第二章　不同的分析与设计方法

曹东刚
caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计

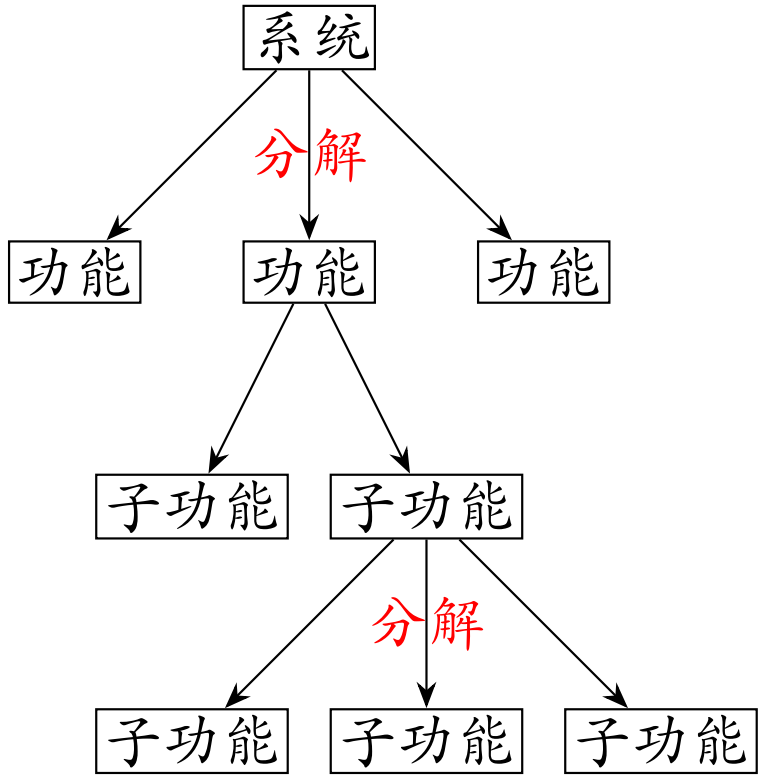
内容提要

功能分解

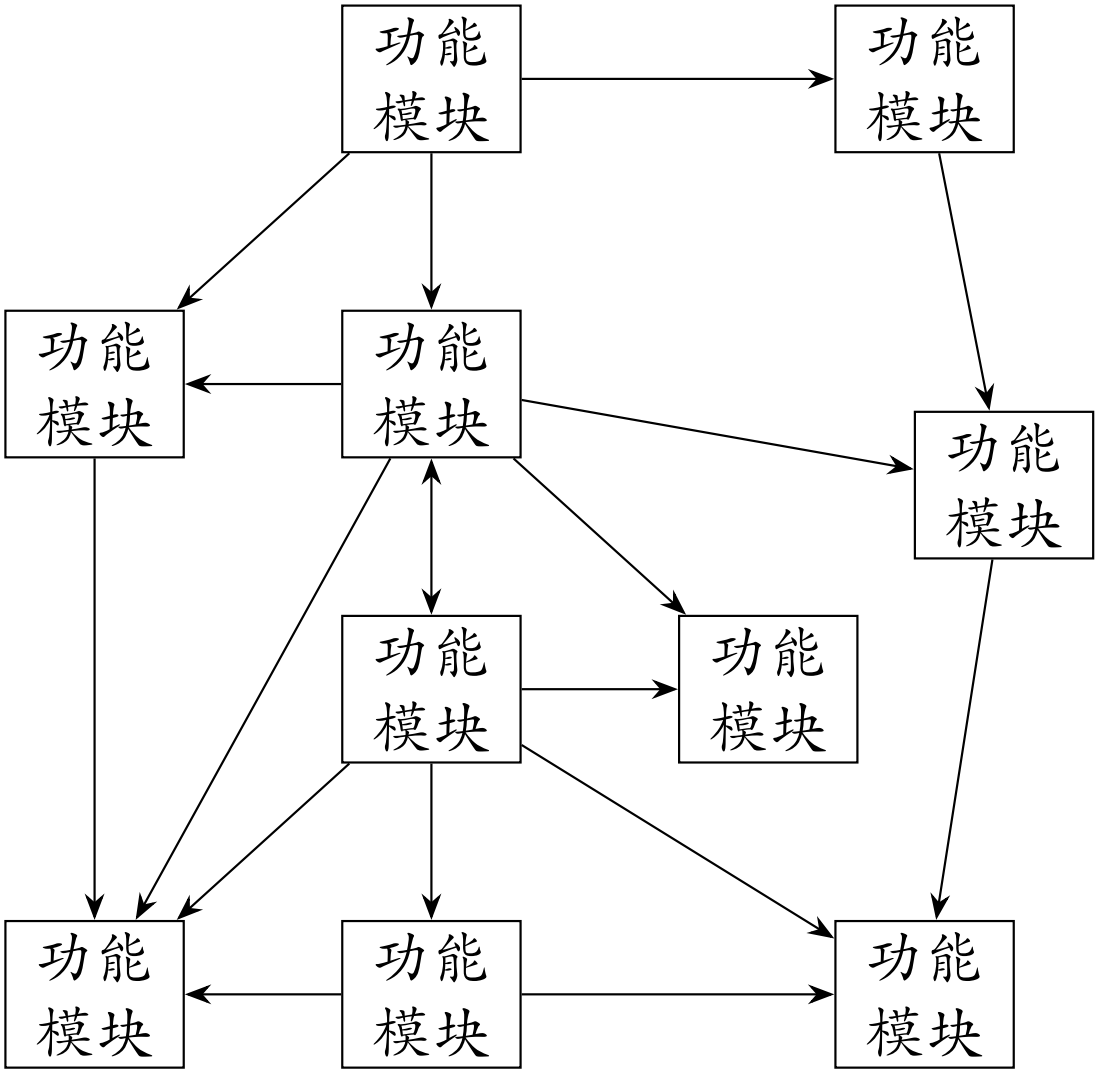
功能分解

以系统功能为中心来组织系统

- 定义各种功能，层层分解为子功能，直到可给出明确的定义
- 根据功能/子功能的需要设计数据结构
- 定义功能/子功能间的接口

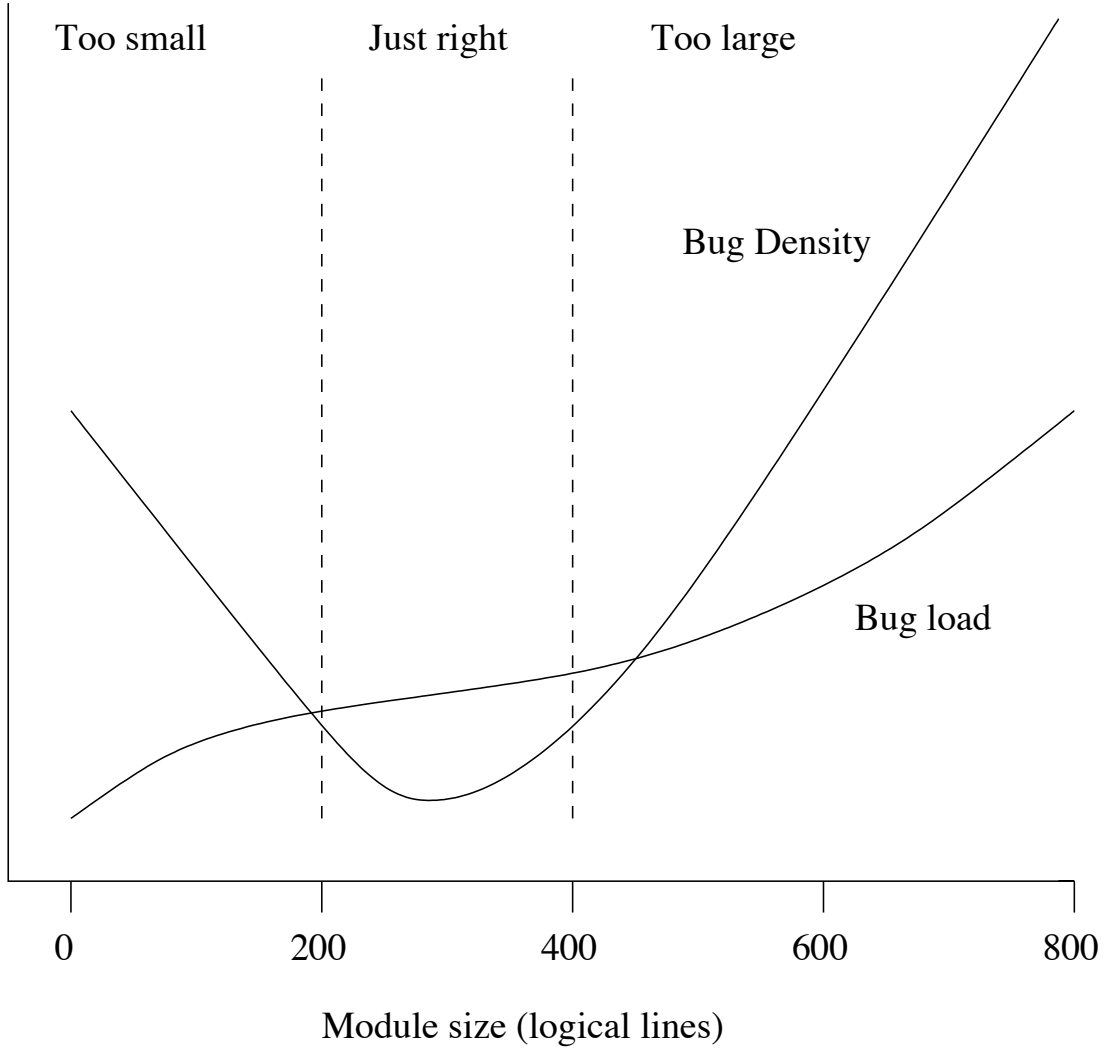


功能分解得到的系统模型：模块＋接口



功能模块与复杂性、Bug 的关系

复杂性来源: 模块内, 模块间



功能分解的特点

- 代表软件工程问世前后人们对软件开发的朴素理解
- 直接地反映用户的需求，所以工作很容易开始
- 分析阶段的概念 (功能) 和设计阶段的概念 (模块) 容易对应
- 不能直接地映射问题域，很难检验结果的正确性
- 对需求变化的适应能力很差，因为功能是最易变的
- 局部的错误和修改很容易产生全局性的影响

功能分解与面向对象

功能分解的一些思想，为后来的面向对象思想吸收
两种模块化方法：

- 对要完成的功能的执行过程的抽象
- 信息隐藏

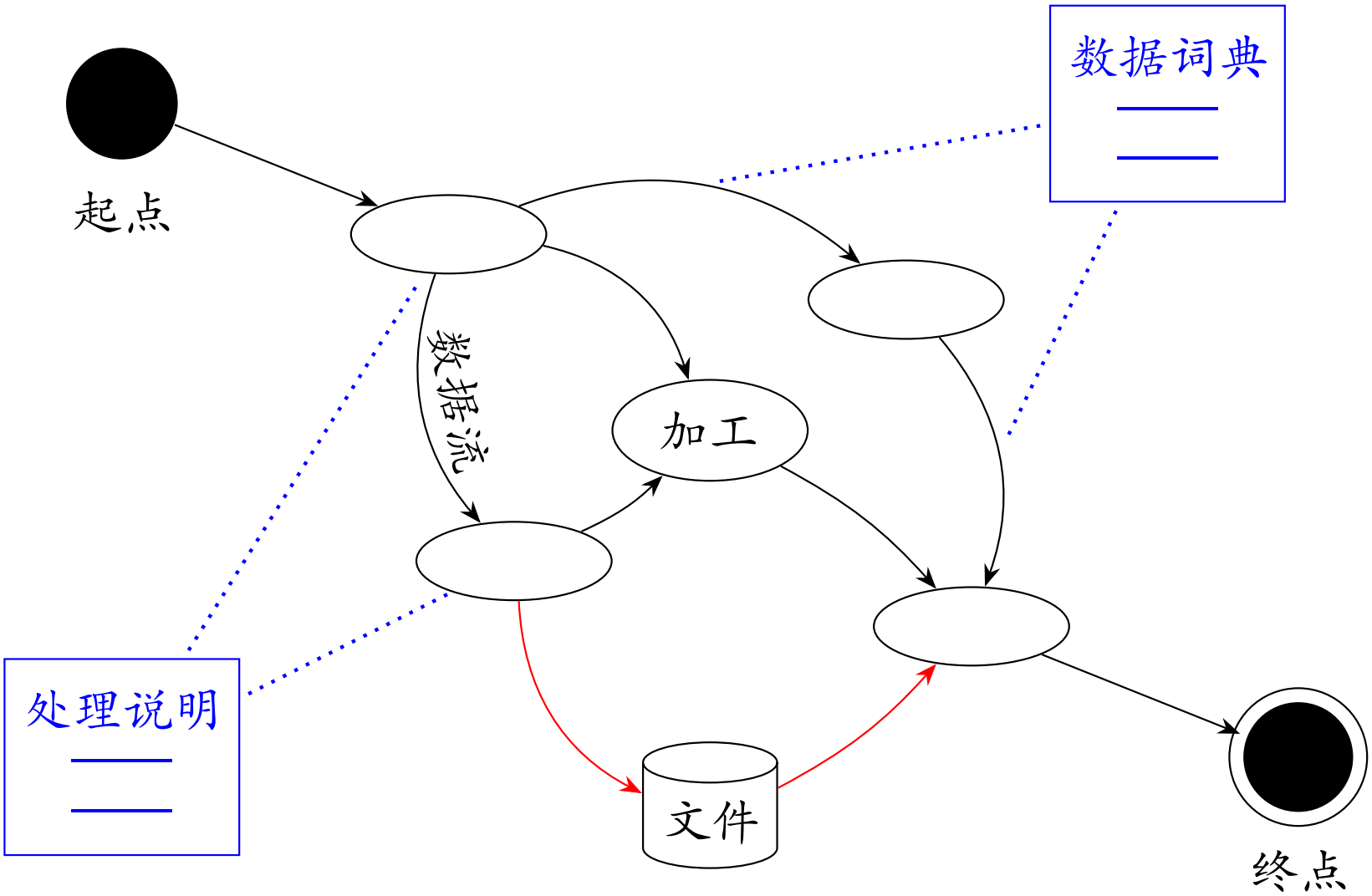
结构化方法: 结构化分析 + 结构化设计

结构化分析: Structured Analysis, SA

又称数据流法，其基本策略是跟踪数据流，即研究问题域中数据如何流动，以及在各个环节上进行何种处理，从而发现数据流和加工

得到的分析模型是数据流图（DFD），主要模型元素是数据流、加工、文件及端点，外加处理说明和数据字典

数据流图: Data Flow Diagram



结构化设计: Structured Design, SD

与功能分解法基本相同，基于模块的概念建立设计模型，分为概要设计和详细设计

- 概要设计：确定系统中包含哪些模块以及模块之间的调用关系，得到模块结构图（MSD）
- 详细设计：描述每个模块内部的数据结构和操作流程

结构化方法的特点

- 有严格的法则，强调研究问题域
- 仍然是间接映射问题域，难以检验分析的正确性
- 与结构化设计的概念不一致，从分析到设计的过渡比较困难
- 数据流和加工的数量太多，引起分析文档的膨胀，导致设计人员和设计人员理解不一致

结构化设计与面向对象

结构化设计的要点:

- 高内聚 (cohesion)
- 低耦合 (coupling)

