_points;
    int p2_points;
  }
}
```

variable symbols of CD in the context of a PingPong game automaton

Scenarios | Overview

Scenario / Use Case	Inheritance	Extension	Embedding	Aggregation
(S1) Modifying a language, tailoring it to a specific use case	suitable	partially	x	x
(S2) Extending a language to a use case while maintaining the integrity of the original models	partially	suitable	x	x
(S3) Combining multiple language components into a modeling language	x	x	suitable	x
(S4) Combining modeling languages into a language family	x	x	suitable	suitable
(S5) Constructing huge languages with different constituents	x	x	suitable	suitable
(S6) Constructing a language or language family with heterogeneous parts for interdisciplinary use	x	x	partially	suitable
(S7) Modularization of model artifacts	x	x	x	suitable

Assist Language Family | Languages and Components



Legend
S1 | Modify DSL
S2 | Extending DSL
S3 | L. Components
S4 | DSLs in Family
S5 | Huge Languages
S6 | Heterogenous parts
S7 | Modularization

Scenarios for one DSL & Language Components

S1 | Modifying a language, tailoring it to a use case

- creating a new DSL and use an existing one as a base language

- Suitable: *Inheritance*
  - e.g., no former models, new tooling
- Partially suitable: *Extension*
  - e.g., reuse tooling

S2 | Extending a language to a use case while maintaining the integrity of the original models

- reuse existing models

- Suitable: *Extension*
  - e.g., ensure to keep modifications genuinely conservative (warnings)
  - *Assist Language, e.g., SI Units*

S3 | Combining multiple lang. components into a DSL

- having reusable language components
- could be incomplete language components

- Suitable: *Embedding*
  - effective when integrated DSLs share common interfaces
  - no glue code necessary
  - developers need to be knowledgeable about the existing components
  - *Assist Language, e.g., MCBasics, MCJavaLiterals*

- Not suitable: *Aggregation*
  - Loose coupling
  - would not complete components into a fully functional DSL

Scenarios for more than one DSL

S4 | Combine DSLs into a language family

- already functional languages
- Suitable: *Embedding, Aggregation*
  - integrated views or separate artifacts
  - *Assist Language, e.g., Context, Task Language, CD4A*

S5 | Construct huge DSLs with different constituents

- already functional languages
- aim: support organization and structuring of larger modeling projects
- Suitable: *Embedding, Aggregation*
  - integrated views or separate artifacts

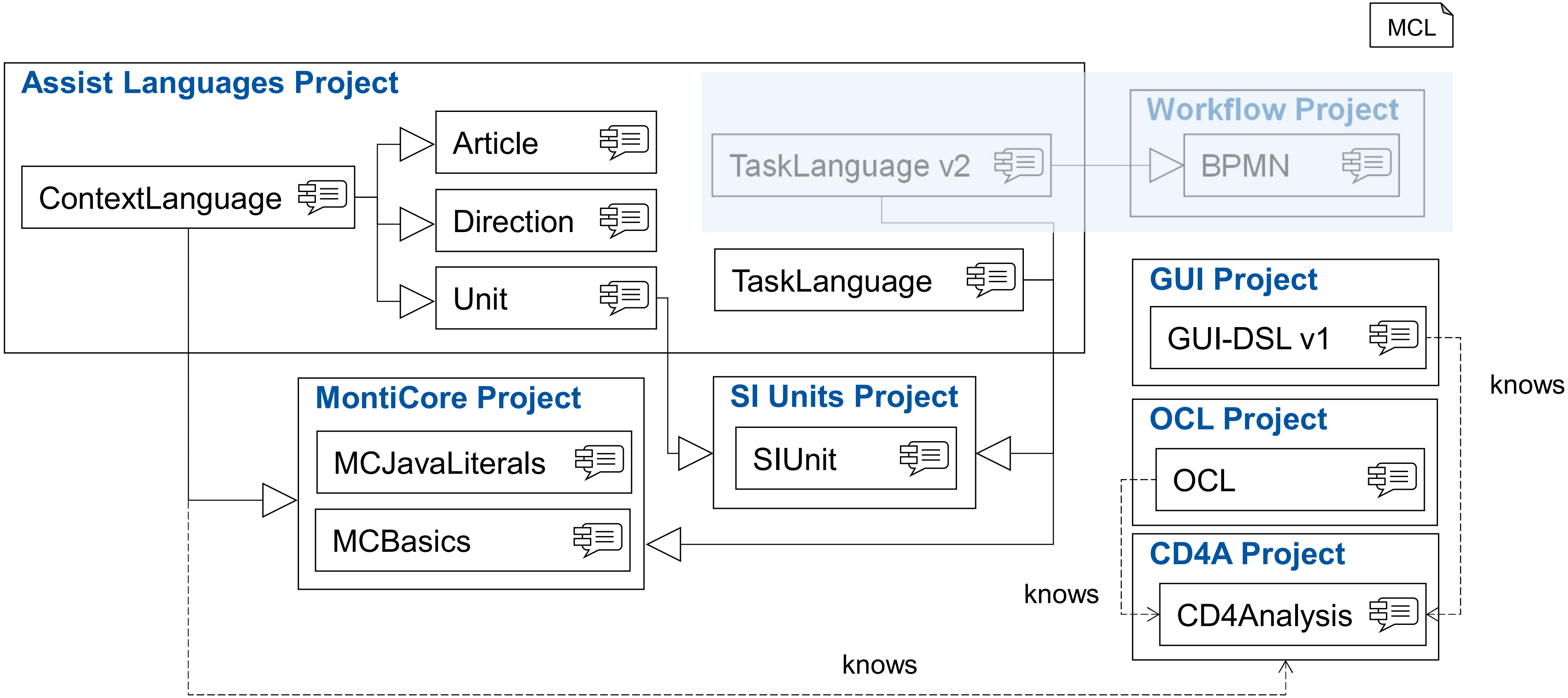
S6 | Constructing a language or language family with heterogeneous parts for interdisciplinary use

- interdisciplinary teams
- artifacts represent different domain-specific views on a system
- Suitable: *Aggregation*
  - enable domain expert views without getting distracted by information of other perspectives
- Partially suitable: *Embedding*

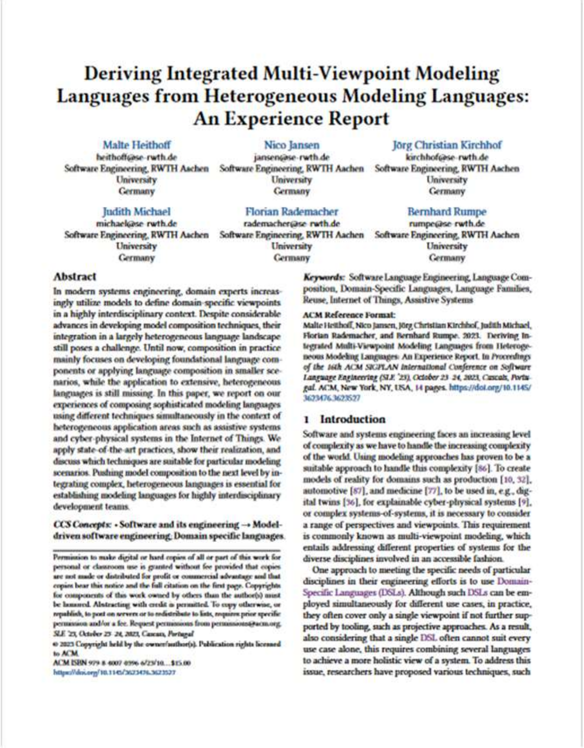
S7 | Modularization of model artifacts

- separation of concerns &
- create a suitable modeling project structure
- Suitable: *Aggregation*

Assist Language Family | Languages and Components



Summary and Discussion



Deriving Integrated Multi-Viewpoint Modeling Languages from Heterogeneous Modeling Languages: An Experience Report

Malte Heithoff
heithoff@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Nico Jansen
jansen@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Jörg Christian Kirchhof
kirchhof@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Judith Michael
michael@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Florian Rademacher
rademacher@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Bernhard Rumpe
