Karlsruhe Institute of Technology

FOR EXTERNAL USE ONLY

Softwaretechnik II

Oliver Hummel, IPD

Topic 8
Software Components

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS

sdq.ipd.kit.edu

Overview on Today's Lecture

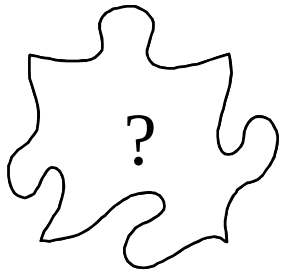
- Content
  - introduction to software components
  - overview of technical component models
    - and web services
  - scientific component models
    - KobrA
    - Palladio
- Learning Goals
  - get acquainted with fundamental ideas of software components
  - be able to apply componentization principles in your software architectures
  - develop a fundamental understanding of Palladio

Software Components

- Building blocks for software
  - a natural concept, however, somewhat difficult to define
- Also, not a new idea!
  - first proposed by McIlroy (1968 at the NATO-Conference in Garmisch)
  - idea was to allow software reuse

- *"Components are for composition, much beyond is unclear…"* (Szyperski)

- Frequently asked questions
  - does a component have state?
  - is a component (more or less than) an object?
  - is a component a module?



Definition Attempts

- The term component is often just used in the generic sense of building block
- However, various authors have attempted to give a more precise definition

> a component represents a modular, deployable, and replaceable **part of a system** that encapsulates implementation and exposes a set of interfaces "
>
> UML 1.5 Specification

> a reusable software component is a logically cohesive, **loosely coupled module** that denotes a single abstraction "
>
> Booch 87

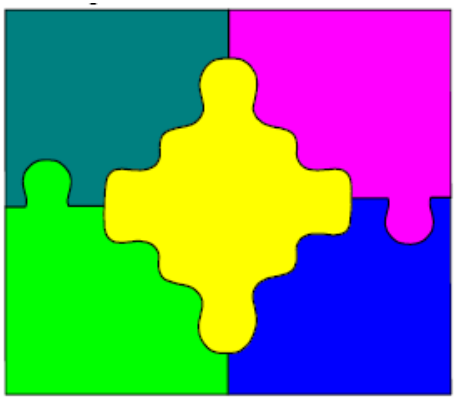
- Today's most commonly accepted definition

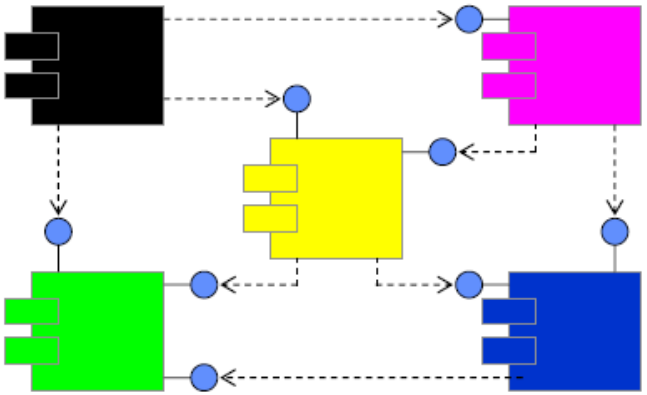
> A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."
>
> WCOP'96 [Szyperski]

This is a Component

- A component is a contractually specified building block for software which can be readily used by third parties without understanding its internal structure. (Reussner)
  - not necessarily black-box: information on component's internals may be provided for tools
  - readily used: effort for deployment, assembly, composition or adaptation should be as low as possible.

➔ Systems can be composed from components





Why Object CANNOT equal Component

➔ Inheritance is conflicting with the Black-Box-Principle of Components

```
class A {
   public m {
     ... x(); ...
   }
   protected x(){...}
}
```

```
class B extends A {
   protected x(){
        ... // redefinition
   }
}
```

```
...
B b = new B();

b.m(); // result unclear
   for developer of B and A!
```

Components in Java

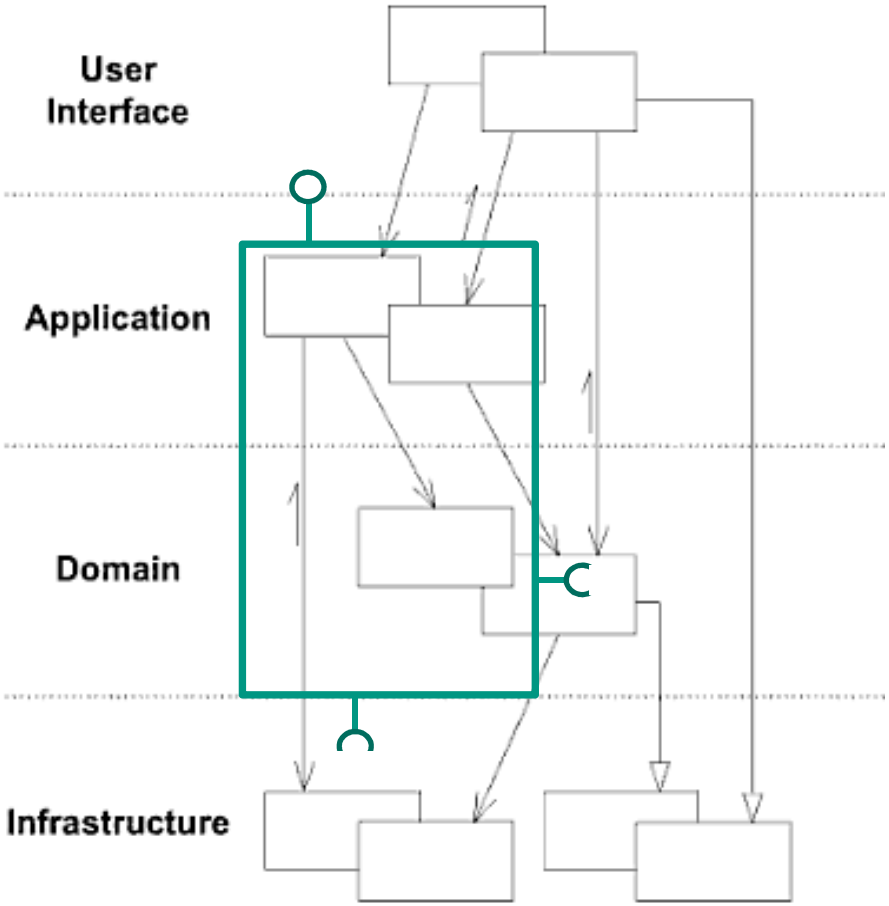
- There is no such thing as a *component in Java*
  - ➔ how can we mimic them there?

Component: Where art Thou?

- Components are feature oriented

➔ they usually comprise the application and domain layer
  - sometimes called business components in an enterprise context

- infrastructure may also form nice components
  - such as JDBC, for example

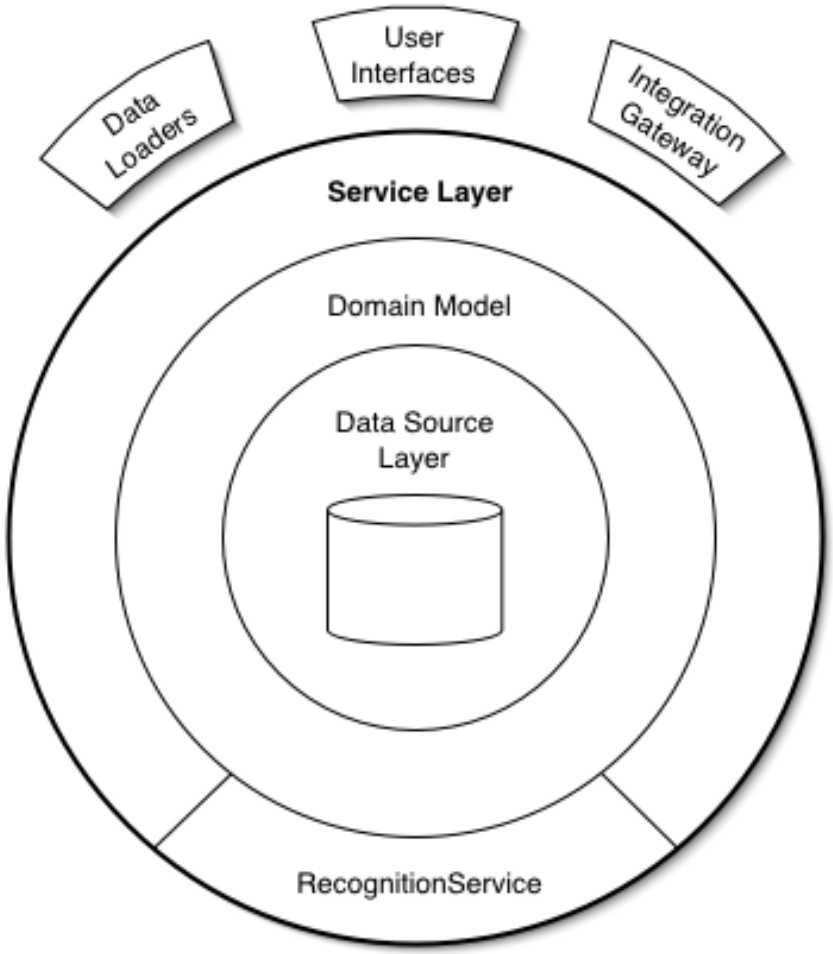


Corresponding Pattern: Service Layer

- *Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation.*

  [Fowler]

