



Combination of Different Modelling Techniques for Software Engineering

Dr. Eckhardt Holz
Institut für Informatik
Humboldt-Universität zu Berlin





Outline

- Why do We Model?
- Combination of Models
 - Different Ways to Combine Models
- Combination of Modelling Techniques
 - Support of Combination
- Conclusion



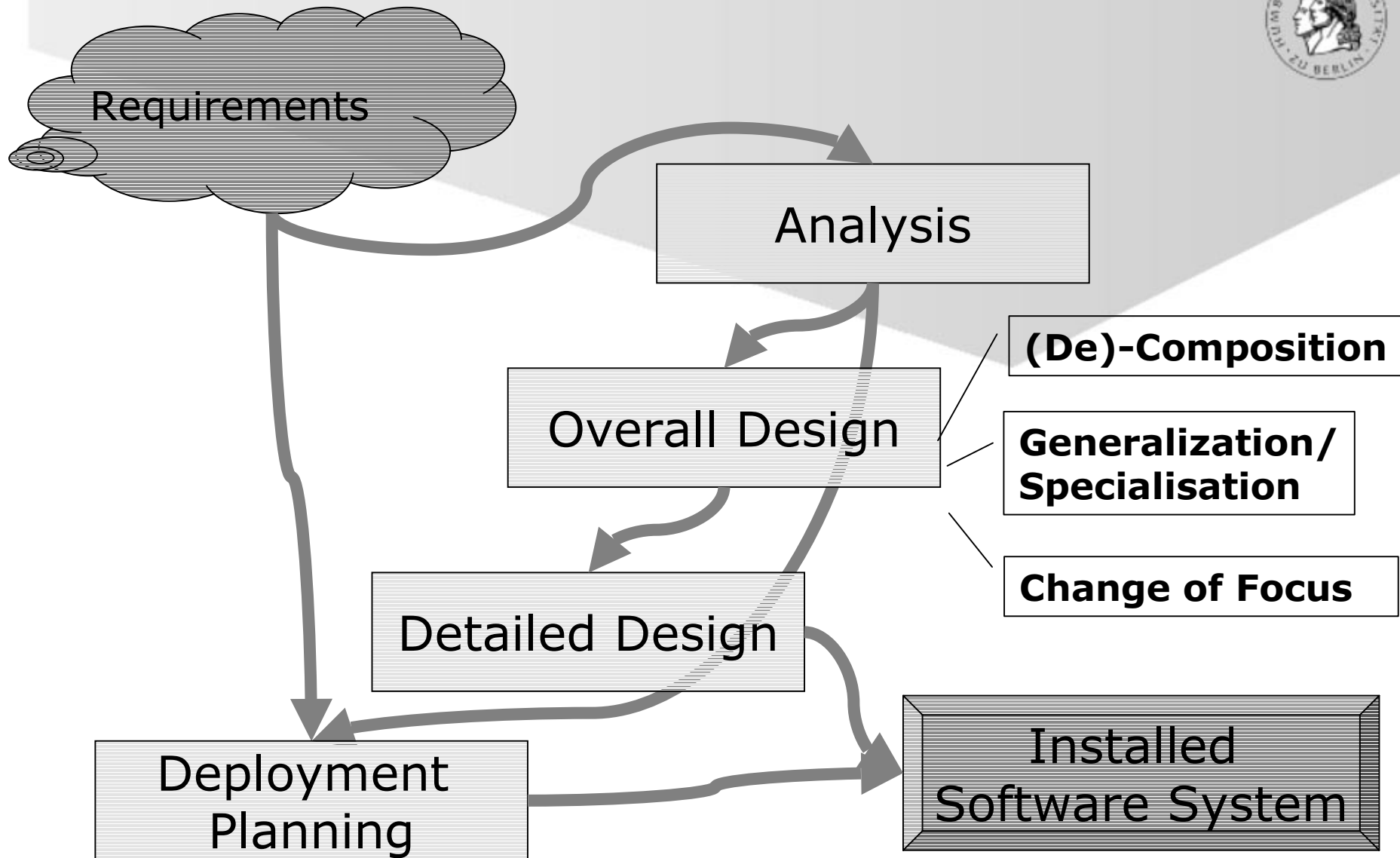
Target Audience

- Industrial Software Engineering
 - Selection of Software Engineering Techniques
 - Process, Language Tools
- Developer of Design & Engineering Languages
- Developer of Tools



Why do We Model in Software Engineering?

- Management of Complexity
- Support of the Design Process
- Automation of Routine Works
- Communication and Documentation
- Verification of the Design
- ...
- *The Overall Goal is the Development of a Software Product!*



