Part II – Black-Box Composition Systems
20. Finding UML Business Components in a Component-Based Development Process

Lecturer: Dr. Sebastian Götz

Prof. Dr. Uwe Aßmann

Technische Universität Dresden

Institut für Software- und Multimediatechnik

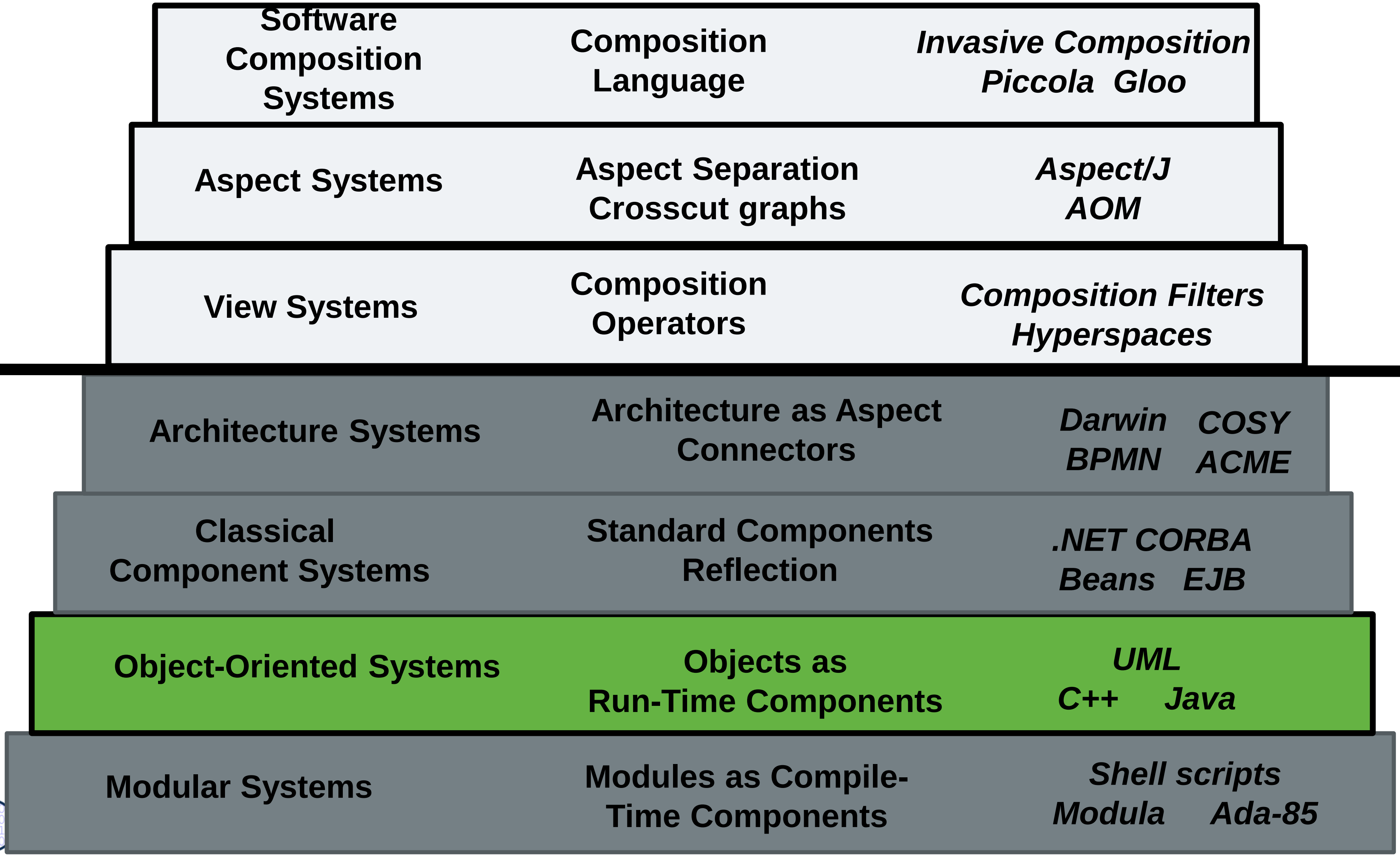
http://st.inf.tu-dresden.de/teaching/cbse

19.04.2018

Literature

- J. Cheesman, J. Daniels. UML Components. Addison-Wesley.

The Ladder of Composition Systems



Software Composition Systems	**Composition Language**	***Invasive Composition Piccola Gloo***
Aspect Systems	**Aspect Separation Crosscut graphs**	***Aspect/J AOM***
View Systems	**Composition Operators**	***Composition Filters Hyperspaces***
Architecture Systems	**Architecture as Aspect Connectors**	***Darwin COSY BPMN ACME***
Classical Component Systems	**Standard Components Reflection**	***.NET CORBA Beans EJB***
Object-Oriented Systems	**Objects as Run-Time Components**	***UML C++ Java***
Modular Systems	**Modules as Compile-Time Components**	***Shell scripts Modula Ada-85***

20.1 The Cheesman-Daniels Business Component Model

- *Problem*: UML classes do not specify required interfaces, which is necessary for UML components
- The Cheesman-Daniels process helps to find components from UML class diagrams
- Using the "Business component model"