rumpe@se-rwth.de
Software Engineering, RWTH Aachen University
Germany

Abstract

In modern systems engineering, domain experts increasingly utilize models to define domain-specific viewpoints in a highly interdisciplinary context. Despite considerable advances in developing model composition techniques, their integration in a largely heterogeneous language landscape still poses a challenge. Until now, composition in practice mainly focuses on developing foundational language components or applying language composition in smaller scenarios, while the application to extensive, heterogeneous languages is still missing. In this paper, we report on our experiences of composing sophisticated modeling languages using different techniques simultaneously in the context of heterogeneous application areas such as assistive systems and cyber-physical systems in the Internet of Things. We apply state-of-the-art practices, show their realization, and discuss which techniques are suitable for particular modeling scenarios. Pushing model composition to the next level by integrating complex, heterogeneous languages is essential for establishing modeling languages for highly interdisciplinary development teams.

CCS Concepts: **• Software and its engineering → Model-driven software engineering; Domain specific languages.**

SLE '23, October 23–24, 2023, Cascais, Portugal

ACM ISBN 979-8-4007-0396-6/23/10...$15.00
https://doi.org/10.1145/3623476.3623527

Keywords: Software Language Engineering, Language Composition, Domain-Specific Languages, Language Families, Reuse, Internet of Things, Assistive Systems

ACM Reference Format:
Malte Heithoff, Nico Jansen, Jörg Christian Kirchhof, Judith Michael, Florian Rademacher, and Bernhard Rumpe. 2023. Deriving Integrated Multi-Viewpoint Modeling Languages from Heterogeneous Modeling Languages: An Experience Report. In *Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering (SLE '23), October 23–24, 2023, Cascais, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3623476.3623527

1 Introduction

Software and systems engineering faces an increasing level of complexity as we have to handle the increasing complexity of the world. Using modeling approaches has proven to be a suitable approach to handle this complexity [86]. To create models of reality for domains such as production [10, 32], automotive [87], and medicine [77], to be used in, e.g., digital twins [36], for explainable cyber-physical systems [9], or complex systems-of-systems, it is necessary to consider a range of perspectives and viewpoints. This requirement is commonly known as multi-viewpoint modeling, which entails addressing different properties of systems for the diverse disciplines involved in an accessible fashion.

One approach to meeting the specific needs of particular disciplines in their engineering efforts is to use Domain-Specific Languages (DSLs). Although such DSLs can be employed simultaneously for different use cases, in practice, they often cover only a single viewpoint if not further supported by tooling, such as projective approaches. As a result, also considering that a single DSL often cannot suit every use case alone, this requires combining several languages to achieve a more holistic view of a system. To address this issue, researchers have proposed various techniques, such

Capturing ***complex systems*** *requires* ***different techniques*** *for composing DSLs*

For the composition of ***language families, embedding*** *and* ***aggregation*** *are needed*

…more details in the SLE paper
and following publications

SLE Language Composition
Paper
se-rwth.de/publications

Languages for
Assistive Systems
Preprint