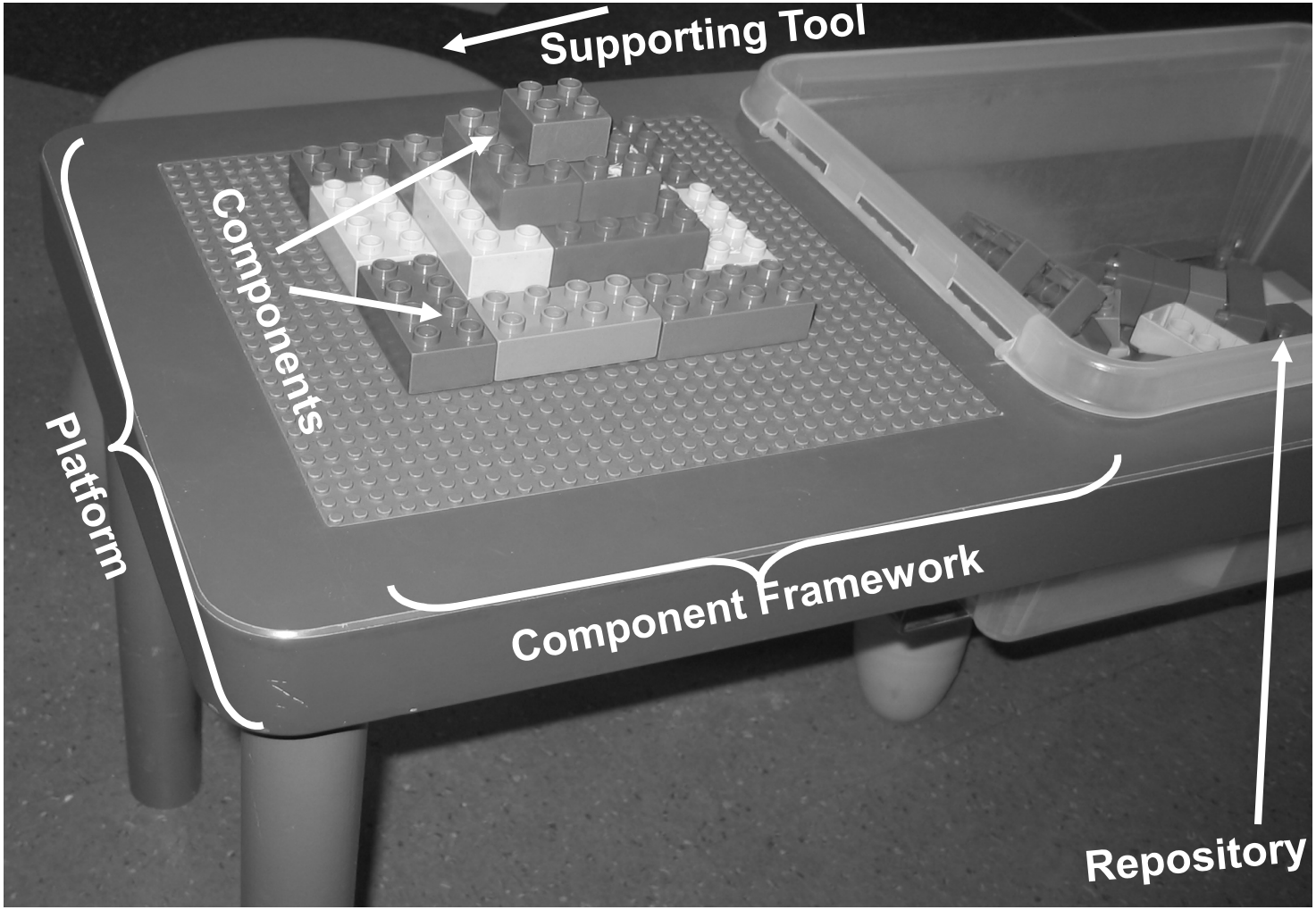
- = Larman‘s Application Layer



Component Model

- Concrete realisation of the principles of CBSE
  - in practice, only components built for the same model can interoperate
- Defines –
  - What is a component?
  - How does a component offer services?
  - How are components connected / composed?
  - How do components communicate?
  - Where can components be found?
- Often also providing an infrastructure for executing components
  - sometimes also called component framework

Component Framework Visualization



(courtesy of Professors Grunske & Crnkovic)

Technical Realisation of Components

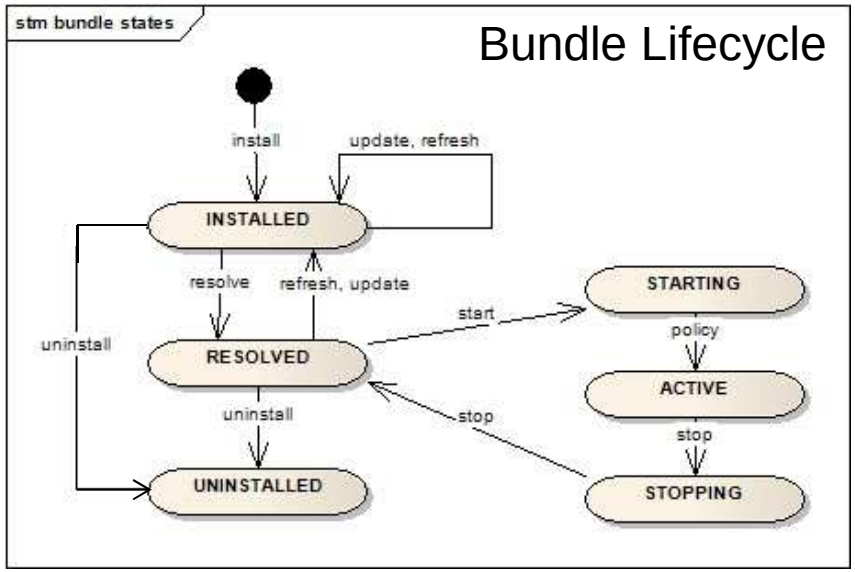
- Technical realisation of CBSE ideas in recent years
  - The OMG way: CORBA, CCM, OMA
  - The Sun way: Java, JavaBeans, EJB
  - The Microsoft way: COM, OLE/ActiveX, COM+, .NET CLR
- Similarities
  - Late binding, encapsulation, interface inheritance, ...
- Differences
  - Memory management, evolution and versioning, and many more
- Main Problem
  - most approaches are rather object-oriented than component-based
    - CORBA = Common Object Request Brokerage Architecture
    - EJBs = POJOs today

➔ *Is OSGi the solution?*

OSGi *(Open Service Gateway Initiative)*

- OO has driven modularisation to the extreme
  - many fine-granular objects
  - but no real module concept „above“ packages
- OSGi offers bundles as a solution
  - basically JAR files defining a public interface via a manifest
  - services of bundles can only be used when explicitly required in consumer bundle
  - integrated into Eclipse
- OSGi provides a registry for services (bundles)
  - only existing during runtime
  - services can come and go

http://it-republik.de/jaxenter/artikel/Erste-Schritte-mit-OSGi-2077.html



What Are Web Services?

"Web services are a new breed of Web application. They are **self-contained**, **self-describing**, modular applications that can be **published**, **located**, and **invoked** across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. …

Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service."

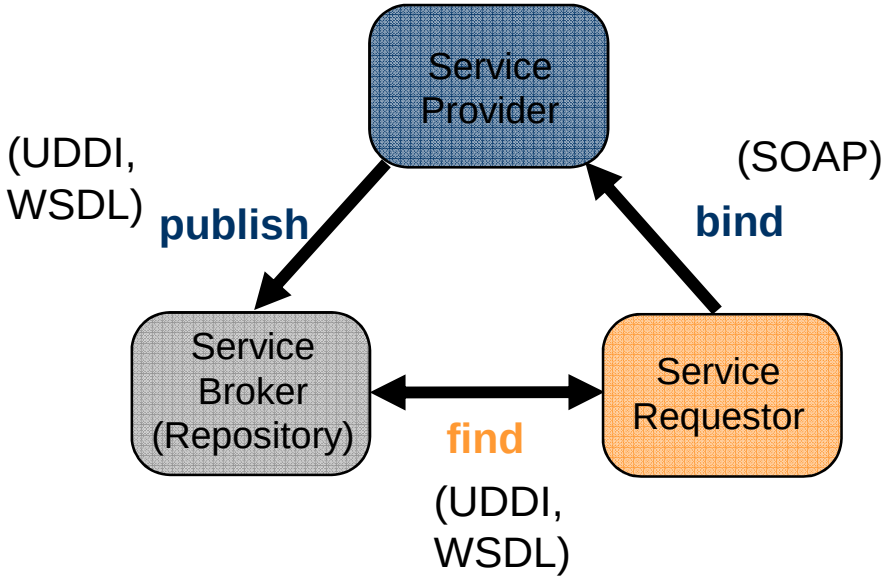
IBM

- self-contained
  - functionality and attributes are exposed in a public interface while implementation is hidden
- self-describing
  - have a machine-readable description that can be used to understand their interface
- modular
  - are reusable and can be composed to generate higher level functionality
- published
  - can be registered in electronic "yellow pages" for easy location by other applications
- located
  - are tied to a fixed, globally unique location identified through a URI
- invoked
  - can be invoked using a standard Internet protocol

➔ *web services can be seen as deployed components*

Service-Oriented Architecture (SOA)

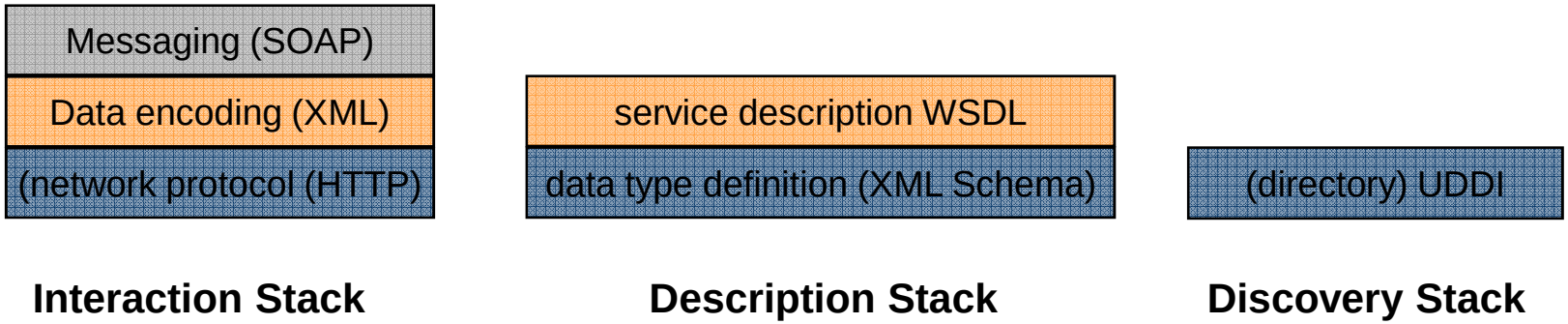
- Elements in a system built from web services play one of three roles
  - service requestor
  - service provider
  - service broker (repository)



- service providers **publish** services by advertising service descriptions in the registry
- service requestors use **find** operation to retrieve service descriptions from the service registry
- service requestors **bind** to service providers using binding information found in service descriptions to locate and invoke a service

Core Web Service Technologies

- **SOAP** (t.a.f.k.a. *Simple Object Access Protocol*)
  - a message layout specification defining a uniform way of passing XML-encoded data
  - a way to simulate RPCs over standard Web communication protocols
- **WSDL** (Web Service Description Language)
  - defines Web Services as collections of network endpoints or *ports*
  - a port is defined by associating a network address with a binding
- **UDDI** (Universal Description, Discovery and Integration)
  - provides a mechanism for clients to find web services
  - the basis for repository services for business applications

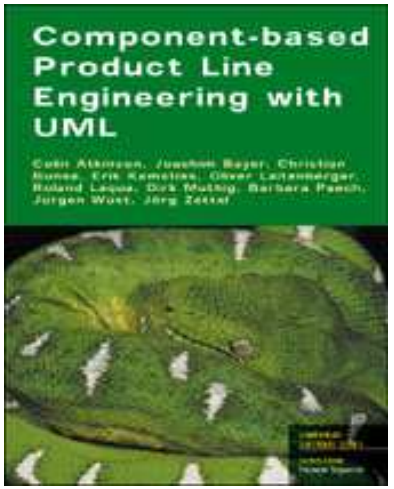


Research-based Component Models

- Aim for a stricter adherence to CBSE principles
- SOFA
  - protocol checking
  - compositionality
  - provides own infrastructure
- ROBOCOP
  - consumer electronics (e.g. TVs)
  - analysis of non-functional properties
  - provides own infrastructure
- KobrA
  - UML-based, hierarchical modelling approach
  - support for software product lines
- Palladio
  - performance prediction at design time
  - support of CBSE role model
  - mapping to EJB possible