Business Objects are Complex Objects

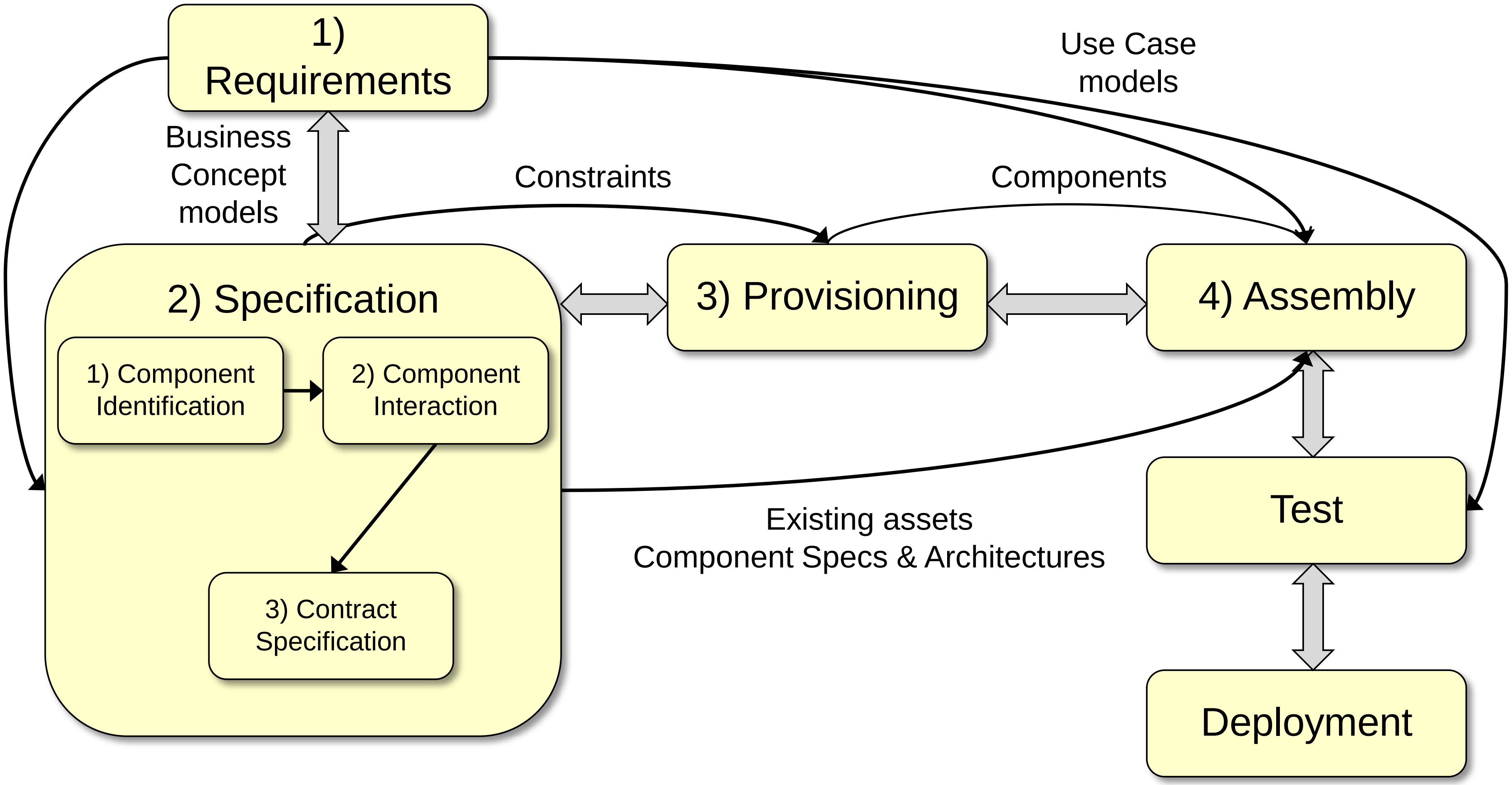
- In the Cheesman-Daniels component model, a **business component** consists of a set of business objects and other business components (part-of relation)
  - The smallest component is a *business object with several provided and required interfaces*
  - The business objects are the logical entities of an application
  - Their interfaces are re-grouped on system components for good information hiding and change-oriented design
- A business component has a specification containing all interfaces and contracts and an implementation
  - UML-CD are used (UML profile with stereotypes)

Goals of the Cheesman-Daniels Process

- The Cheesman-Daniels Process identifies UML components in UML class diagrams
  - It bridges
    - *domain modelling* with
    - *use case modelling* (functional requirements)

Identifying Business Components with the Cheesman-Daniels Process

- Overall development process



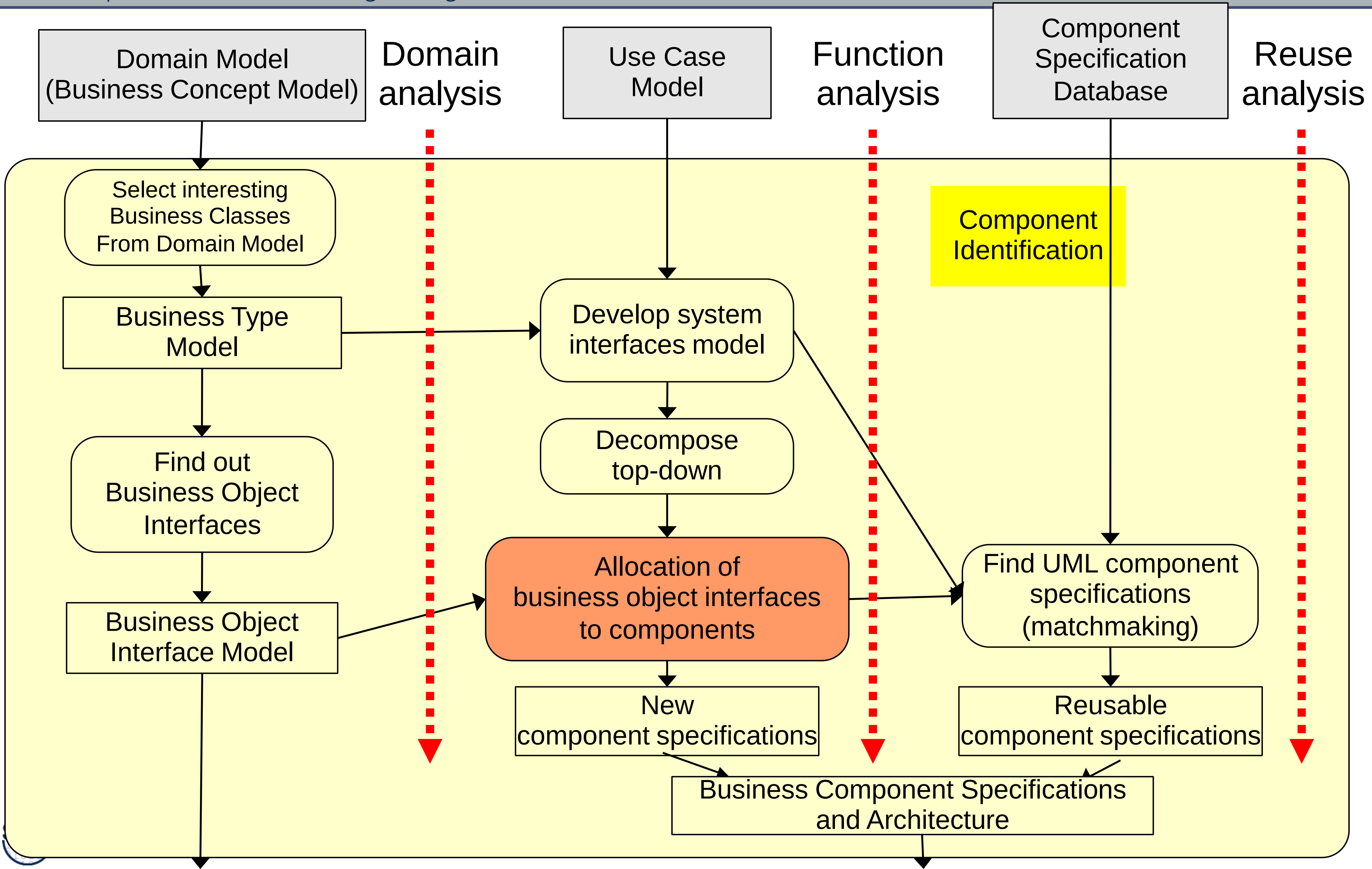
Simplified version of Fig. 2.1 from Cheesman/Daniels