最好的达到高内聚的方法，是从问题域，而不是解空间中，寻找依据。后来的面向对象方法采纳了该思想。

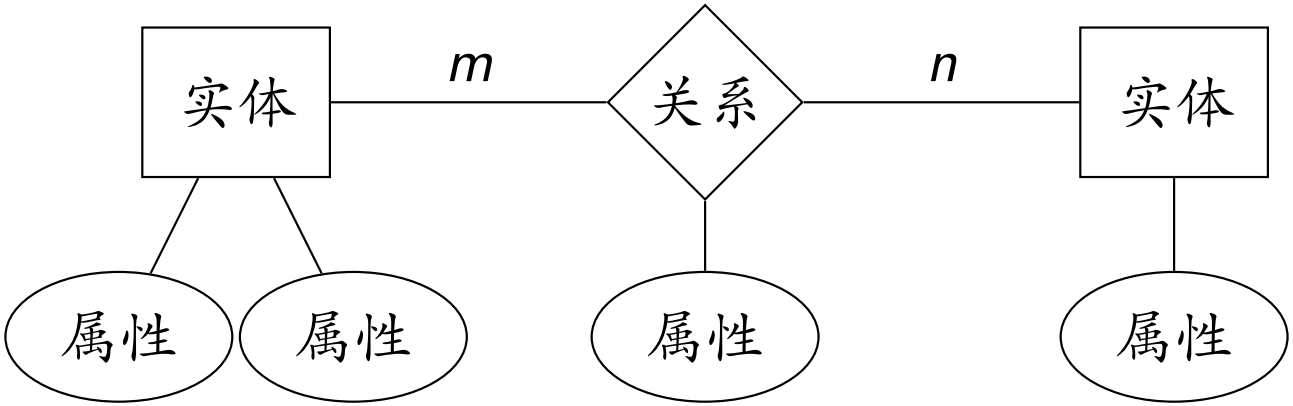
信息建模法: information modelling

信息建模法

由实体-关系法（E-R 方法）发展而来，核心概念是实体和关系

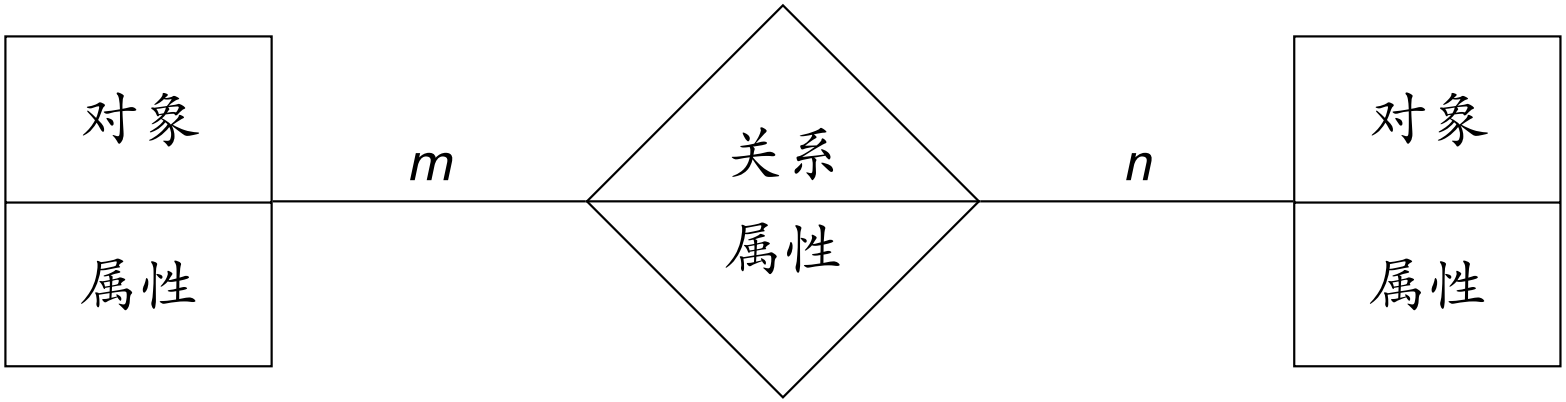
- 实体描述问题域中的事物，包含一组对事物数据属性的描述
- 关系描述事物之间在数据方面的联系，有自己的属性
- 发展之后的方法也把实体称作对象，并使用了类型和子类型的概念，作为实体（对象）的抽象描述

信息模型



E-R 图

信息模型



信息模型

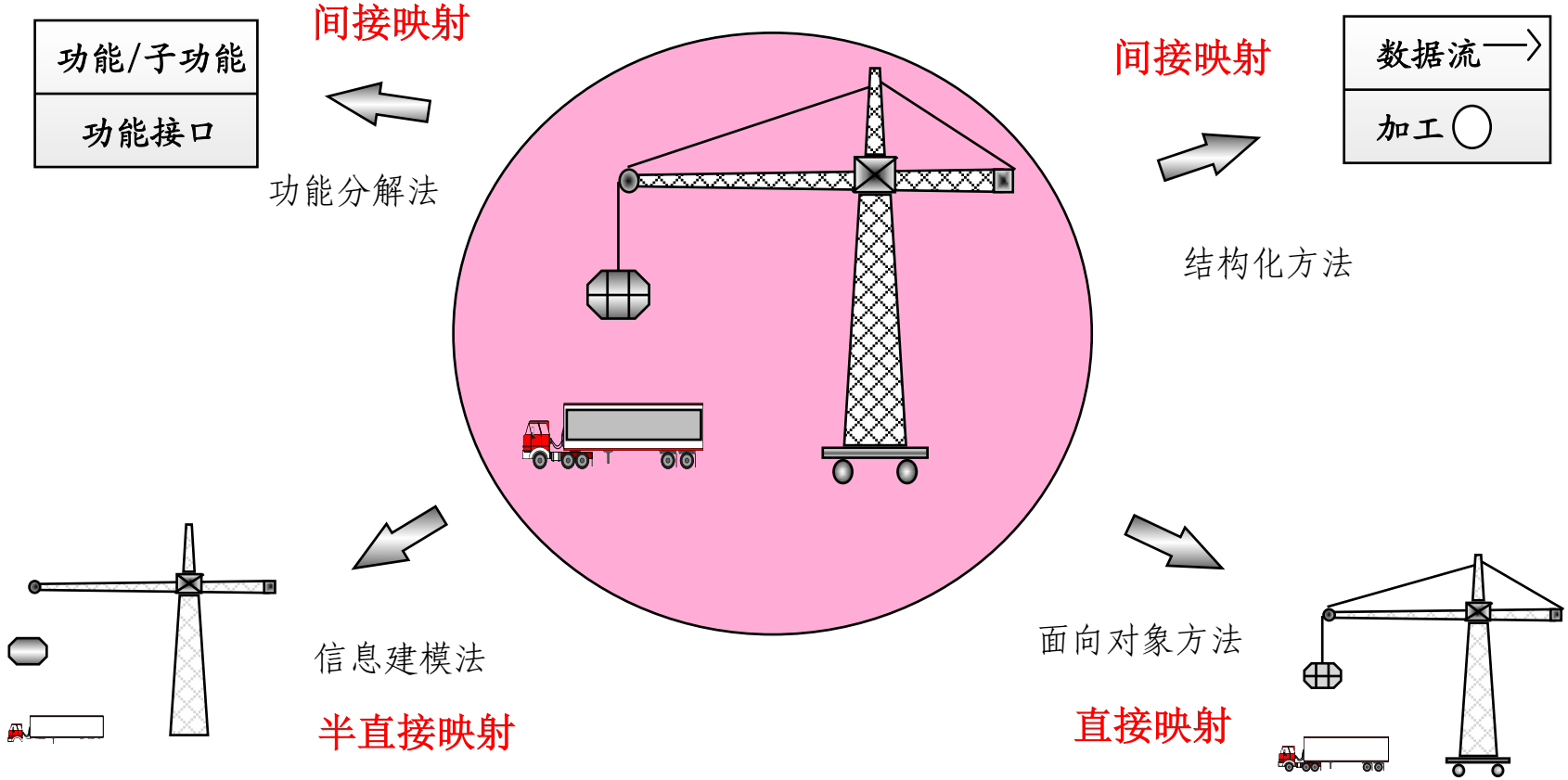
信息建模方法的特点

- 从问题域中的具体事物出发认识问题域，系统模型对问题域的映射比功能分解法和数据流法更直接
- 与面向对象方法相比，是半直接映射
  - 强调的重点是信息建模和状态建模，而不是对象建模
  - 没有把对实体属性所进行的操作封装到实体对象中
  - 只有属性的继承，不支持操作的继承
  - 没有采用消息通讯

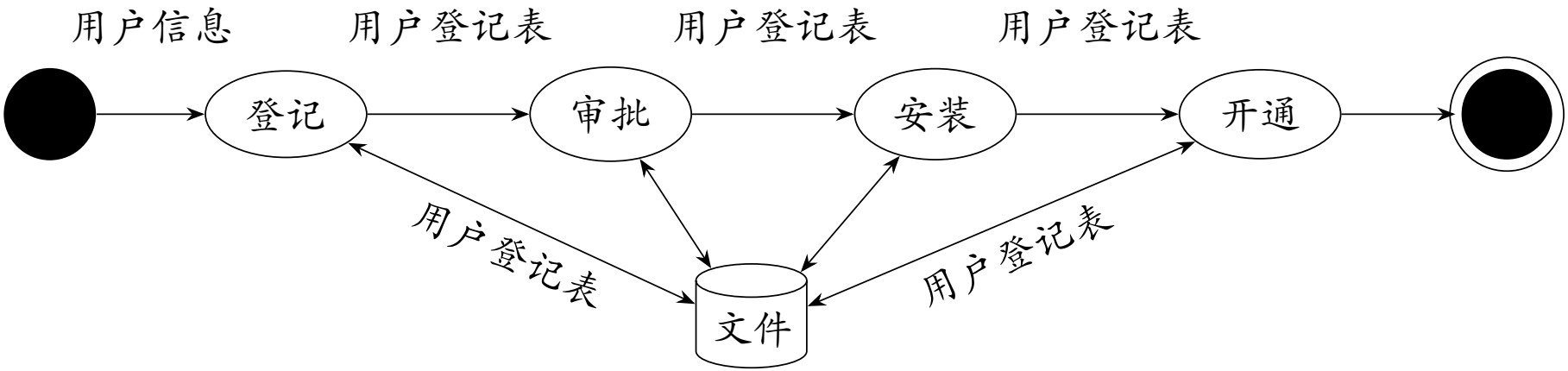
面向对象方法：面向对象的分析 + 面向对象的设计

- 面向对象方法把问题域中的事物抽象为对象，以对象作为系统的基本构成单位
- 对象的属性和操作刻画了事物的静态特征和动态特征，完整地刻画了问题域中事物
- 对象间的继承、聚合、关联、消息等关系，如实地表达了问题域中事物之间的各种关系，符合人类的日常思维，使系统的复杂性得到控制
- 得到的系统模型可以直接映射问题域

面向对象方法和其他方法的比较

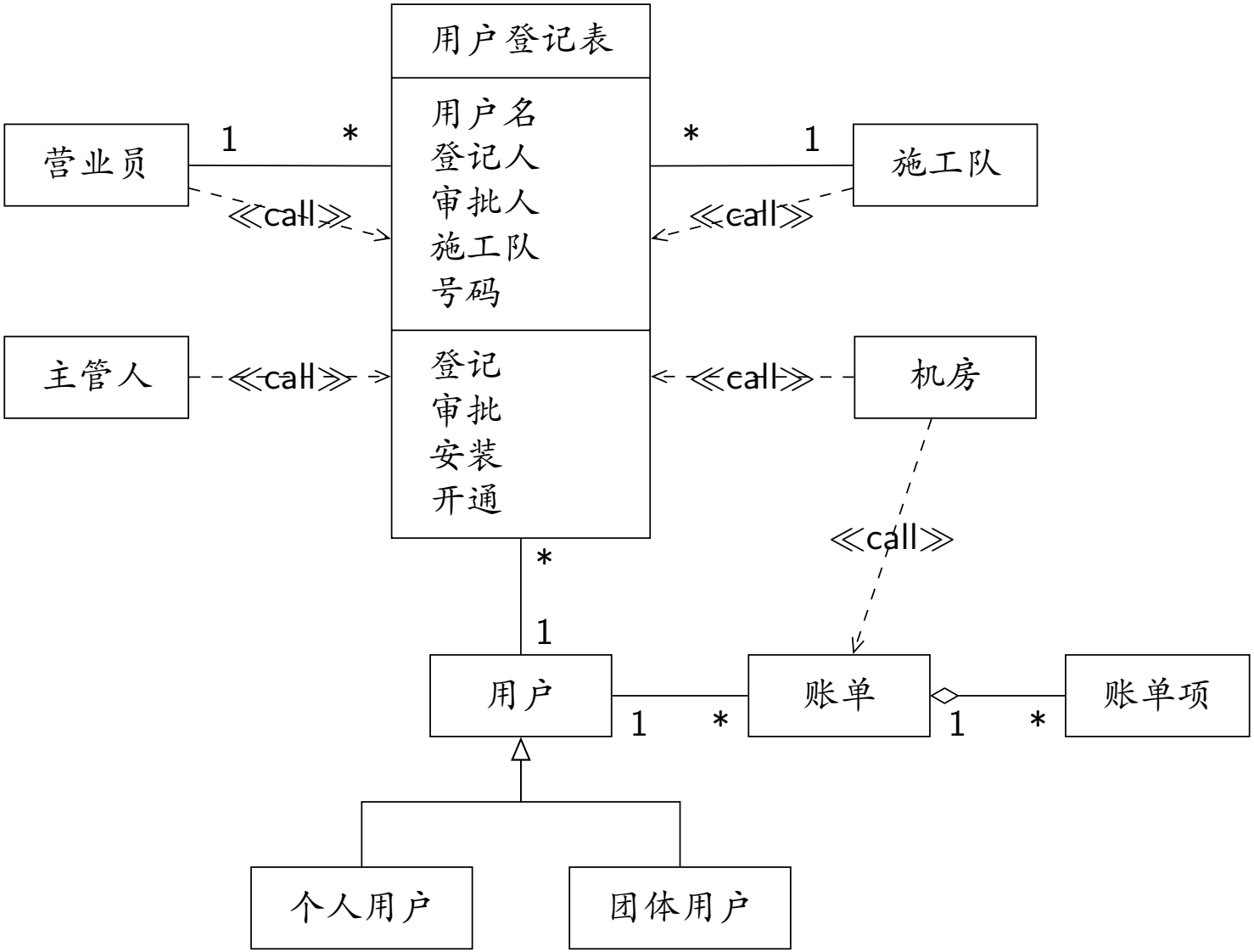


OO vs 结构化：数据流和加工



- 不直接映射问题域，与问题域事物相关的数据和操作不是围绕这些事物来组织的，而是分散在数据流和加工中
- 信息膨胀，模型中的多个数据流实现中可能只是一项数据
- 分析模型难以与设计模型及源程序对应

OO vs 结构化: 对象及其关系



面向对象分析 OOA

- 定义：OOA 是软件生存周期的一个阶段，具有一般分析方法共同具有的内容、目标及策略；但是，它强调运用面向对象方法进行分析，用面向对象的概念和表示法表达分析结果
- 任务：运用面向对象的概念对问题域进行分析和理解，将问题域中与系统责任有关的事物抽象为系统中的类和对象，定义其属性与操作，以及它们之间的各种关系
- 目标：建立一个满足用户需求、直接映射问题域的 OOA 模型及其规约

面向对象设计 OOD – 早期

- 不是基于 OOA 的，大多基于结构化分析结果
- 是 OO 编程方法的延伸，多数方法与编程语言有关，特别受 Ada 影响很大
- 不是纯 OO 的，对某些 OO 概念（如继承）缺少支持，挽杂一些非 OO 概念（如数据流、包、模块等）
- 不是只针对软件生存周期的设计阶段，涉及一些系统分析问题（如识别问题域的对象）

面向对象设计 OOD – 现今

- 以 OOA 为基础，一般不依赖结构化分析
- 和 OOA 方法构成一种共同方法体系，采用一致的概念与原则，但分属软件生存周期的不同阶段，有不同的目标及策略
- 较全面地体现面向对象方法的概念与原则
- 大多数方法独立于编程语言

面向对象设计 OOD – 现今

- 以 OOA 为基础，一般不依赖结构化分析
- 和 OOA 方法构成一种共同方法体系，采用一致的概念与原则，但分属软件生存周期的不同阶段，有不同的目标及策略
- 较全面地体现面向对象方法的概念与原则
- 大多数方法独立于编程语言

面向对象设计 OOD

就在是 OOA 模型基础上运用面向对象方法进行系统设计，产生一个符合具体实现条件的 OOD 模型

软件建模面临的挑战：问题域和系统责任的复杂性

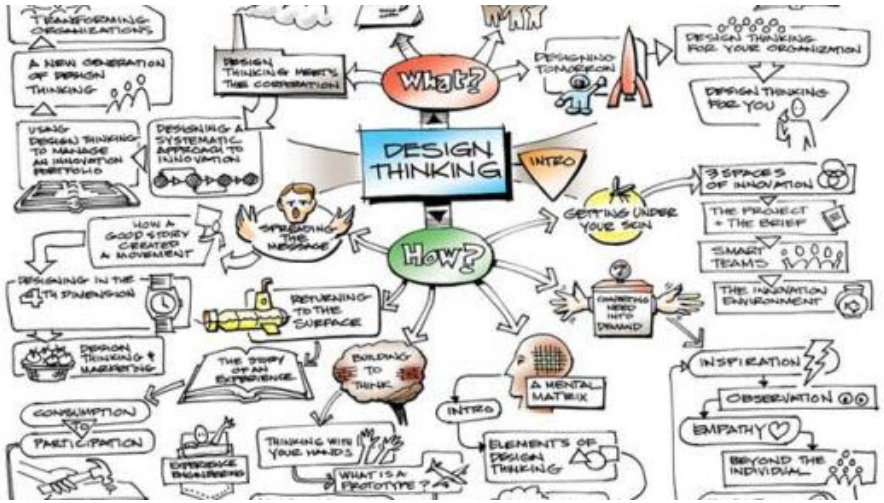
随着硬件性能的提高和价格的下降，软件系统所面临的问题域和系统责任越来越复杂，因此系统也越来越庞大