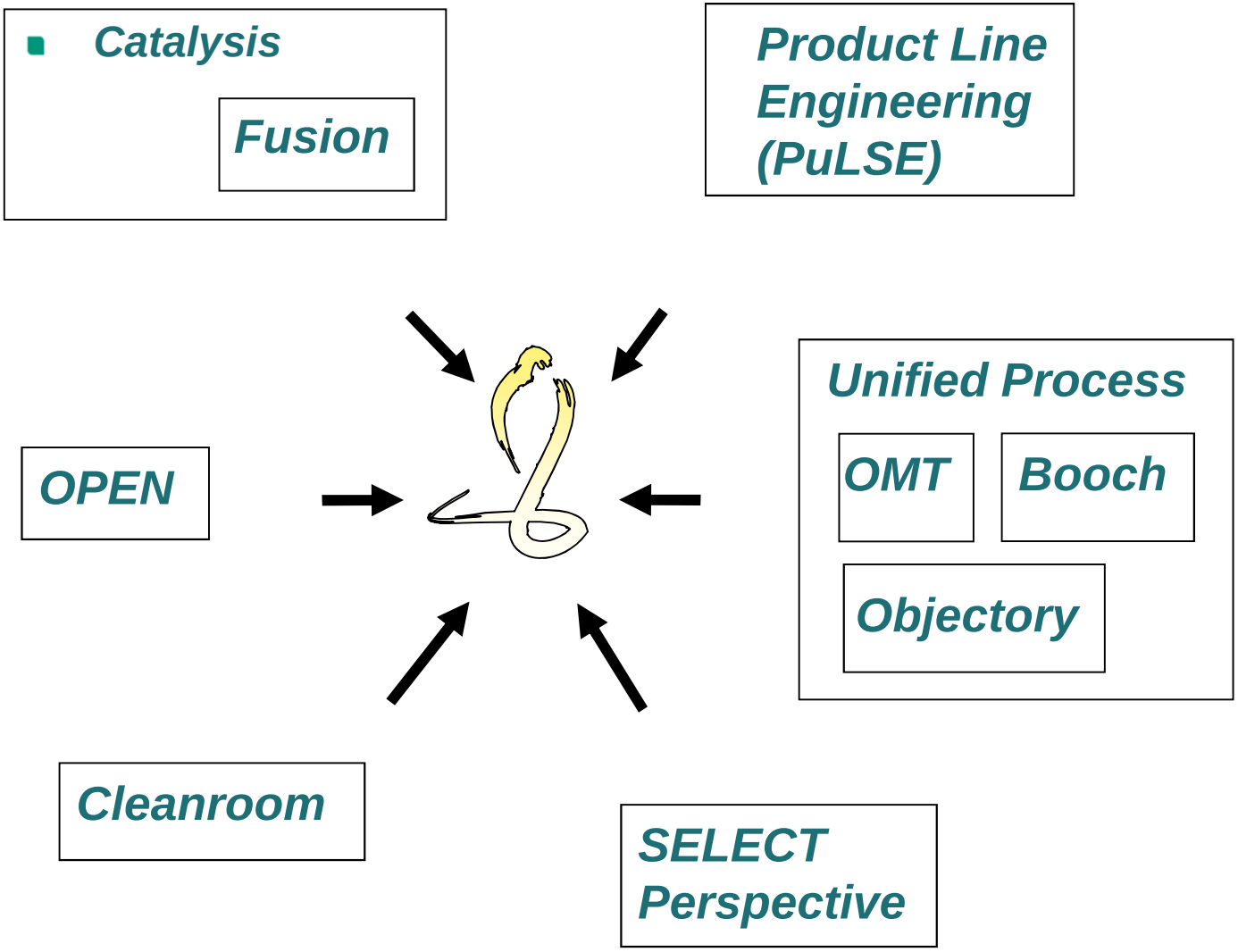
The KobrA Approach…

- … was developed in the "Komponentenbasierte Anwendungsentwicklung" project by Fraunhofer IESE et al.
  - well documented in the "KobrA Book"
  - updated by LS SWT in Mannheim in 2008



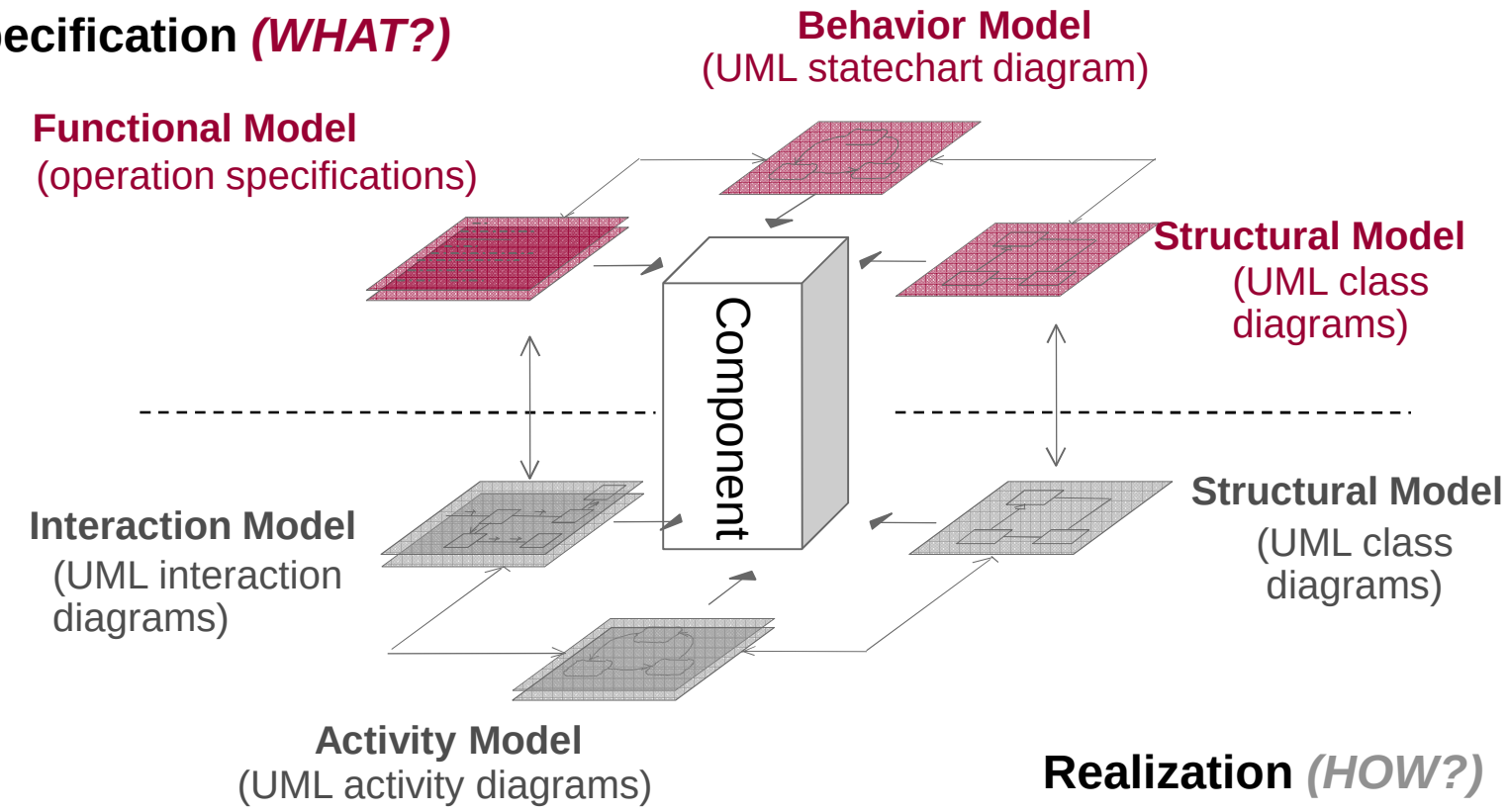
- …promotes the hierarchical (recursive) modelling of systems based on the following modelling principles
  - **Uniformity**
    - all behavior rich elements should be viewed as components, including (sub)systems
    - component assembly = component development
  - **Parsimony**
    - minimal set of concepts (no redundancy)
  - **Locality**
    - all models should be local to a component
  - **Encapsulation**
    - component specifications (what) must be separated from component realizations (how)

Methodological Influences on KobrA



KobrA Component Modeling

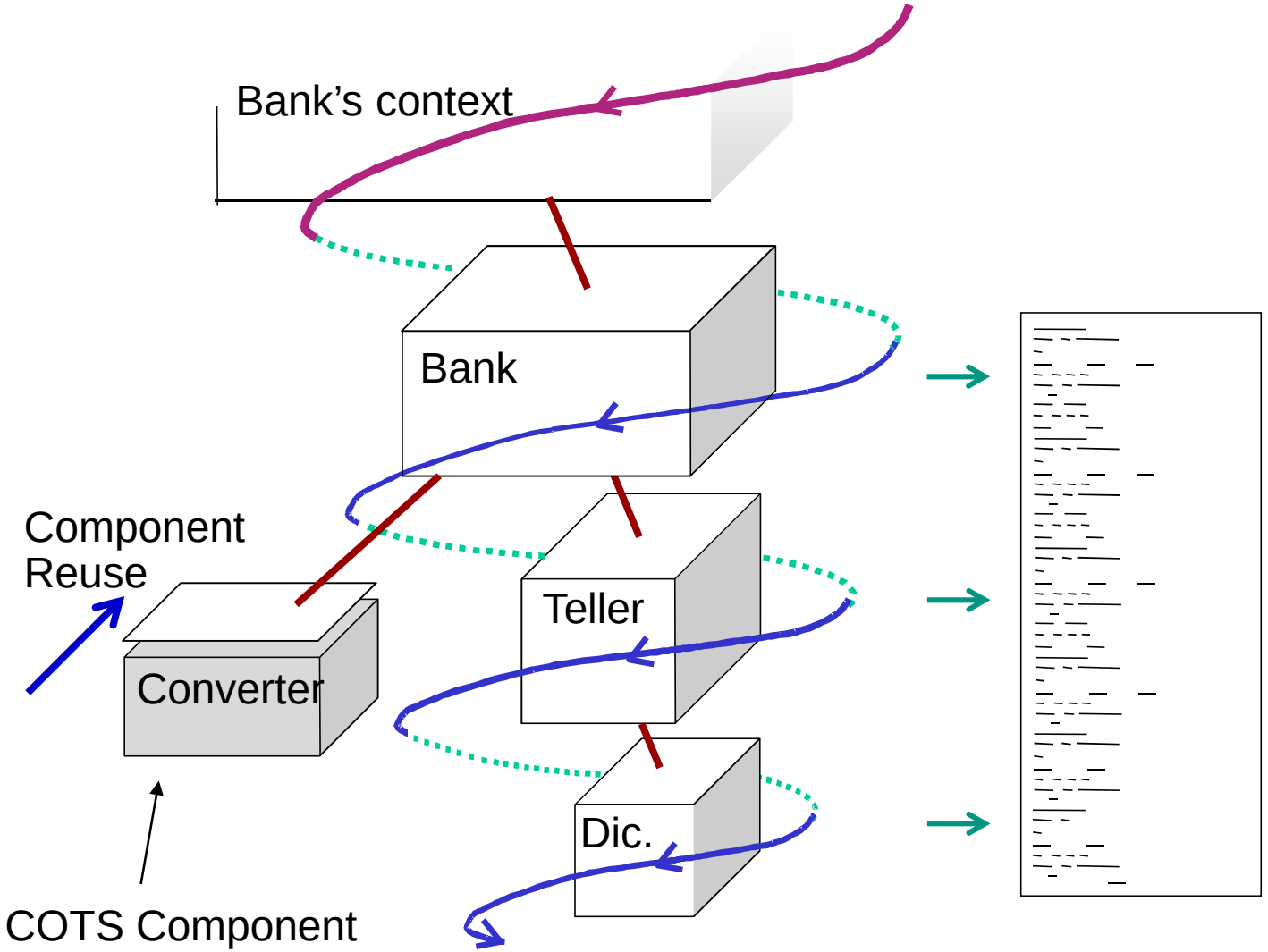
- KobrA components are modelled in a uniform way
  - requiring three black-box and three white-box models for each
    - similar to the models we have seen within the RUP