Artifacts of the Cheesman/Daniels Process

- Requirement artifacts:
  - *Domain model (business concept model):* describes the business domain (application domain)
  - *Use case model* (requirements model)
- System artifacts, derived from the business concept model:
  - *Business type model*, class diagram derived from domain model:
    - Represents the system's perspective on the outer world (more attributes, refined class structures from the system's perspective)
  - *Business object interface model*, identifies the business objects and all their interfaces
  - *Business object model*, derived from the business object interface model by adding additional operations
- System component artifacts
  - Component interface specifications: one contract with the client
  - Component interface information model (state-based model)
  - Component specifications: all interface specifications of a component plus constraints.
  - Component architecture: wiring (topology) of a component net.

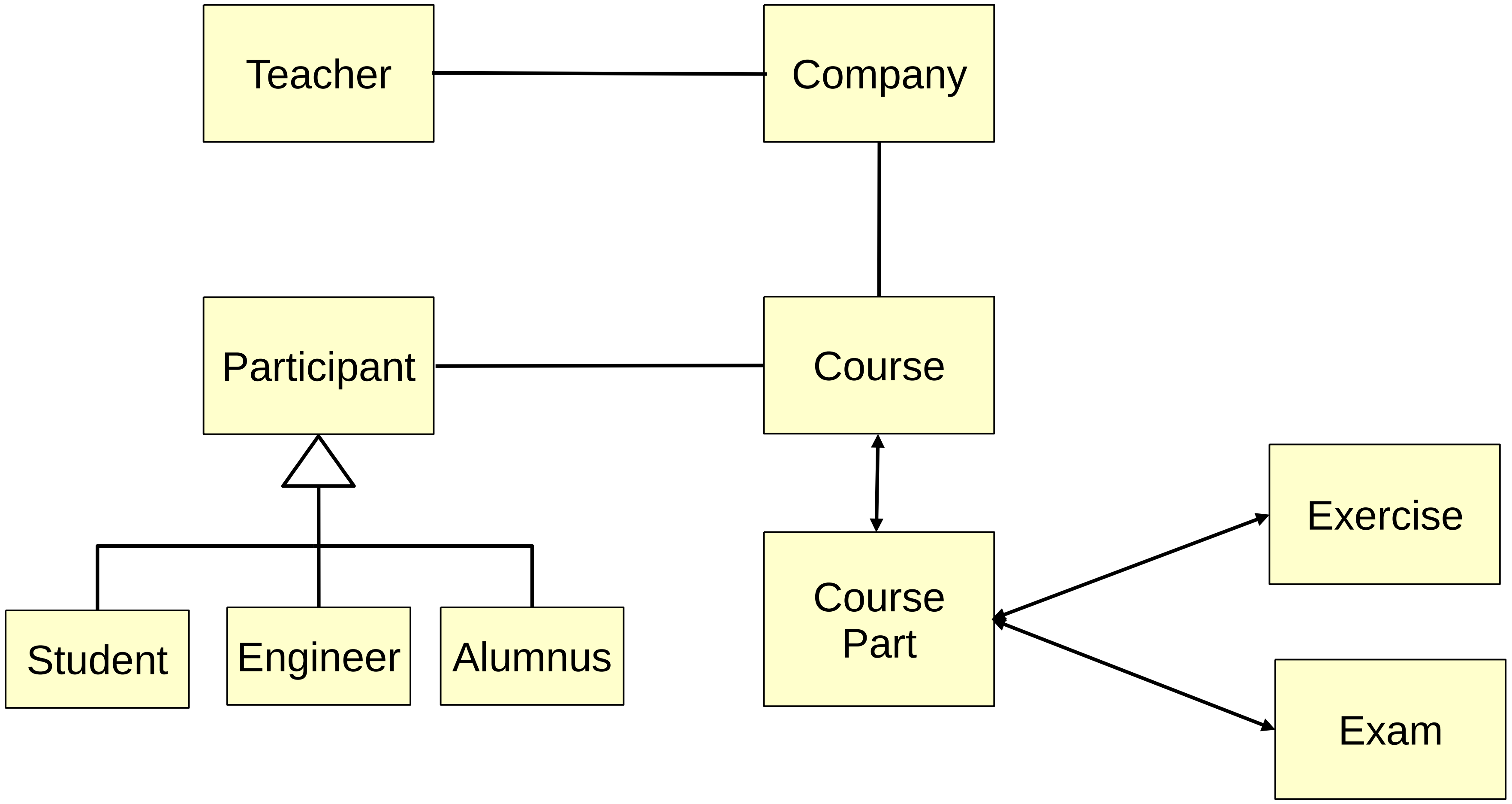
20.2. Identifying Business Components

Component Identification (Step 2.1)


