基于面向对象的架构设计
Architectural Design

Yang YI, Ph.D

issyy@mail.sysu.edu.cn

Architectural Design

Agenda

- Objectives(本章教学目标)
- Context in the RUP(在整个设计流程中的位置)
- Architectural Design steps(步骤)
- Exercises (实验任务)

Objectives: Architectural Design

- Explain the purpose of Architectural Design and where it is performed in the lifecycle.
  - (架构设计的目标及其在软件生命周期中的位置)
- Describe a representative architectural pattern and set of analysis mechanisms, and how they affect the architecture.(一些相关的概念)
- Describe the rationale and considerations that support the architectural decisions.(架构设计中常见的错误)
- Show how to read and interpret the results of Architectural Design:(步骤)
  - Architectural layers and their relationships
  - Key abstractions

Architectural Design

Agenda

- Objectives
- Context in the RUP
- Architectural Design steps
- Exercises

What is Software Architecture?

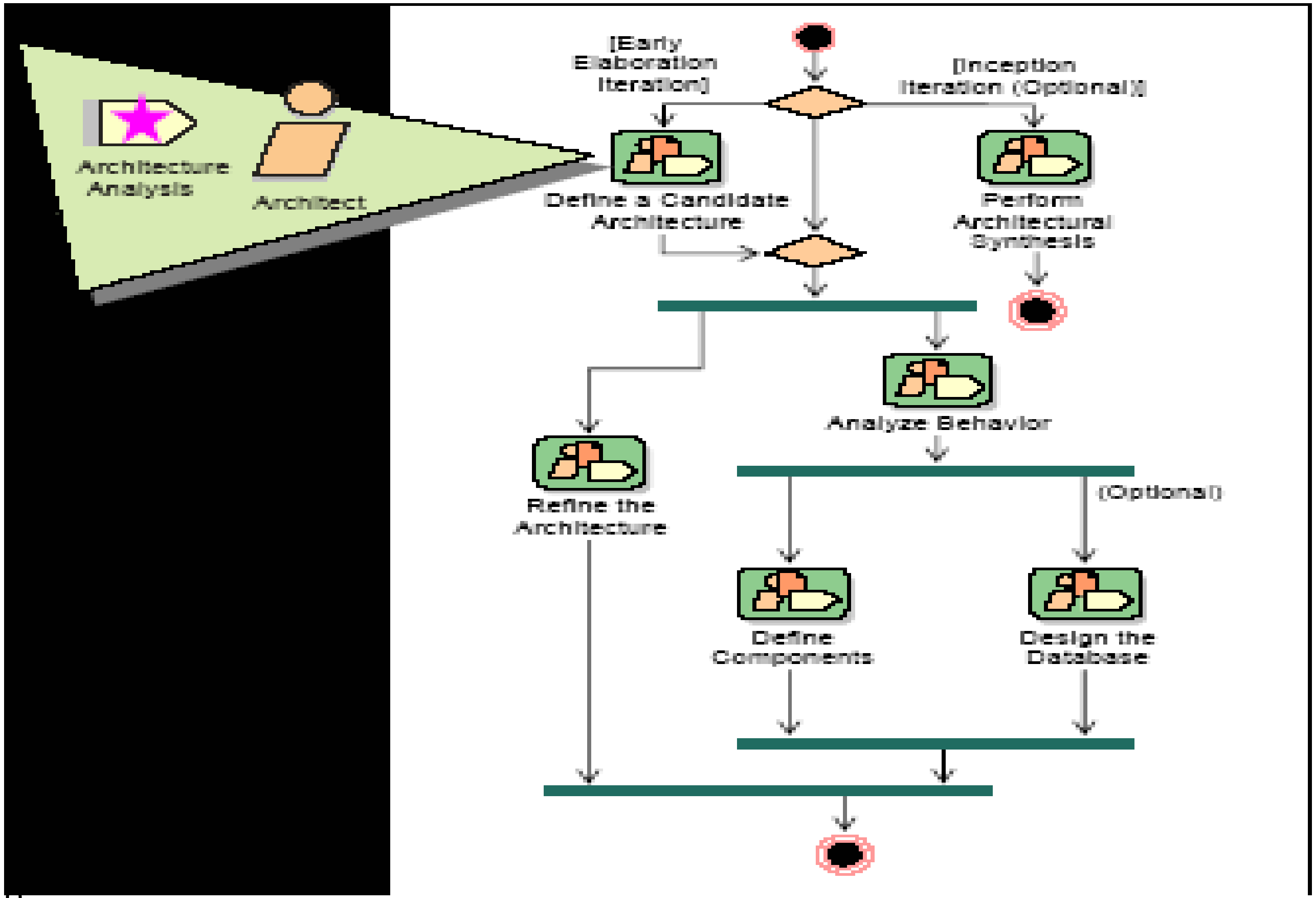
- The process of defining a solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability.
- It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

- Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman derived and refined a definition of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996).
- Their definition is:

- "Software architecture encompasses the set of significant decisions about the organization of a software system, including
  - the selection of the structural elements and their interfaces by which the system is composed;
  - behavior as specified in collaboration among those elements;
  - composition of these structural and behavioral elements into larger subsystems;
  - an architectural style that guides this organization.

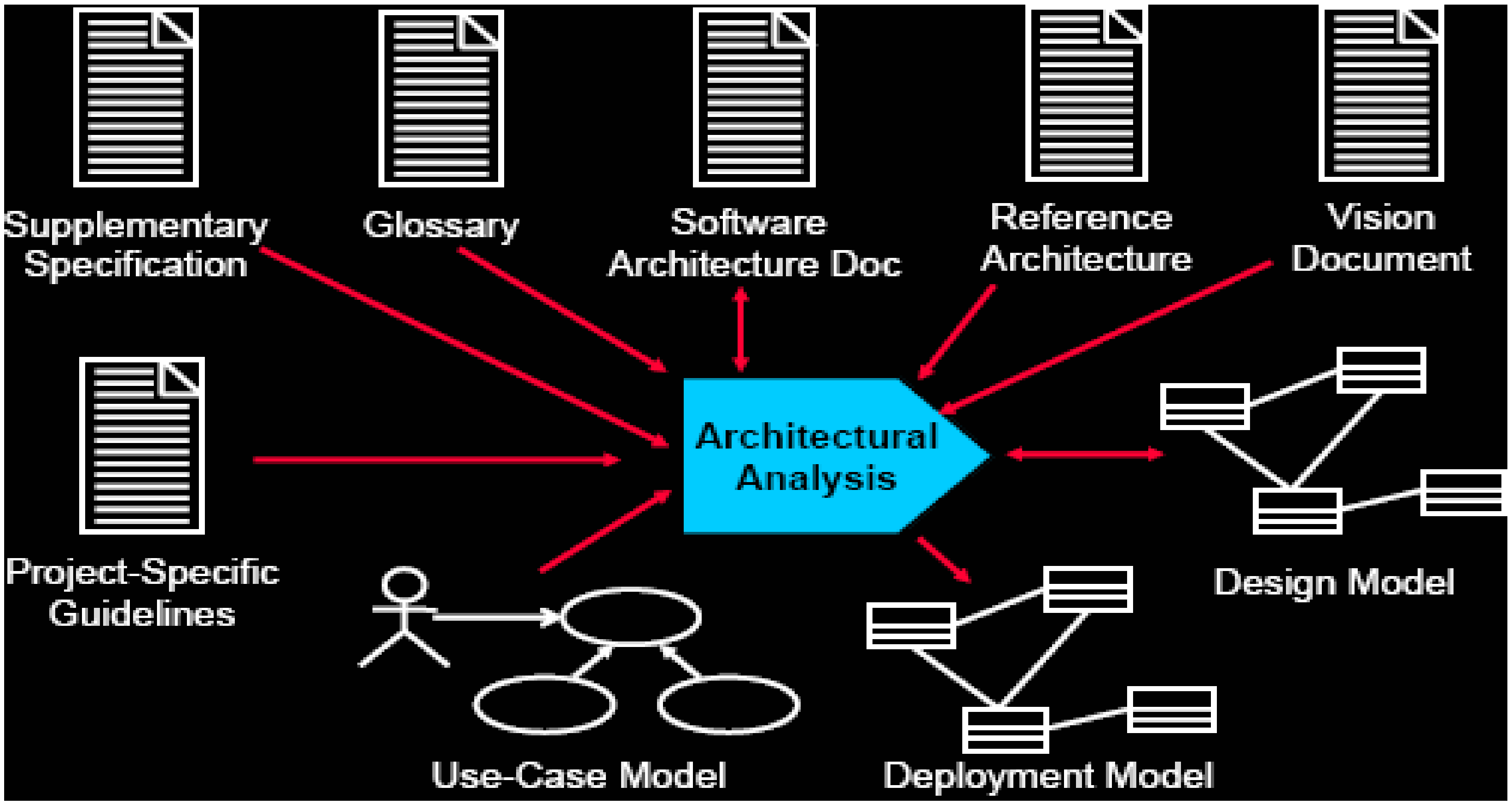
- Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns

Architectural Design in Context of System Design


Architecture Analysis
Architect
[Early Elaboration Iteration]
[Inception Iteration (Optional)]
Define a Candidate Architecture
Perform Architectural Synthesis
Analyze Behavior
Refine the Architecture
(Optional)
Define Components
Design the Database

Architectural Design Overview

架构设计涉及的制品



Architectural Design

Agenda

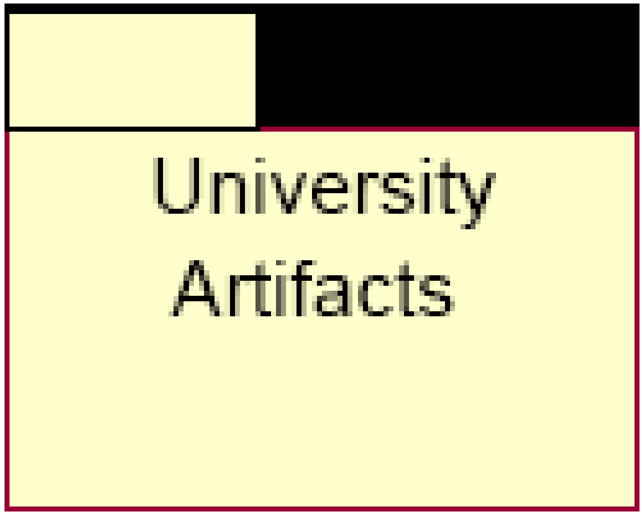
- Objectives
- Context in the RUP
- Architectural Design steps
- Exercises

Architectural Design Steps（步骤）

1. Key Concepts
2. Define the High-Level Organization of Subsystems
3. Identify Key Abstractions
4. Checkpoints

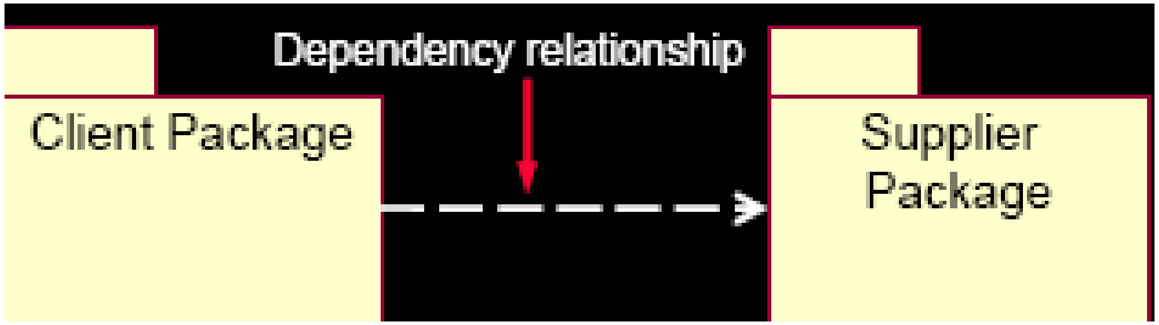
Review: What Is a Package?