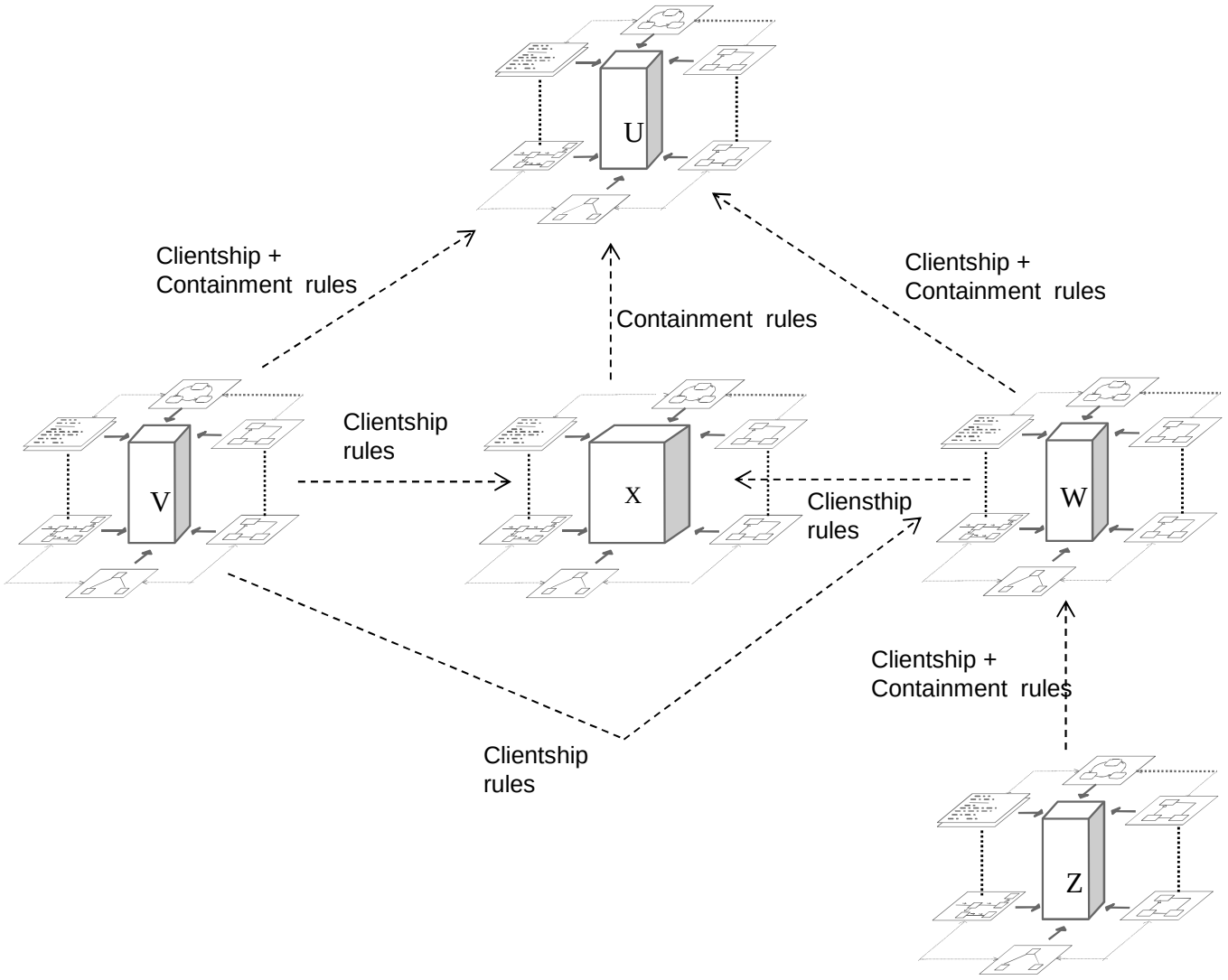
Context Realization Activities

- The Context Realization is the starting point for system modeling in KobrA, comprising –
  - Enterprise modeling
    - who does what to what and when
    - actors, activities, data and rules
    - described at “business” level of abstraction
  - Data modeling
    - identify organizational and technological data
    - identify information and material data
  - Usage modeling
    - activity modeling
      - decompose activities that involve the “system”
    - Incorporates use case analysis
  - Interaction modeling
    - integrate actors, activities, data and rules

Component Engineering Process



Framework Correctness Rules



Product Line Engineering

- Most software development organizations develop a line of similar but slightly different products
- KobrA describes a family of similar systems (i.e. components) by modeling
  - variabilities
  - decisions
- A component (class) with (non-empty) decision models (i.e. optional features) is known as a **generic component**
- A tree of generic components represents a generic framework
  - decision models are related hierarchically
  - decision model resolution implies the resolution of all lower decision models

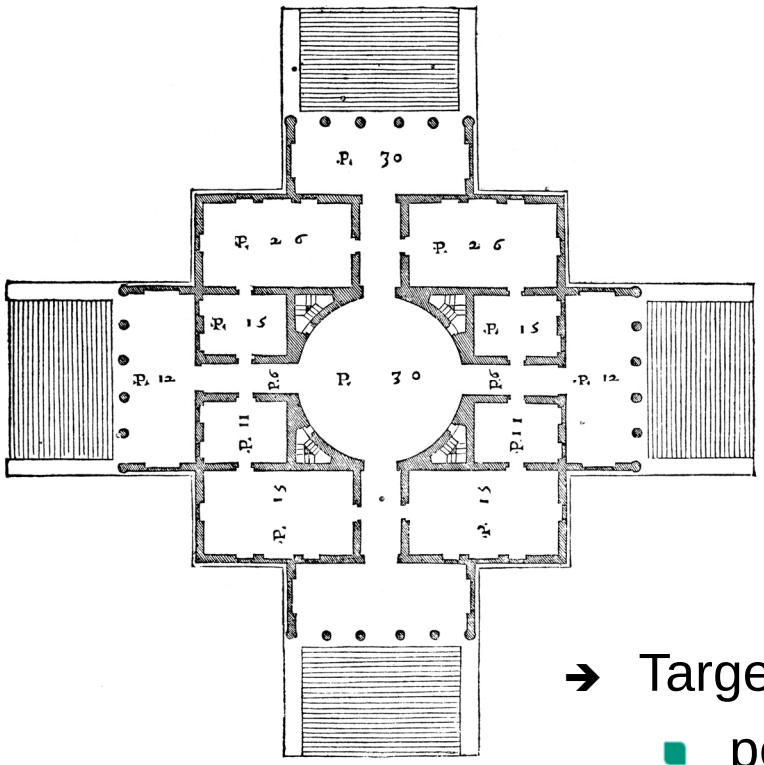
➔ Incremental introduction of product lines

Andrea Palladio

- Andrea Palladio (30 November 1508 – 19 August 1580) was an Italian architect active in the Republic of Venice.
- Palladio, influenced by Roman and Greek architecture, primarily by Vitruvius, is widely considered the most influential individual in the history of Western architecture.
- All of his buildings are located in what was the Venetian Republic, but his teachings, summarized in the architectural treatise, The Four Books of Architecture, gained him wide recognition.

[Wikipedia]

Palladio Component Model (1)

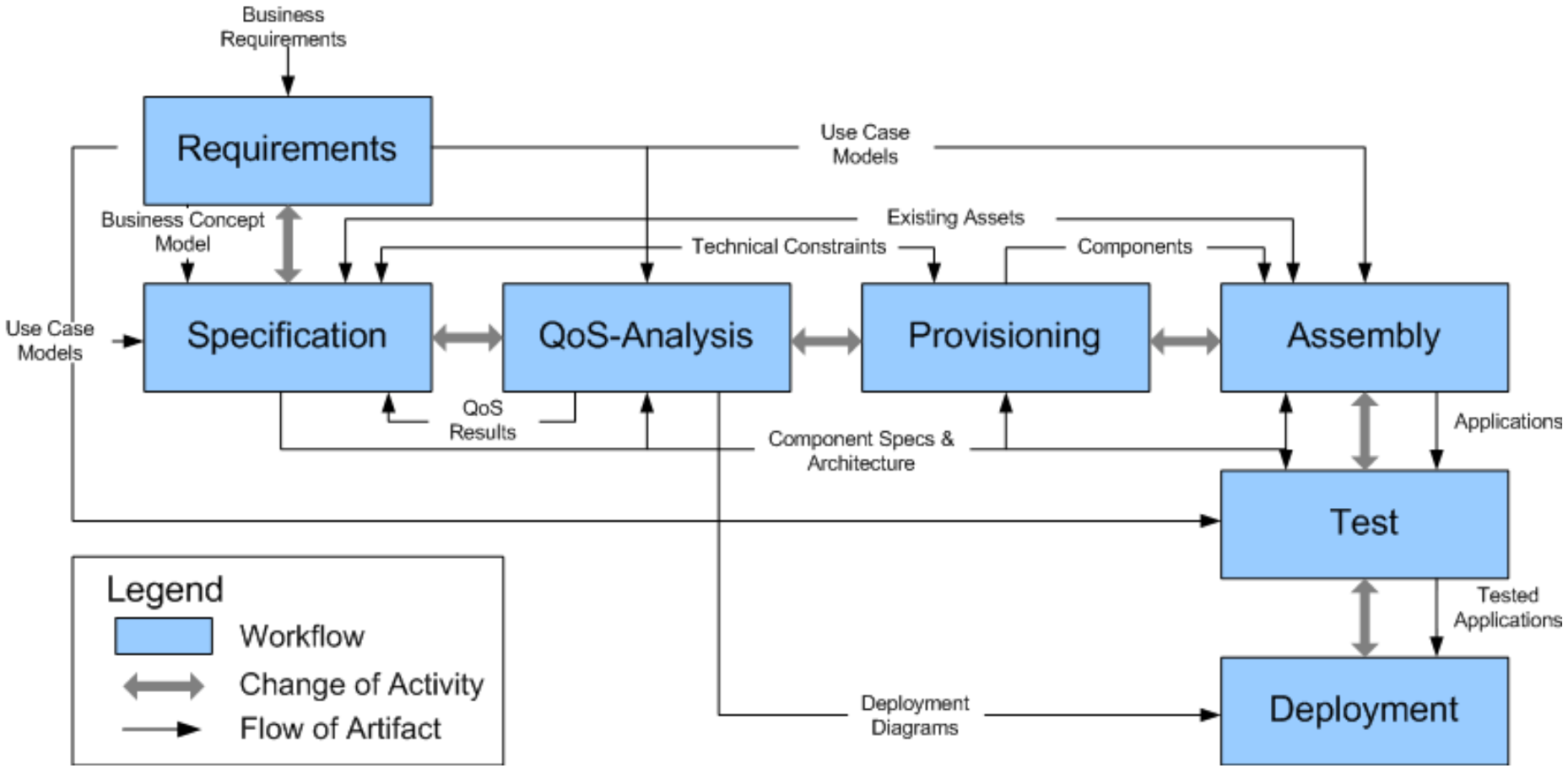


- The Palladio Component Model (PCM) is a domain specific modelling language (DSL)
- Designed to enable early performance predictions for software architectures
- Aligned with a component-based software development process

➔ Targets at –

- performance prediction for component-based software architectures
- business information systems