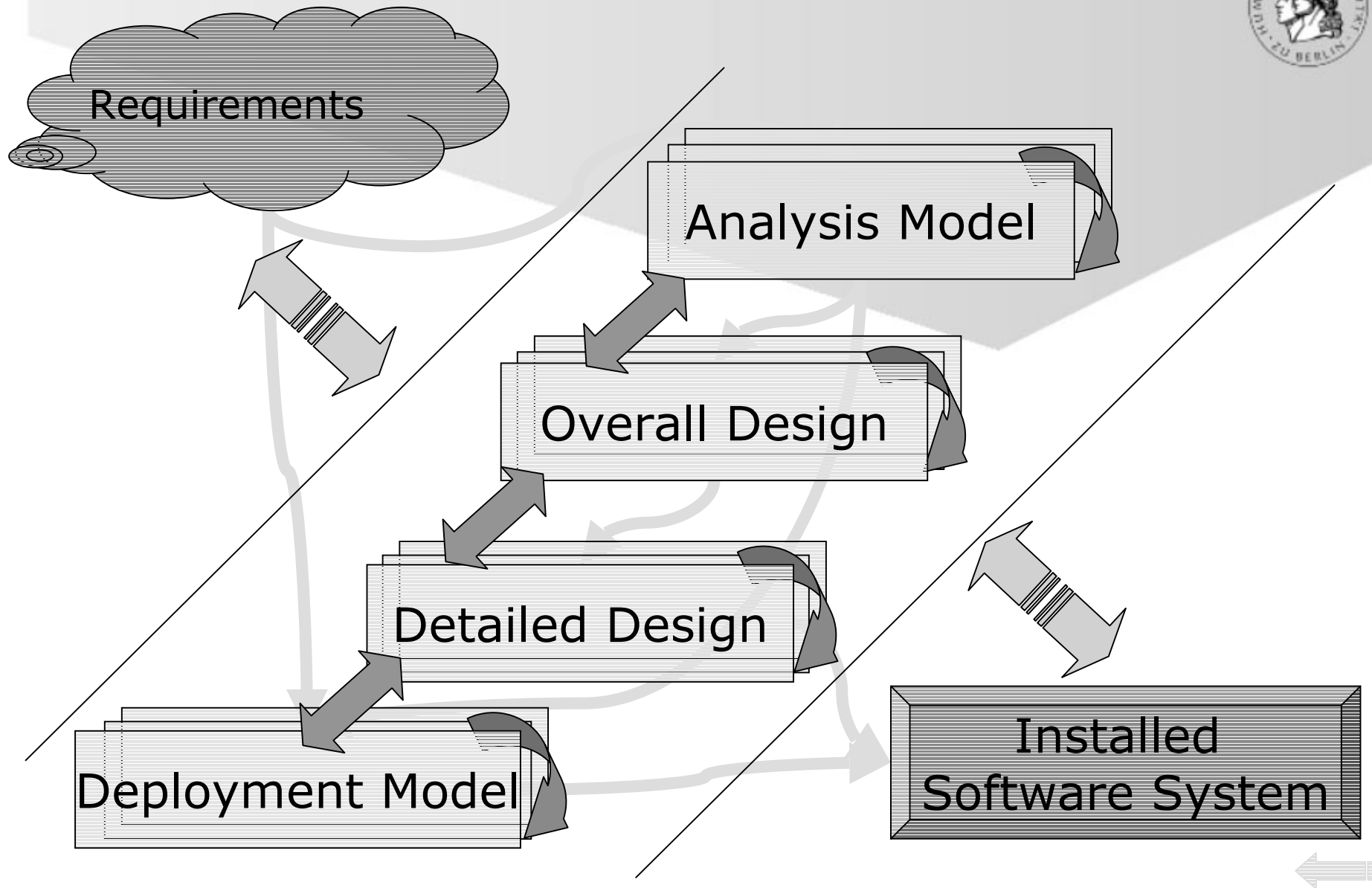


Software Engineering ...

... Deploys a *Variety of Models*

- Conceptual Abstraction
 - Decomposition and Refinement
 - Abstract Model Provides Context for Refinements
- Separation of Concern
 - Views
 - Each Model Describes the System from a Limited Point of View







Models and Specifications ...

- ... may be available beforehand in different modelling techniques
- ... may be developed using different modelling techniques

- The Design of Complex Software Systems uses a *Variety of Modelling Techniques.*





Formal Description Techniques

- **SDL**
 - Specification and Description Language
- **MSC**
 - Message Sequence Charts
- **VHDL**
 - Very High Speed Integrated Circuit Hardware Description Language
- **ASM**
 - Abstract State Machines
- **LOTOS**
 - Language of Temporal Ordering Specifications
- **Petri Nets**

Semi-formal Design Techniques

- **UML**
 - Unified Modeling Language
- **OMT**
 - Object Modeling Technique
- **Booch**
- **ER-Diagrams**
 - Entity Relationship Diagrams
- **State Charts**

Programming Language Style

- **IDL**
 - Interface Definition Language
- **ODL**
 - Object Definition Language
- **ASN.1**
 - Abstract Syntax Notation Nr. 1

Informal Techniques

- Requirements Specification Document
- Structured Text





Combination of Models

- How do the Information hold by the different models relate to each other?
- Classification of the Relationships between Models
- Strategies for the Transition between or the Connection of Models





Combination of Models

The Development of Complex Software System
Requires the Heterogenous Application of Different
Models and Modelling Techniques ...

... for the different Design Phases

- Analysis, Design, Implementation, ...
- *Process Driven Combination*

... within a single Phase

- Single Components, Views,...
- *Structural or Abstraction Driven Combination*

... for Different Design Activities

- Refinement, Implementation , Verification/Validation
- *Process Driven Combination*



Process Driven Combination

- **Synopsis:**
 - Models of one Design Phase or Activity serve as Starting Point for the next Phase or Activity
 - Different Languages are use for the Activities or Phases
- **Needs:**
 - Mapping between Models
- *Models of the single phases are related to each other*



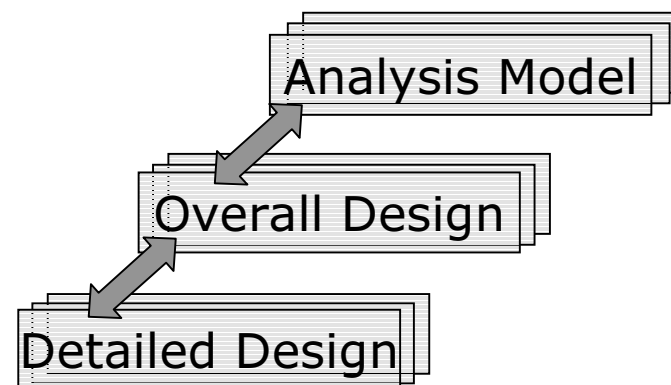


Esprit-Project InSyDe

- *Integrated System Design*
- Combination of Object-oriented Techniques with Domain-specific Techniques
 - OMT
 - SDL
 - VHDL
- Hardware-Software-Co-Design
- Project Partner
 - HU Berlin, VUB Brussels, DCU Dublin
 - Alcatel, Verilog, Intracom



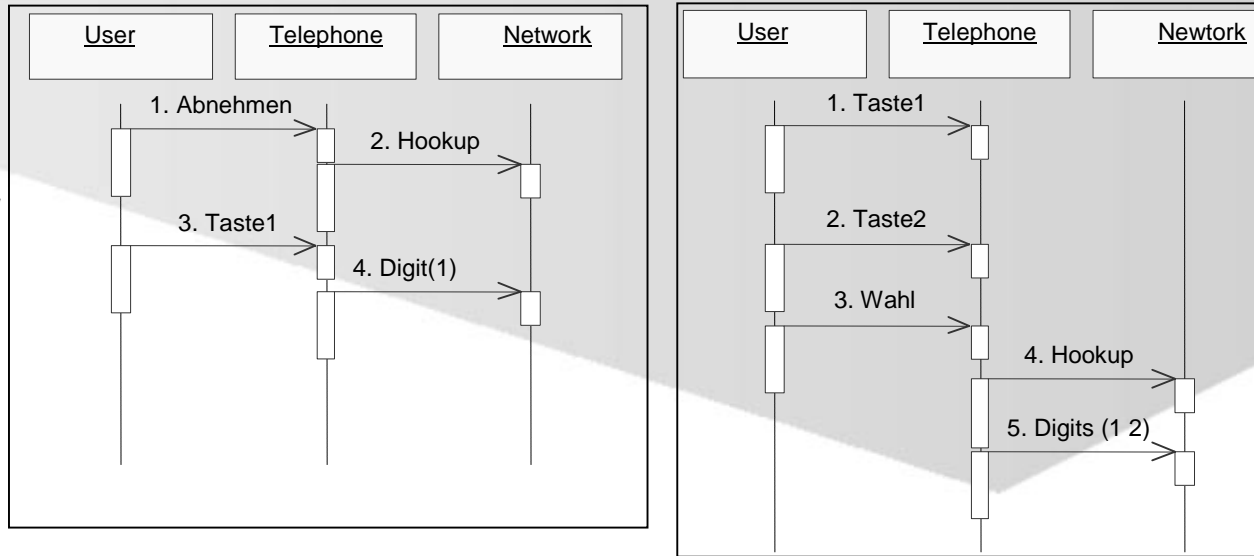
- **INSYDE**
 - Analysis Model: UML resp. OMT
 - Overall Design: UML resp. OMT*
 - Detailed Design: SDL and VHDL
 - Mapping between Phases by Translation