问题域（problem domain）

所开发系统的应用领域，即业务范围

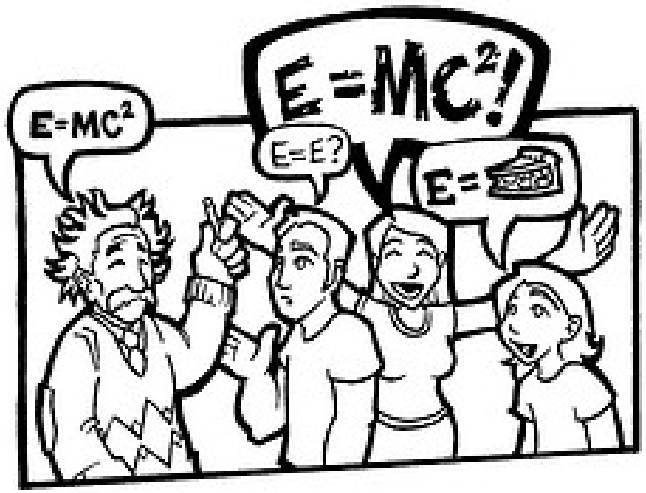
系统责任（system responsibilities）

所开发系统应该具备的职能



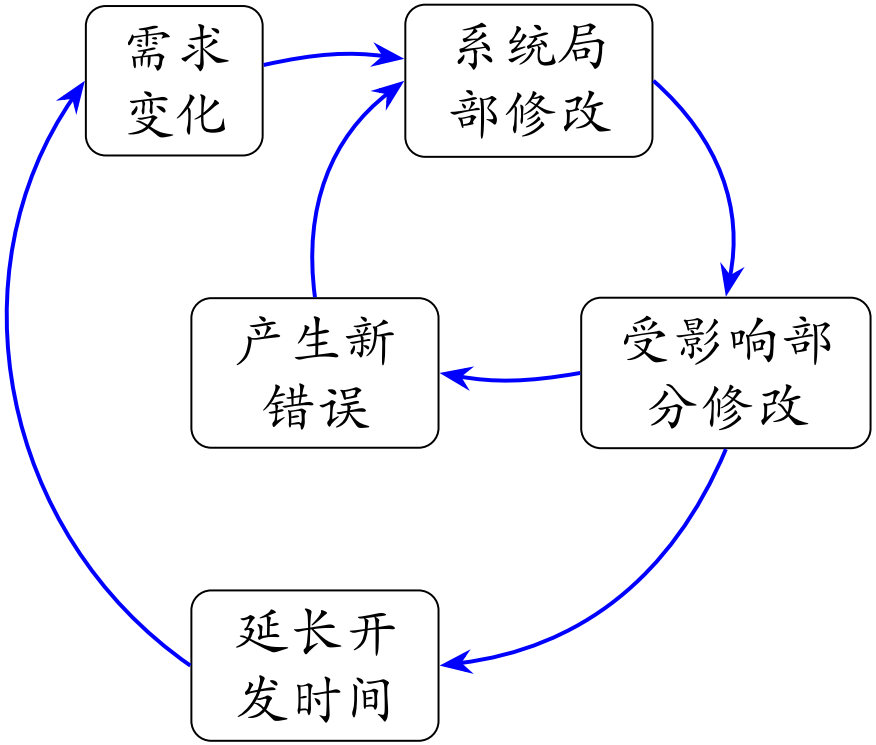
软件建模面临的挑战：交流问题

- 软件系统的开发需要各类人员之间频繁交流，领域的多样性使软件工程中的交流问题比其他工程更为突出
- 有效的交流需要一种彼此都能理解的共同语言，否则将使彼此的思想难以沟通，很容易隐藏下许多错误



软件建模面临的挑战：需求变化

开发者须接受和适应需求变化：用户、竞争、经费等因素...



- 功能：最易变
- 外部接口：很易变
- 属性：较易变
- 对象：较稳定

软件建模面临的挑战：复用

复用级别提高——分析结果复用，要求分析模型的基本成分可以在多个系统中复用，要求一个分析模型可以在多种条件下设计和实现

面向对象方法的优势

- 对问题域和系统责任的复杂性具有较强的处理能力
  - 系统模型能直接地映射问题域
- 对需求的变化具有较强的适应性
  - 按封装原则把系统中最容易变化的因素隔离起来
- 为实现分析与设计级别的软件复用提供了有力支持
  - 封装、继承、聚合等原则，对象的完整性、独立性等
- 提供了便于各类相关人员交流共同语言
  - 贯穿软件生存周期全过程的、与问题域一致的概念、词汇、原则及表示法

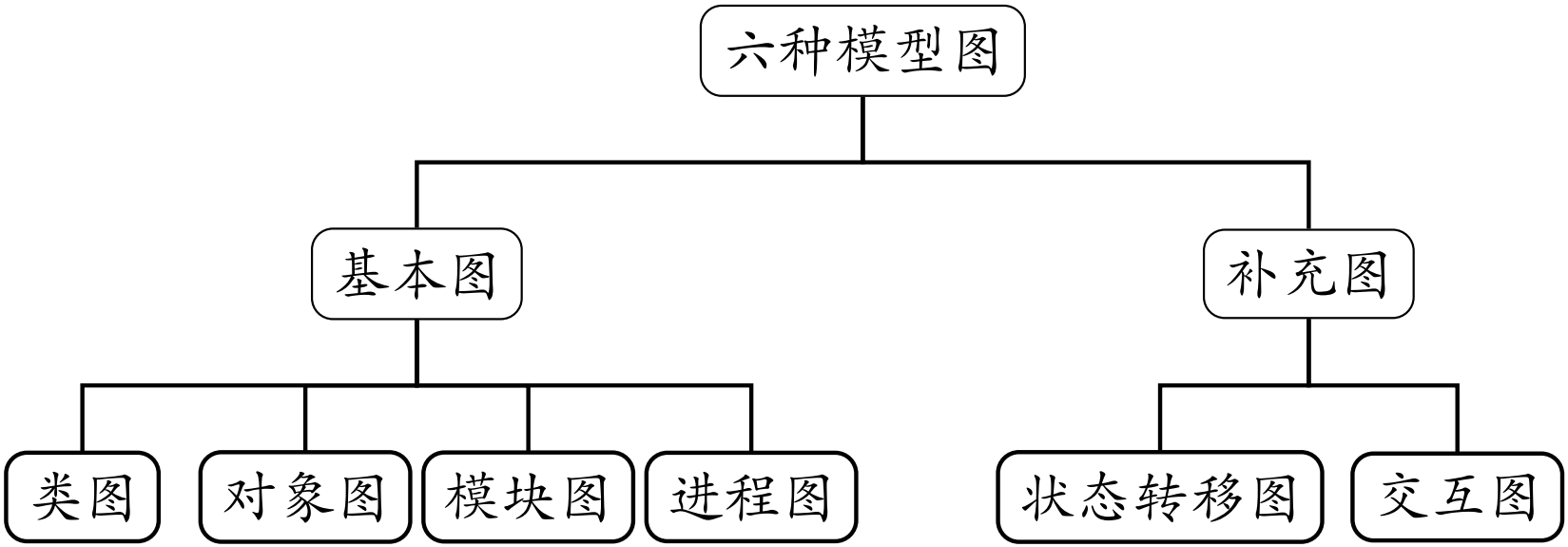
几种典型的 OO 方法

方法的异同

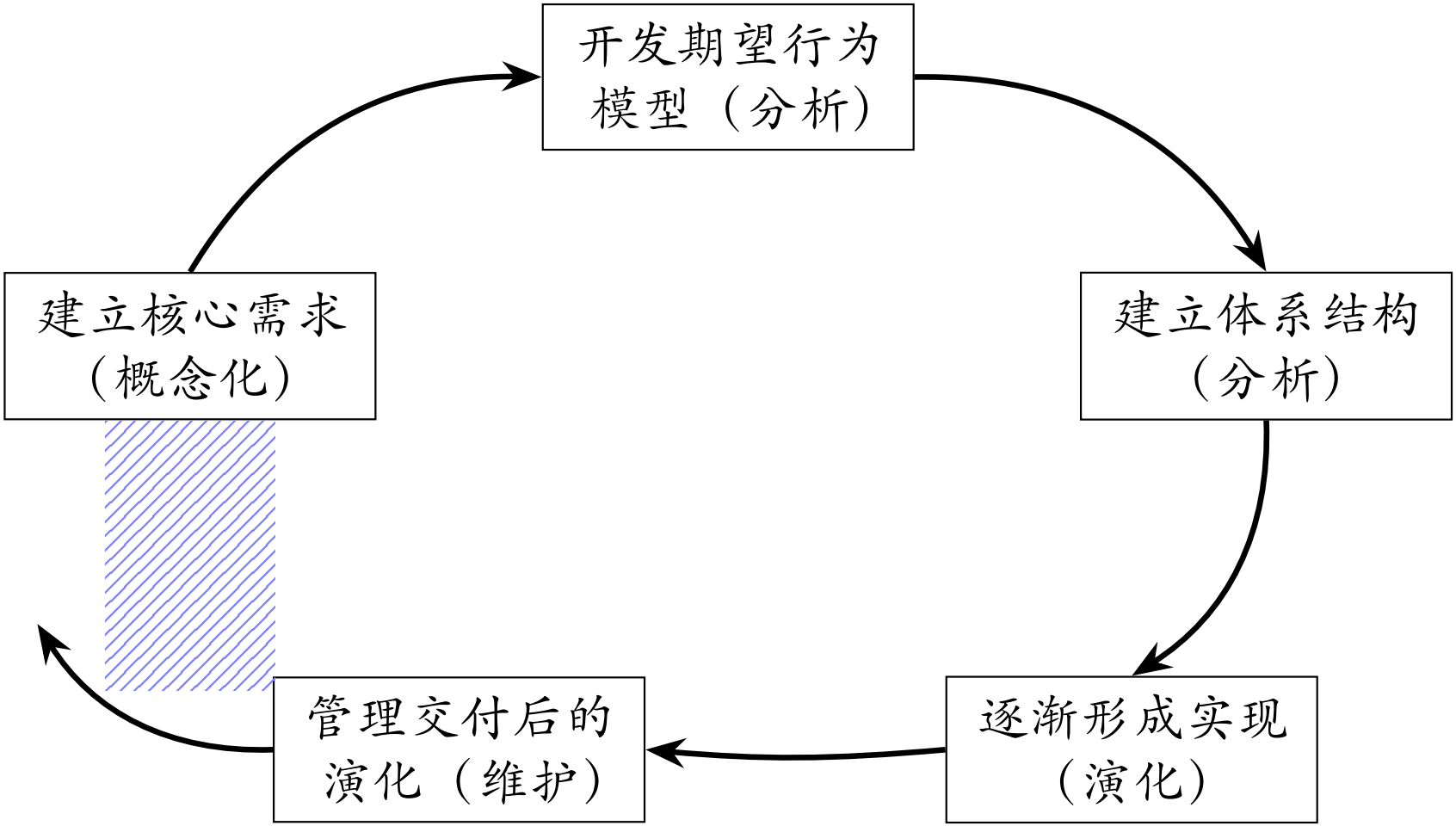
概念、表示法、系统模型、开发过程、可操作性、技术支持等

- Booch 方法
- Coad-Yourdon 方法
- Jacobson 方法
- Rumbaugh 方法
- ...