- **A package is a general-purpose mechanism for organizing elements into groups.**
- **It is a model element that can contain other model elements.**
- **A package can be used**
  - **To organize the model under development.**
  - **As a unit of configuration management.**

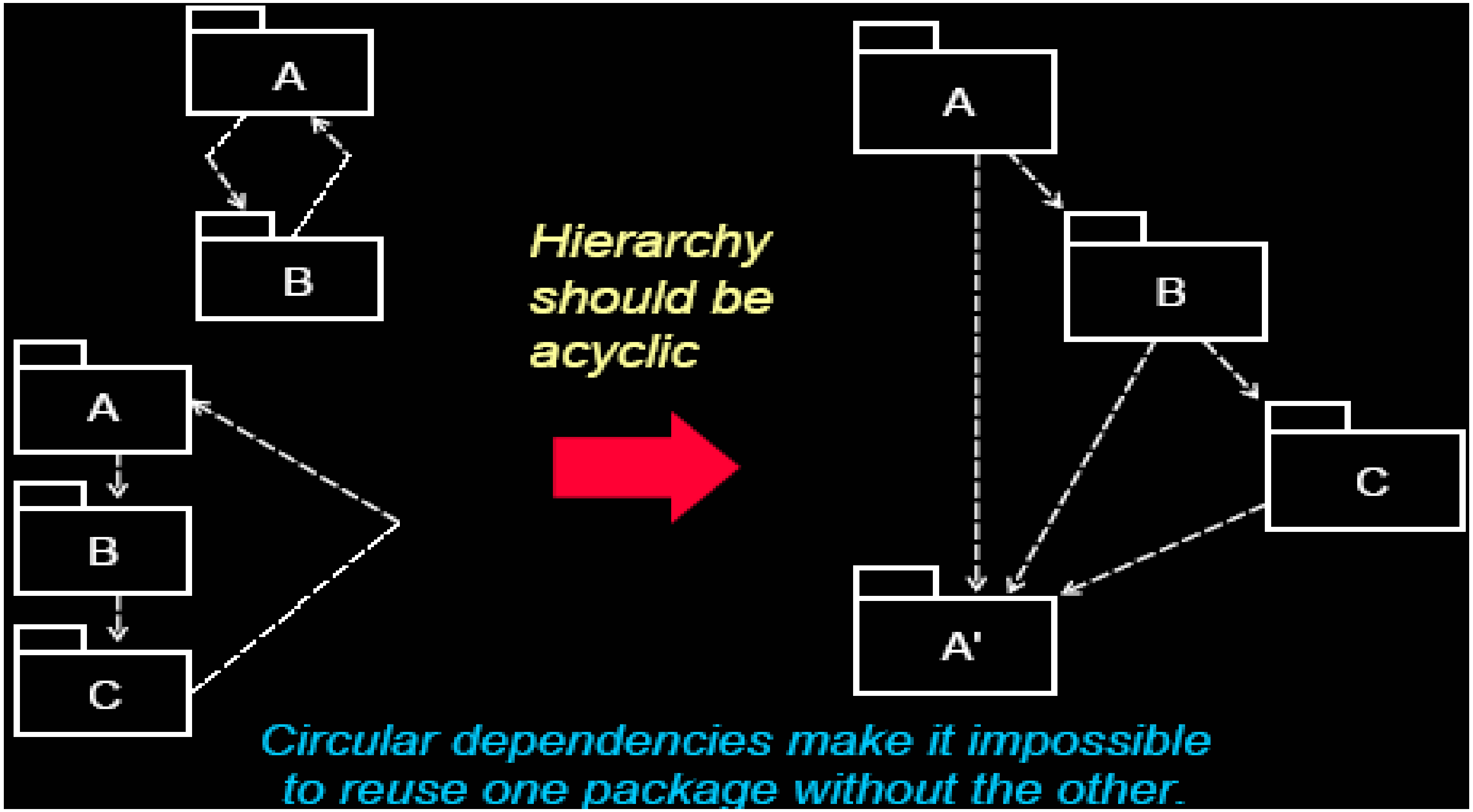


Package Relationships: Dependency (依赖关系)

- **Packages can be related to one another using a dependency relationship.**
- **Dependency Implications**
  - **Changes to the Supplier package may affect the Client package.**
  - **The Client package cannot be reused independently because it depends on the Supplier package.**



Avoiding Circular Dependencies


A
B
A
B
C
Hierarchy should be acyclic
A
B
C
A'
Circular dependencies make it impossible to reuse one package without the other.

Architectural Design Steps

1. Key Concepts
2. Define the High-Level Organization of Subsystems
3. Identify Key Abstractions
4. Checkpoints

Patterns and Frameworks

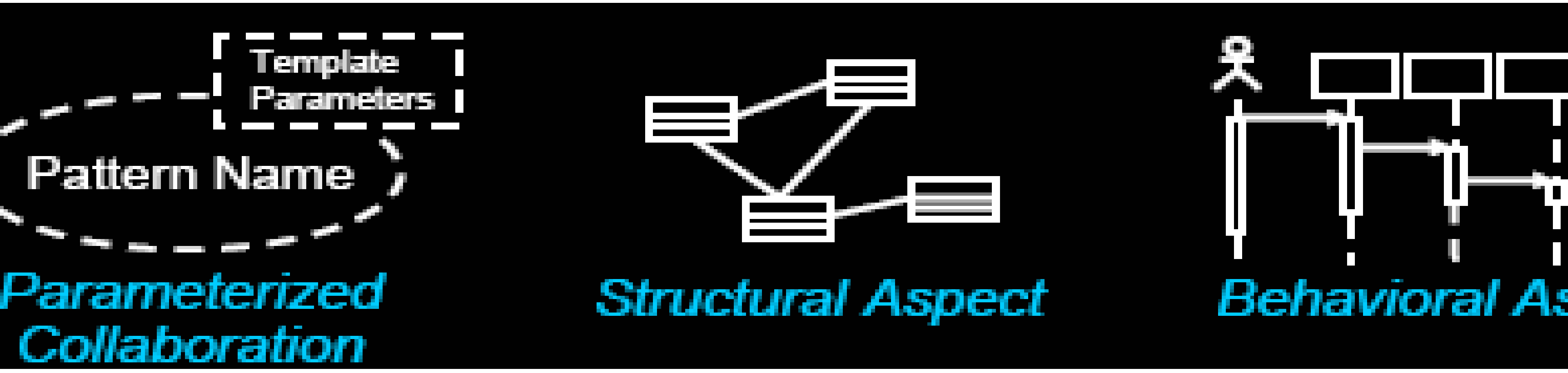
- **Pattern（模式）**
  - Provides a common solution to a common problem in a context
  - Analysis/Design pattern
  - Provides a solution to a narrowly-scoped technical problem
  - Provides a fragment of a solution, or a piece of the puzzle
- **Framework（框架）**
  - Defines the general approach to solving the problem
  - Provides a skeletal solution, whose details may be analysis/Design patterns

Architecture/Pattern/Framework

名词	中文释义	功能	范围	是否有代码实现？
Architecture	架构	说明方案	全局	无
Pattern	模式		局部	无
Framework	框架		局部	有部分

What Is a Design Pattern?

- **A design pattern is a solution to a common design problem.**
  - **Describes a common design problem**
  - **Describes the solution to the problem**
  - **Discusses the results and trade-offs of applying the pattern**
- **Design patterns provide the capability to reuse successful designs.**



What Is an Architectural Pattern?

- An architectural pattern expresses a fundamental structural organization schema for software systems.
- It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them
  - – *Buschman et al, "Pattern-Oriented Software Architecture — A System of Patterns"*

➢ Layers

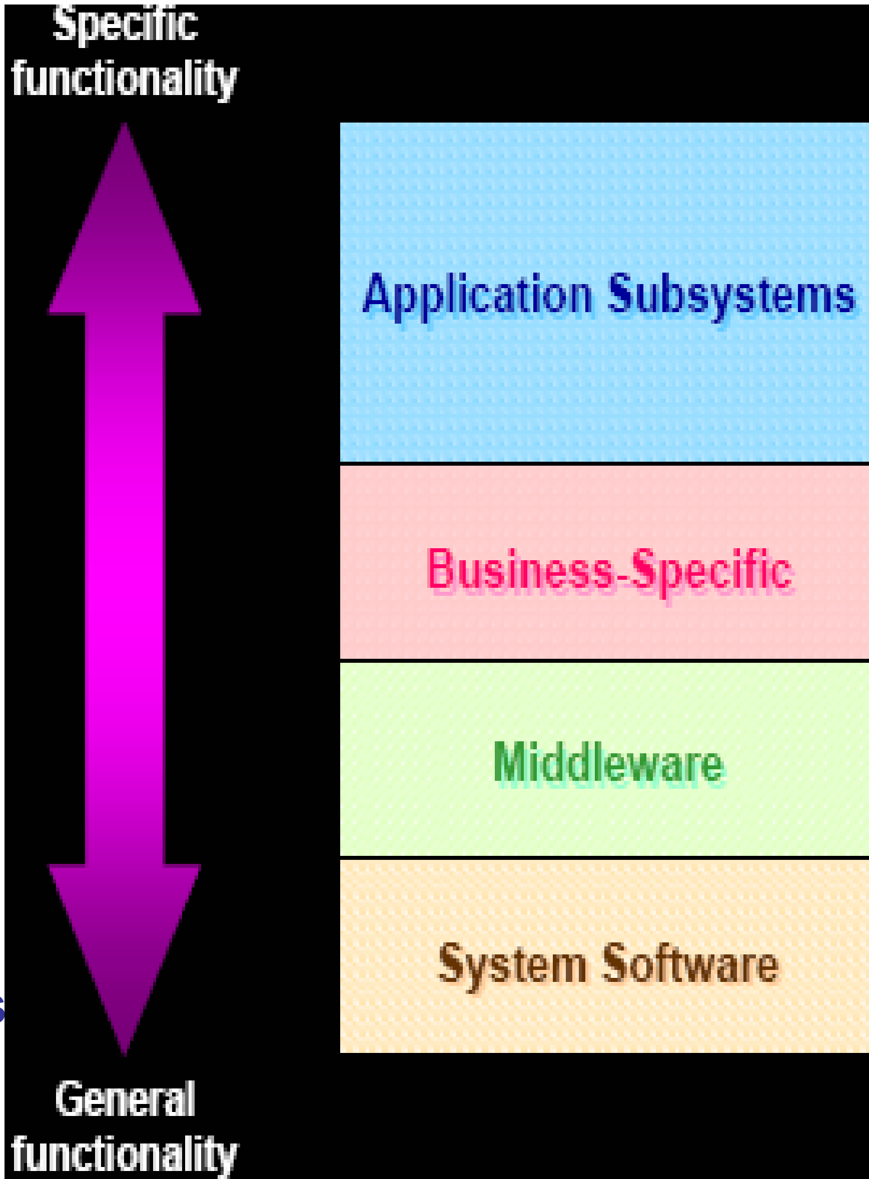
➢ Model-View-Controller (M-V-C)

➢ Pipes and filters

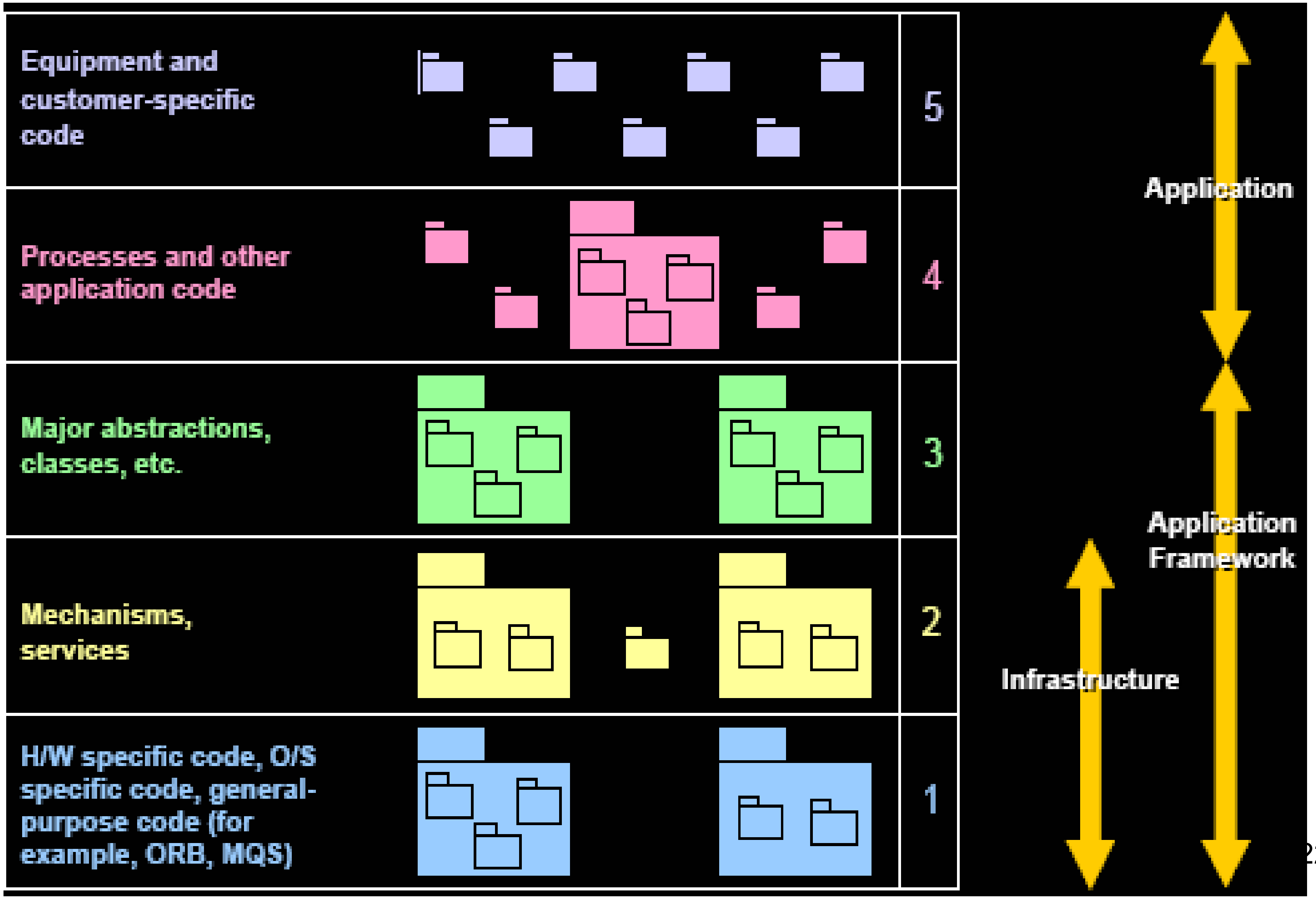
➢ Blackboard

Case: Typical Layering Approach

- **Distinct application subsystems that make up an application — contains the value adding software developed by the organization.**
- **Business specific — contains a number of reusable subsystems specific to the type of business.**
- **Middleware — offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.**
- **System software — contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers, and so on.**