Domain Model (Business Concept Model)
Domain analysis
Use Case Model
Function analysis
Component Specification Database
Reuse analysis
Select interesting Business Classes From Domain Model
Component Identification
Business Type Model
Develop system interfaces model
Decompose top-down
Find out Business Object Interfaces
Allocation of business object interfaces to components
Find UML component specifications (matchmaking)
Business Object Interface Model
New component specifications
Reusable component specifications
Business Component Specifications and Architecture

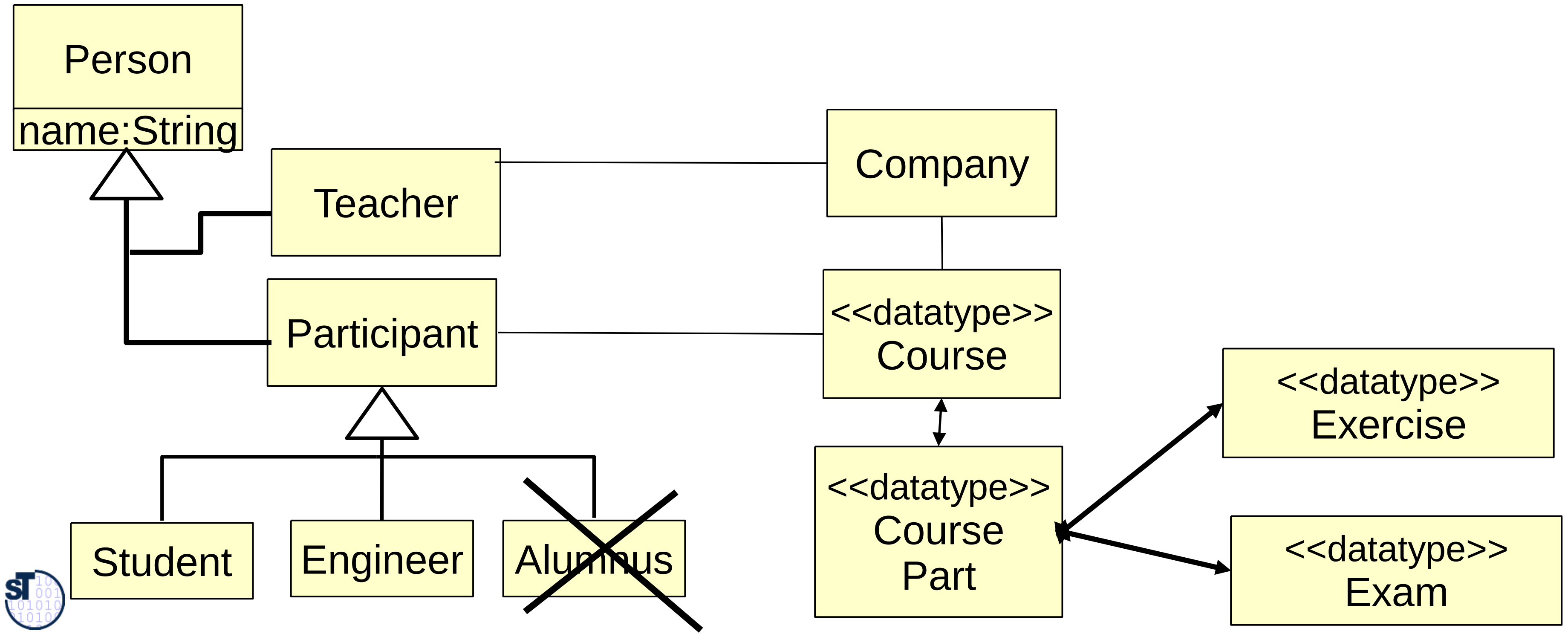
Ex.: Domain Model of a Course-Management System

- Collects all concepts of the domain (aka business concept model)