Booch 方法

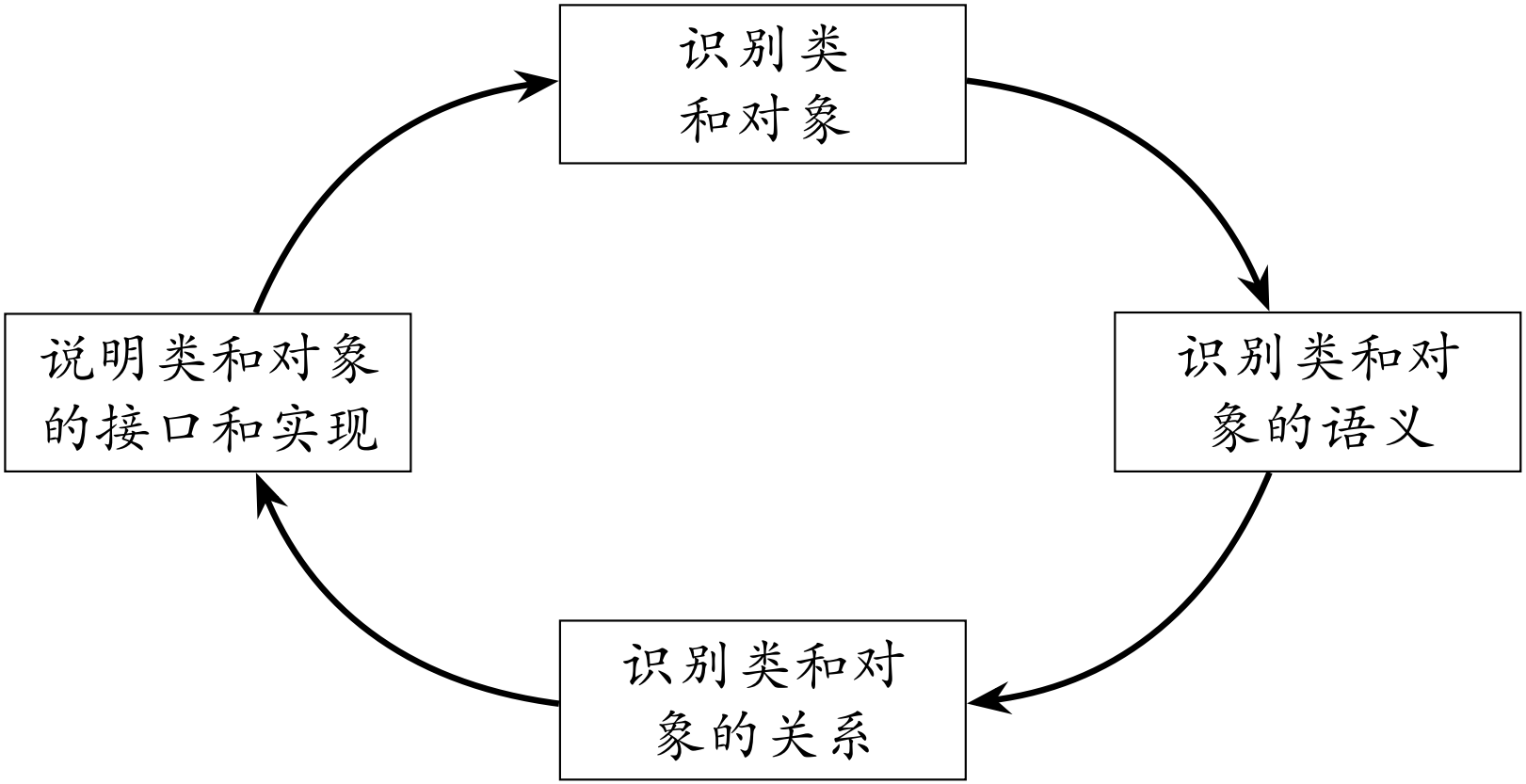


Booch 方法宏过程



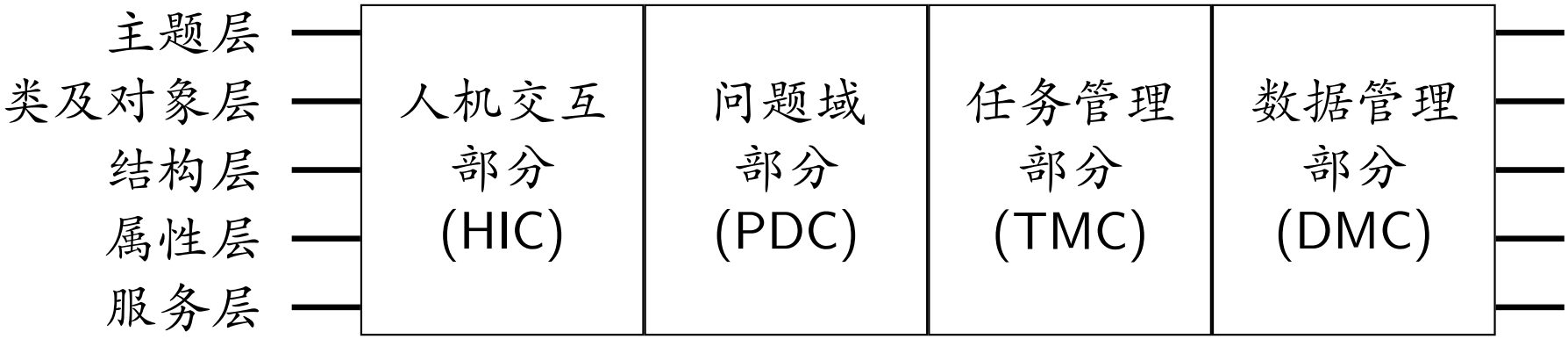
Booch 方法微过程

- 类图和对象图并存



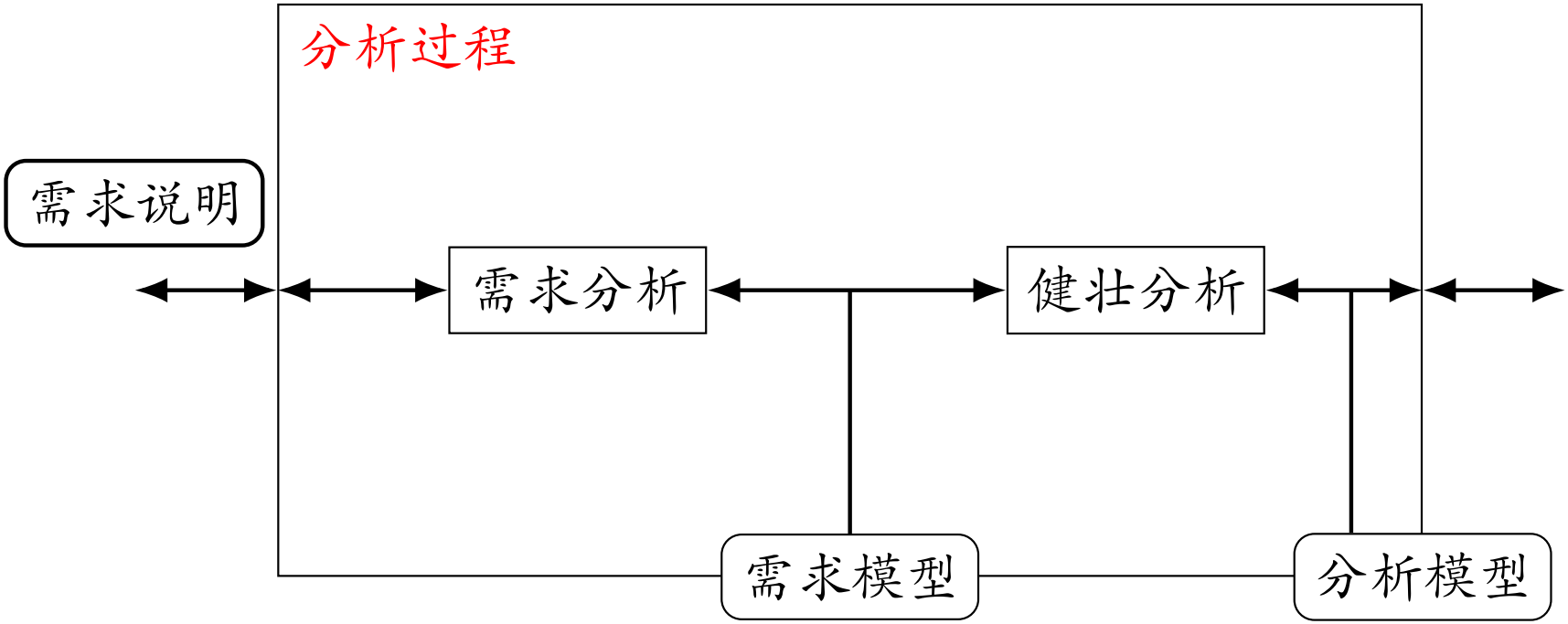
Coad/Yourdon 方法

- 概念简练，过程清晰
- 强调概念的一致性
- 过程指导不够具体

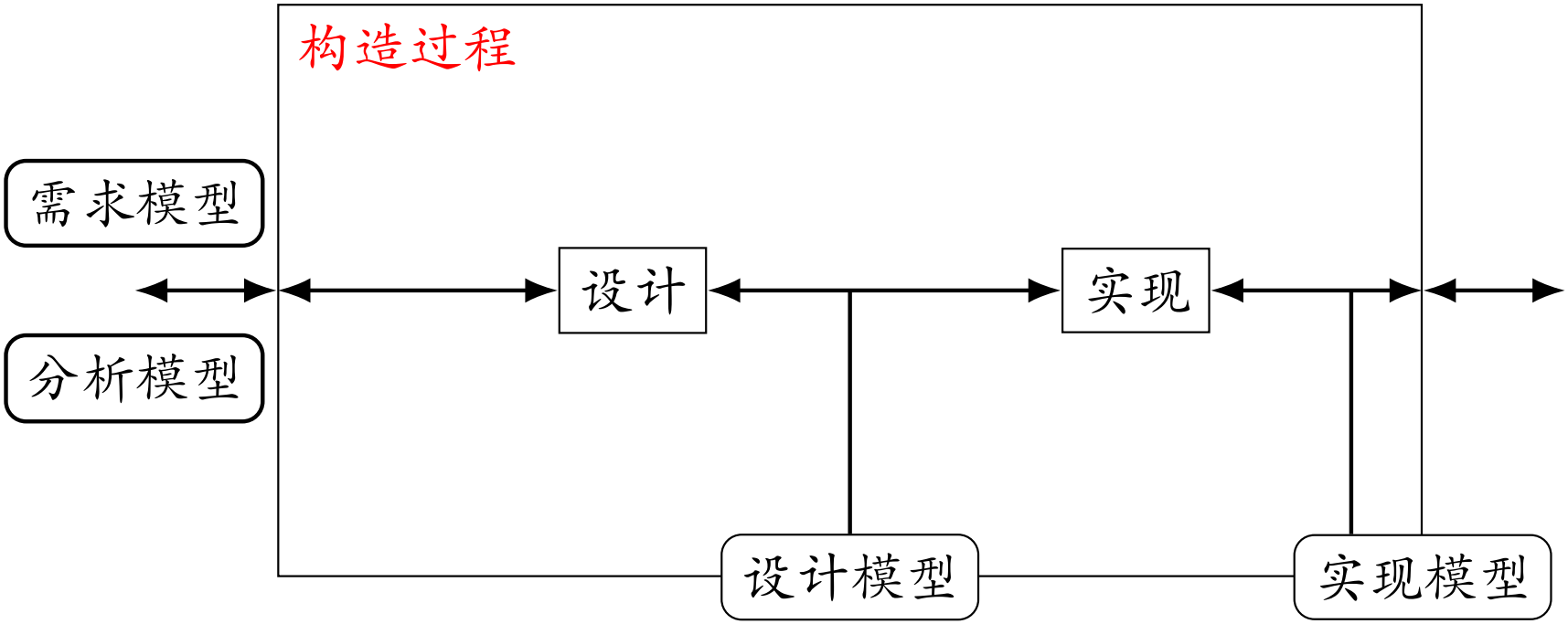


OOD 模型的 5 个层次和 4 个部分

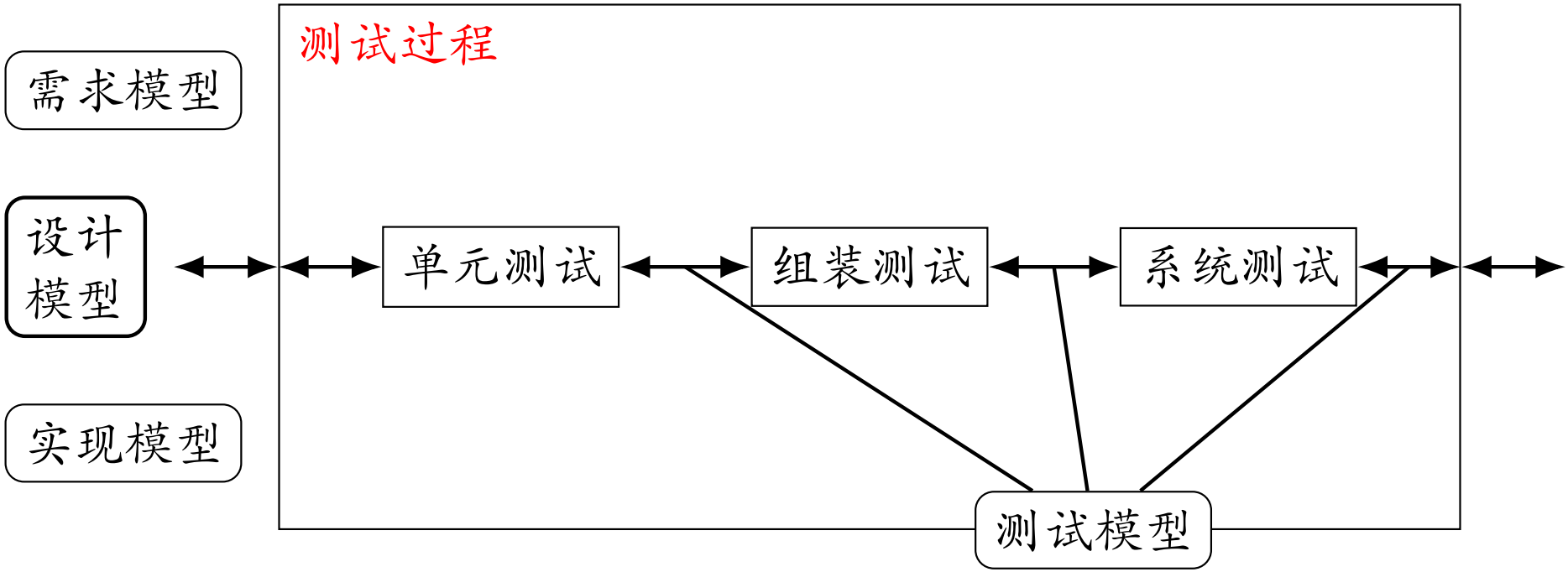
Jacobson 方法 (OOSE): 用况驱动



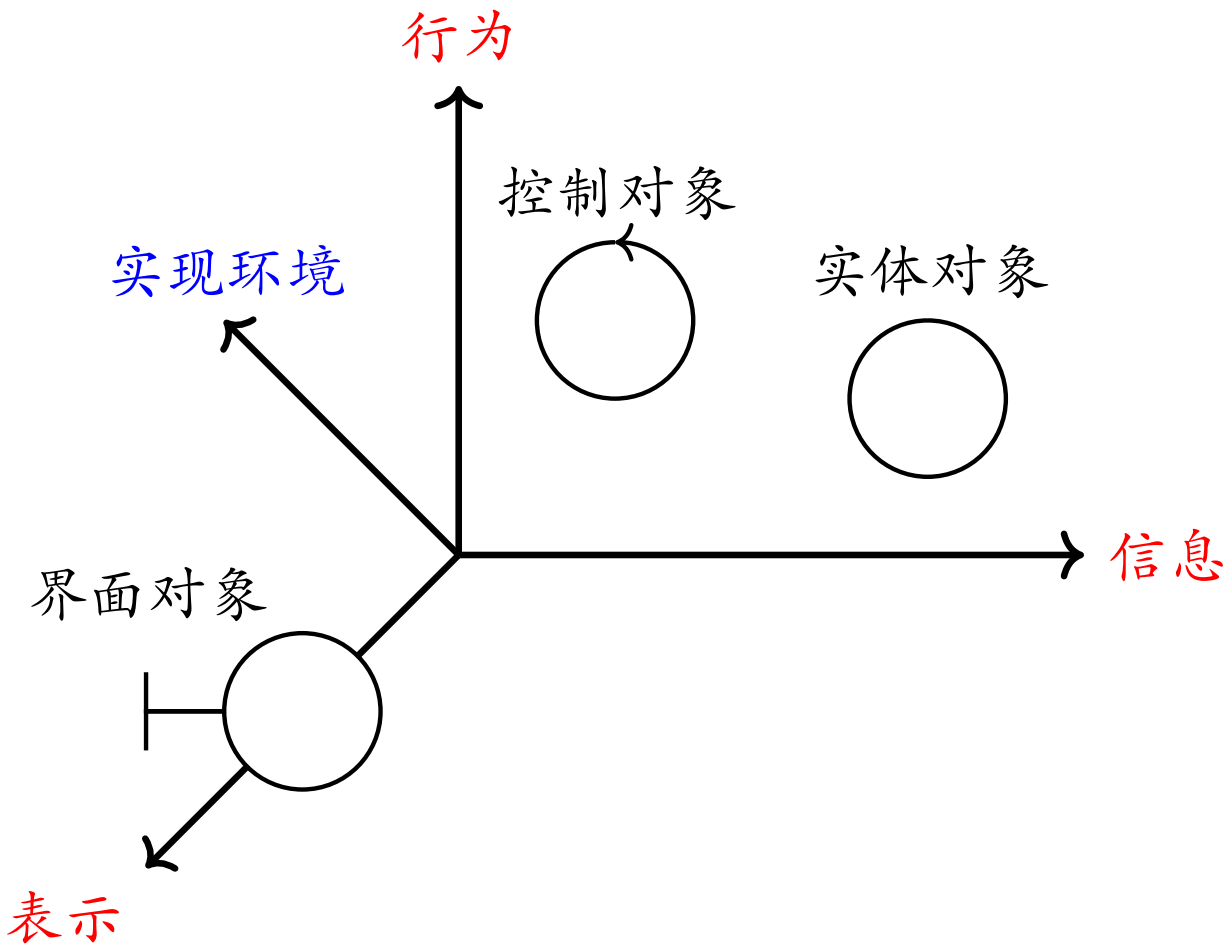
Jacobson 方法 (OOSE): 用况驱动



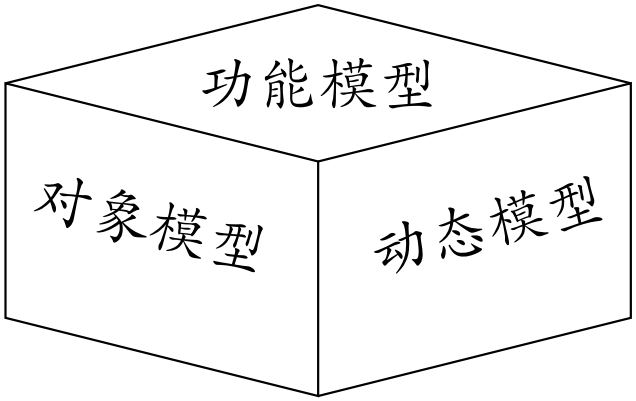
Jacobson 方法 (OOSE): 用况驱动



Jacobson 方法 (OOSE): 用况驱动



Rumbaugh 方法 (OMT)



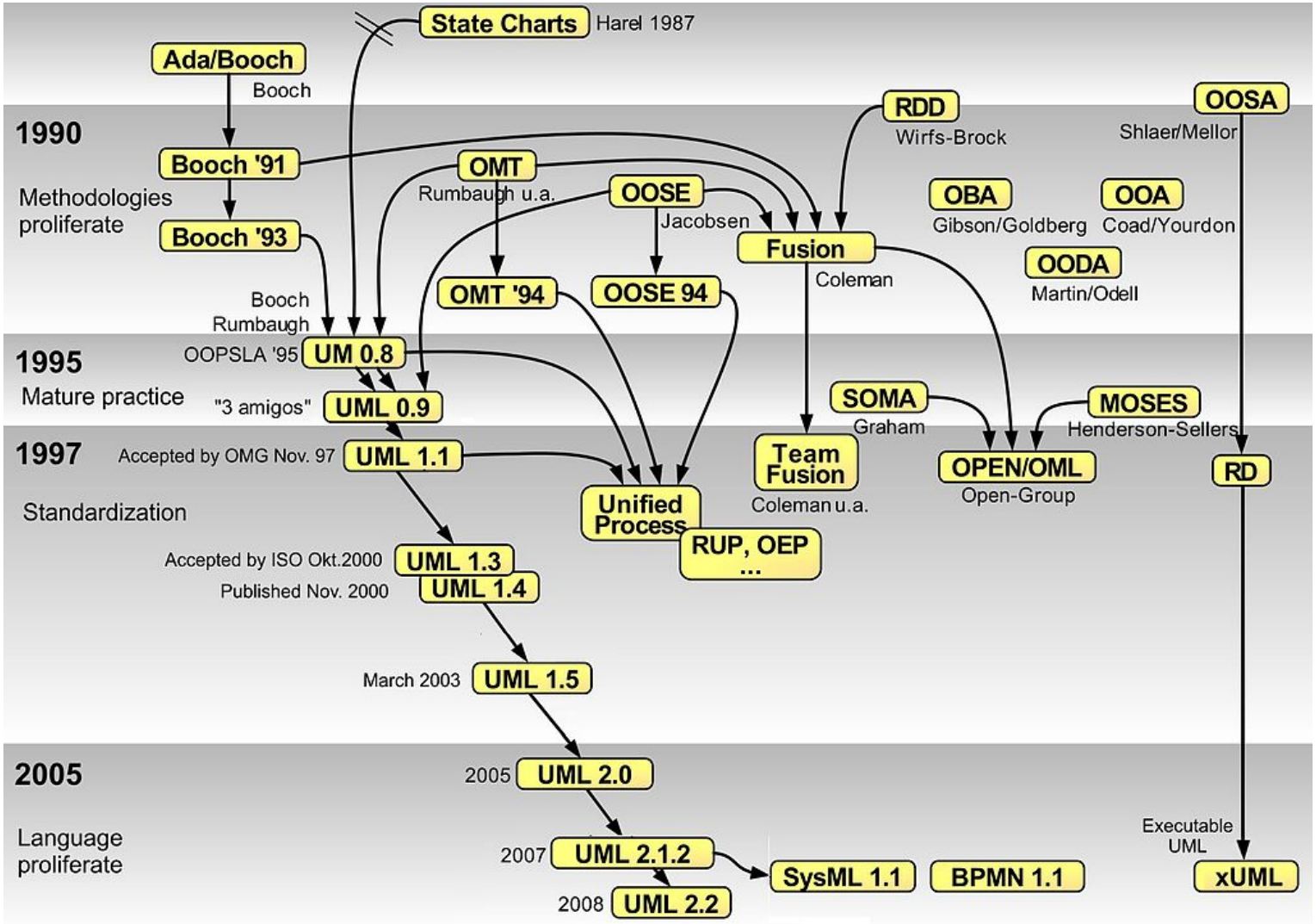
过程

- 分析（面向对象）
- 系统设计（传统方法）
- 对象设计（面向对象）
- 实现

特点

概念严谨，阐述清楚；过程具体，可操作性强；包含了许多非 OO 的内容，提出若干扩充概念，偏于复杂

对象技术发展脉络


State Charts
Harel 1987
Ada/Booch
Booch
1990
Booch '91
Booch '93
Methodologies
proliferate
OMT
Rumbaugh u.a.
OOSE
Jacobsen
RDD
Wirfs-Brock
OOSA
Shlaer/Mellor
OBA
Gibson/Goldberg
OOA
Coad/Yourdon
Fusion
Coleman
OODA
Martin/Odell
OMT '94
OOSE 94
Booch
Rumbaugh
1995
OOPSLA '95
UM 0.8
Mature practice
"3 amigos"
UML 0.9
SOMA
Graham
MOSES
Henderson-Sellers
1997
Accepted by OMG Nov. 97
UML 1.1
Team
Fusion
Coleman u.a.
OPEN/OML
Open-Group
RD
Unified
Process
RUP, OEP
...
Standardization
Accepted by ISO Okt.2000
UML 1.3
Published Nov. 2000
UML 1.4
March 2003
UML 1.5
2005
2005
UML 2.0
Language
proliferate
2007
UML 2.1.2
SysML 1.1
BPMN 1.1
Executable
UML
xUML
2008
UML 2.2