Architectural Pattern: Layers



Equipment and customer-specific code
5
Processes and other application code
4
Major abstractions, classes, etc.
3
Mechanisms, services
2
H/W specific code, O/S specific code, general-purpose code (for example, ORB, MQS)
1
Application
Application Framework
Infrastructure

Layering Considerations

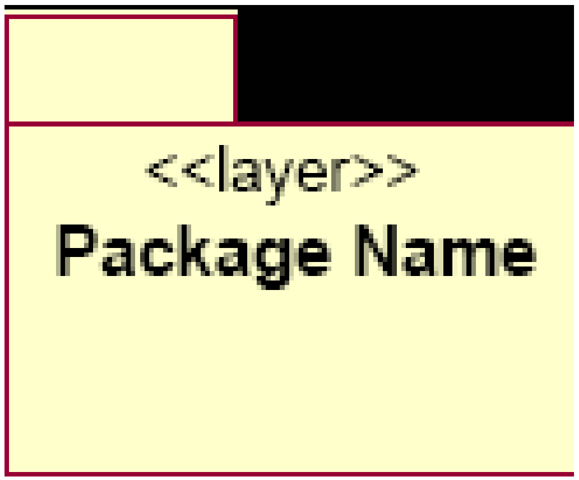
- Level of abstraction （抽象级别）（每层的抽象度一致。从逻辑上，级别一样）
  - Group elements at the same level of abstraction
- Separation of concerns （关心分离）（每一层有每一层自己的关注点，例如，V关注与外部世界的交互，C关注服务）
  - Group like things together
  - Separate disparate things
  - Application vs. domain model elements
- Resiliency （弹性）（鲁棒性，适用范围比较广）
  - Loose coupling
  - Concentrate on encapsulating change
  - User interface, business rules, and retained data tend to have a high potential for change

Modeling Architectural Layers （层次架构的建模）

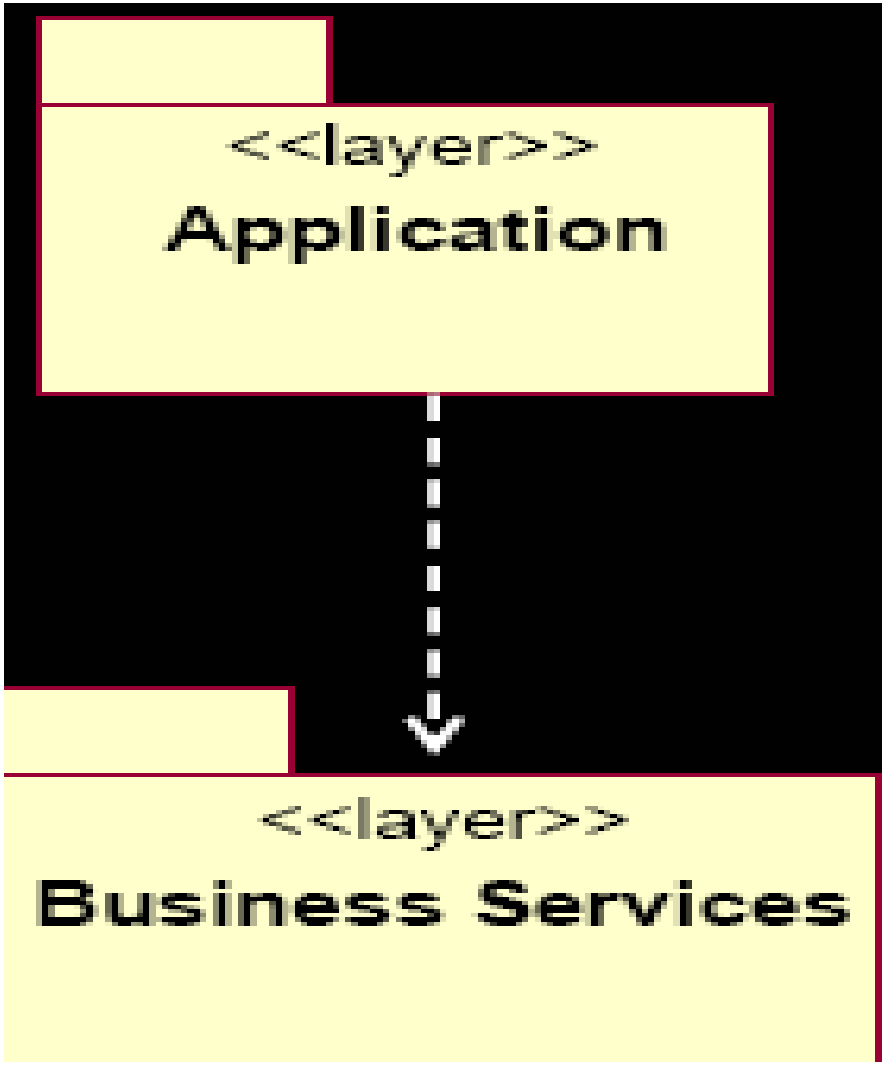
- **Architectural layers can be modeled using**

stereotyped packages.