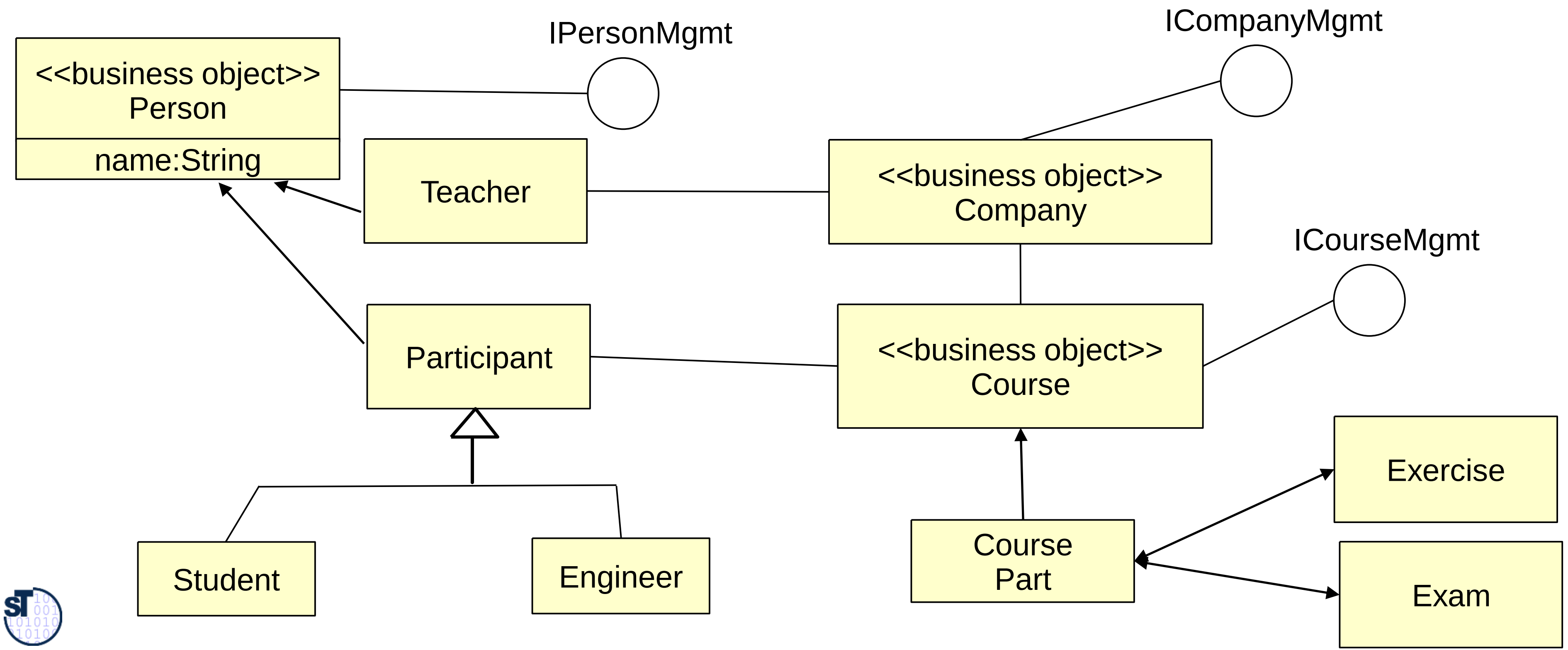
Step 2.1a) Business Type Model

- Shorten the domain model by selecting system types from the domain model
  - Eliminates superfluous concepts
  - Adds more details
  - Distinguish datatypes (passive objects, materials, persistent entities)



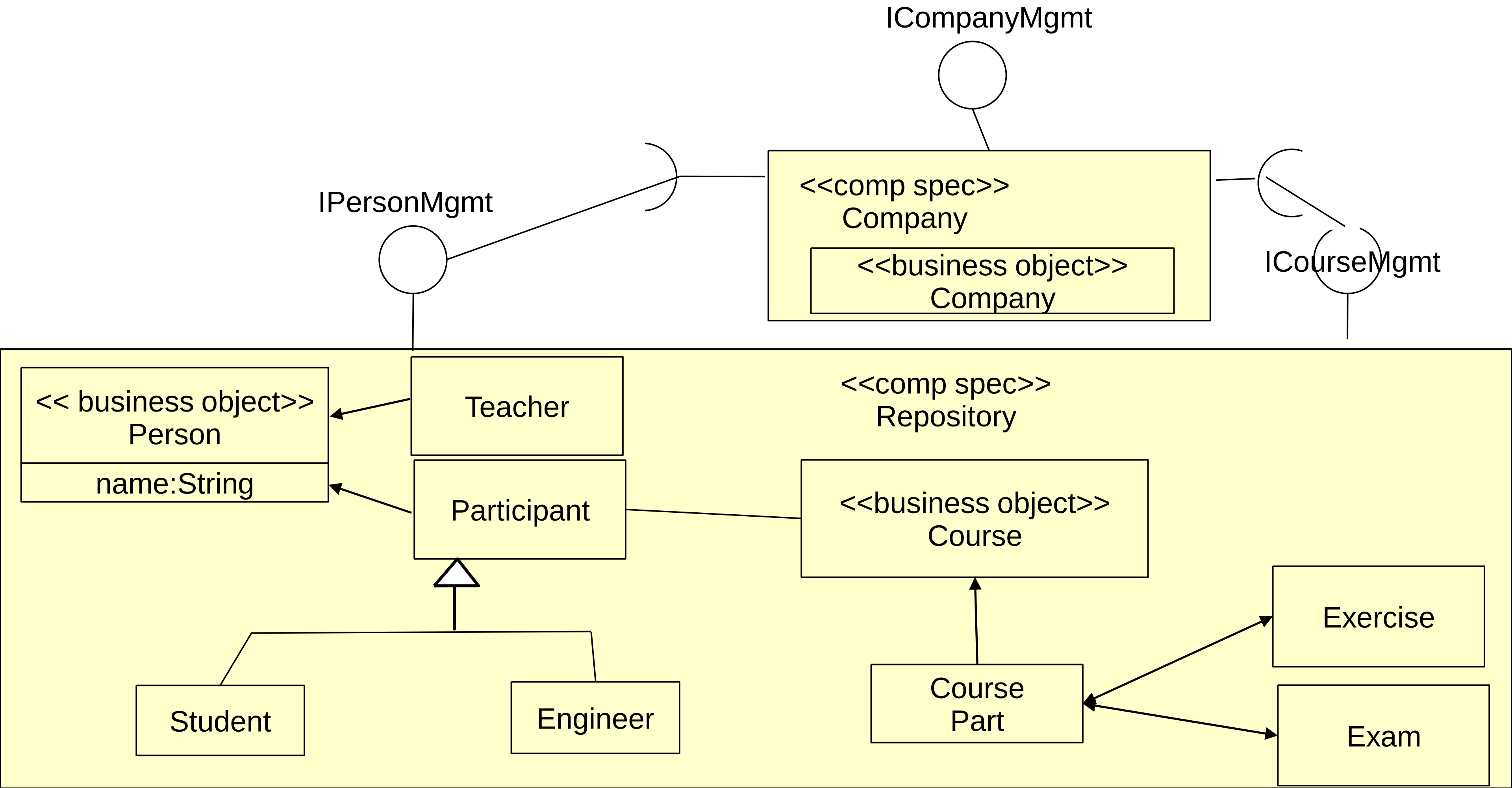
Step 2.1b) Identifying Business Object Interfaces

- Identifies business objects from the business type model
  - And defines *management interfaces* for them
  - Here, only Company, Course, Person are business objects, all others are dependent types