内容提要

诞生背景

- 面向对象方法种类繁多
  - 1989 年约 10 种，1994 年达到 50 种以上
- 概念、表示法、过程策略及文档组织等方面的差异
  - 使用户在选择建模方法和工具时无所适从，不利于技术交流
- 迫切需要 OO 概念及表示法走向统一和标准化
  - 统一建模语言 UML 应运而生

第一阶段: OO 方法学家的联合行动

1995.10: G. Booch 与 J. Rumbaugh 联合, 推出 Unified Method 0.8

1996.06: I. Jacobson 加入，推出 UML 0.9 (Unified Modeling Languange)

第二阶段: 公司的联合行动

1996: 成立了 UML 伙伴组织，12 家公司加入

1997.01: 推出 UML1.0，另外 5 家公司加盟

1997.09: 形成 UML1.1，提交 OMG 作为建模语言规范提案

1997.11: UML1.1 被 OMG 正式采纳

第三阶段: OMG 主持下的修订

1997-2002: OMG 成立 UML 修订任务组，先后产生 UML 1.2、1.3、1.4、1.5 等版本

2000-2001: OMG 发布 4 个提案需求 RFP，征集对 UML 的显著改进提案

2002- : 先后形成 4 个 UML2.0 规范，并继续改进

UML 是什么

统一建模语言（UML）

是一种用来对软件密集型系统制品进行可视化、详述、构造和建档的图形语言，也可用于业务建模以及其它非软件系统建模

- 是一种建模语言，不是一种建模方法
- 用于建立系统的分析模型和设计模型，而不是用于编程
- 是一种已被 OMG 采纳的建模语言规范（specification）
- 部分地采用了形式化语言的定义方式，但并不严格，不能编译执行或解释执行

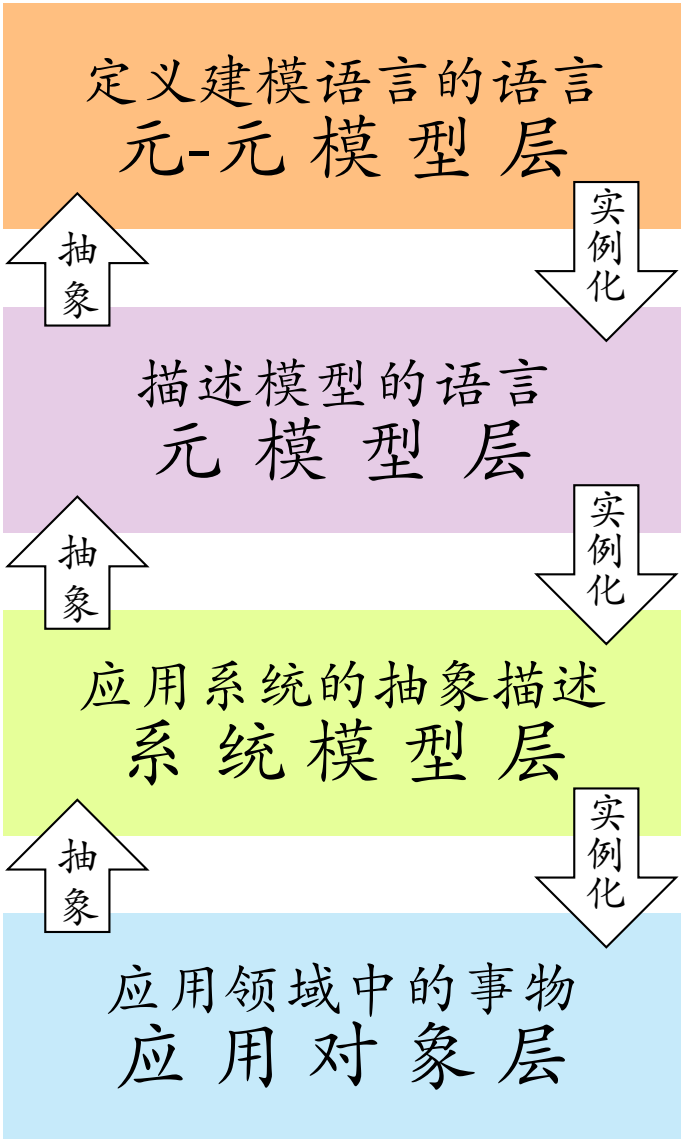
UML1 主要规范

- UML 概要（UML Summary）
- UML 语义（UML Semantics）
- **UML** 表示法指南（UML Notation Guide）
- UML 外廓范例（UML Example Profiles）
- UML 模型交换（UML Model Interchange）
- 对象约束语言 OCL （Object Constraint Language）

UML1 的 9 种模型图

- 静态结构图（Static Structure Diagram）
  - 类图（Class Diagram）
  - 对象图（Object Diagram）
  - 用况图（Use Case Diagram）
- 交互图（Interaction Diagram）
  - 顺序图（Sequence Diagram）
  - 协作图（Collaboration Diagram）
  - 状态图（State chart Diagrams）
  - 活动图（Activity Diagrams）
- 实现图（Implementation Diagrams）
  - 构件图（Component Diagram）
  - 部署图（Deployment Diagram）

OMG 四层元模型体系结构



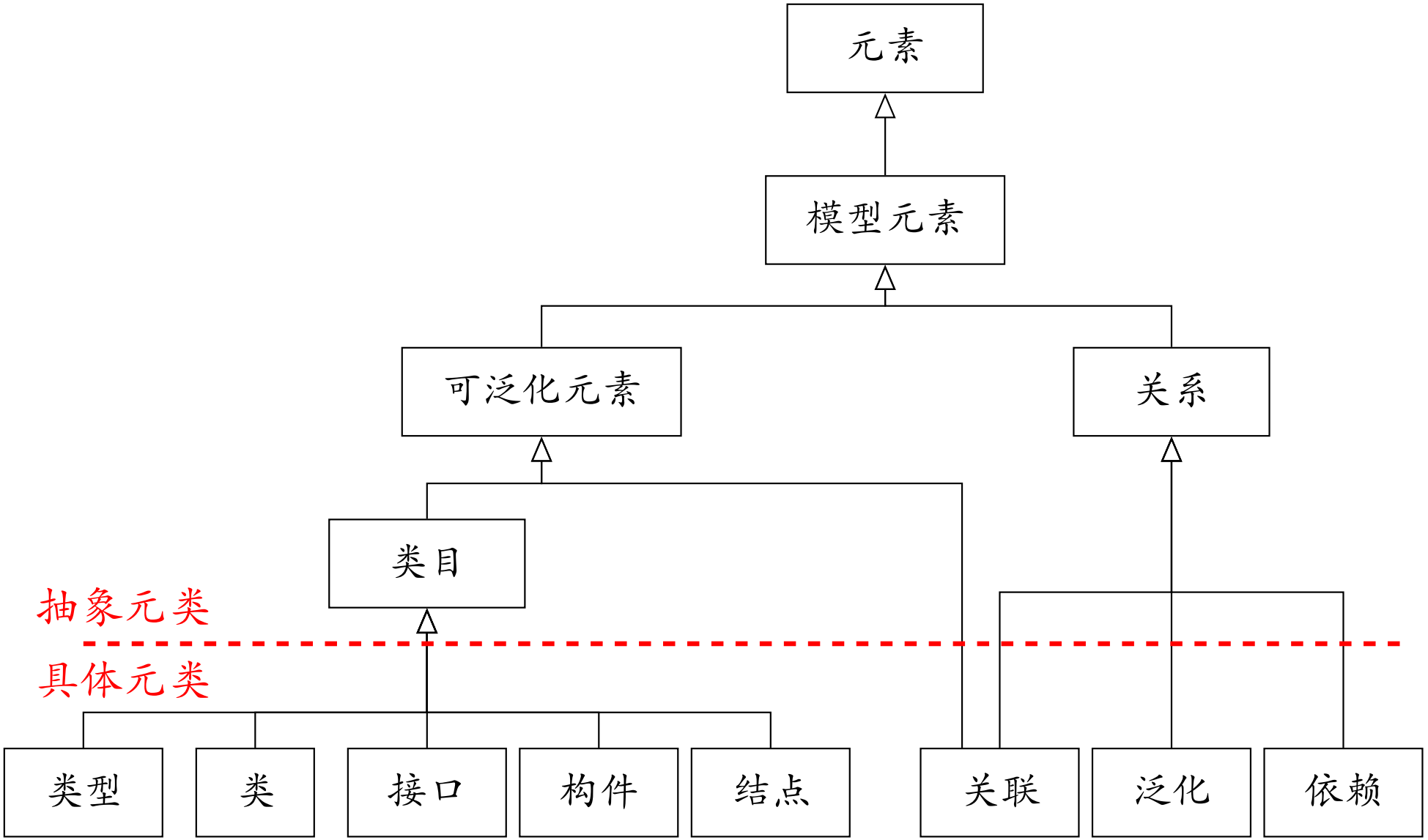
元-元模型（meta-metamodel）：元模型的基础体系结构，定义一种说明元模型的语言。例如：MOF

元模型（metamodel）：元－元模型的一个实例，定义一种说明模型的语言。例如：UML

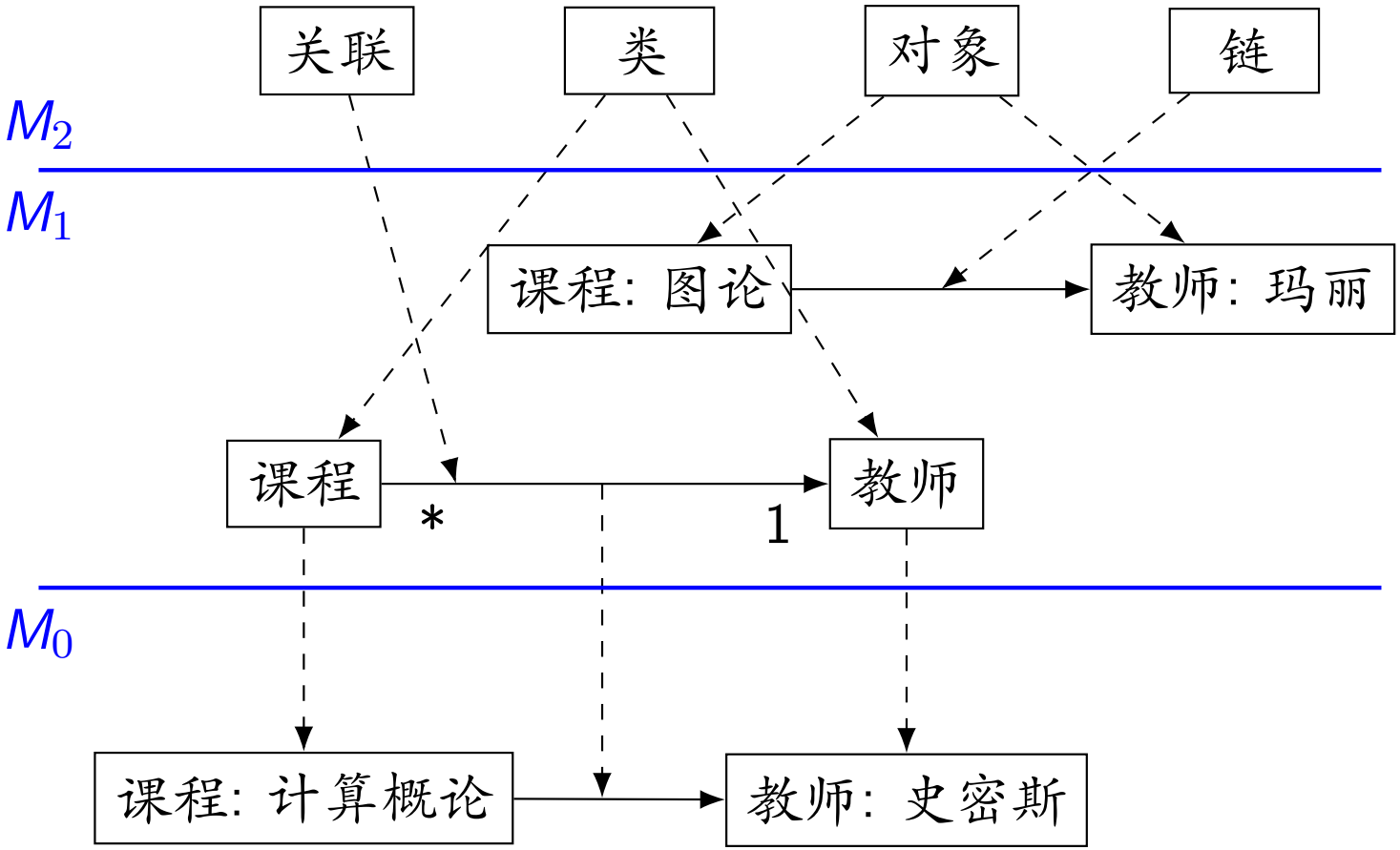
模型（model）：元模型的一个实例，定义一种语言来描述信息领域。例如：教学管理系统中的教室类、学生类、课程类

用户对象（user object）：模型的一个实例，定义一个特定的信息领域。例如：一个学校——某老师，某学生，某课程

抽象元类与具体元类



元模型中实例级概念引起体系结构层次的混乱



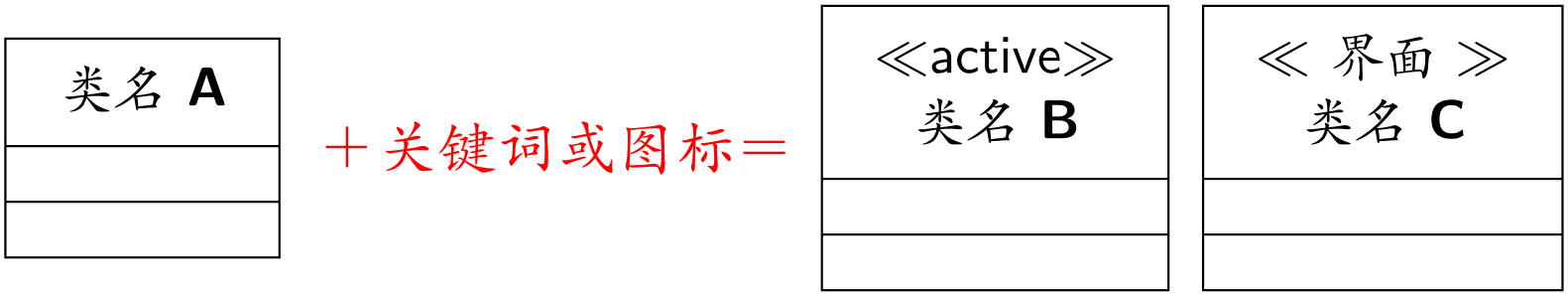
扩展机制

扩展机制

附加到其他模型元素之上以，将原有的建模元素特化成一种语义较特殊的新变种，或者表示出它们的某些细节

- 约束（constraint）：用于说明某些必须保持为真的命题
- 注释（comment）：对模型元素的细节所进行的解释
- 标记值（Tagged Value）：表示模型元素的附加的特征
- 衍型（stereotype）：附加到其他模型元素之上，从而将原有的建模元素定制成一种语义较为特殊的新变种