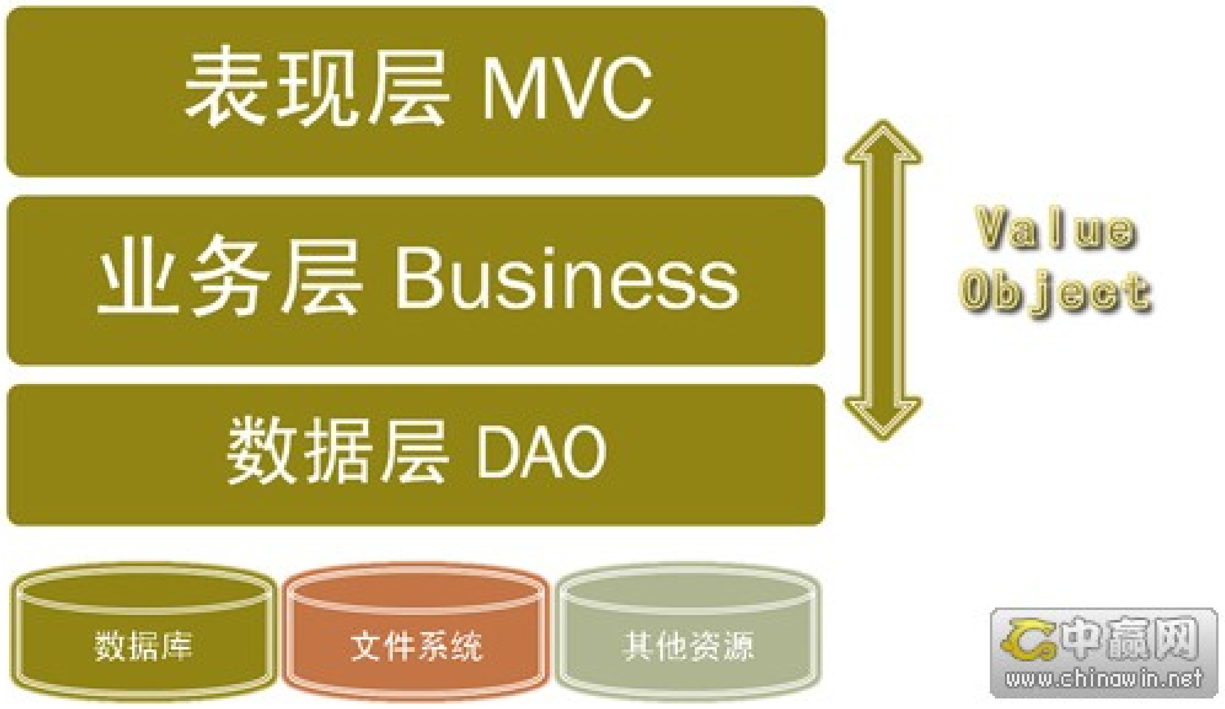
- **<<layer>> stereotype**



Example: High-Level Organization of the Model



MVC
Model-View-Controller



MVC
Model-View-Controller

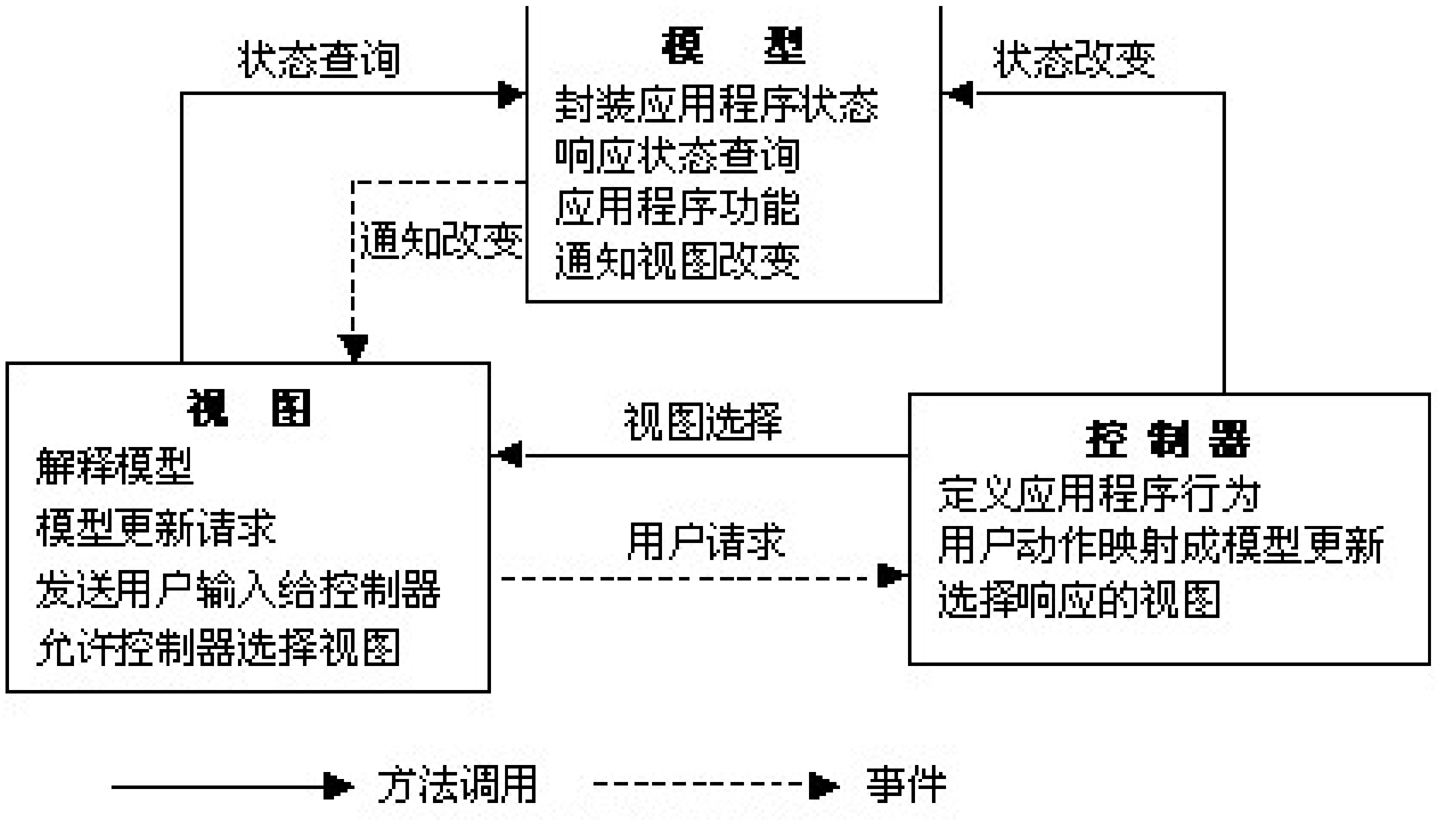
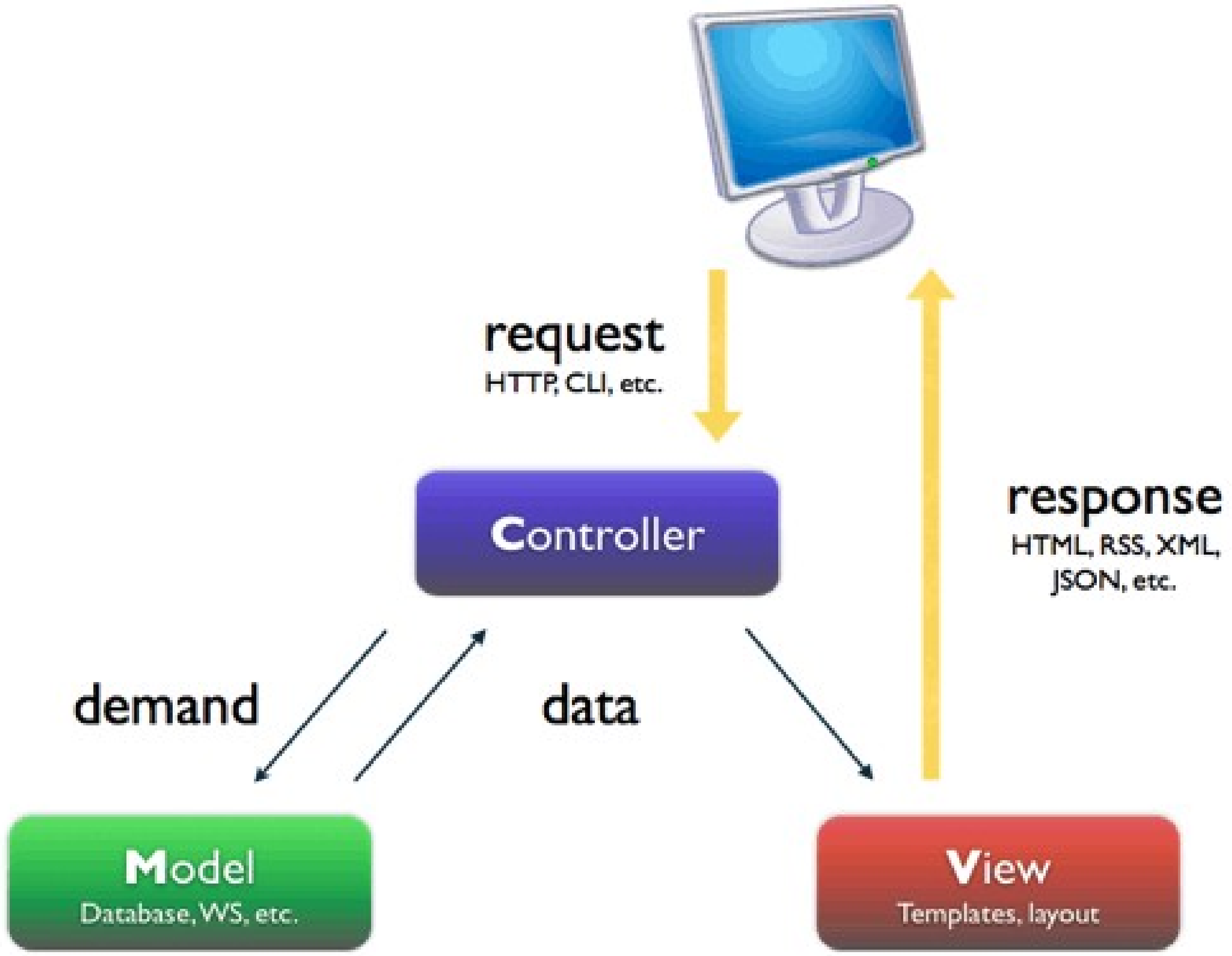
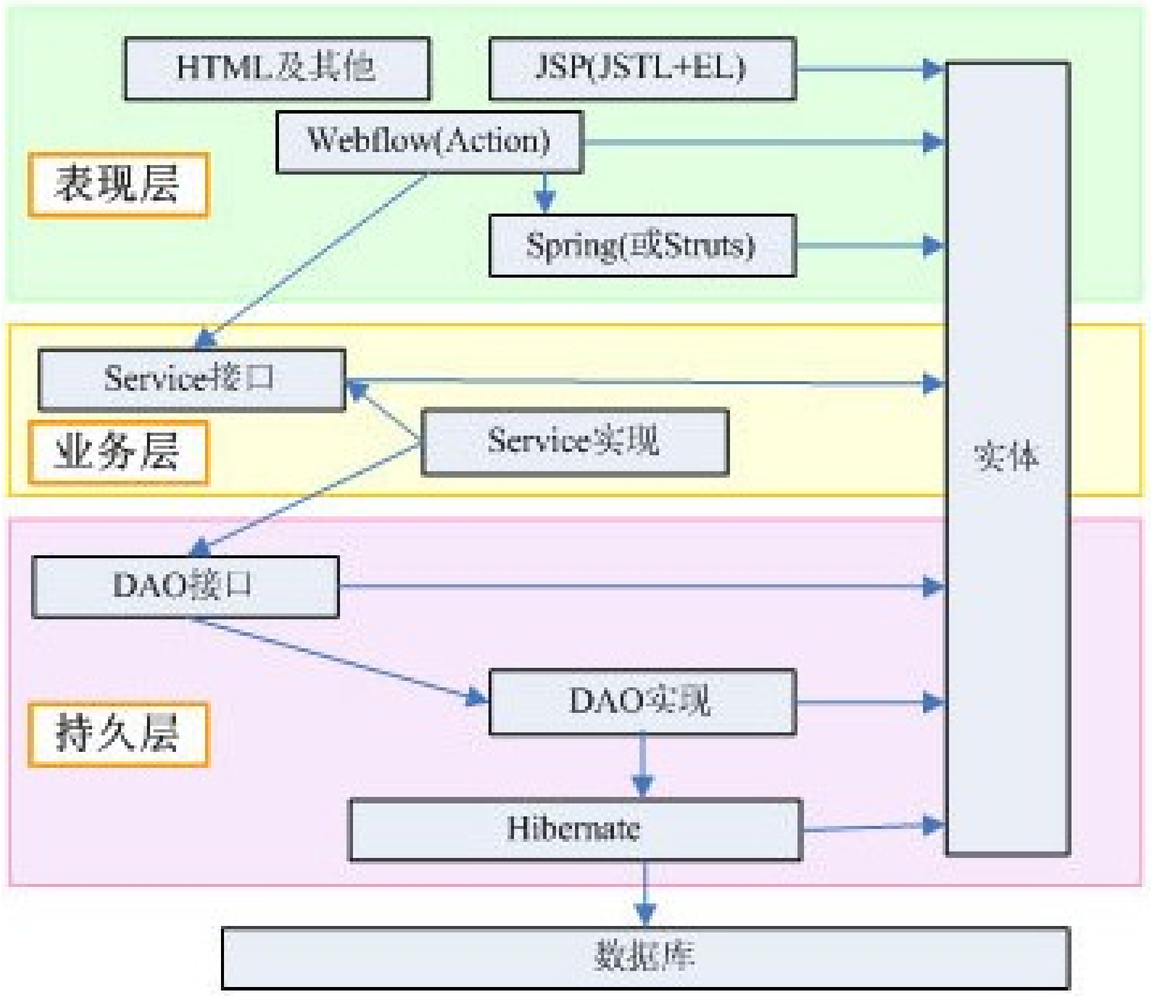