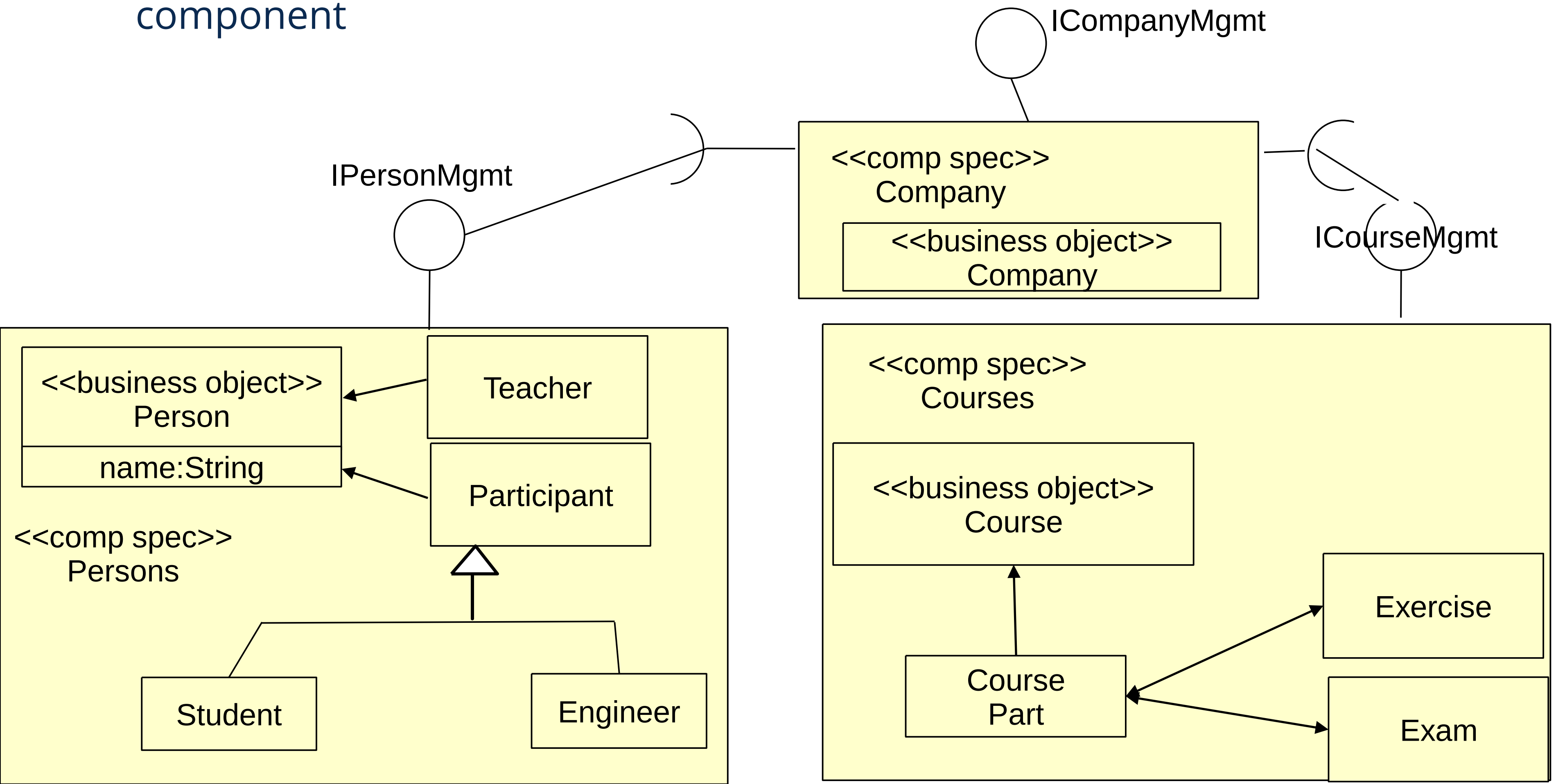
Step 2.1c) Component Grouping

- Group classes and interfaces into reusable components



Alternative Component Grouping (Version 0.2)

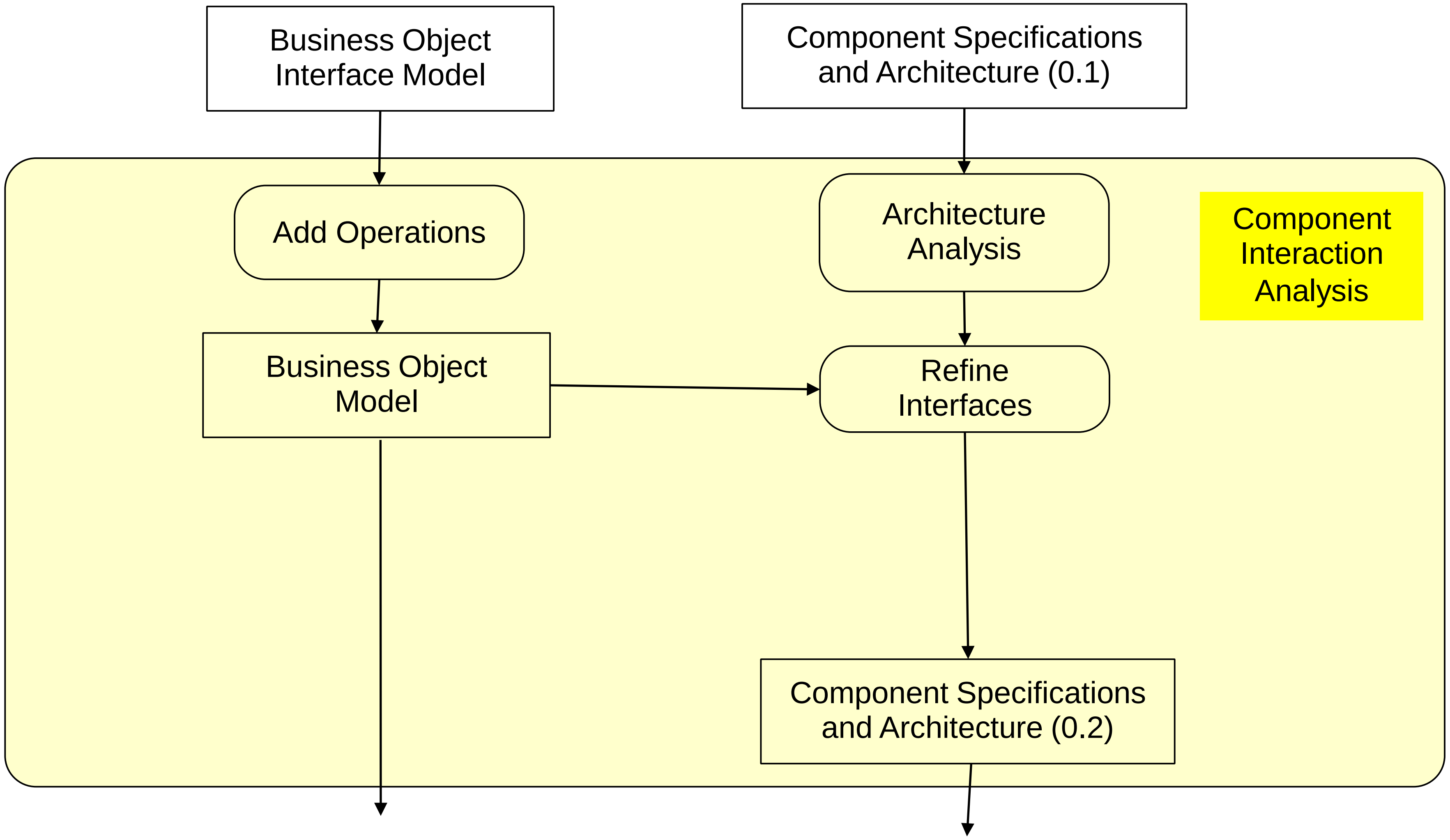
- Often, classes and interfaces can be grouped in several ways into components. Goal: think about what is reusable
- Here: Person management might be reuseable, so make it a separate component