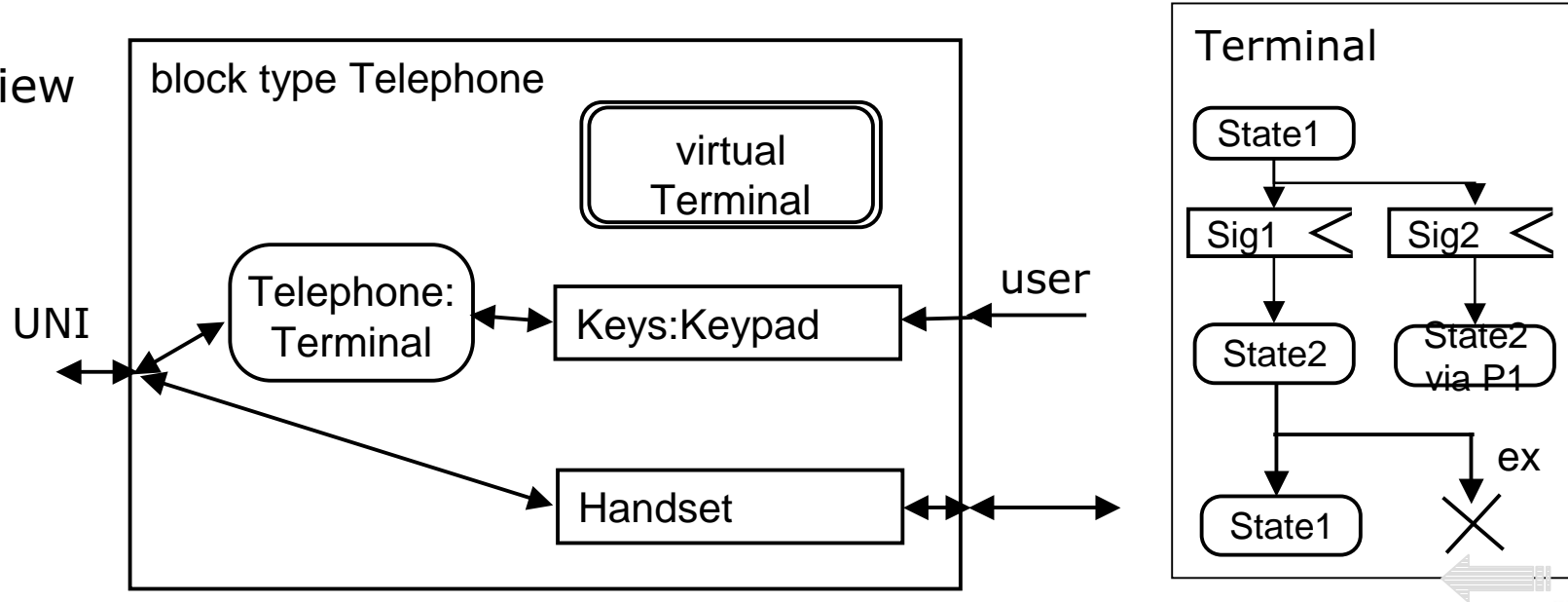
Combination of Models



External View
MSC



Internal View
SDL





Structural Combination

- Synopsis:
 - Decomposition into Sub-Models
 - Application of different Domain or Task Specific Languages for the Sub-Models
- Needs:
 - Description of the overall System (Context)
 - Interface Specifications
 - Open Sub-Models
- *Overall Model is Composition of the Sub-Models*