图 1 MVC 組件类型的关系和功能

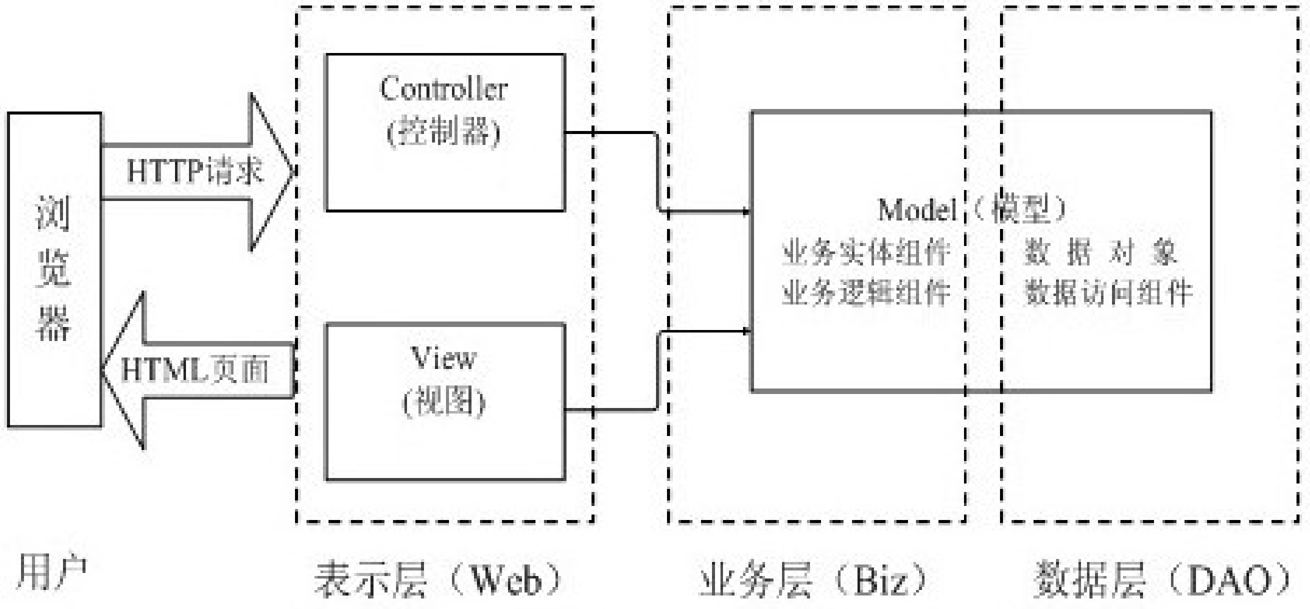
MVC
Model-View-Controller

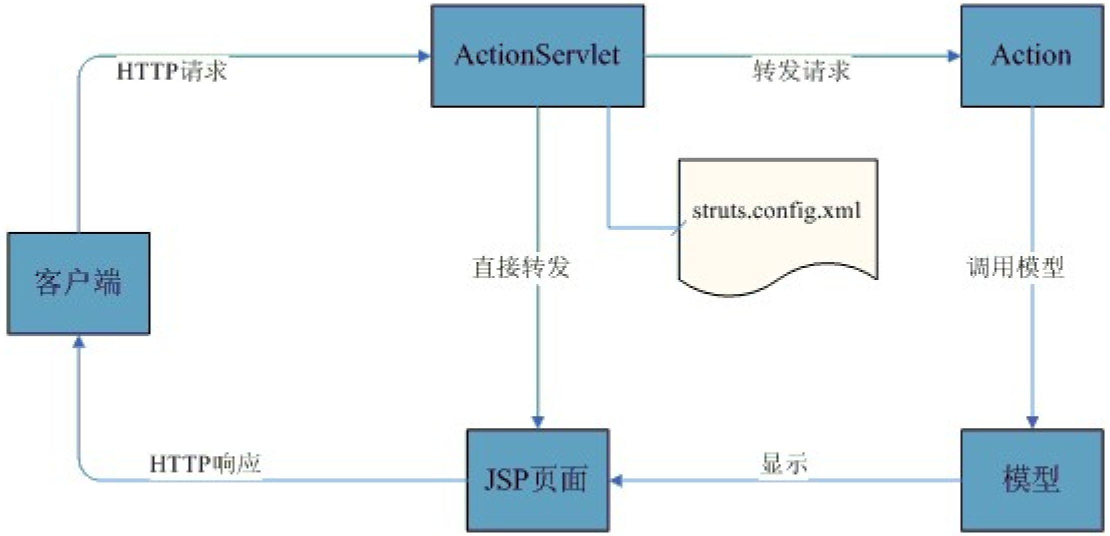


MVC
Model-View-Controller

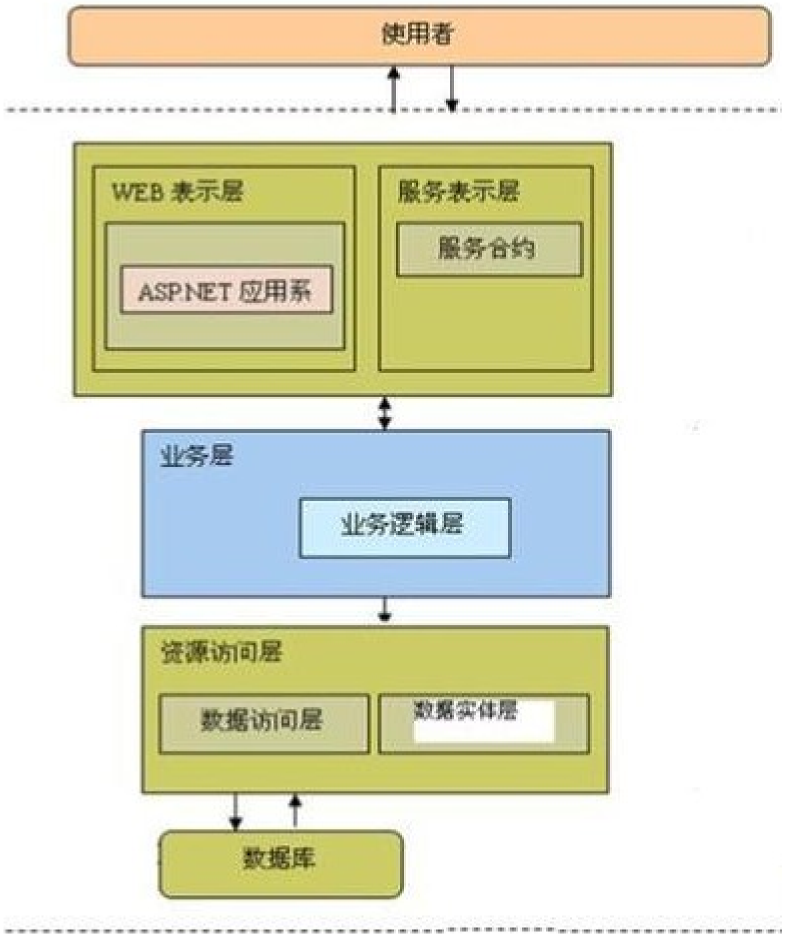


MVC
Model-View-Controller

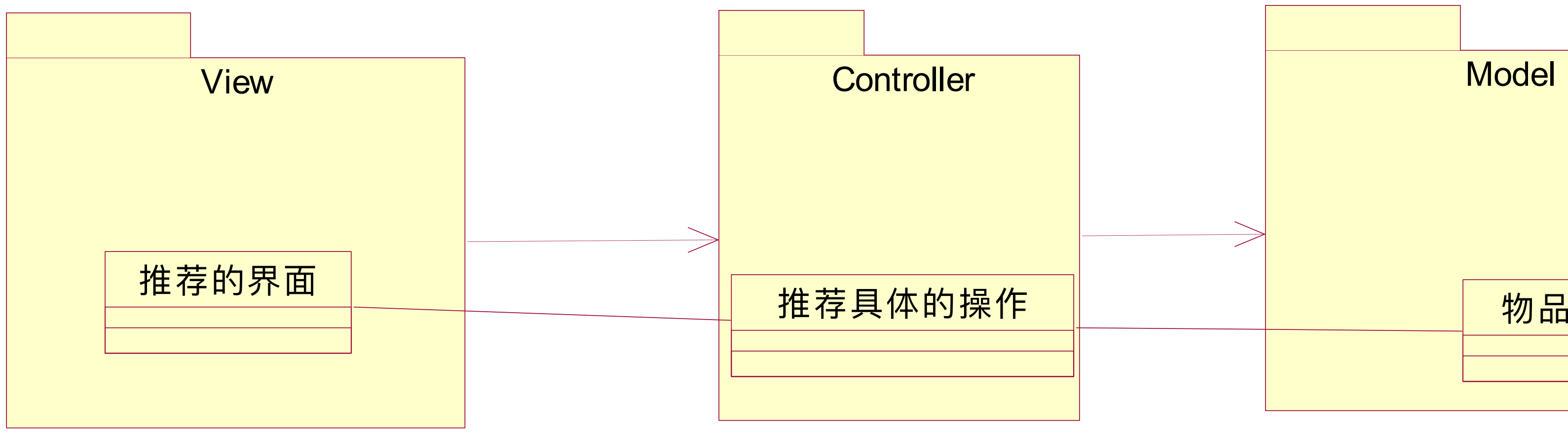




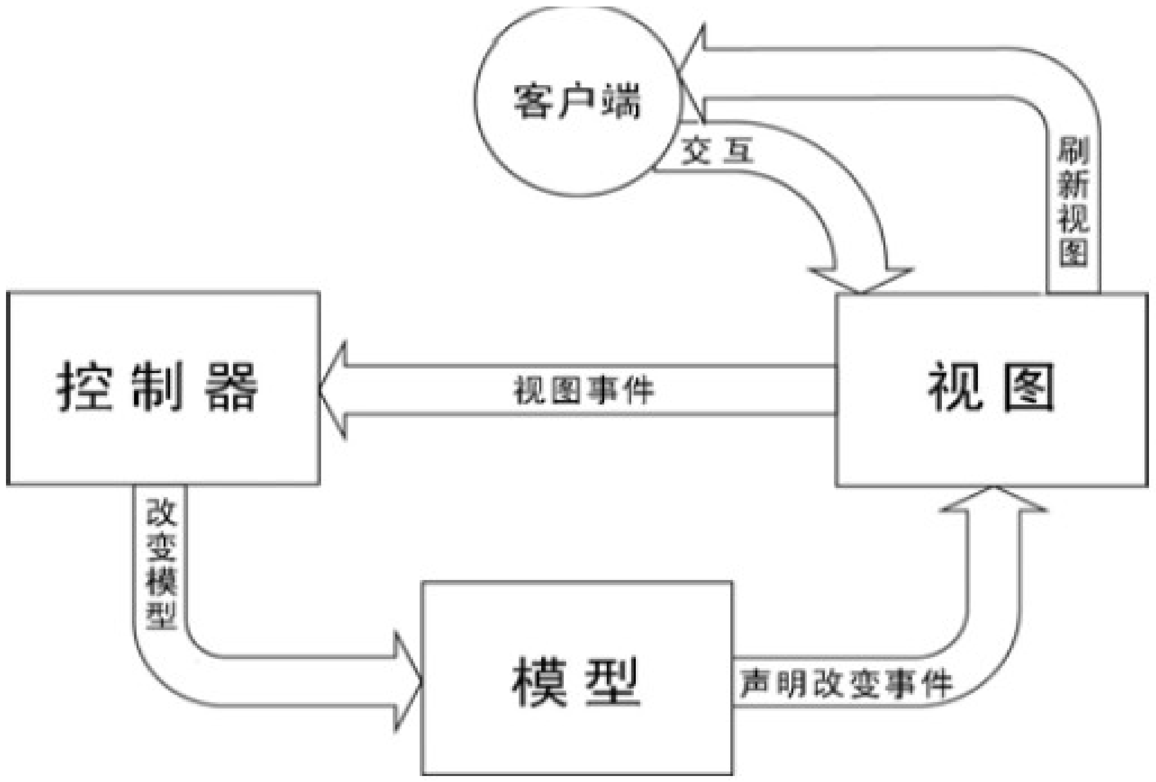
ActionServlet
Action
HTTP请求
转发请求
客户端
struts.config.xml
直接转发
调用模型
HTTP响应
JSP页面
显示
模型



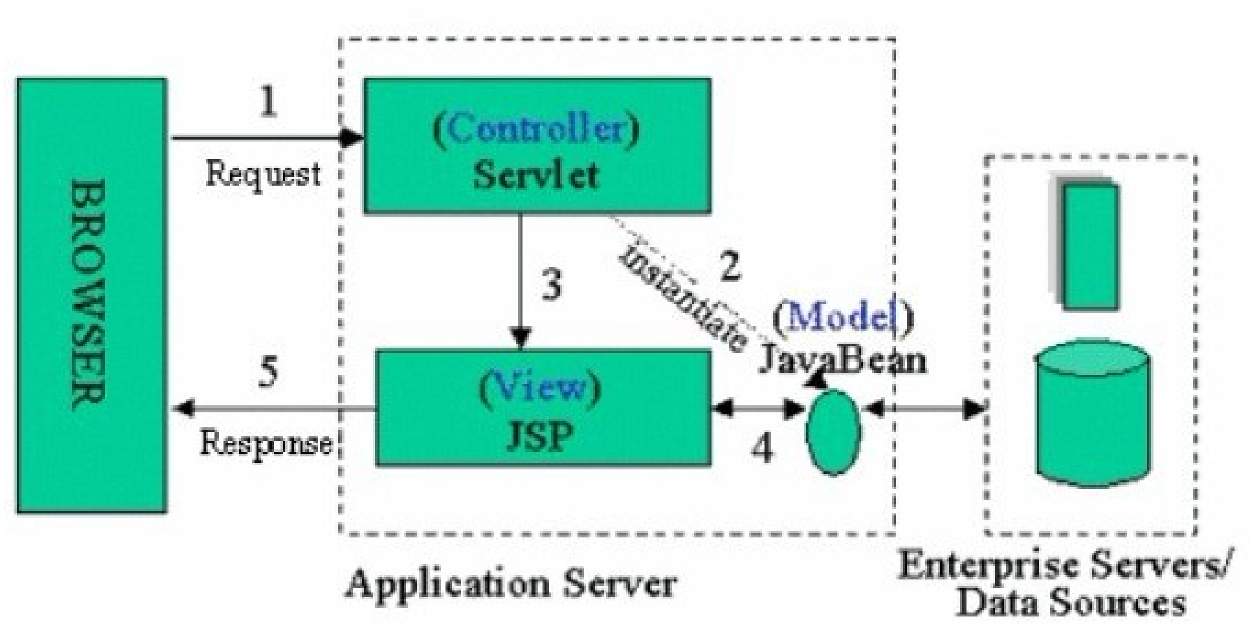
使用者
WEB 表示层
服务表示层
ASP.NET 应用系
服务合约
业务层
业务逻辑层
资源访问层
数据访问层
数据实体层
数据库