衍型的表示法和例子



UML2 概况

UML Superstructure

提供可直接用来构造用户系统的各种模型元素，以及从不同的视角对系统进行建模的各种模型图

UML Diagram Interchange

给出在不同的建模工具之间实现模型交换的规范

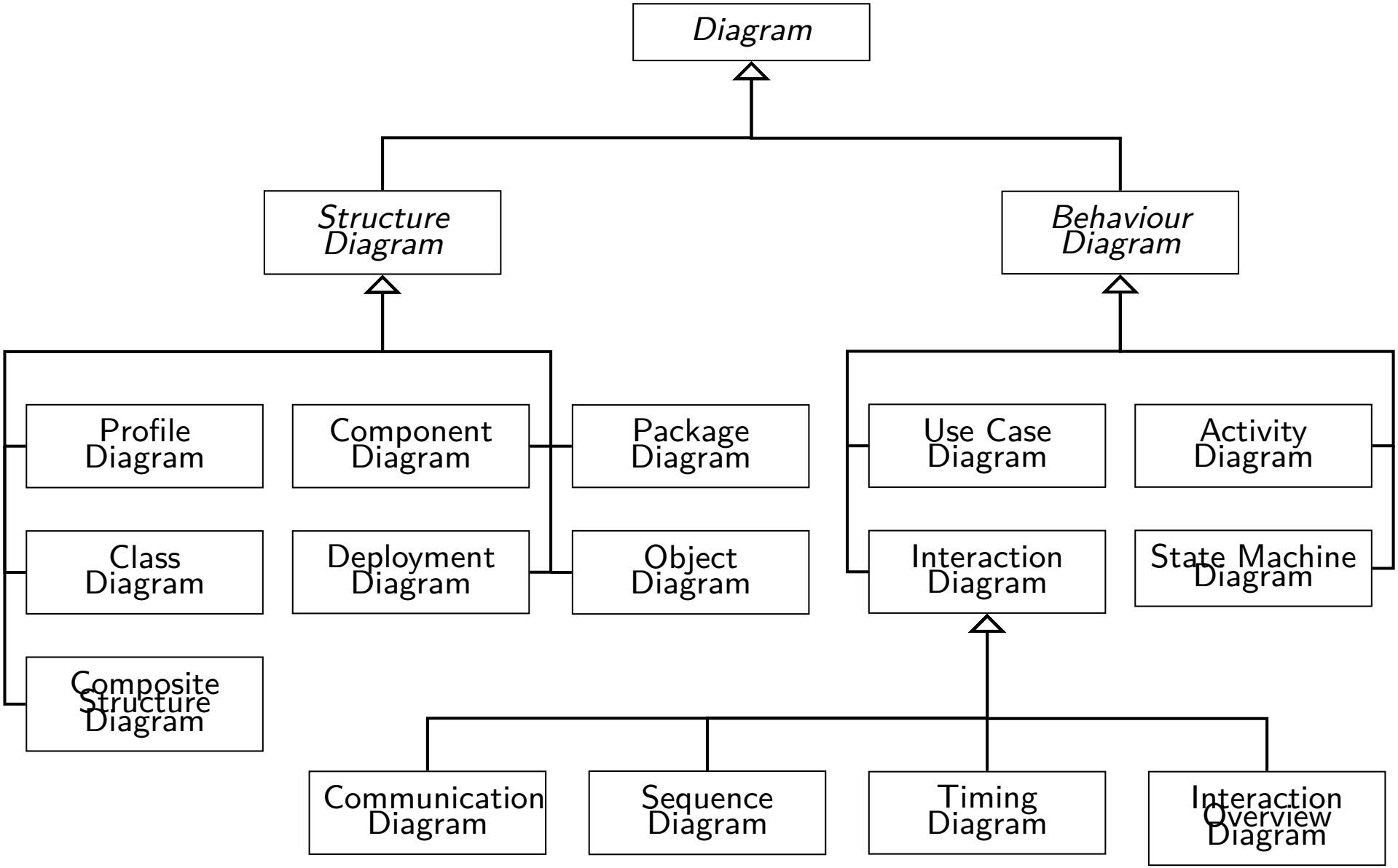
UML Infrastructure

定义一个可复用的元语言核心，用来定义 UML、MOF 和 CWM 等元模型

UML OCL

一个形式化的语言，描述模型约束信息

UML2 的 14 种模型图



学习建议

- 重点掌握 UML 直接提供给应用模型开发者使用的建模元素，即“具体元类”；熟练地掌握面向对象方法最基本的概念
- 掌握类图、用况图、顺序图、活动图、状态机图、构件图
- 着眼于实际应用，从 UML 的复杂性中解放出来
- UML 只是一种建模语言，不是建模方法，它不包括过程，而且是独立于过程的
- 动手实践，使用工具；选择合适的项目开始实际应用

内容提要

宗旨

- 充分运用面向对象方法的基本概念，限制扩充概念
- 加强过程指导
  - 给出用最基本的 OO 概念自然而有效地解决建模问题的策略
- 强调在类的抽象层次上建立系统模型
  - 所有对象的属性和操作以及对象之间的关系，都通过它们的类来描述，而不是针对个别对象实例进行描述

主要建模元素

- 对象、类（所有的对象都通过类来表示）
- 属性、操作（类属性和实例属性，被动操作和主动操作）
- 一般-特殊关系，一般-特殊结构
- 整体-部分关系，整体-部分结构
- 关联（二元关联、多元关联）
- 消息（控制流内部的消息，控制流之间的消息）