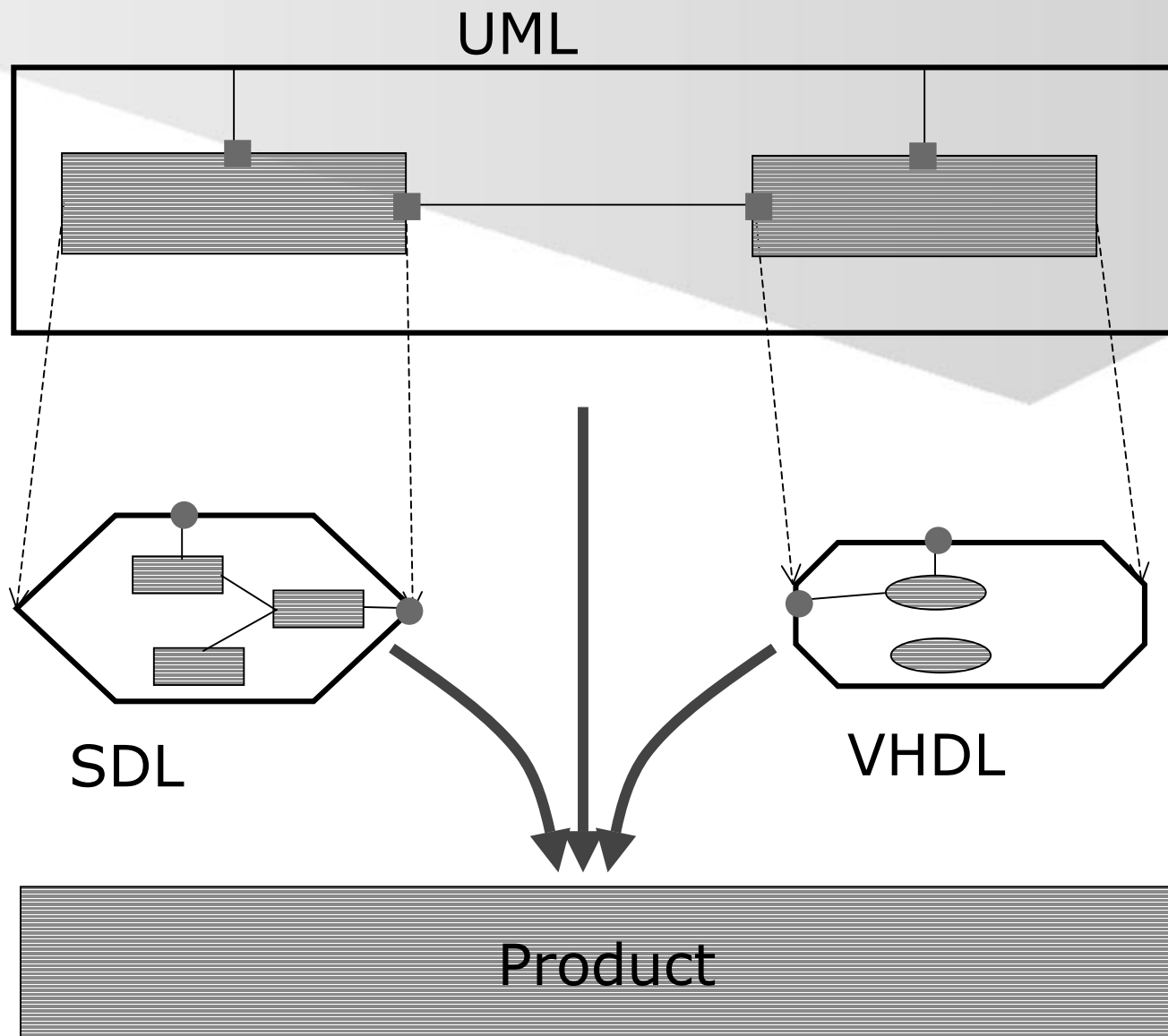




- **INSYDE**
 - Interface Definition/Overall Structure in OMT* resp. UML
 - Common VHDL- and SDL-Communication Concepts
 - Additional Reflection in SDL and UML
 - Sub-Models in SDL and VHDL (Detailed Design)
 - Interfaces Provide Basis for Connection Points to Environment

Overall Design

Detailed Design





- **SDL-ASN.1**
 - Overall System (Structure and behaviour) in SDL
 - Data Structures and Signatures in ASN.1
 - Embedded Sub-Models
 - SDL provides Context





Abstraction Driven Composition

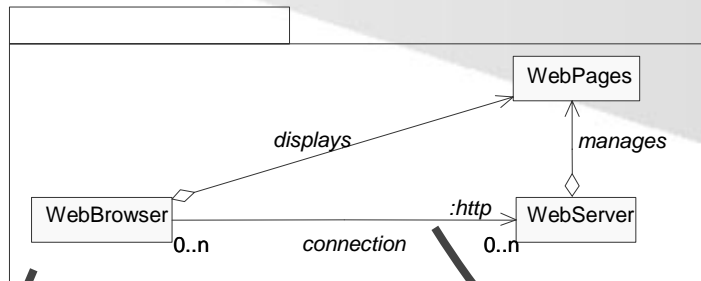
- Synopsis:
 - Models describe the System within the same Phase but using different Abstractions (views)
 - Views deploy different Languages
- needs:
 - Overall View resp. Relation between Views
 - Identification of Reference Points
- *Overall Model is Superimposition of the Sub-Models*



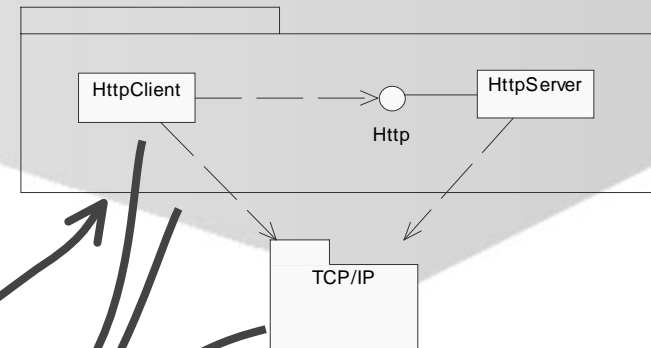
Combination of Models



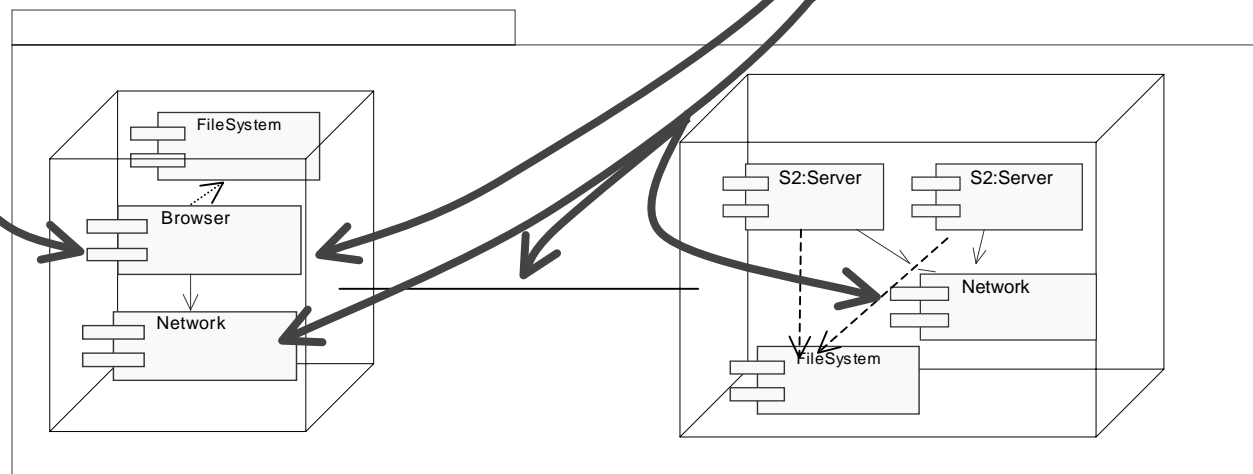
Functional View



Inter-process Communication



Configuration View





- **SDL and UML (Z.100)**
 - Integration of UML-Notation in SDL (*class, association*)
 - Reflection of UML-Concepts with SDL
- **UML and SDL (Z.109)**
 - Identification of Common Concepts
 - Model Construction Rules
 - Model Notation Rules





Support of Model Combination

Language Integration

- Single Language
- Complex Language, Semantic Problems
- Example: UML (Booch + OMT + OOSE)
- Example: ESPRIT-Project SPECS (SDL+LOTOS+Estelle)
- Includes all Elements of the Single Languages
 - Least Common Multiple
 - Unification of Similar Concepts





Translation

- Adapted and Limited Concept Space
- No Explicit Grammar for Semi-formal Techniques
- Strong Dependence on Grammar Changes
- Example: ESPRIT-Project INSYDE (OMT-OMT*-SDL/VHDL)
- Example: SDL-ASM
- Concentration on a Common Kernel
 - Greatest Common Denominator
 - All Specific Concepts of the Single Languages are derived from the Kernel





Support of Model Combination

Cooperation

- Based on Concept Space (Meta-Model, abstract Grammar)
- No Dependency to Concrete Notations
- Example: SDL-UML (Z.109)
- Example: OMG *Model Driven Architecture* (MDA)