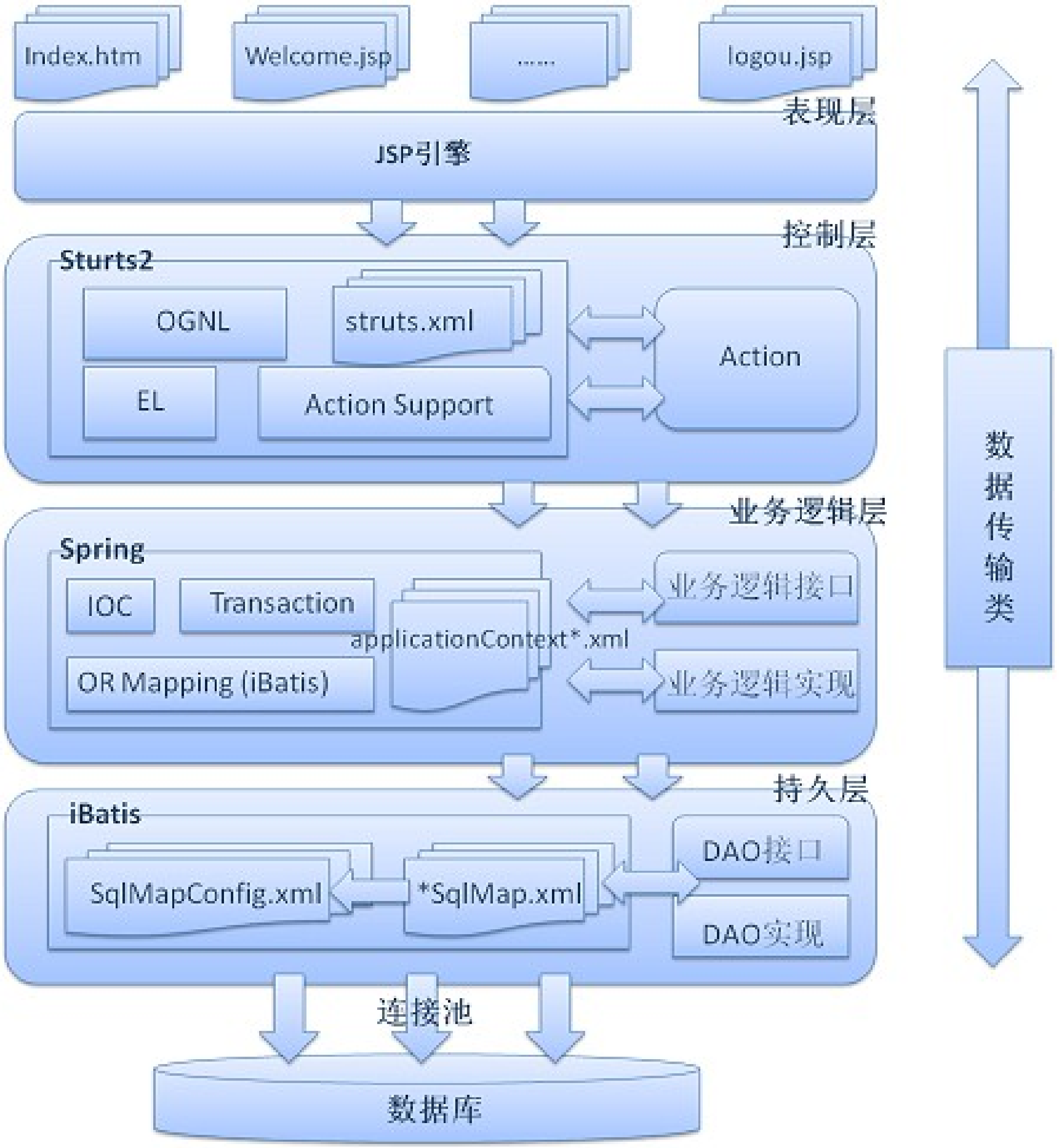

View
推荐的界面
Controller
推荐具体的操作
Model
物品

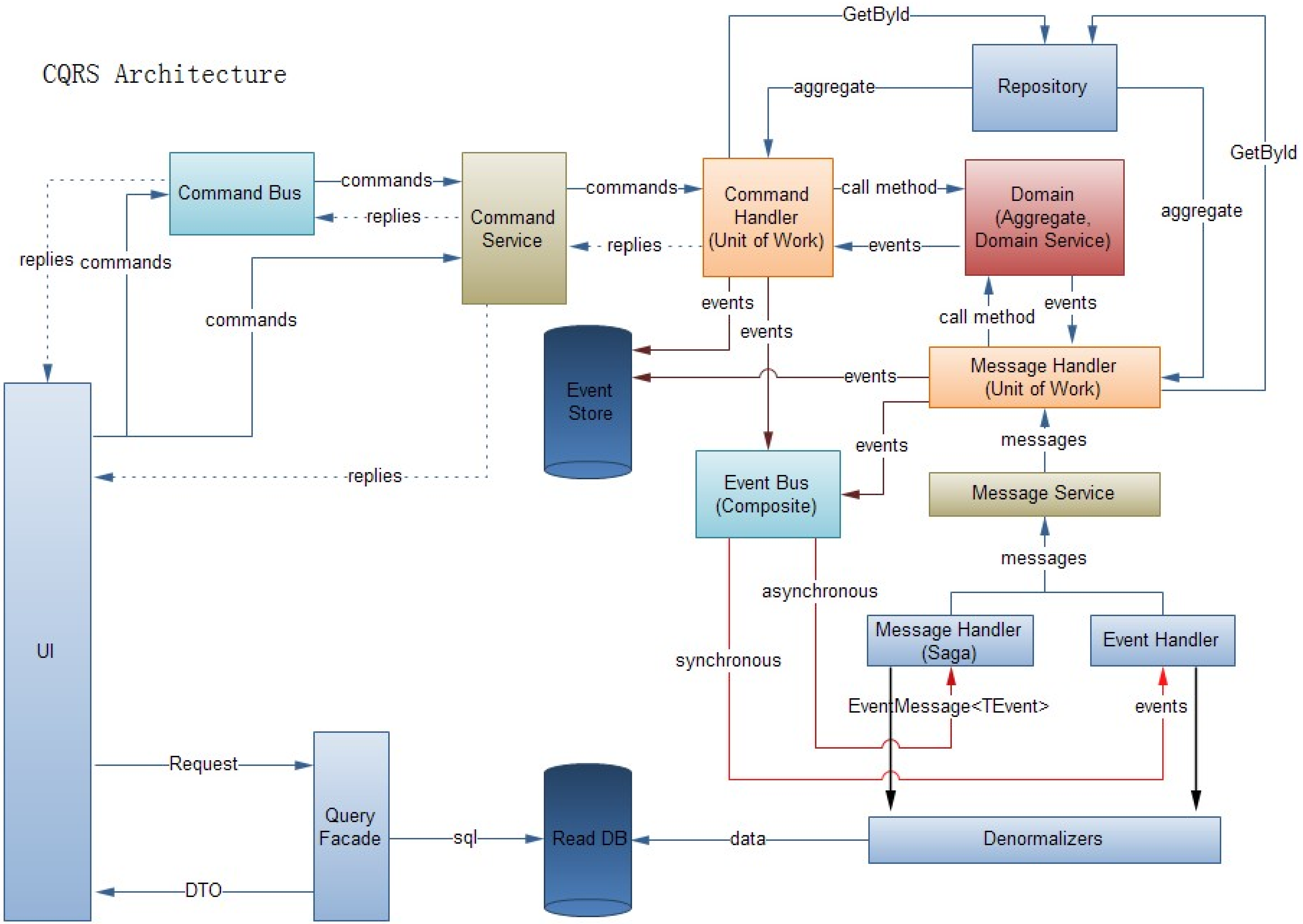
MVC
Model-View-Controller



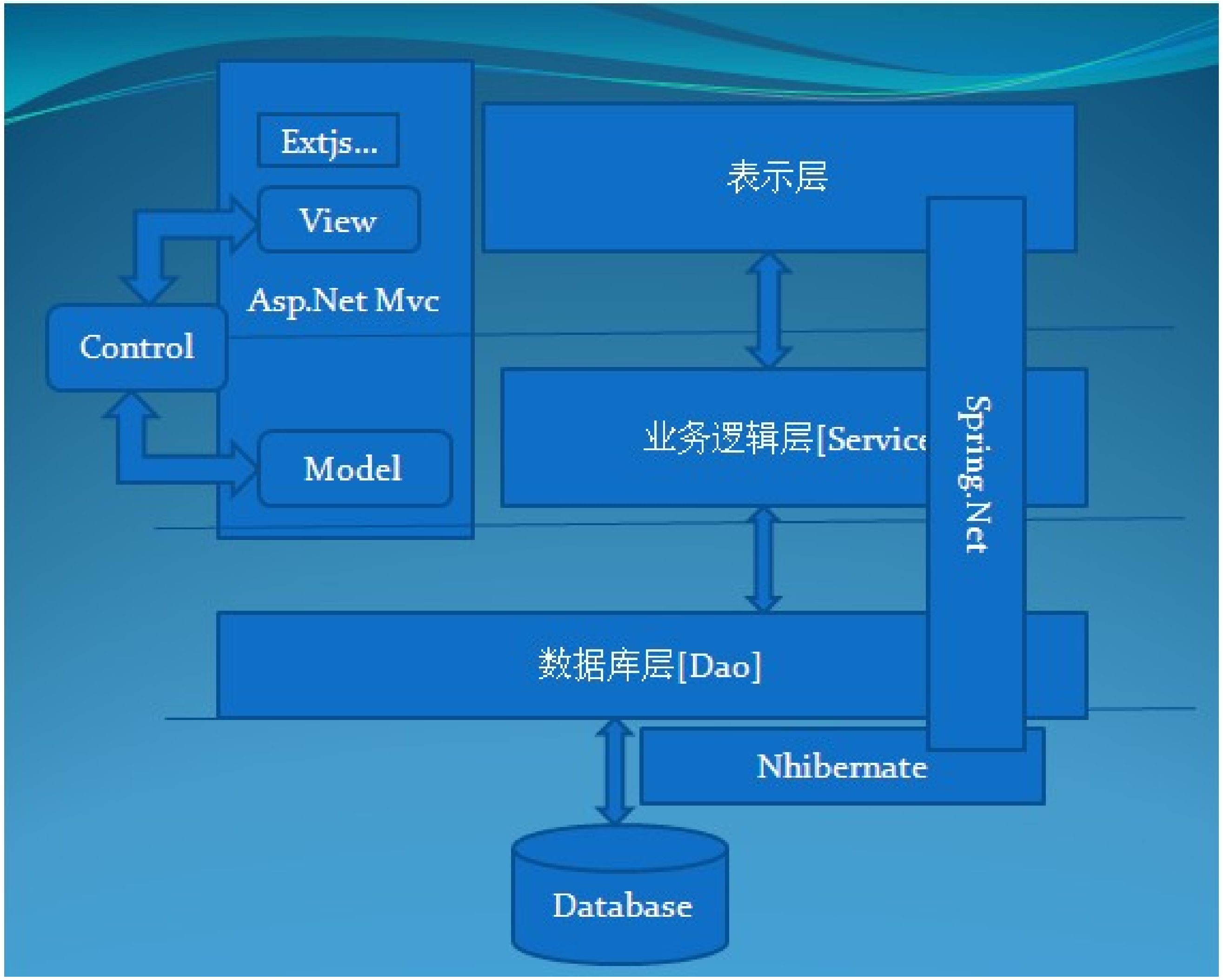
MVC
Model-View-Controller



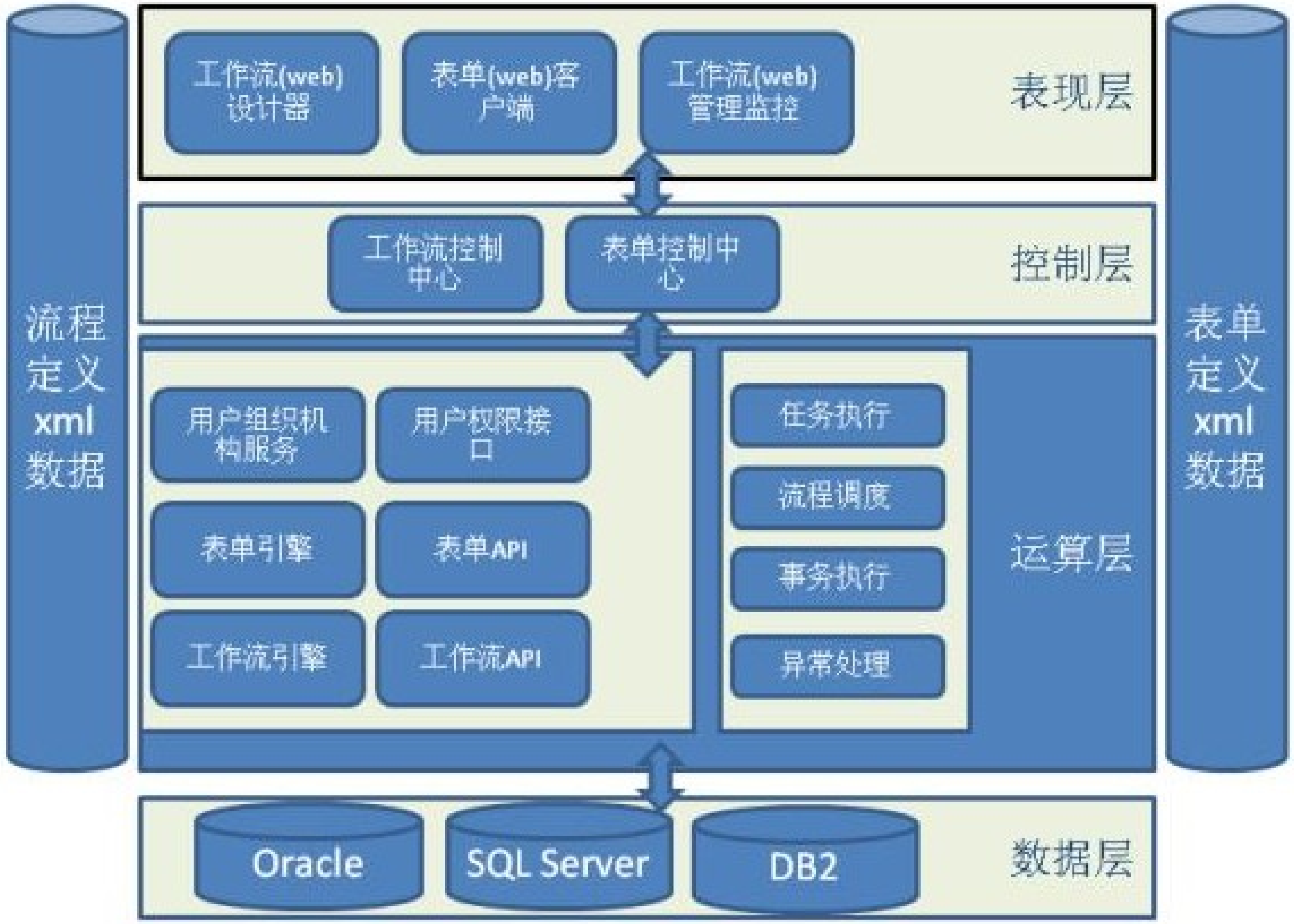

Index.htm
Welcome.jsp
......
logou.jsp
表现层
JSP引擎
控制层
Sturts2
OGNL
struts.xml
Action
EL
Action Support
业务逻辑层
Spring
IOC
Transaction
applicationContext*.xml
业务逻辑接口
OR Mapping (iBatis)
业务逻辑实现
持久层
iBatis
DAO接口
SqlMapConfig.xml
*SqlMap.xml
DAO实现
连接池
数据库
数据传输类