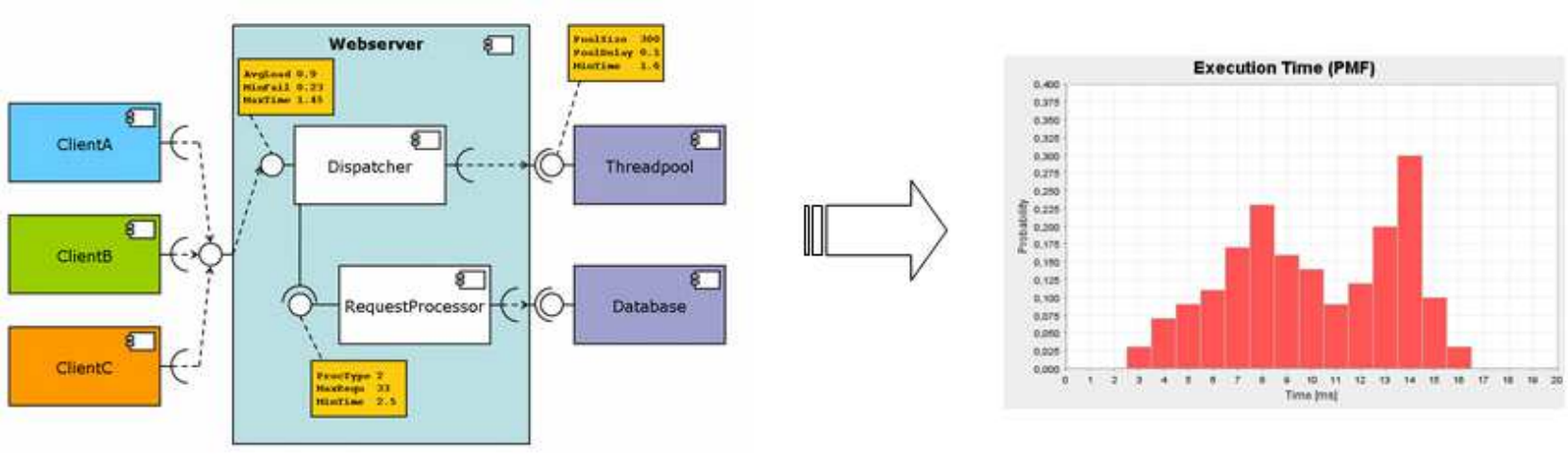
CBSE Development Process



[Cheeseman2000, Koziolek2006a]

General Idea

- Prediction of non-functional properties on a model base
  - for systematic design of software systems
- Describe a component as a set of models

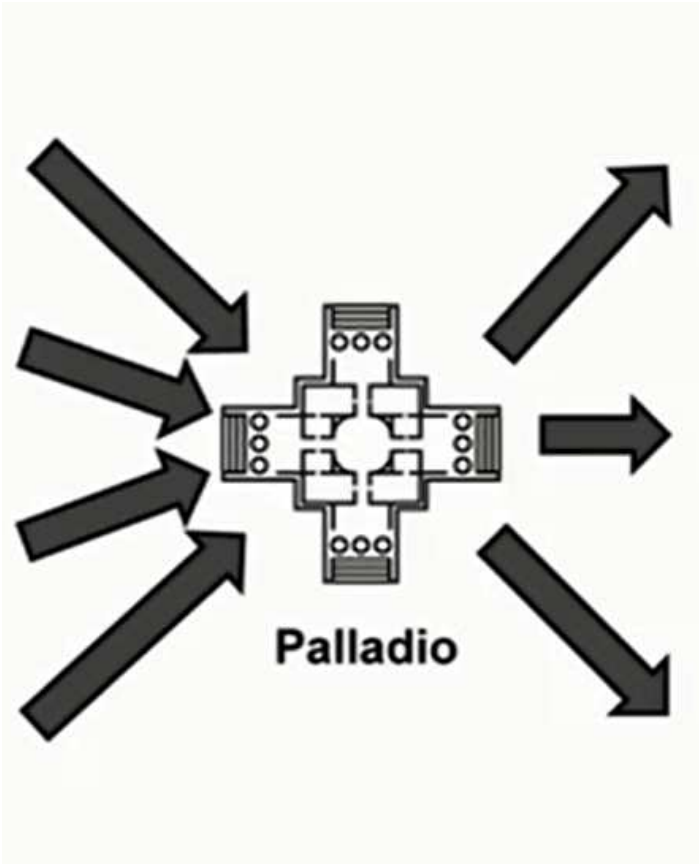


- Derive performance metrics from the models using
  - analytical techniques and
  - simulation

Component Description

- What are the intuitive inputs and outputs for a performance prediction model?

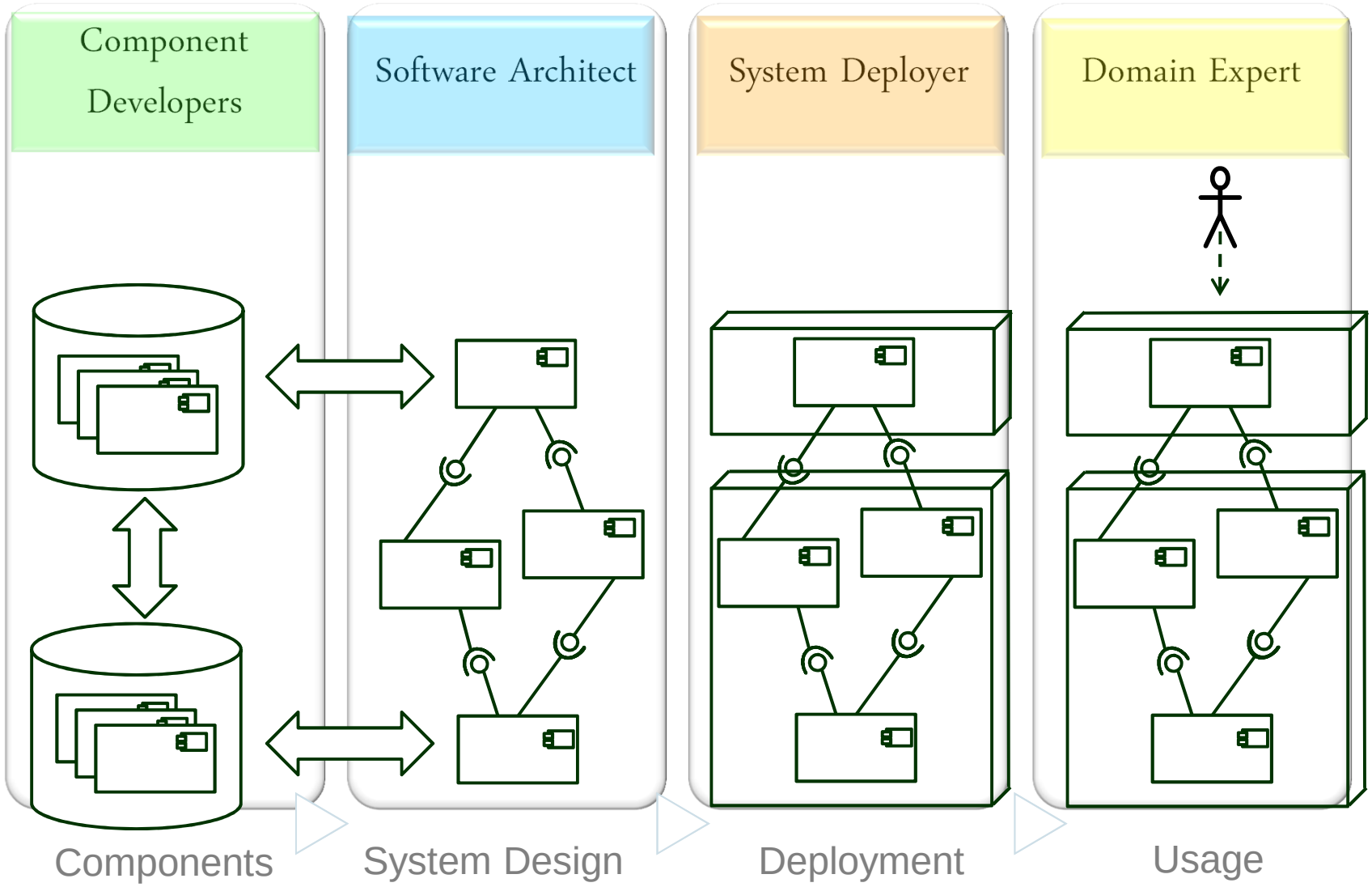
?



?

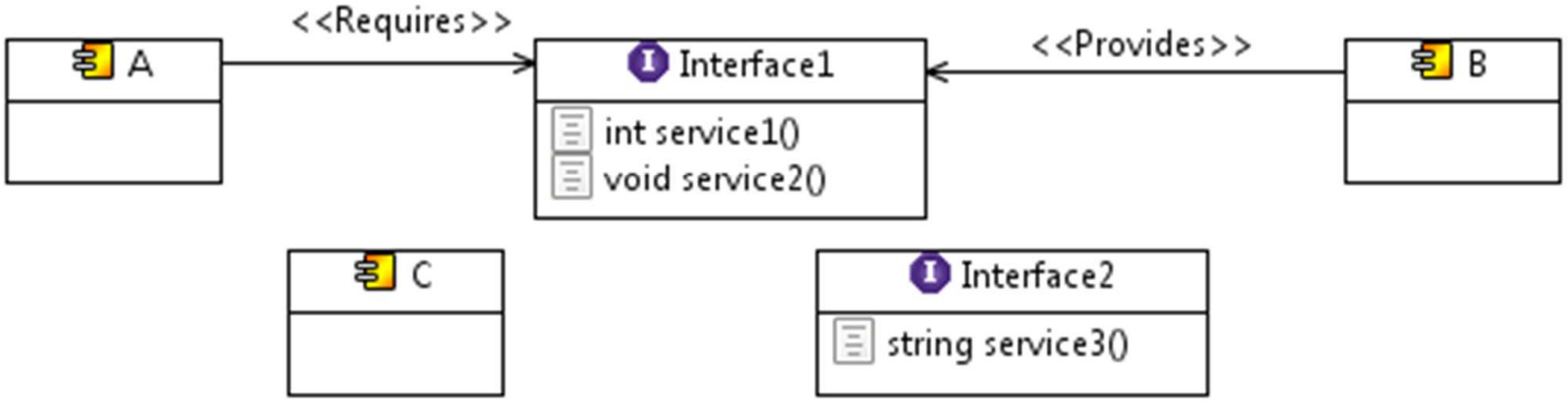
also see: http://www.youtube.com/watch?v=H0Gj-kdGhRs

Models an their Creators



Component Description

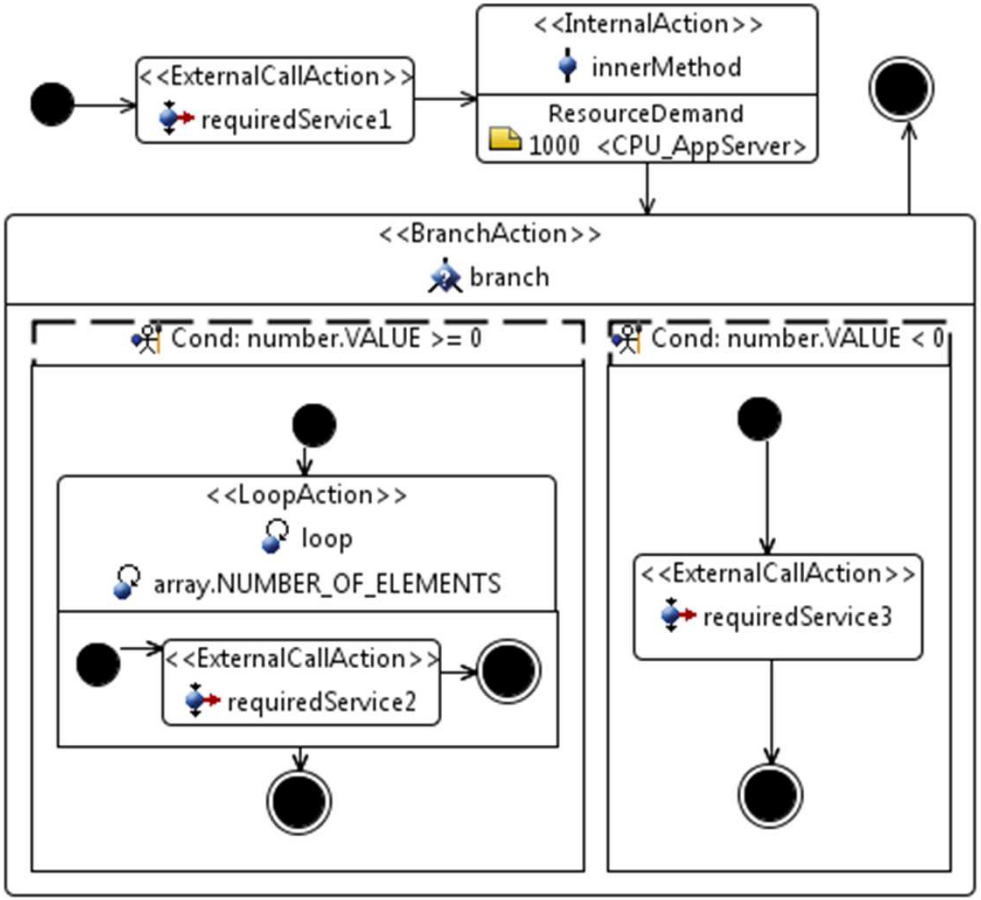
- Component interfaces need to be described
- Created components are stored in a repository