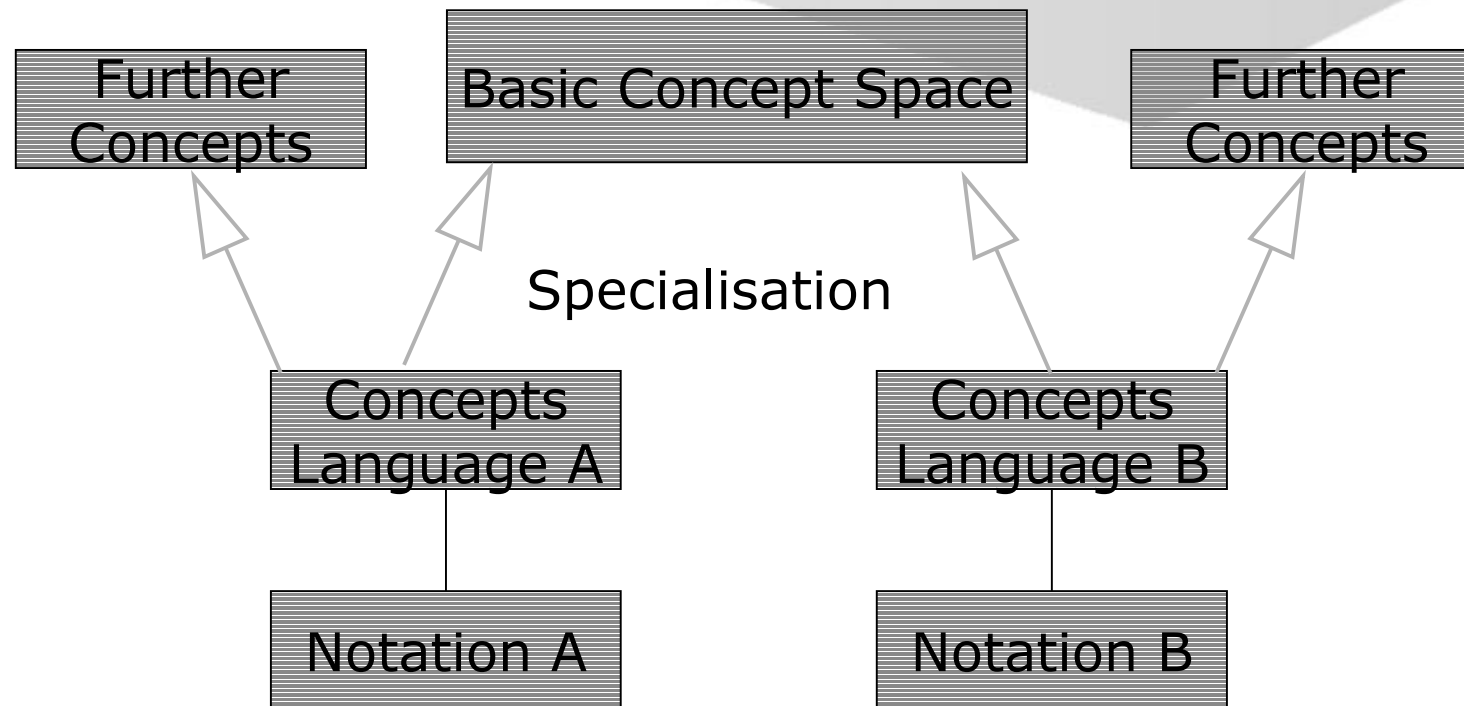


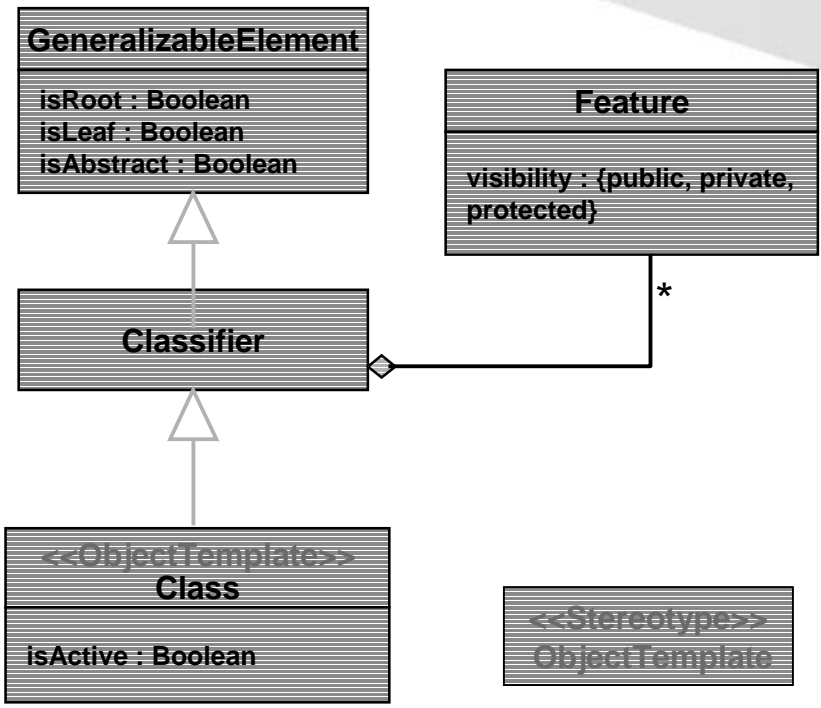
Concept Space based Combination

- Synopsis:
 - Models are made of Concepts
 - Notations visually Reflect all or a Subset of the Concepts
 - Concept Space Definition given by
 - Abstract Grammar
 - Meta-Model
- Combination Styles
 - Common Concept Space as Foundation
 - Relation to a Common Concept Space



Common Foundation





```

Object-template      :: Agent-type-definition /
                    Data-type-definition
Agent-type-definition :: Agent-type-name
                    Agent-kind
                    [ Agent-type-identifier ]
                    Agent-formal-parameter*
                    Data-type-definition-set
                    (...)
                    Procedure-definition-set
                    Agent-definition-set
                    Gate-definition-set
                    Channel-definition-set
                    [ State-machine-definition ]
    
```

Meta-Model

Abstract Grammar





- UML
 - each Class-Definition is an Object-Template
- SDL
 - each Agent-Type-Definition is an Object-Template
 - each Data-Type-Definition is an Object-Template