CQRS Architecture
GetById
Repository
aggregate
GetById
aggregate
Command Bus
commands
Command Service
commands
Command Handler (Unit of Work)
call method
Domain (Aggregate, Domain Service)
replies
replies
events
replies
commands
commands
events
events
call method
events
Event Store
events
Message Handler (Unit of Work)
events
messages
replies
Event Bus (Composite)
Message Service
messages
asynchronous
Message Handler (Saga)
Event Handler
synchronous
EventMessage<TEvent>
events
UI
Request
Query Facade
sql
Read DB
data
Denormalizers
DTO

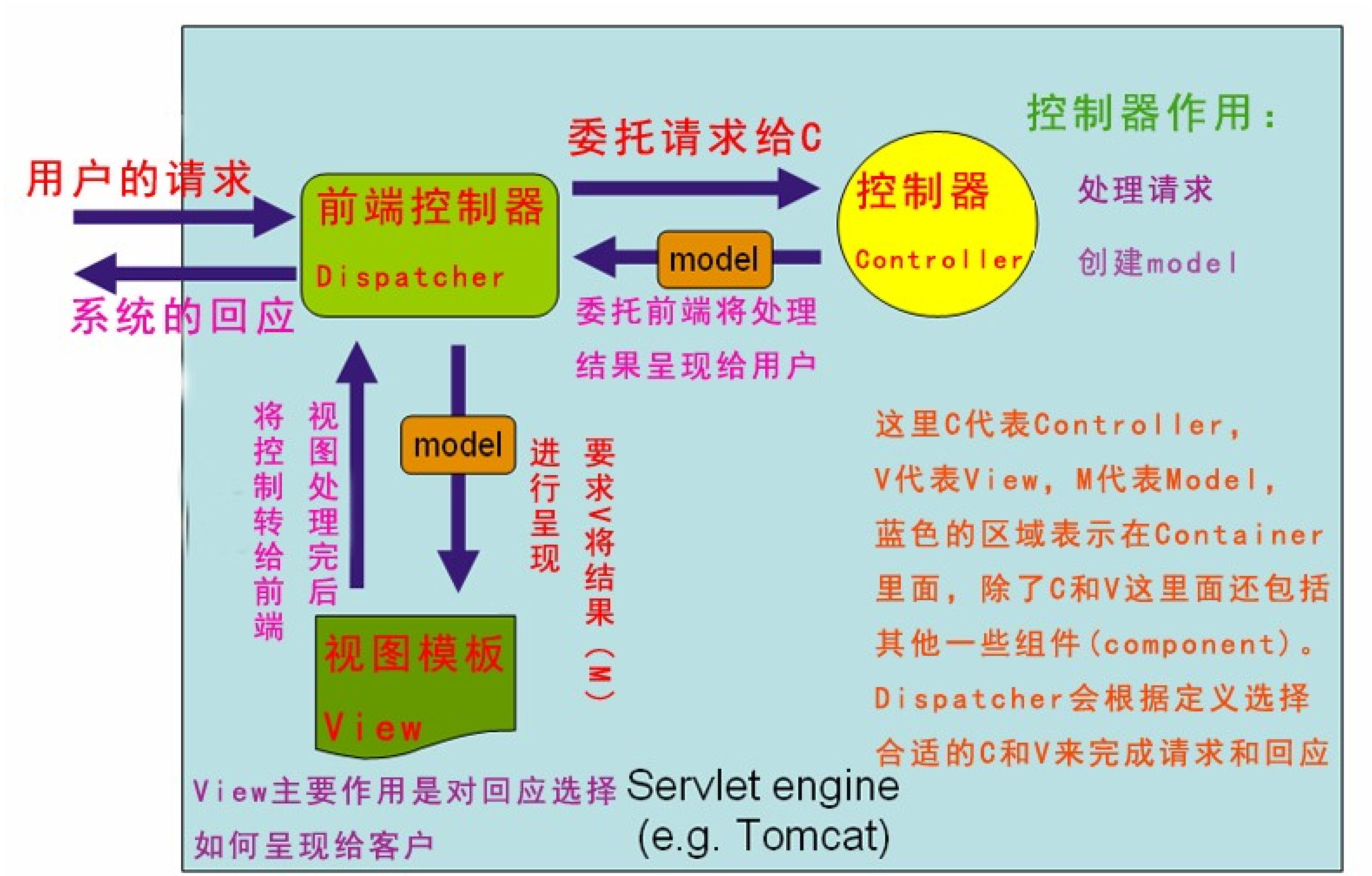
MVC
Model-View-Controller

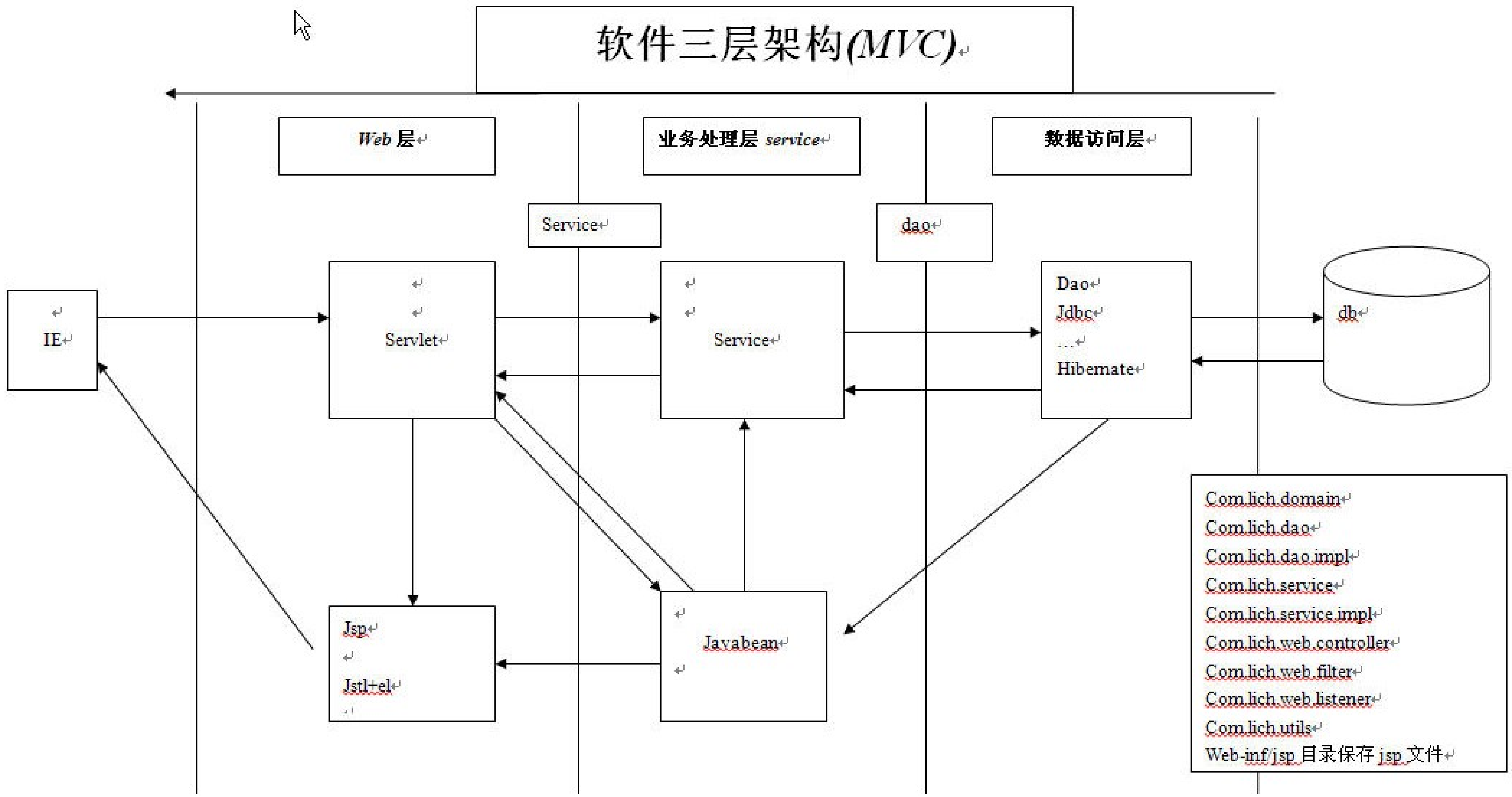


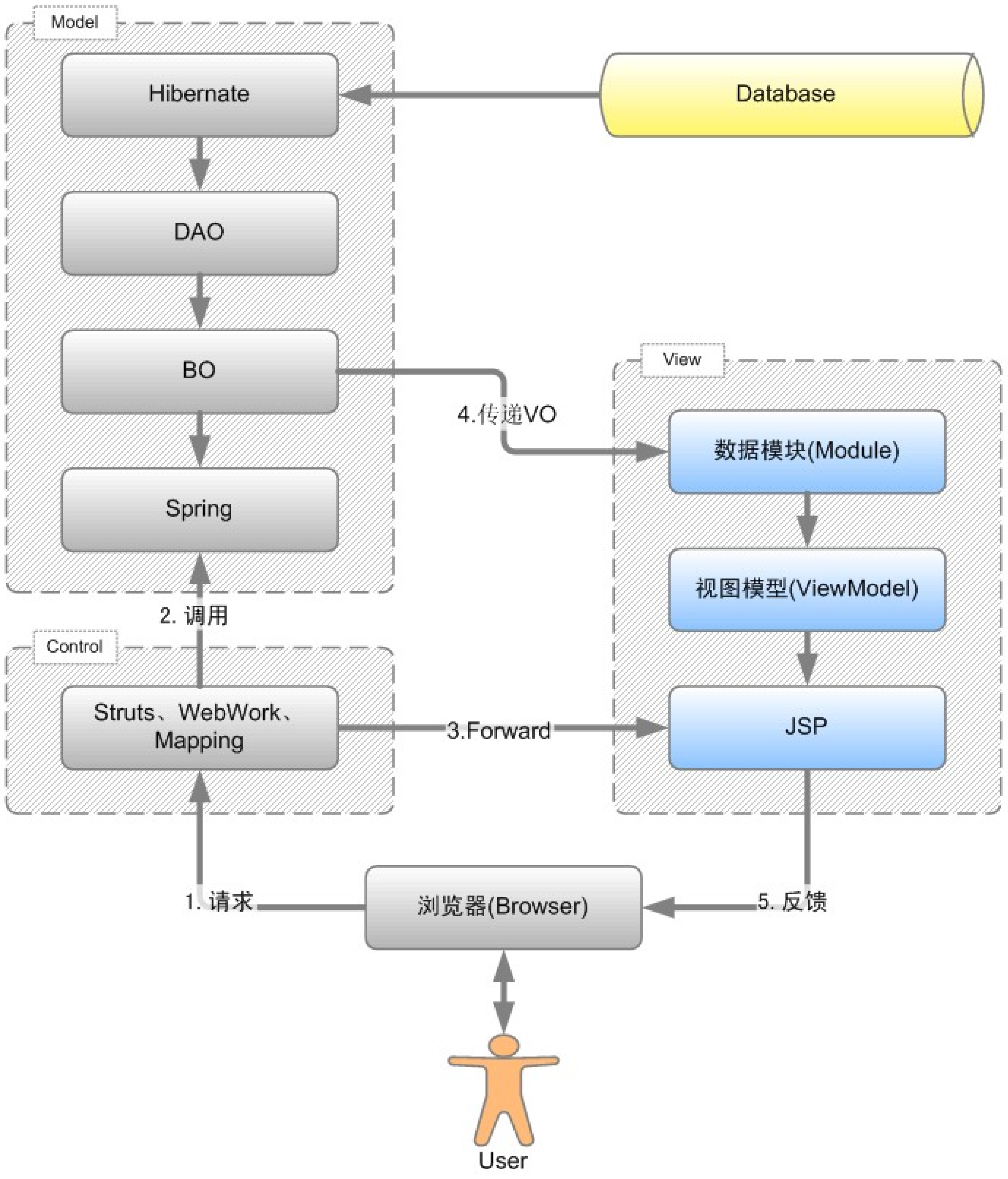
MVC
Model-View-Controller



MVC
Model-View-Controller




软件三层架构(MVC)
Web 层
业务处理层 service
数据访问层
Service
dao
IE
Servlet
Service
Dao
Jdbc
…
Hibernate
db
Jsp
Jstl+el
Javabean
Com.lich.domain
Com.lich.dao
Com.lich.dao.impl
Com.lich.service
Com.lich.service.impl
Com.lich.web.controller
Com.lich.web.filter
Com.lich.web.listener
Com.lich.utils
Web-inf/jsp 目录保存 jsp 文件


Model
Hibernate
DAO
BO
Spring
Database
4.传递VO
View
数据模块(Module)
视图模型(ViewModel)
JSP
2. 调用
Control
Struts、WebWork、Mapping
3.Forward
1. 请求
浏览器(Browser)
5. 反馈
User

Architectural Design Steps

1. **Key Concepts**
2. **Define the High-Level Organization of Subsystems**
3. **Identify Key Abstractions**
4. **Checkpoints**

What Are Key Abstractions?