Component Developer

Behavior Specification

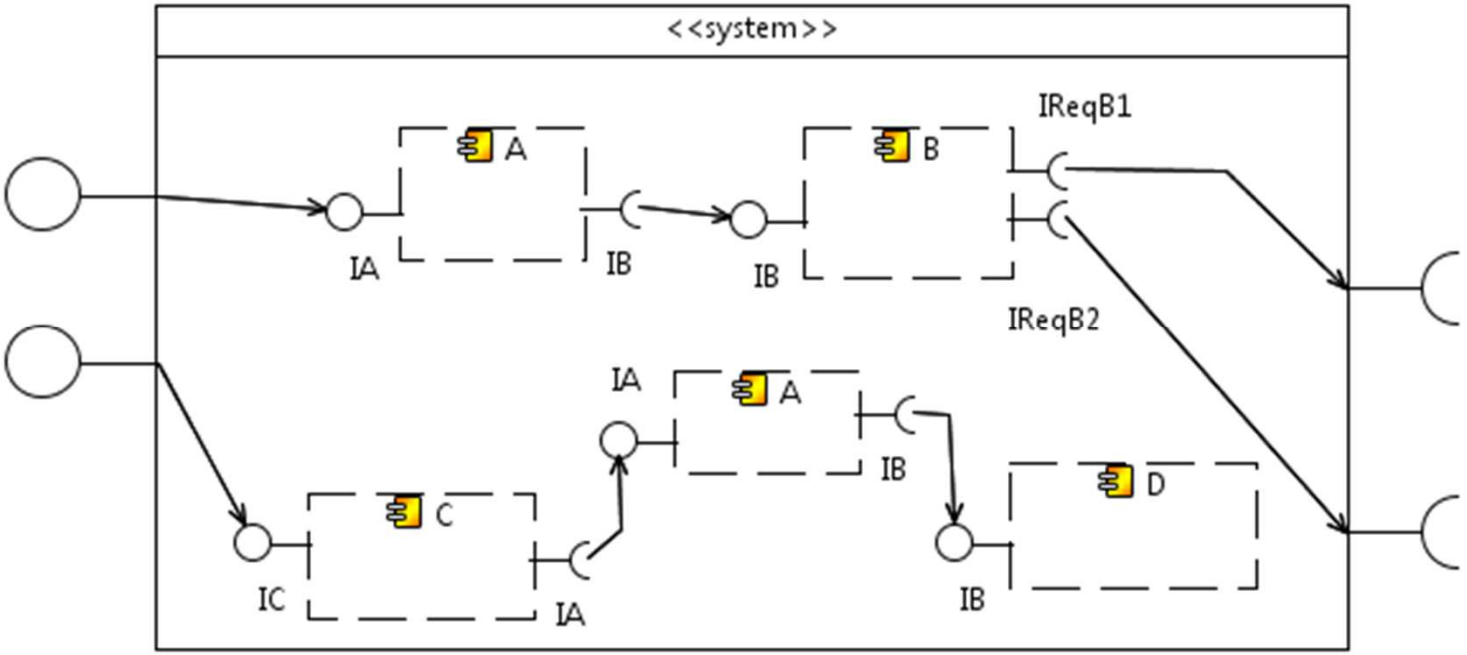
- Component behavior needs to be described in so-called Service Effect Specification (SEFF)



Component Developer

System Composition

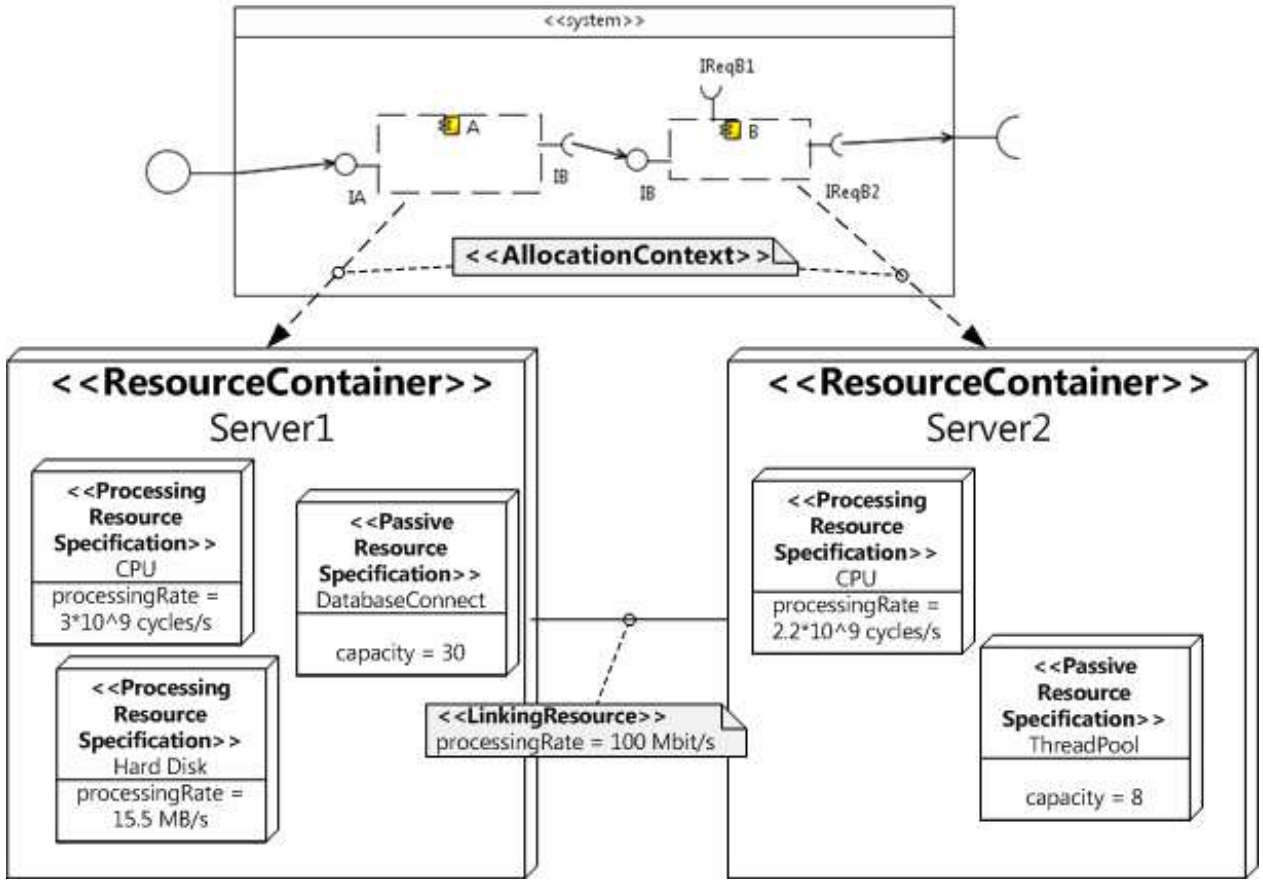
- System is composed of components from repository



Software Architect

Resource Description

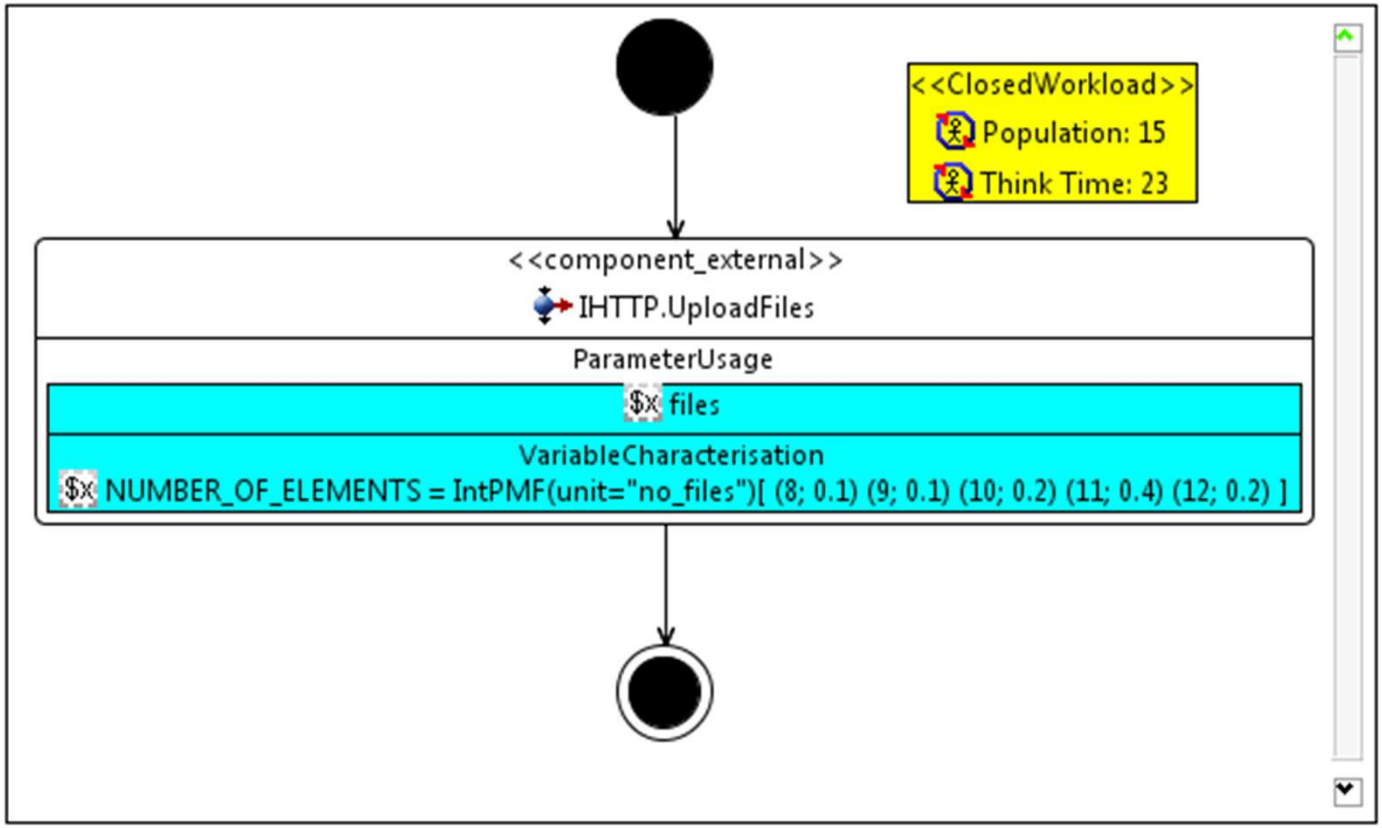
- The deployment enviroment of the system needs to be specified