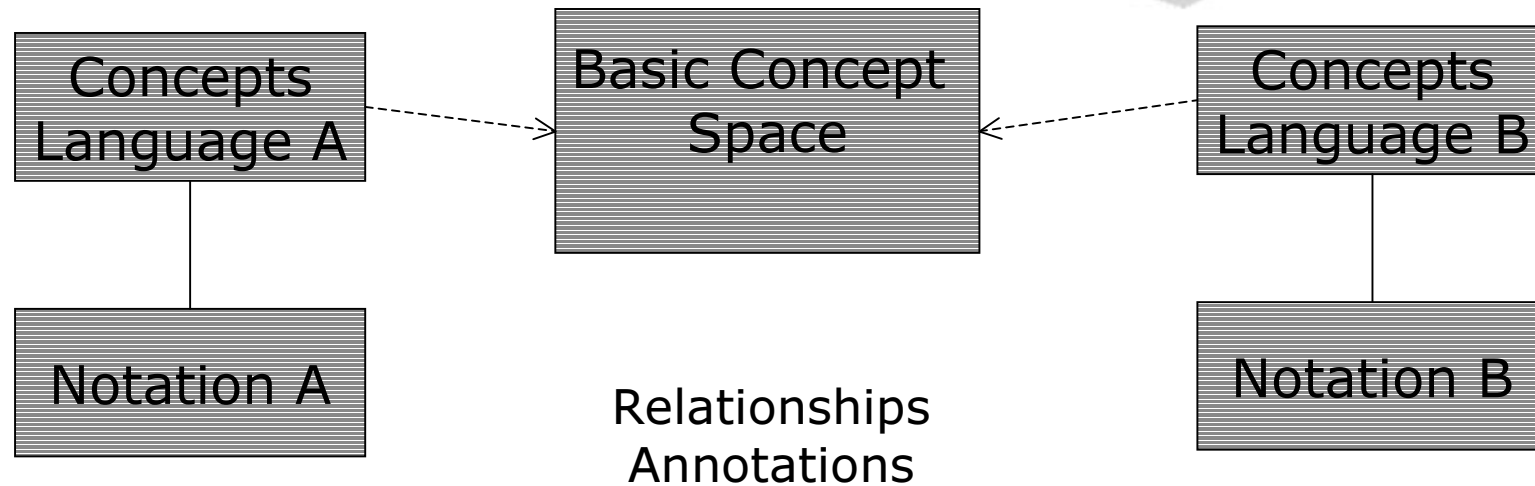


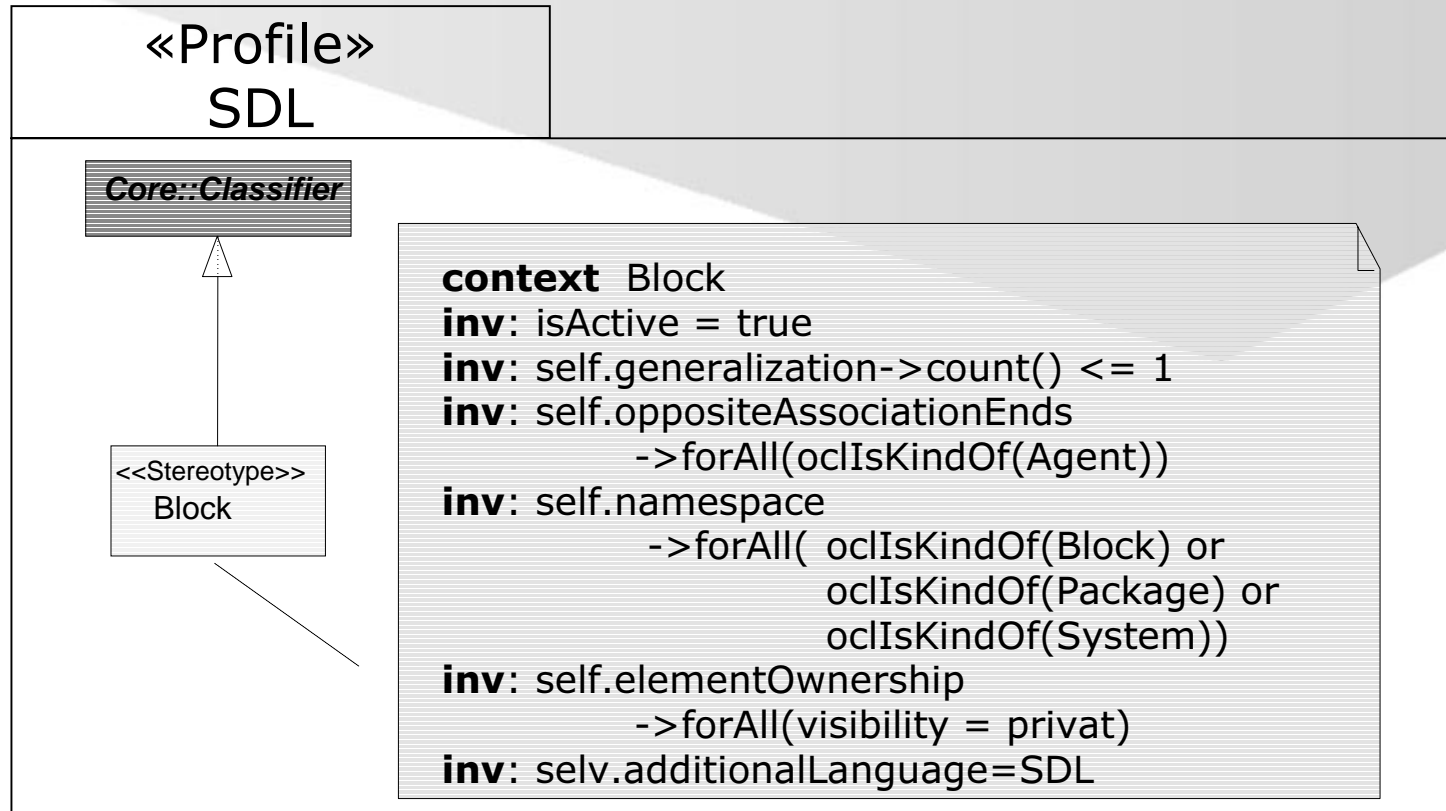
- Common Concept Space as Foundation
 - Requires a Modification of the Language Definition(s)
 - Meta-Model
 - Abstract Grammar
 - Can be Hidden at the Notation Level
 - Is Suited for a Process Driven Combination