主要原则

- 抽象: 数据抽象，过程抽象
- 分类
- 封装
- 继承
- 聚合
- 关联
- 消息通信
- 粒度控制
- 行为分析

OOA 模型和 OOD 模型

模型：正式理解

一个系统模型，应包括建模过程中产生的图形、文字等各种形式的文档。因为，所谓“模型”是指某一级别上的系统抽象描述，构成这种描述的任何资料都是模型的一部分

模型：习惯说法

大部分 OOA/OOD 著作谈到“模型”，一般是指 OOA 或 OOD 过程中产生的图形文档

本课程采用习惯说法，将模型和模型规约分开讨论

三种模型

基本模型—类图

面向对象的建模中最重要、最基本的模型图，可从对象层、特征层、关系层看

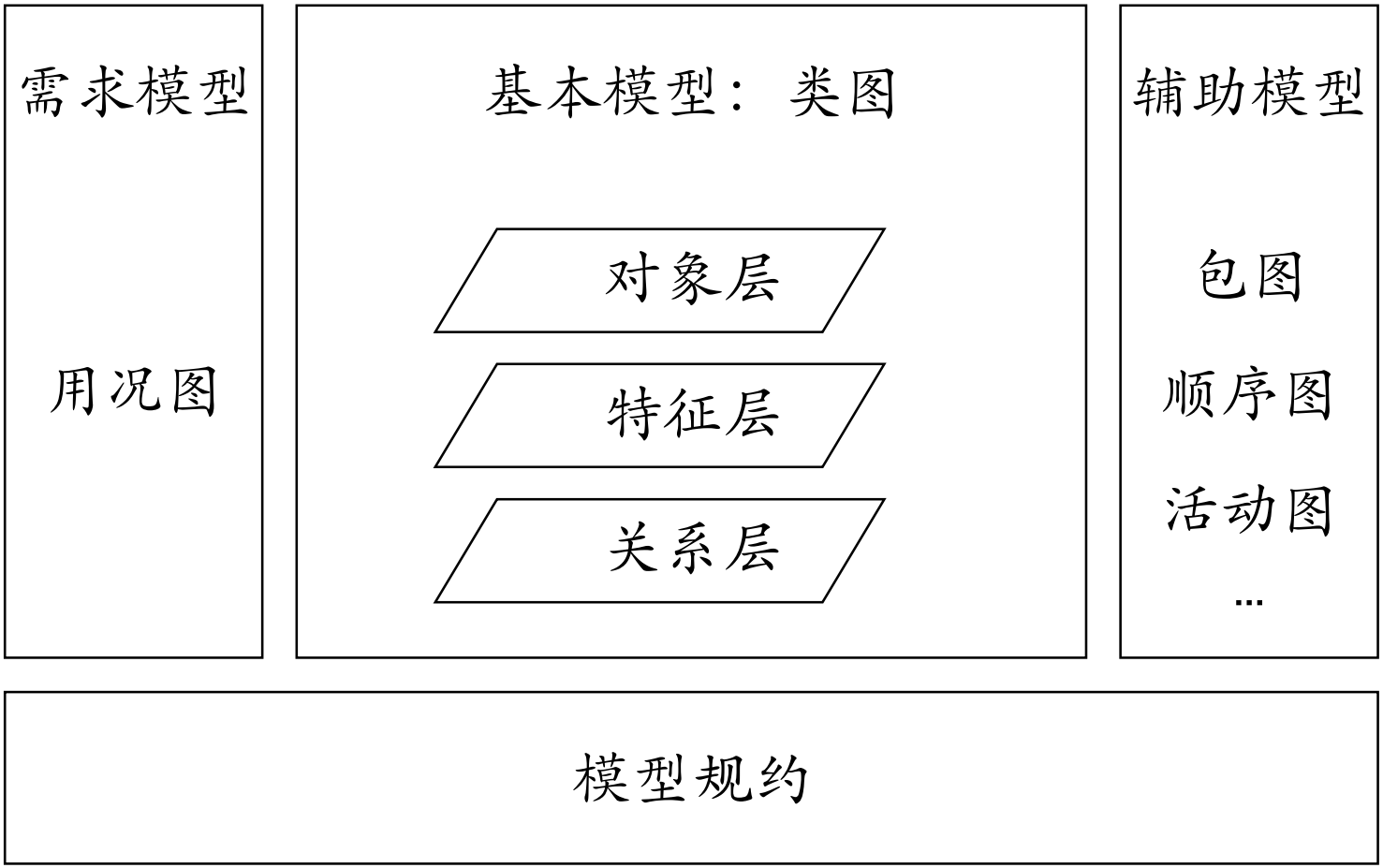
需求模型—用况图

用况是一项系统功能使用情况的说明。把所有参与者对系统功能的使用情况确切描述出来，便全面定义了系统的功能需求

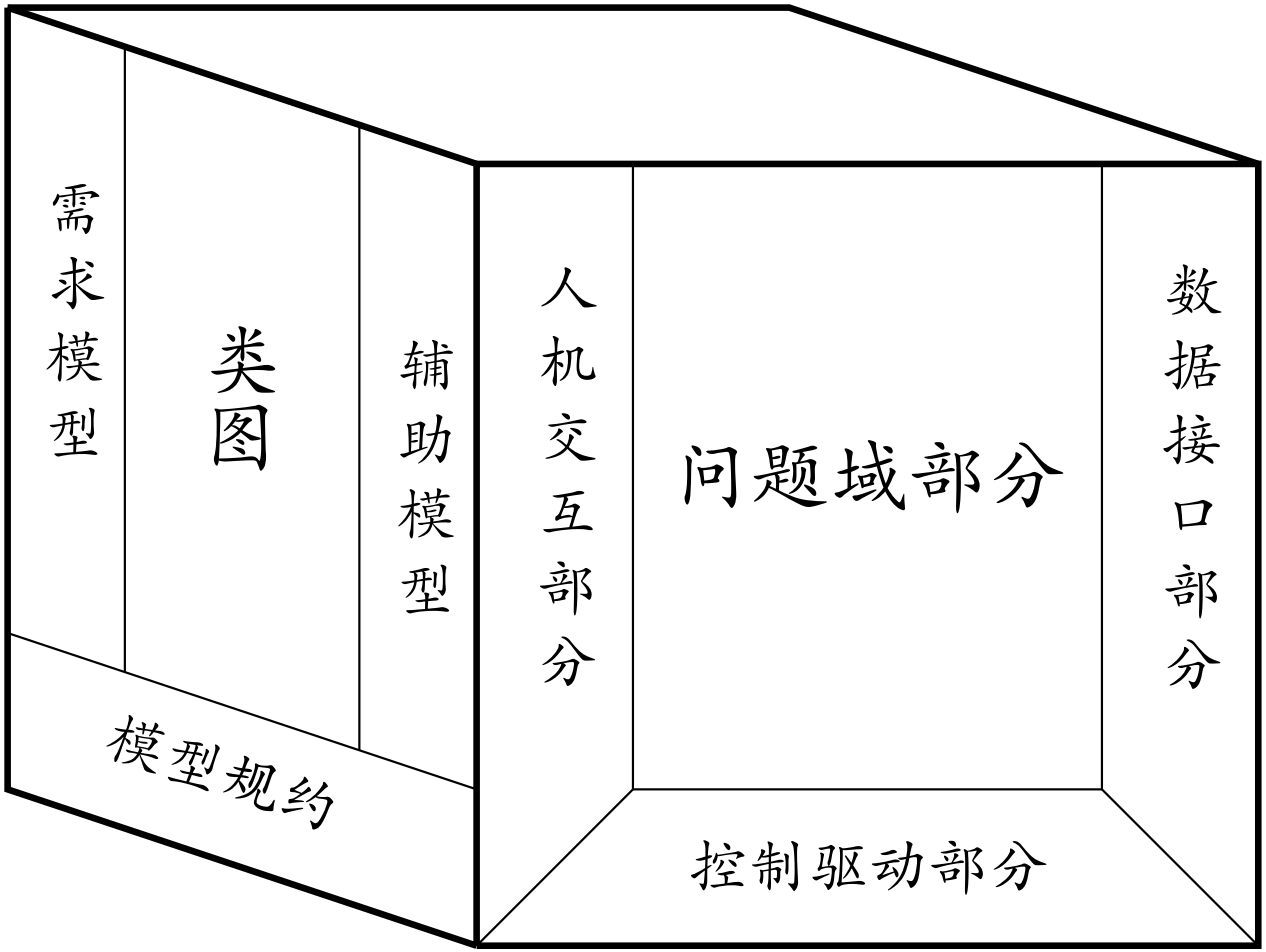
辅助模型—其他图

对类图起到辅助作用，提供更详细的建模信息，或者从不同的视角来描述系统。例如包图、顺序图、活动图等

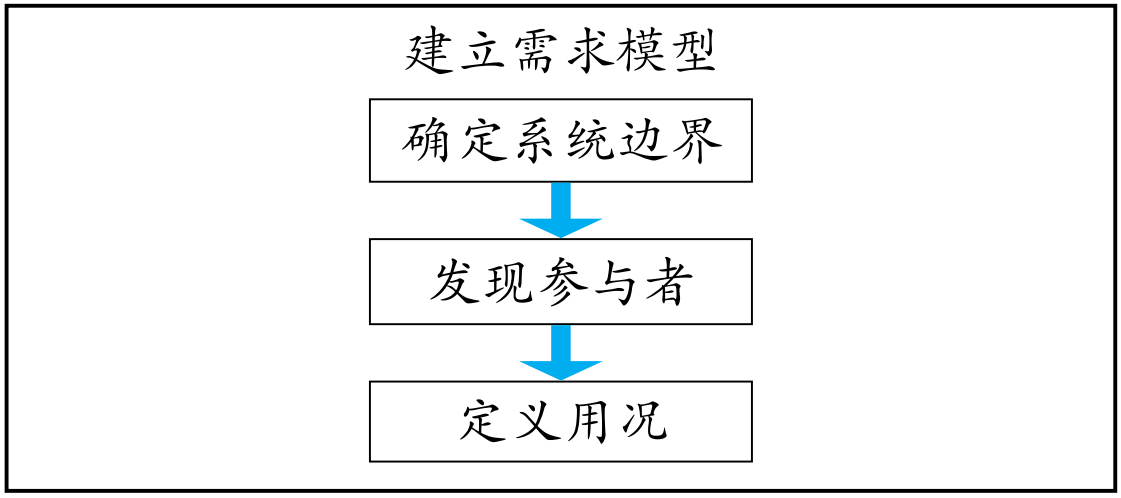
OOA 模型框架



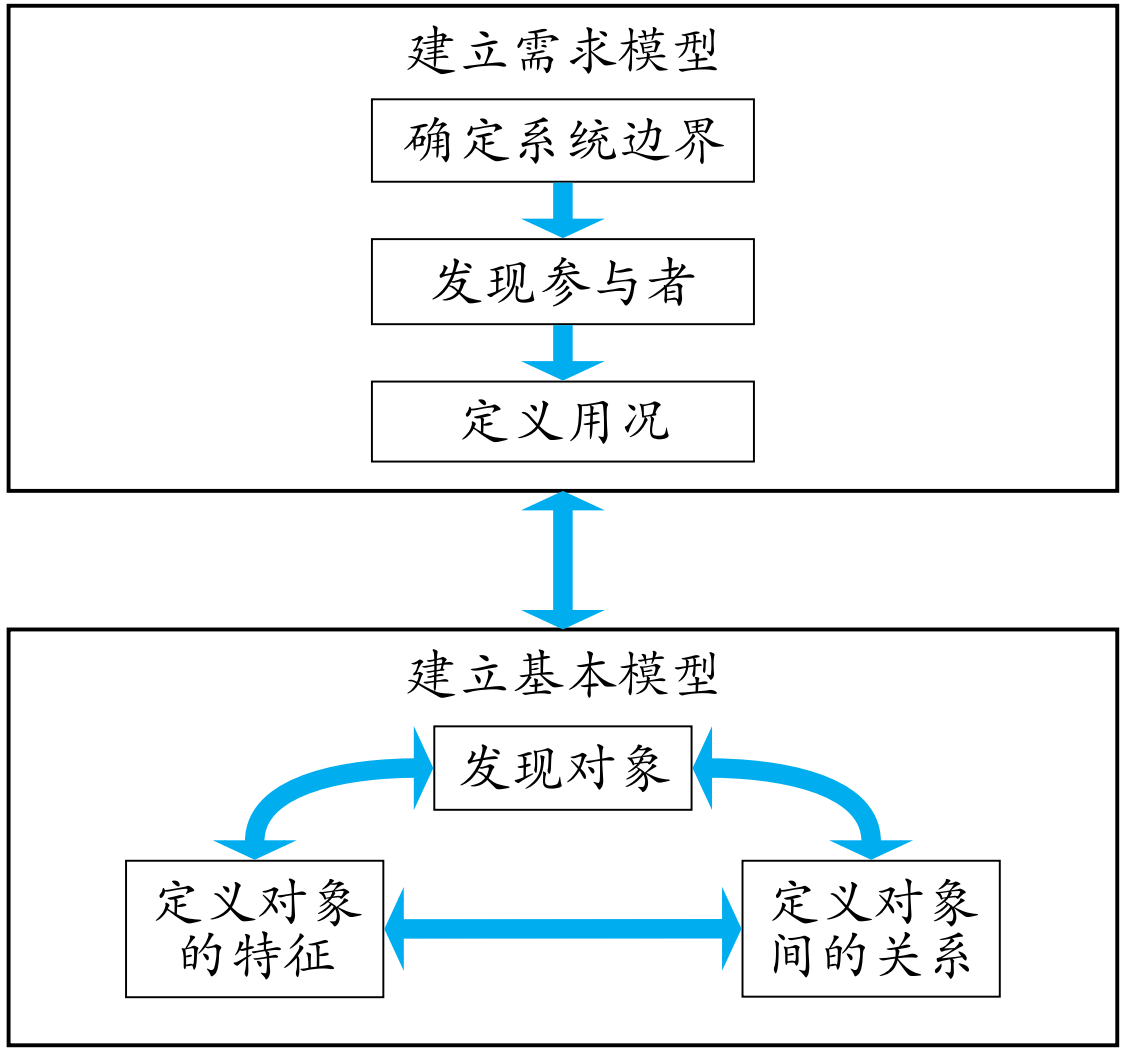
OOD 模型框架



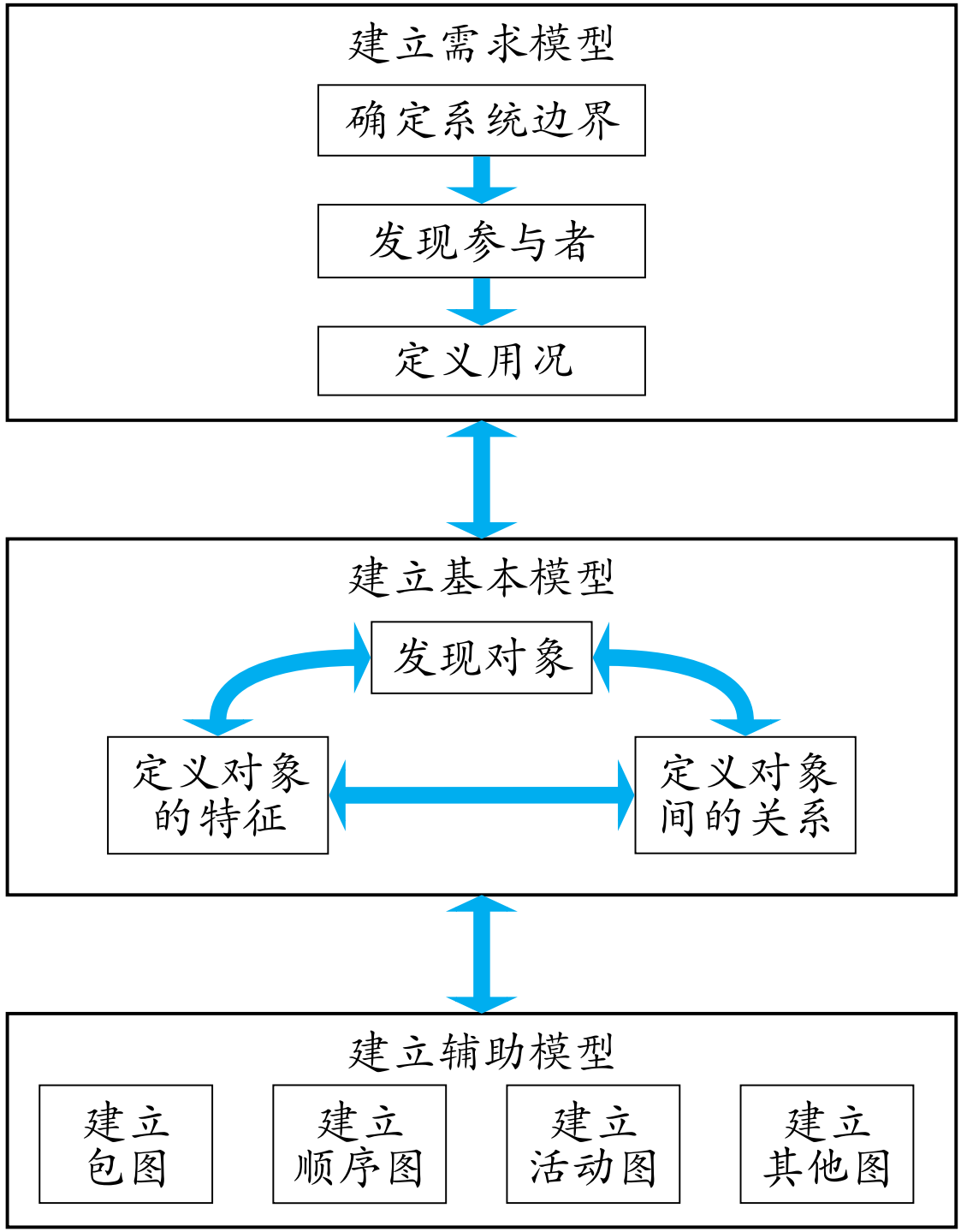
OOA 过程



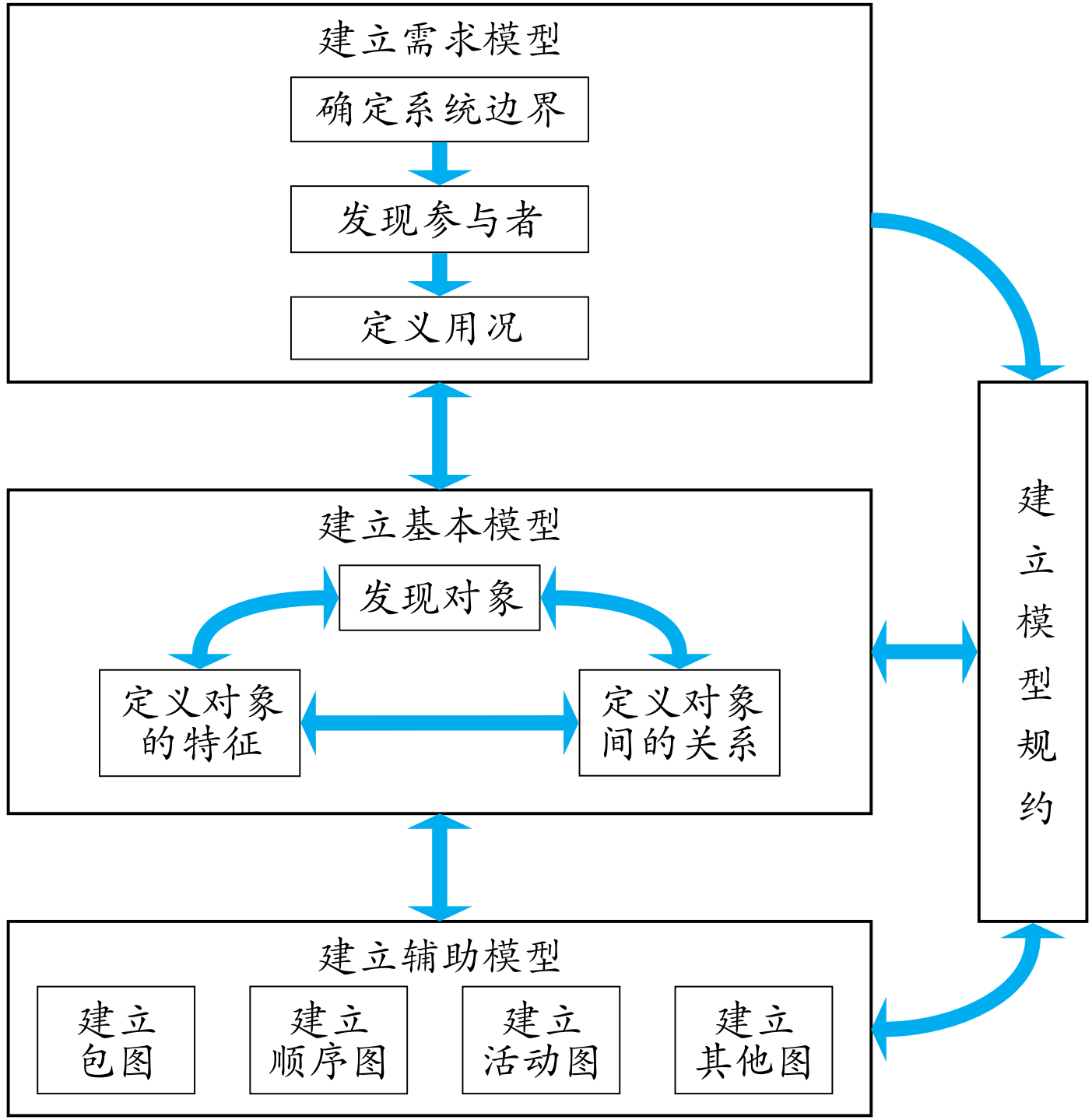
OOA 过程



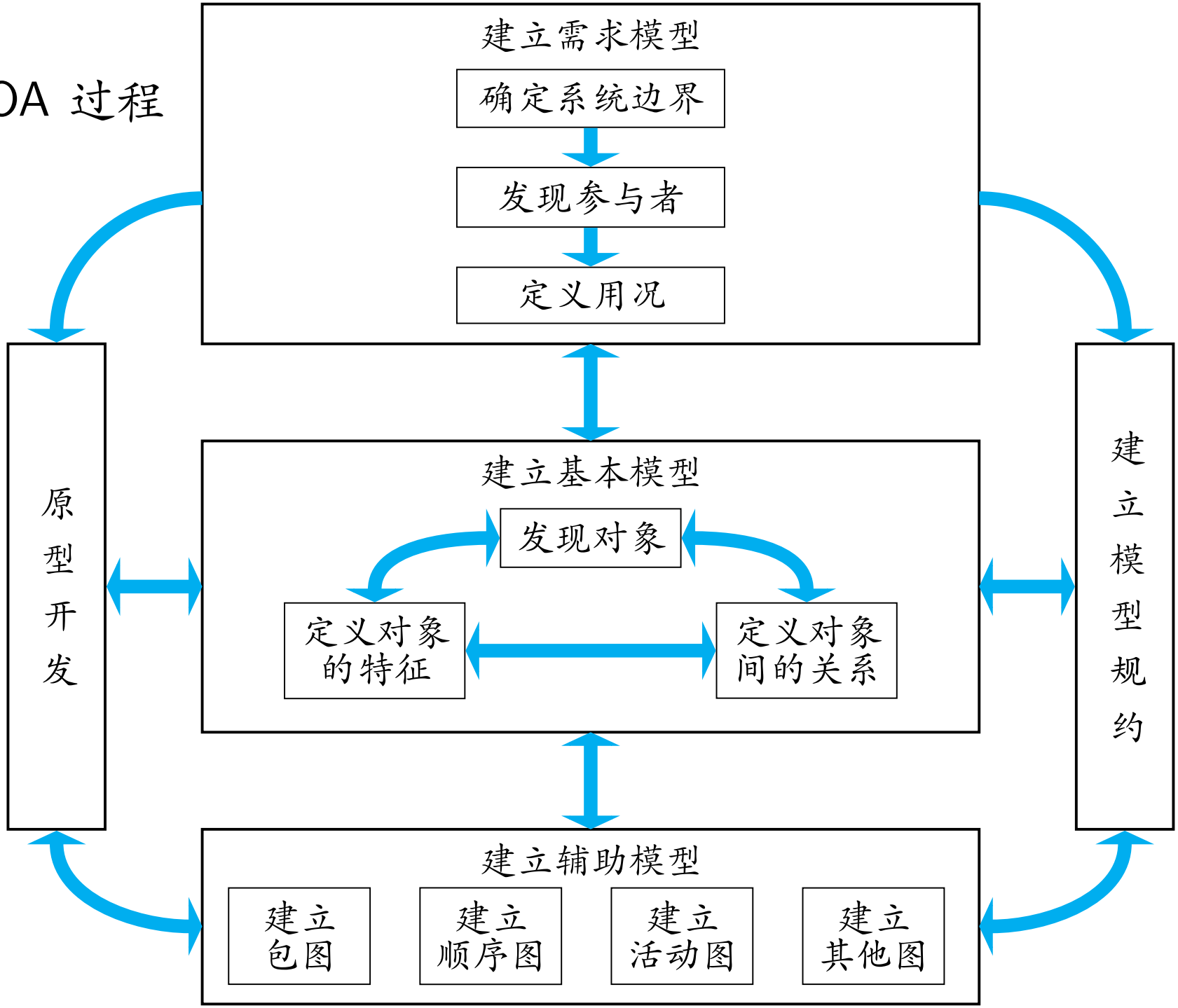
OOA 过程


建立需求模型
确定系统边界
发现参与者
定义用况
建立基本模型
发现对象
定义对象的特征
定义对象间的关系
建立辅助模型
建立包图
建立顺序图
建立活动图
建立其他图

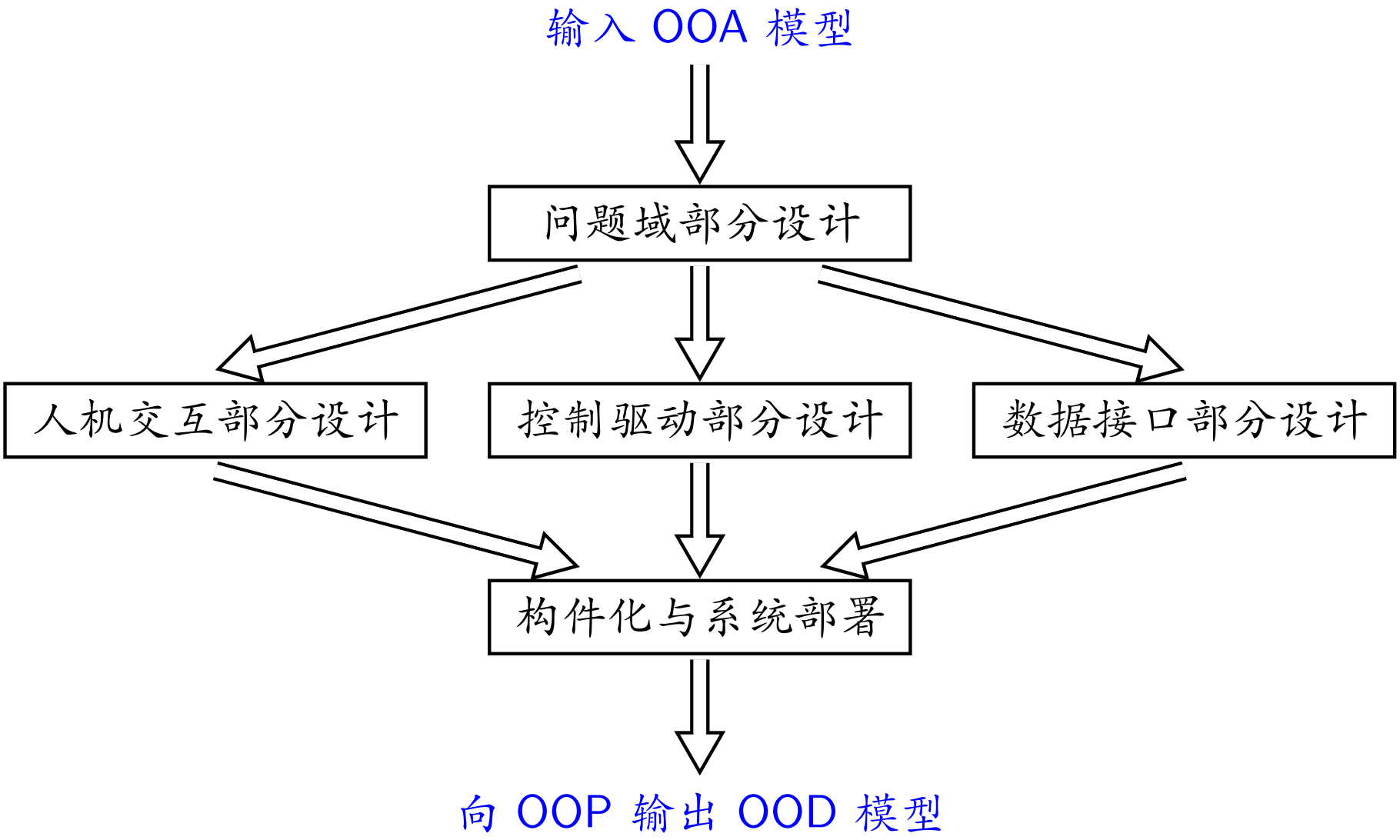
OOA 过程


建立需求模型
确定系统边界
发现参与者
定义用况
建立基本模型
发现对象
定义对象的特征
定义对象间的关系
建立辅助模型
建立包图
建立顺序图
建立活动图
建立其他图
建立模型规约