- A key abstraction is a concept, normally uncovered in Requirements, that the system must be able to handle
- Sources for key abstractions
  - Domain knowledge
  - Requirements
  - Glossary
  - Domain Model, or the Business Model (if one exists)

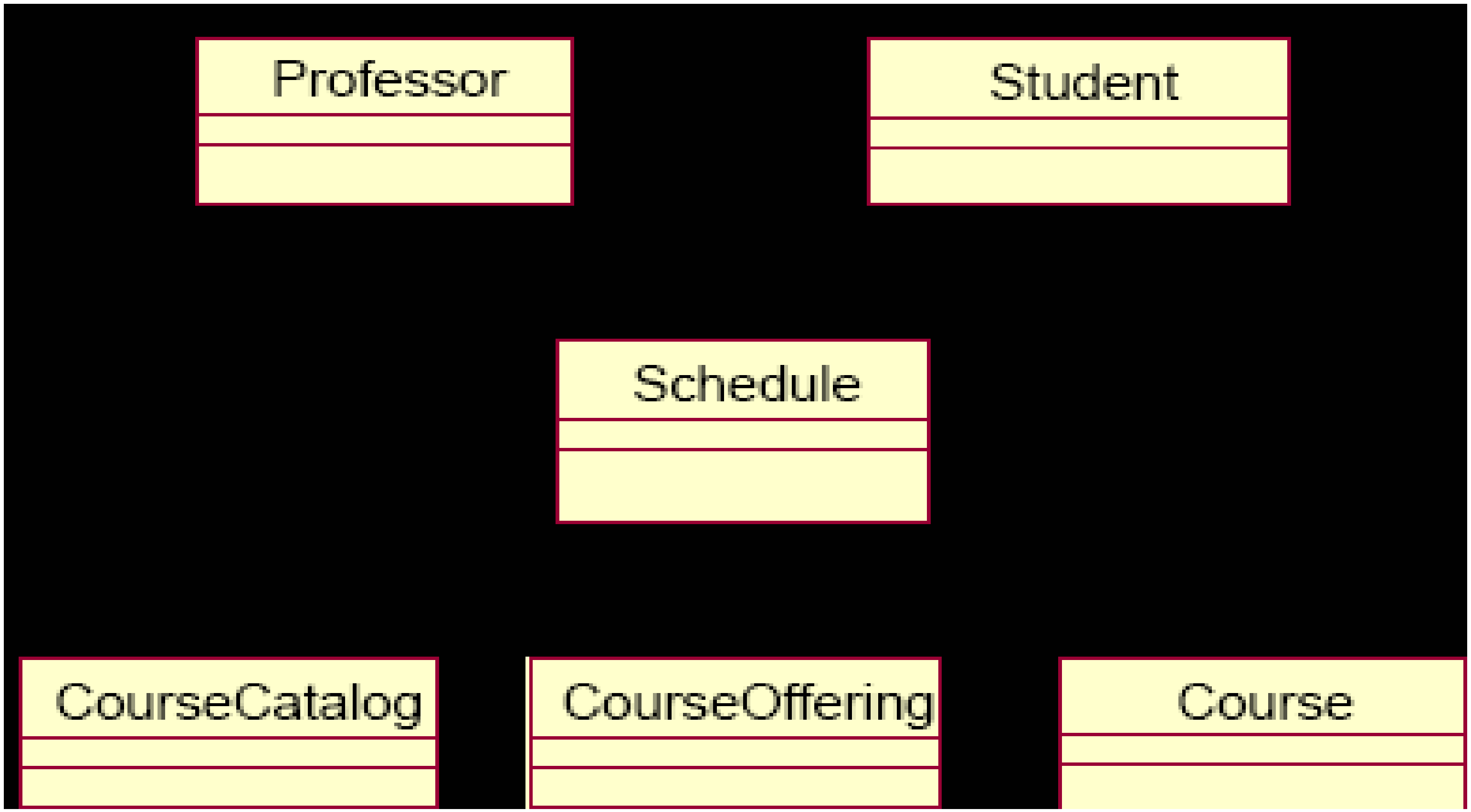
Defining Key Abstractions *Steps*

- Define analysis class relationships
- Model analysis classes and relationships on class diagrams
  - Include brief description of analysis class
- Map analysis classes to necessary analysis mechanisms

基于面向对象的软件架构设计的步骤和制品

1. 架构描述
   - 用一段文字，描述你的项目将采用哪种架构，例如 MVC ，SringMVC,…. 也可多做些解释。
2. 架构图
   - 基于上面 1 ）画出你系统的架构图，就是用包来描述，包里面是空的。
3. 关键抽象
   - 找出系统的数据类（ entity classes) ，并且用类图描述出来

Example: Key Abstractions



Architectural Design Steps

1. **Key Concepts**
2. **Define the High-Level Organization of Subsystems**
3. **Identify Key Abstractions**
4. **Checkpoints**

Architectural Design Steps

- Key Concepts
- Define the High-Level Organization of Subsystems
- Identify Key Abstractions
- Checkpoints

Checkpoints

- General
  - Is the package partitioning and layering done in a logically consistent way?
  - Have the necessary analysis mechanisms been identified?
- Packages
  - Have we provided a comprehensive picture of the services of the packages in upper-level layers?

Checkpoints (cont.)

- Classes
  - Have the key entity classes and their relationships been identified and accurately modeled?
  - Does the name of each class clearly reflect the role it plays?
  - Are the key abstractions/classes and their relationships consistent with the Business Model, Domain Model, Requirements, Glossary, etc.?

作业 -7th

1. What is the purpose of Architectural Design?
2. What is used in architecture diagram?
3. What key abstractions are identified during Architectural Design? Why are they identified here?
4. What is Design Pattern, and Framework?
5. How many Design Pattern can you describe?
6. 请描述架构设计的步骤及其制品
7. 请描述，采用基于结构化与基于面向对象的软件架构设计的相同点及不同点（建议对比步骤和制品）（选做）

Lab: Architectural Design

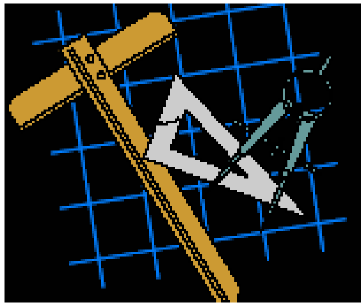
- **Given the following:**
  - **Some results from the Requirements discipline:**
- **Problem statement**
  - **Use-Case Model main diagram**
  - **Glossary**
  - **Some architectural decisions:**
  - **(textually) The upper-level architectural layers and their dependencies**

Lab: Architectural Design (cont.)

- **Identify the following:**
- **The key abstractions**

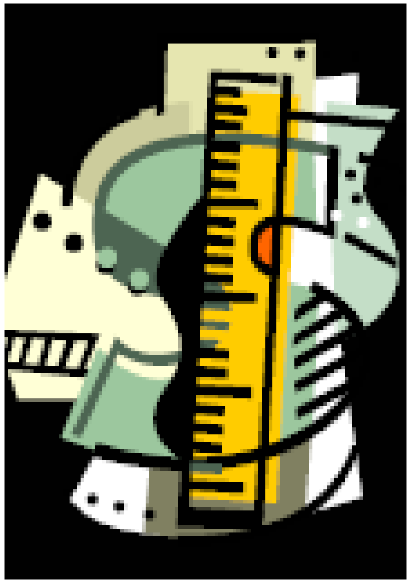
Lab: Architectural Design (cont.)

- **Produce the following:**
  - **Architecture Doc**
  - **Architecture Diagram (Class diagram containing the upper-level architectural layers and their dependencies (the architecture design of your project)**
  - **Class diagram containing the key abstractions**
  - **Other DOC if necessary, describing the above artifacts**



Lab: Review

- **Compare your key abstractions with the rest of the class**
  - **Have the key concepts been identified?**
  - **Does the name of each class reflect the role it plays?**
- **Compare your class diagram showing the upper-level layers**
  - **Do the package relationships support the System architecture?**



- Architectural Design focuses on defining a candidate architecture and constraining the architectural techniques to be used in the system.
- It relies on gathering experience gained in similar systems or problem domains to constrain and focus the architecture so that effort is not wasted in architectural rediscovery.
- In systems where there is already a well-defined architecture, Architectural Design might be omitted; Architectural Design is primarily beneficial when developing new and unprecedented systems.

Activity diagram

- **The workflow of a use case describes what needs to be done by the system to provide the value that the served actor is looking for.**
- **It consists of a sequence of activities that, together, produce something for the actor.**
- **The workflow often consists of a basic flow and one or several alternative flows.**
- **The structure of the workflow can be described graphically with the help of an activity diagram.**

Patterns

- offer a way to capture, reuse, and share solutions to common problems and a common language for describing problems and solutions.
- implemented in a development tool like Rational Software Architect help automate access to, and the application of, patterns.
- Design patterns and transformations in Rational Software Architect help automate routine modeling and coding tasks.

Continue……

- Grady Booch's definition of a pattern is still one of the best: "a solution to a recurring problem in a given context."
- Patterns provide a standard way of capturing and naming solutions, programming idioms, and best practices.
- As more developers have researched and understood patterns, patterns have become a standard way for practitioners to communicate and share what they know with each other.

Continue……

- For an architect leading a large team of developers, with tools to share and apply patterns automatically, patterns can become a way to enforce coding standards and ensure consistency in the way a design for a system is implemented.
- For the developer, a set of carefully selected patterns, customized for a specific project or application, can reduce mundane coding tasks and head off confusion about how to approach the implementation of design elements in the system.

- Most encounters with patterns occur through pattern specifications, descriptions of patterns in documentation, books, articles, and Web sites. An important function for patterns is to enhance communication and education within the community of designers and developers. The efficiency of using patterns is greatly enhanced by documenting them in a standard way. Usually pattern specifications include at least these three main topics:
  - Context: When would you apply this pattern?
  - Problem: A precise statement of the problem solved by the pattern
  - Solution: The description of the solution provided by this pattern
- This list is often extended with other topics such as consequences of applying the pattern, examples, keywords, and classification.
- Pattern specifications are critical, but they provide only part of the promise of patterns.

- Patterns allow you to create artifacts that can be more easily reused and customized for each use.
- The power of patterns is not realized by simply copying and reusing artifacts as-is (unlike cutting and pasting lines of code, or deploying an existing component into a new application). Such procedures are common and do constitute reuse, but patterns take reuse to a higher level by allowing points of variability.
- The key to pattern automation is the use of pattern implementation tools that expose the variability choices but hide the substitution details.

- Design Patterns
- Design patterns allow you to make use of existing solutions developed in the same type of model that you are working on.
- So, for example, the Observer GoF pattern contains design-level UML classes that can be applied in a design-level UML class model.
- Patterns have parameters so that you can customize them for a specific context, but patterns do not automatically translate themselves to work in different model types.
- You cannot, for example, apply a design pattern and get a code-level idiom in Java code without using transformations.

- Transformations
- Transformations take elements from one model and translate them into elements of a different model.
- For example, you can apply a transformation to move from a platform-independent model to a platform-specific model as you add in details about the platform and get closer to the implementation.
- When adding levels of refinement, you can transform from a platform-specific model to another, adding more details without changing the model type.
- Transformations are often applied to whole models, but they can be applied to selections from models as well.