 - UML-Concepts in SDL (*Type references*)
 - from SDL to UML

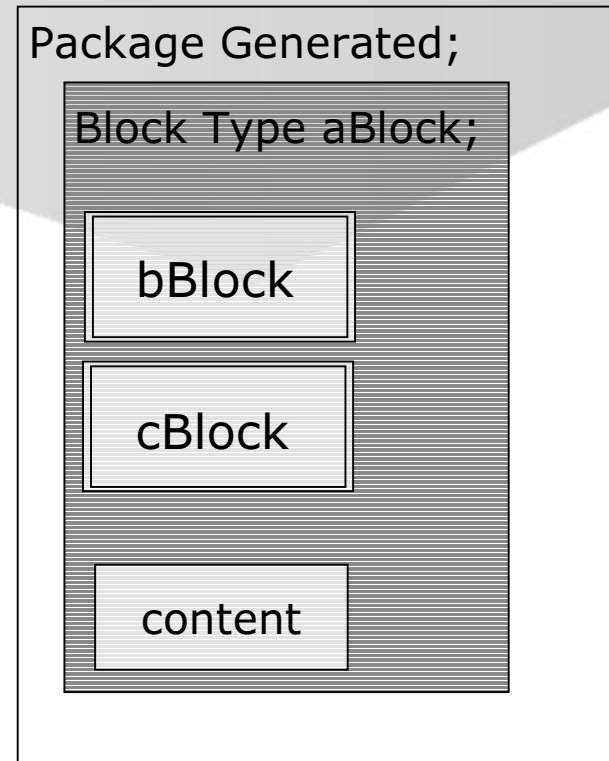
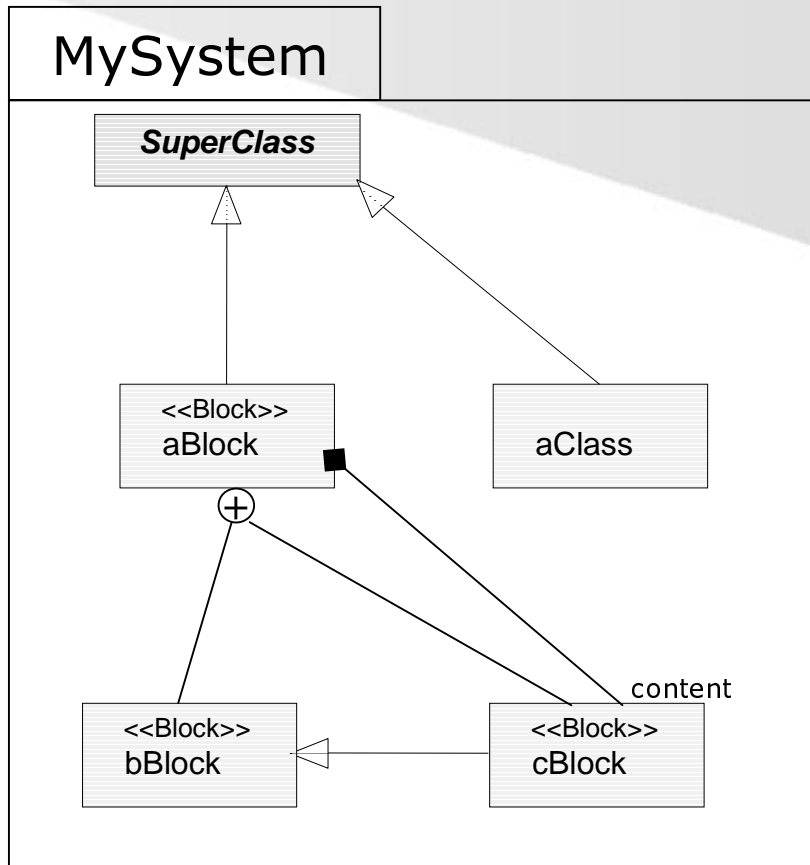


Relation to a Common Concept Space





at Model Level





- UML
 - The Tagged Classes have Block-Properties
- SDL
 - Only those Classes which are Tagged in the UML-Model are Reflected





- Relation to a Common Concept Space
 - requires Modification to the Language Definition(s) or Extensions at **Model Level**
 - requires Extensibility Features like
 - special Attributes within the Abstract Grammar
 - or Stereotypes within the Meta-Model
 - Is Reflected at Notation Level
 - selective Designation (Attribute Values or Tags)
 - Standard Notation or Dedicated Symbols
 - Is Suited for a Structural and a Abstraction Driven Combination

 - UML-SDL-Profil
 - from UML to SDL



Requirements on Language Development

Concept Based Language Definition

- Concept Space is Primary
 - Specify Concept Space as Meta-Model or Abstract Grammar
- Notation is Derived
 - Assign Notation Elements to Concepts
i.e. Graphical Symbols to Concepts



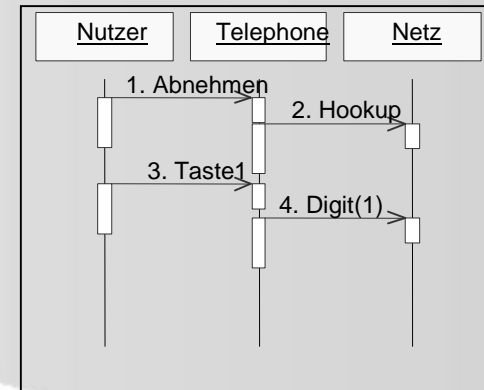
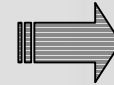
Relation to Basic Concept Space

- Meta-Model-Stereotypes/Attributation
 - Modular Language: Core and Advanced Concepts
 - Clearly defined Points and Mechanisms to Extended and/or specialize the Language
- Profile/Sub-Language
 - Set of Extensions/Restrictions
 - Selection/Visualization at Noation Level

Concept Based Interface to CASE Tools

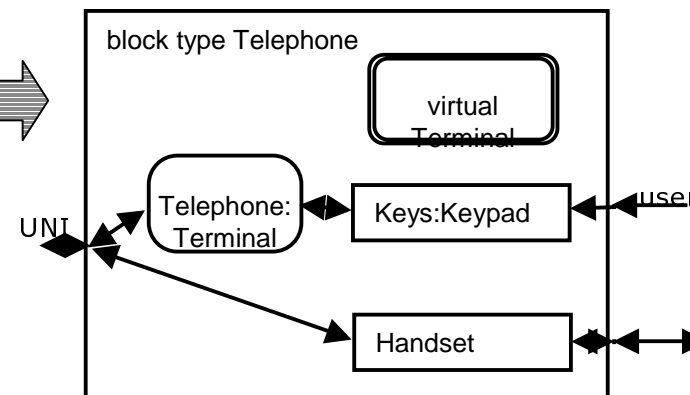
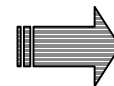
- APIs
 - Tools are Based on Concept Space
Interface corresponds to Concept
 - cf. UML-Facility
 - Navigation, Modification, Extraction Creation of Models
- Explicit and flexible Exchange Format
 - cf. XML/XMI
 - at least all Concepts – additionally Notation Information
- Model Repositories