OOA 过程


建立需求模型
确定系统边界
发现参与者
定义用况
原型开发
建立基本模型
发现对象
定义对象的特征
定义对象间的关系
建立模型规约
建立辅助模型
建立包图
建立顺序图
建立活动图
建立其他图

OOD 过程


输入 OOA 模型
问题域部分设计
人机交互部分设计
控制驱动部分设计
数据接口部分设计
构件化与系统部署
向 OOP 输出 OOD 模型

OOA 与 OOD 的关系

一致的概念与表示法、不同的抽象层次

OOA

研究问题域和用户需求，运用面向对象的观点发现问题域中与系统责任有关的对象，以及对象的特征和相互关系。目标是建立一个直接映射问题域，符合用户需求的 OOA 模型

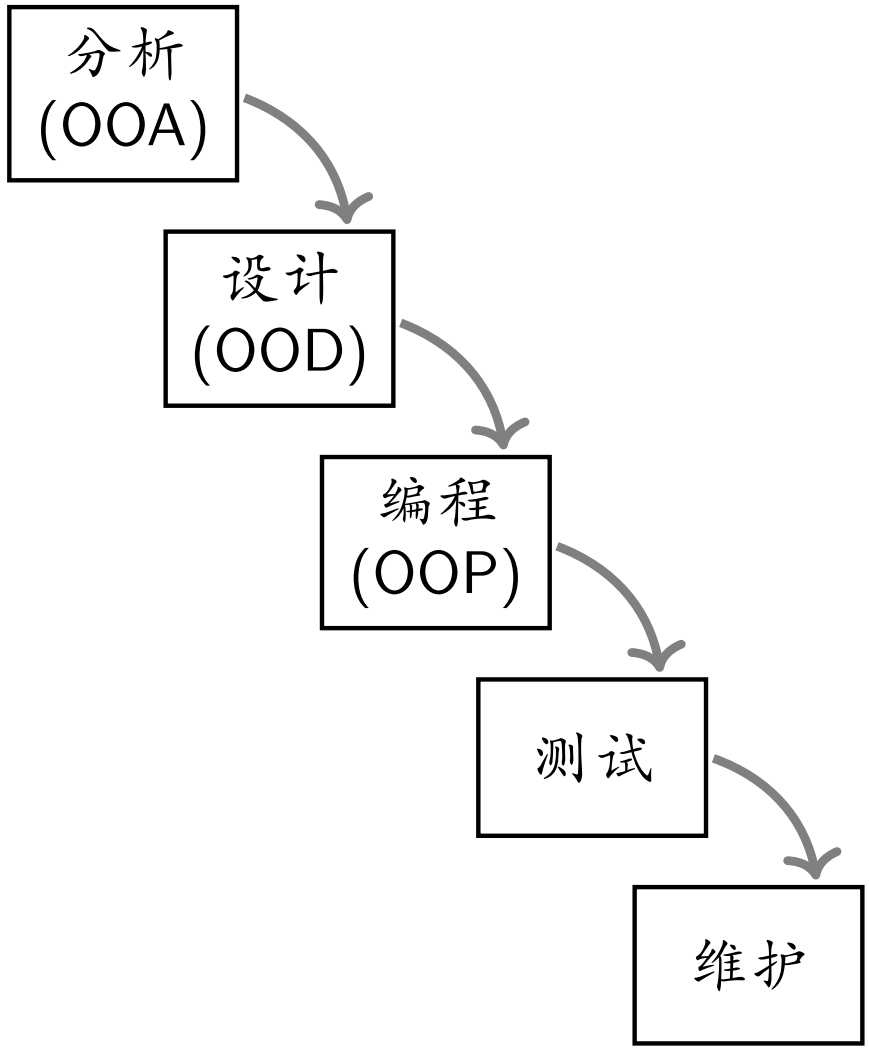
OOD

在 OOA 模型基础上，针对选定的实现平台进行系统设计，按照实现的要求进行具体的设计，目标是产生一个能够在选定的软硬件平台上实现的 OOD 模型

OOA 和 OOD 适应各种软件生存周期模型

瀑布模型

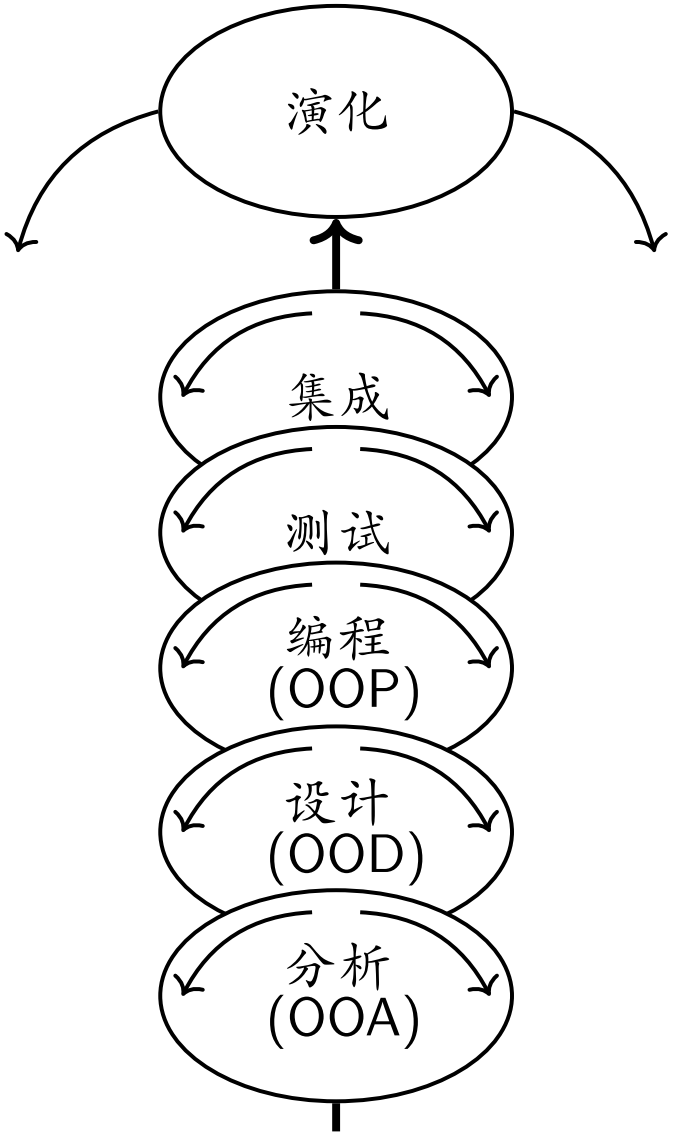
强调严格的阶段划分和前后次序
先做完 OOA 再进行 OOD



OOA 和 OOD 适应各种软件生存周期模型

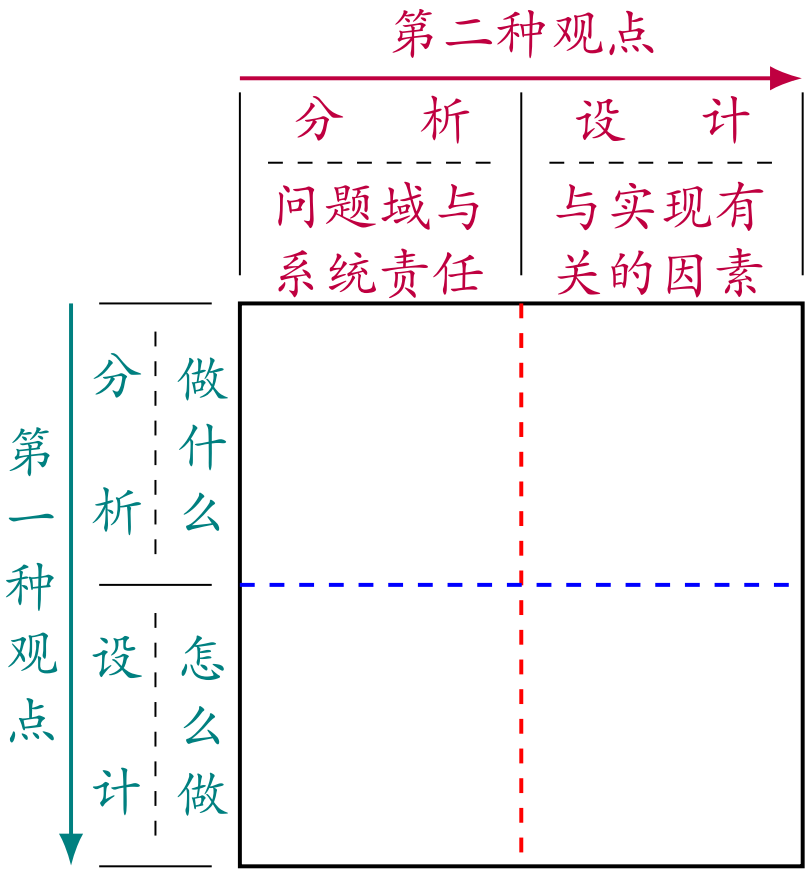
喷泉模型

各个阶段之间没有严格的界限，
其活动可以交叠和回溯
有些工作既可在 OOA 中进行，
也可在 OOD 中进行



OOA 和 OOD 分工的两种观点

关键问题：对象的特征细节在分析还是设计时定义？

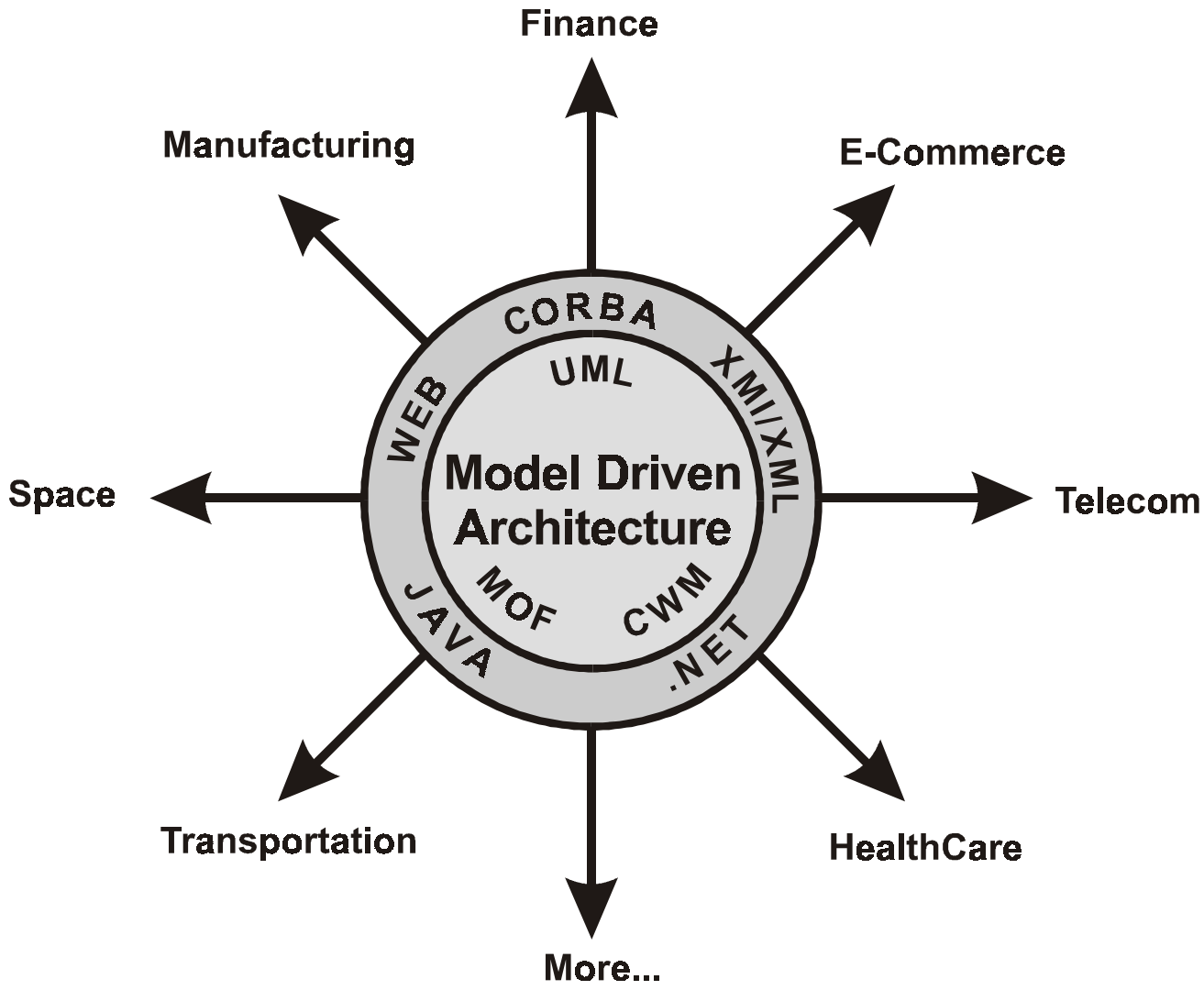


第二种观点

对与问题域和系统责任紧密相关的对象细节，分析人员比设计人员更有发言权

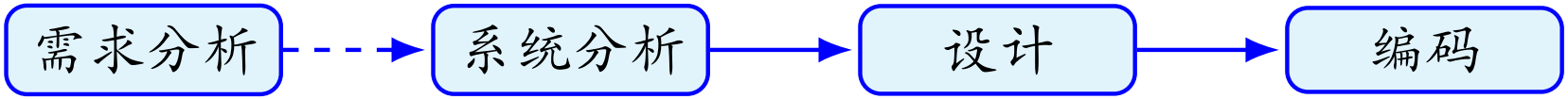
OOA 阶段建立 PIM，OOD 阶段建立 PSM

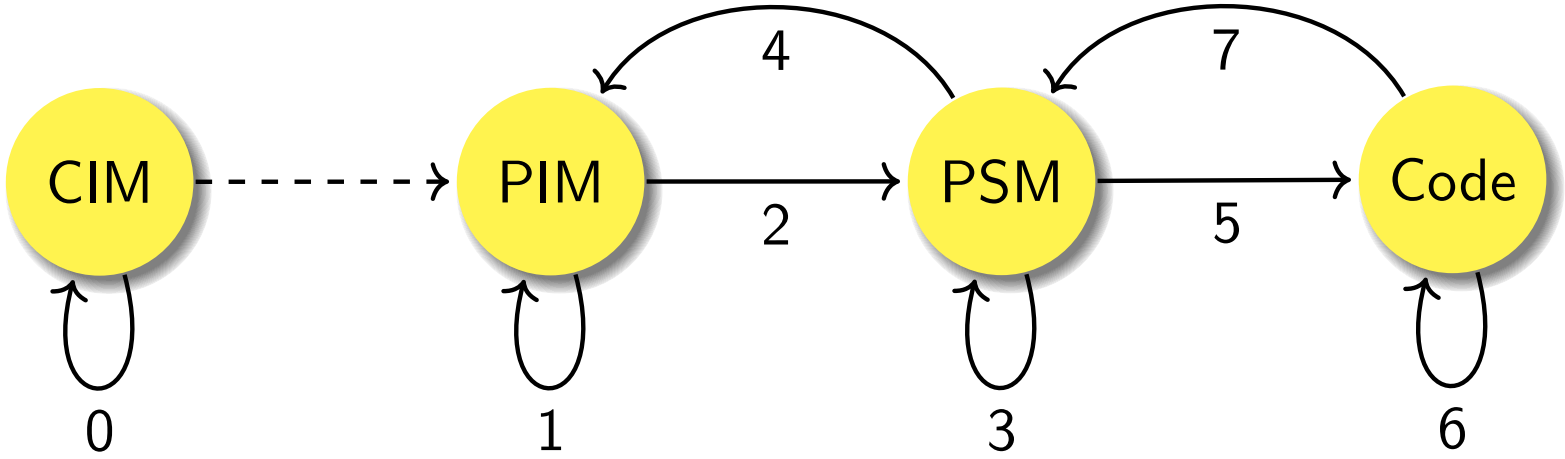
从 MDA 看 OOA 与 OOD



模型转换: model transformation

Model once, generate everywhere!





OOA 和 OOD 的 MDA 视图

OOA 可得到一种平台无关的 PIM 模型

OOD 可得到一种平台相关的 PSM 模型