Component Identification

- The **component identification** subprocess attempts to
  - Create a business object interface model from the domain model (still without methods)
  - Attempts to group these interfaces to initial *system component specifications*
    - The grouping is done according to
      - *information hiding*: what should a component hide, so that it can easily be exchanged and the system can evolve?
      - *Reuse considerations:* which specifications of components are found in the component specification repository, so that they can be reused?
- There is a tension between business concepts, coming from the business domain (problem domain), and system components (solution domain). This gap should be bridged.

Step 2.2: Component Interaction Analysis for Refinement of Component Interfaces

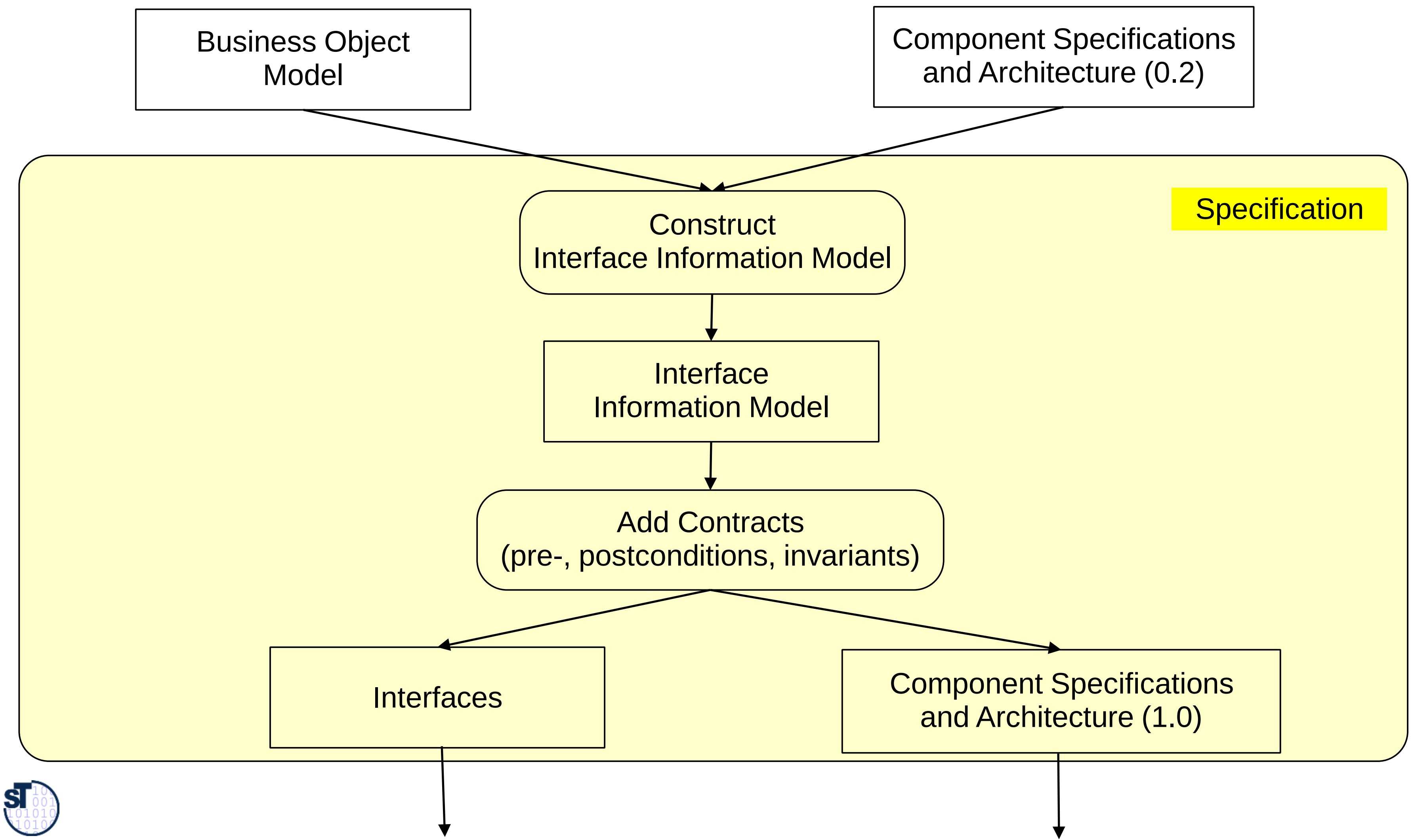

Business Object Interface Model
Component Specifications and Architecture (0.1)
Add Operations
Architecture Analysis
Component Interaction Analysis
Business Object Model
Refine Interfaces
Component Specifications and Architecture (0.2)

Component Interaction Analysis

- Component Interaction Analysis refines the results of the first stage
  - Removing,
  - Regrouping,
  - Augmenting,
  - Adding interfaces
  - Producing component specifications and wirings in a version 0.2
- Additionally, operations are added to business object interfaces
  - And mapped to internal types.