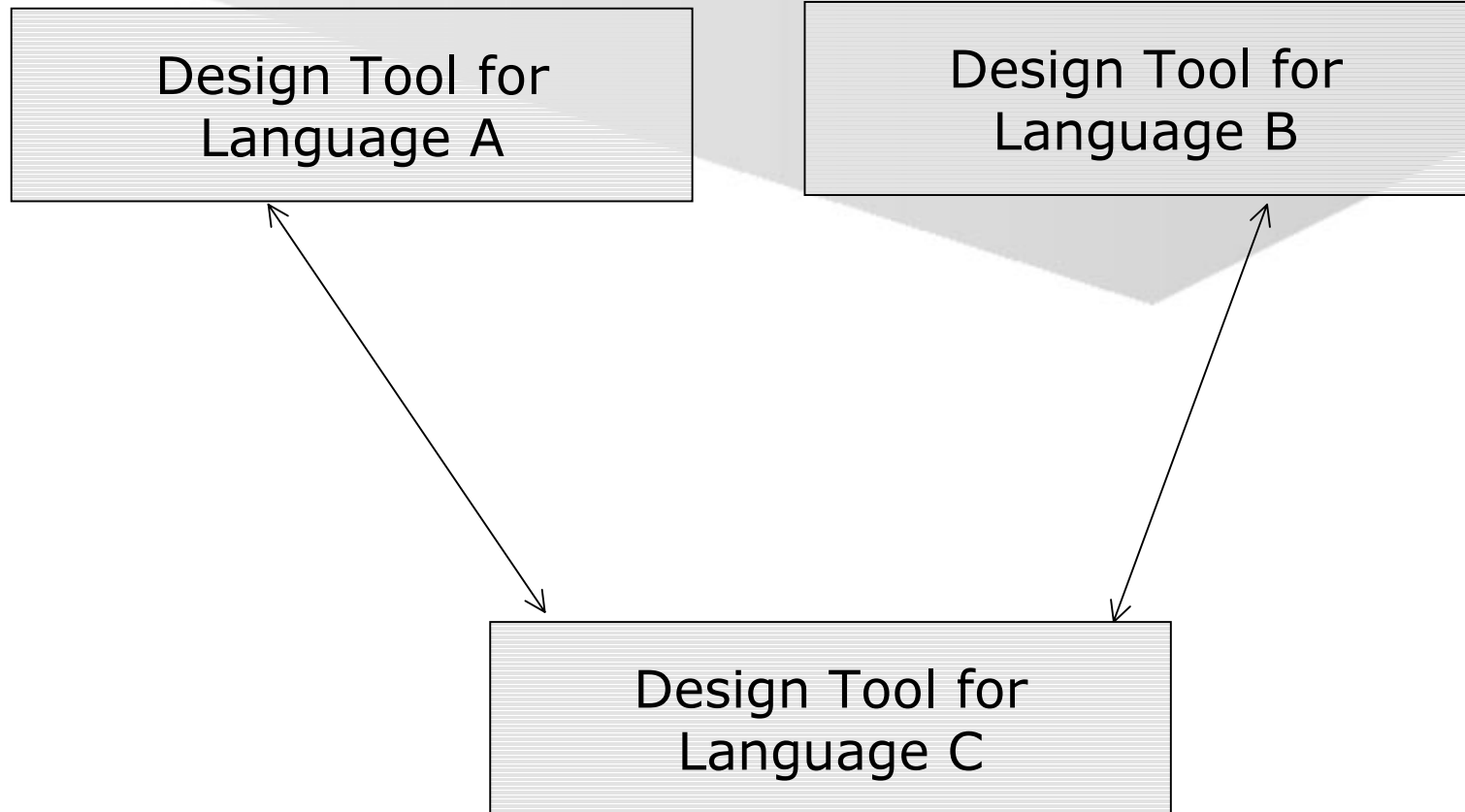
Design Tool for Language A



getModellElements(type, tag)
addModellElements(type, tag)

Design Tool for Language B

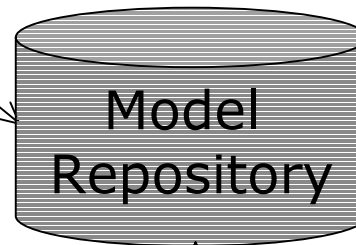




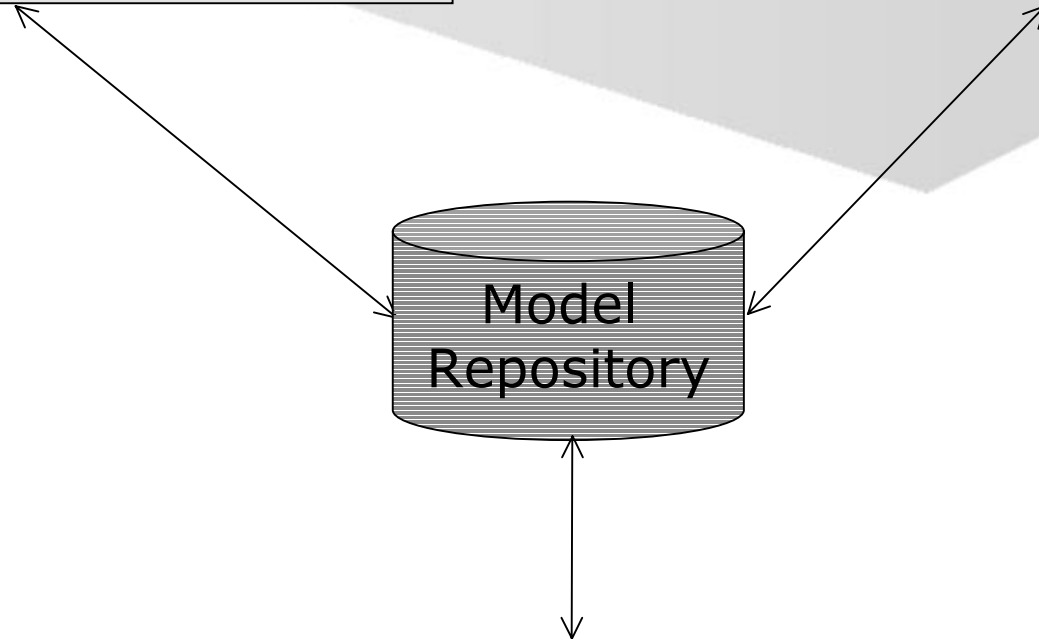


Design Tool for
Language A

Design Tool for
Language B



Design Tool for
Language C





Summary and Conclusion

- Software Engineering deploys a variety of Models and Modelling Techniques
- The Relations between Models are Determined by
 - Development Process
 - Structural Requirements
 - Level of Abstraction
- Language Integration and Translation have only a Limited Applicability
 - Semantic Problems
 - Expressivness
 - Size of the Language



- An Concept Spacebased Approach ...
 - Allows a Selective Combination
 - Is to a High Degree Stable against Notation Changes
 - Supports all Kinds of Model Combination
 - Puts higher Requirements on Design Tools
- Future Language Developments Should Include Extensibility Features