System Deployer

Usage Model

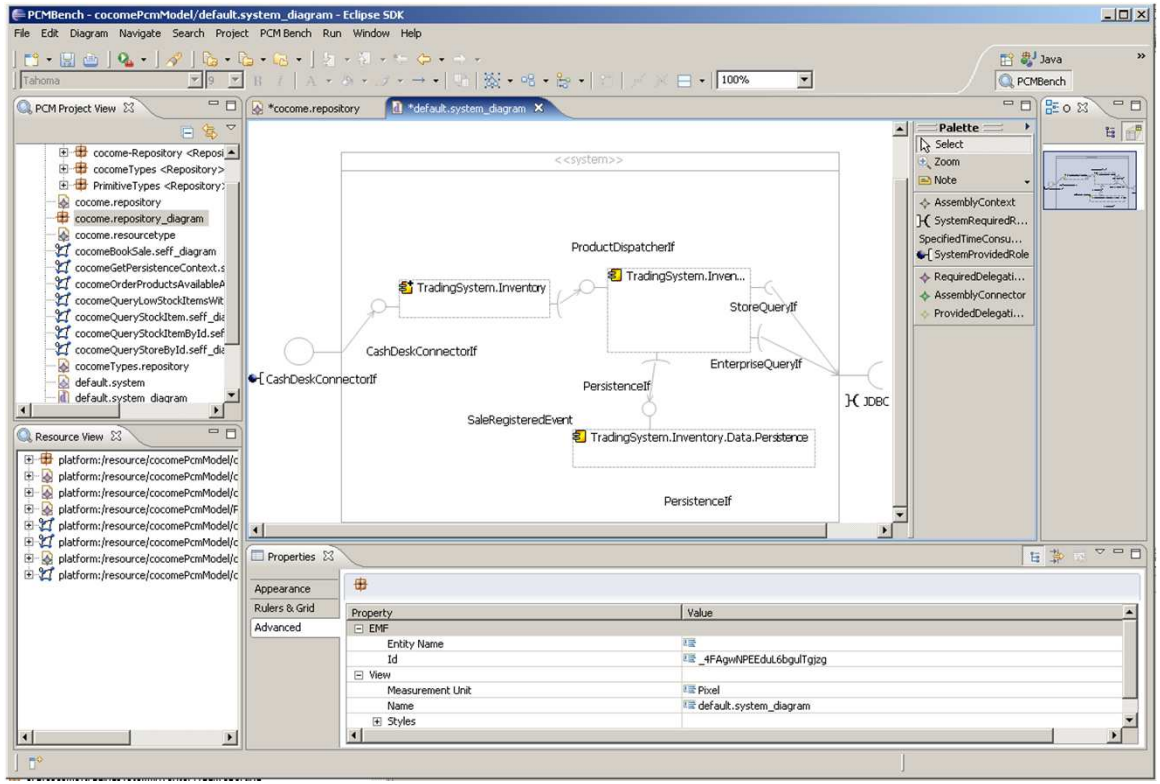
- Finally, it needs to be specified how the system is used



Domain Expert

Tool Time: PCMBench

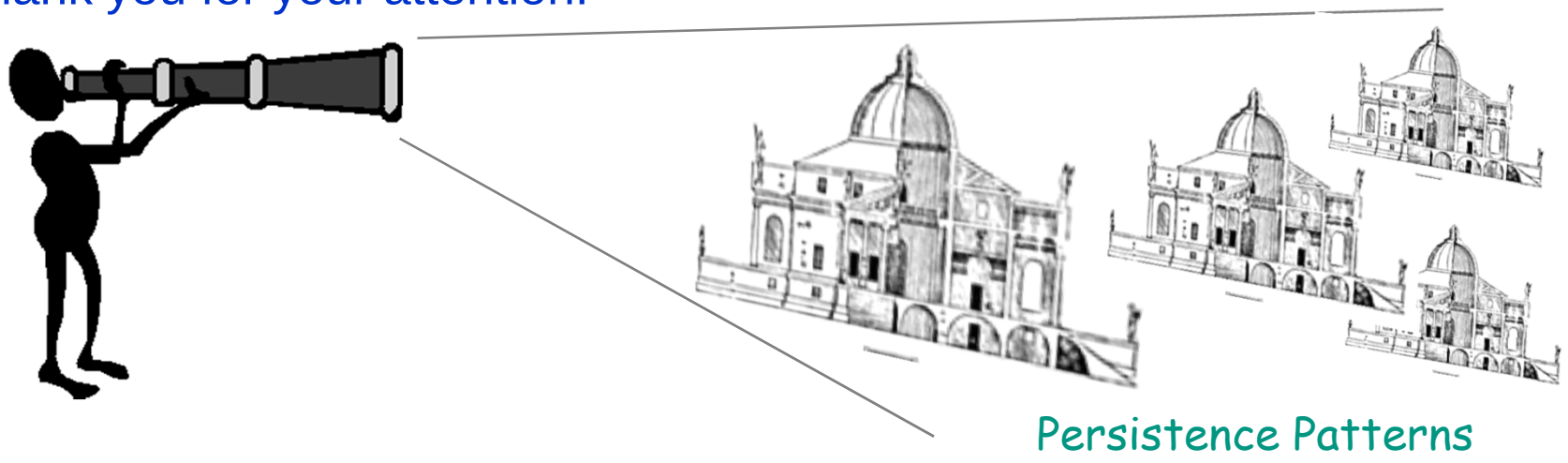
- Supports the whole component-based design process
- Analysis approaches provide hints on performance bottlenecks / issues



see http://www.palladio-simulator.com

Conclusion

- Components are supposed facilitate the composition of software systems
  - various component models are available
  - however, as so often, the concept is still overloaded
    - and there is no „one size fits all“
- *More to come next semester in „Komponentenbasierte Software-Architekturen!*
- Thank you for your attention!

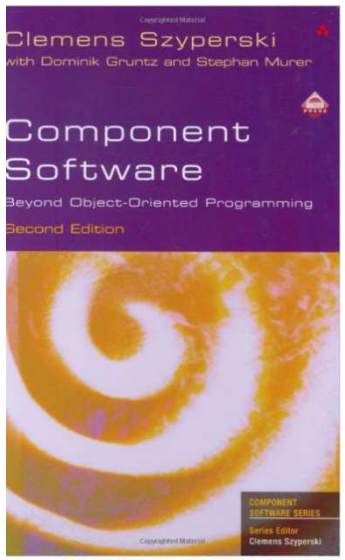


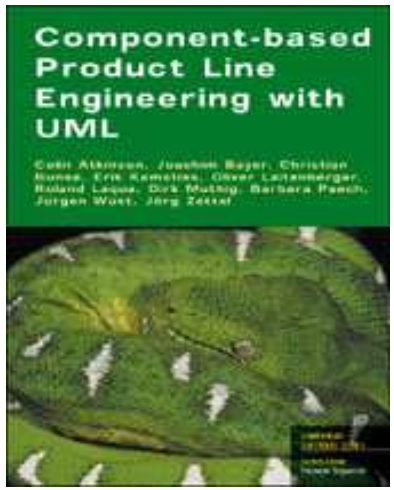
Reminder

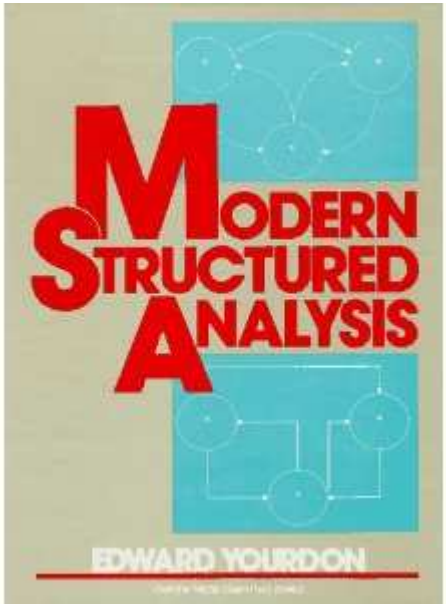

Winter Day 2013
Mi 4.12. 15:00–19:00
Infobau Raum 333
SDQ
FZI
Descartes
KIT AREGroup
ABB
SAP
IBM

References (1)

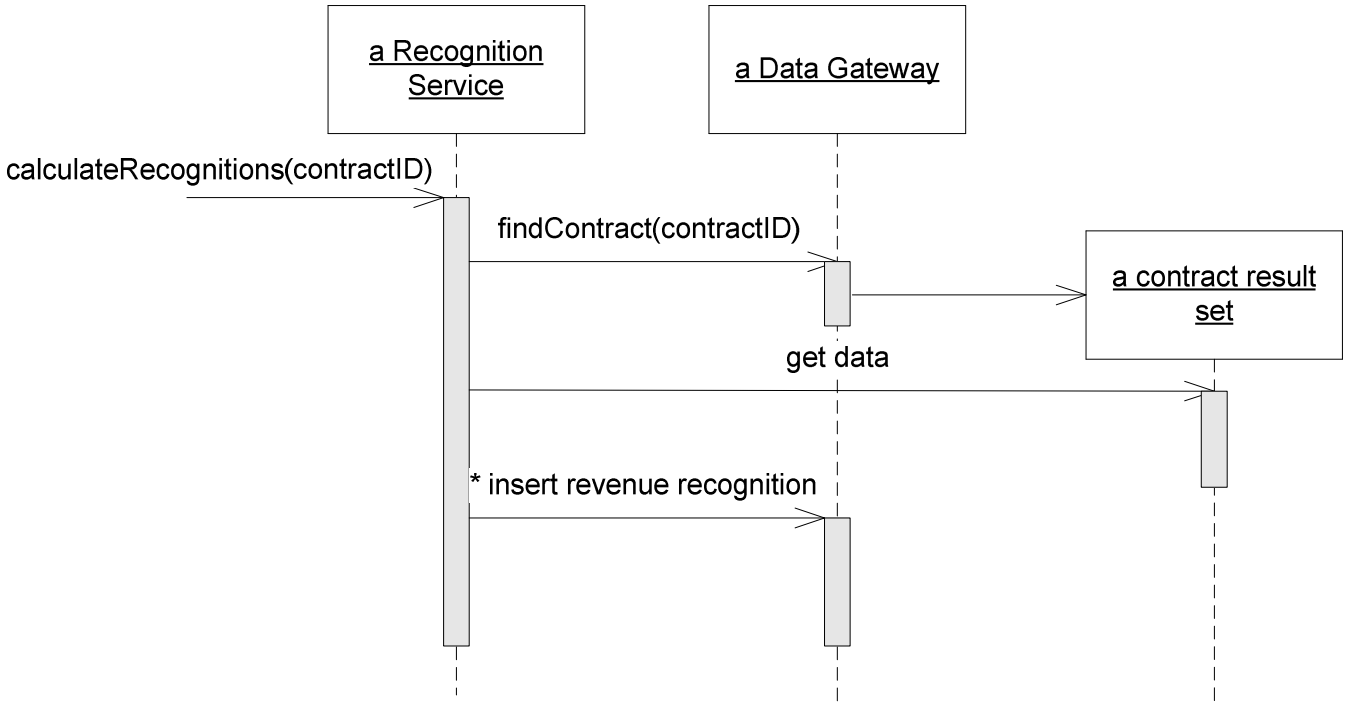
- *Steffen Becker, Palladio Screencast: http://www.palladio-simulator.com/tools/screencasts*
- *Becker, Koziolek & Reussner: Palladio Component Model: http://www.sciencedirect.com/science/article/pii/S0164121208001015*