Step 2.3: Contract Specification

- Enrich the interfaces with contracts


Business Object Model
Component Specifications and Architecture (0.2)
Specification
Construct Interface Information Model
Interface Information Model
Add Contracts (pre-, postconditions, invariants)
Interfaces
Component Specifications and Architecture (1.0)

Contract Specification in OCL (Step 2.3)

- Specification of declarative contracts for UML classes in OCL
- **Invariants**:
  - Evaluate business domain rules and integrity constraints
  - Example:

```
context r: Course
-- a course can only be booked if it has been allocated in the
   company
inv:  r.bookable = r.allocation->notEmpty
```

- **Pre**- and **Postconditions** for operations (**assumptions** and **guarantees**)
  - Can only be run on some state-based representation of the component
  - Hence, the component must be modeled in an *interface information model*
  - Or: be translated to implementation code (e.g. Java using an OCL2Java Compiler)

```
context Course::book(cert:Certification)
-- a course can only be booked if the booker has an A-level
   certificate
pre:  cert.instanceOf A-level
```

Step 3: Provisioning (Realization, Implementation, Publishing)

- Provisioning selects component implementations for the specifications
  - Choosing a concrete implementation platform (EJB, CORBA, COM+, ...)
  - Look up component implementations in implementation repositories
    - Write adapters if they don't fit exactly
  - Program missing components
- And makes them available in component repositories
  - Store component implementations and specifications in database for future reuse

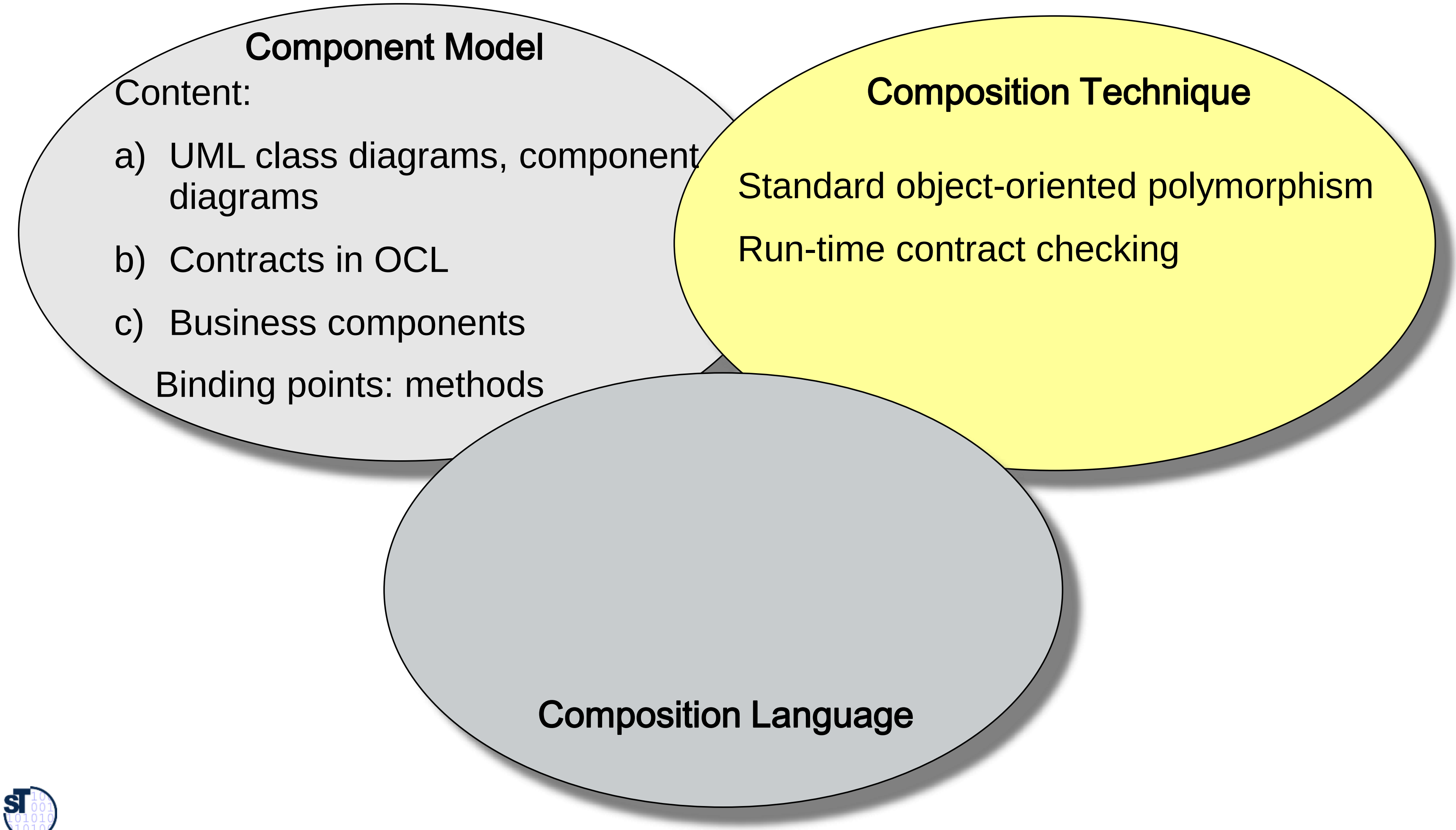
Step 4: Assembly

- Puts together architecture, component specifications and implementations, existing components
  - We will see more in the next lectures

20.3 Evaluation of Cheesman-Daniels Business Components

- No top-down decomposition of components, only bottom-up grouping from class diagrams
  - part-of relationship is not really supported
- Reuse of components is attempted, but
  - Finding components is not supported
    - Metadata
    - Facet-based classification

Cheesman-Daniels' Business Component Model as Composition System



Component Model

Content:

a) UML class diagrams, component diagrams

b) Contracts in OCL

c) Business components

Binding points: methods

Composition Technique

Standard object-oriented polymorphism

Run-time contract checking

Composition Language

The End