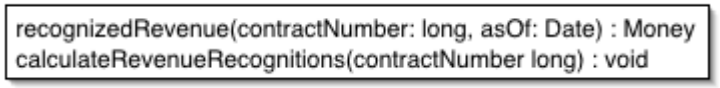
Appendix: Transaction Script



- Put all logic for this transaction in a procedure
- Retrieve contracts from data source, calculate, and store results
- *Procedural counterpart to "Domain Model"*
  - *third possibility is "Table Module"*

[Fowler]

Transaction Script

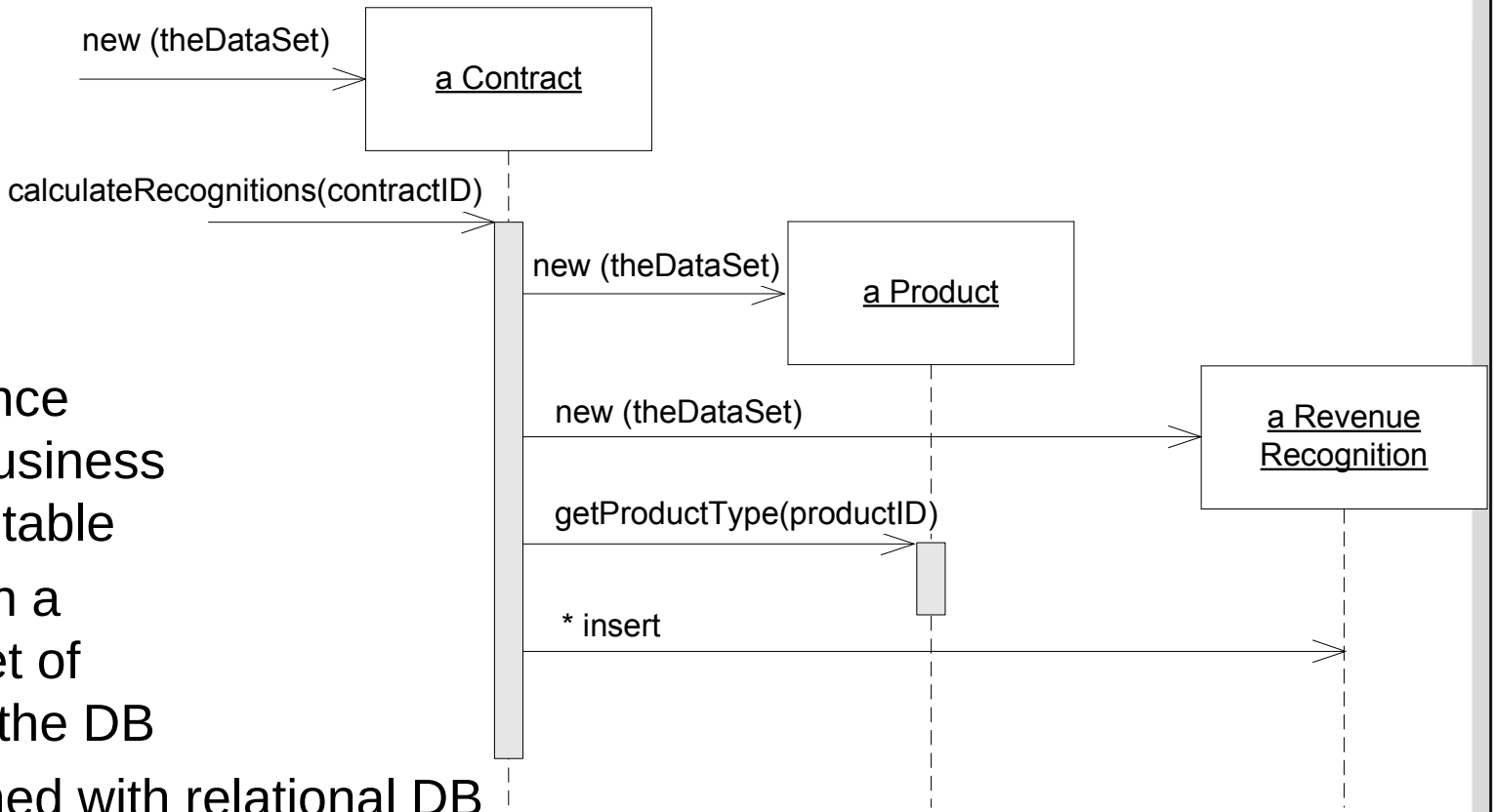


- Single procedure per transaction type
- Factor common behaviour out into subroutines

- Advantages:
  - Simple procedures that developers understand
  - Easy to connect to simple data sources
  - Transaction boundaries are easy to determine
- Problems:
  - Does not scale well with complex logic
  - Tends to have duplicate code then

[Fowler]

Appendix: Table Module

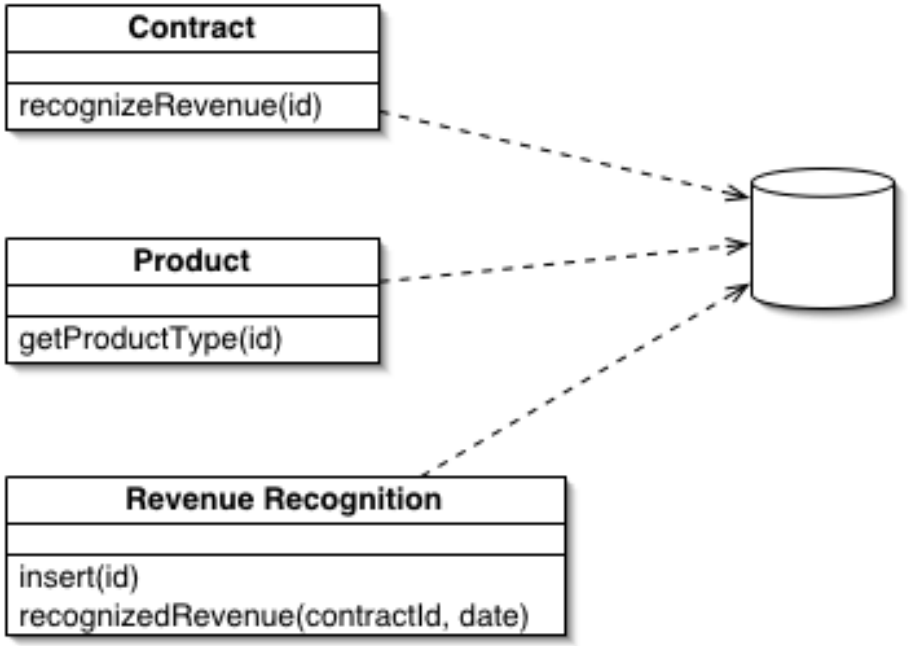


- One instance handles business logic for a table
- Called with a Record Set of data from the DB
- More aligned with relational DB
- Often one class per table, handles result of query

[Fowler]

Table Module

- Objects do not have identity
- Works with table-like data structure



- Advantages
  - Straightforward mapping to data
  - Separates logic for different concepts
  - Useful if used technology supports it (COM, .NET)
- Problems
  - No object instances: can be bad for complex logic

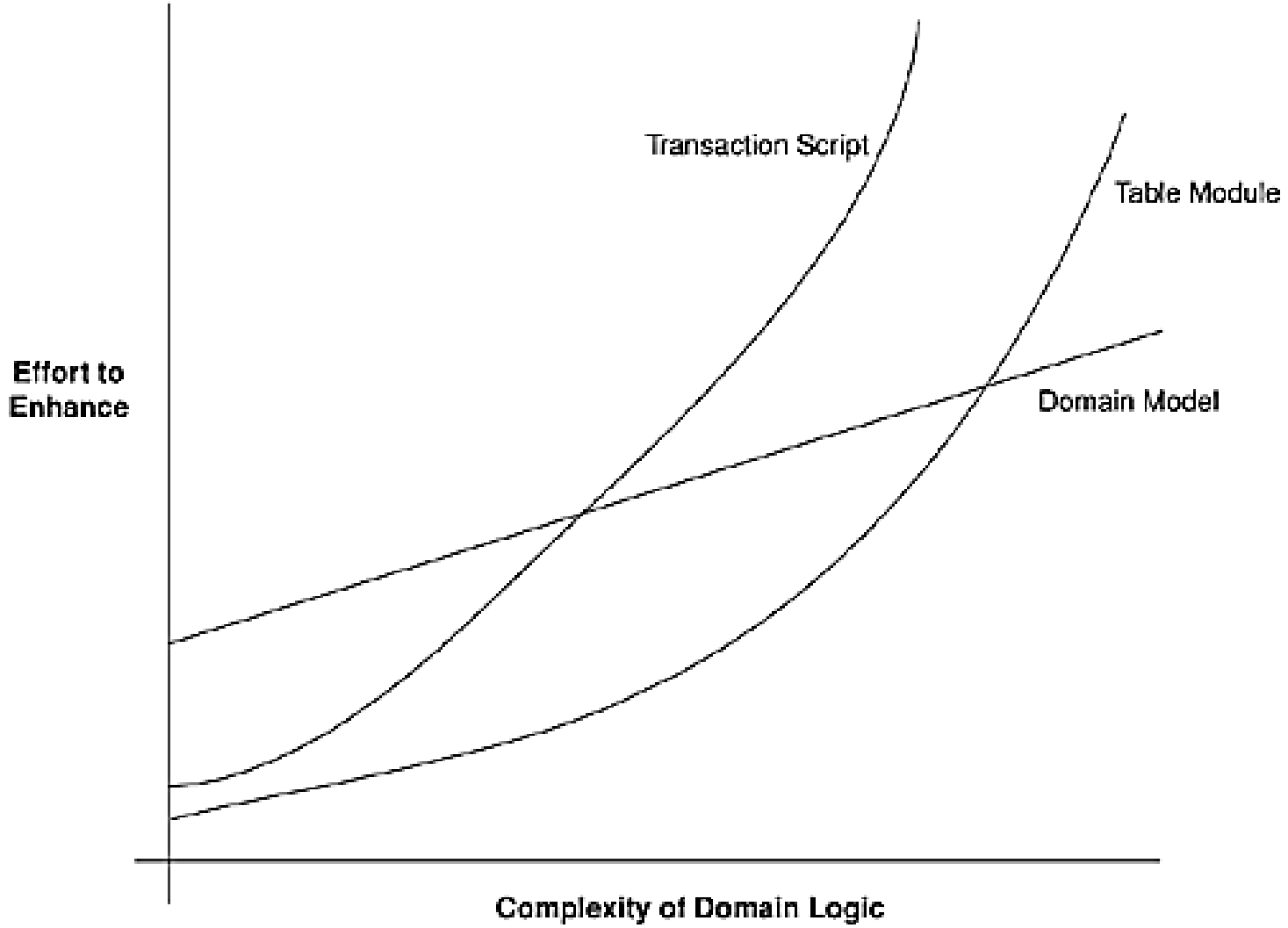
[Fowler]

Appendix: When to use what

- Strongest factor: How complex is domain logic?
  - See graph on previous slide
- How difficult to map to data source?
  - What choices on data source architectural level?
- Are developers familiar with domain models?
  - If yes, less disadvantages, so more attractive
  - Domain model must be carefully designed and adhered to be successful.
- What tools do you use?
  - Development environments / tools may favour a pattern
- Possible to combine all three

[Fowler]

Simplified View on Complexity and Effort



[Fowler]