中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC　　Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

Basic Concepts

- Formally define software architecture
- Distinguish prescriptive Versus descriptive architectures
- List the causes and types of architectural degradation, and the challenges of architecture recovery
- Understand elements of software architecture and differentiate between components and connectors
- Delineate the role of architectural styles and patterns in a software architecture

- 正式定义软件架构
- 区分说明性的和描述性的架构
- 列举架构退化的原因和类型，以及架构复苏的挑战
- 了解软件体系结构的元素并区分组件和连接器
- 描述软件体系结构中风格和模式的作用

- **Definition:**
  - 软件系统的体系结构是一组关于系统的主要设计决策
    A software system's architecture is the set of *principal design decisions* about the system
- 软件体系结构是软件系统构建和发展的蓝图
  Software architecture is the blueprint for a software system's construction and evolution
- 设计决策包括正在开发的系统的每个方面
  Design decisions encompass every facet of the system under development
  - Structure
  - Behavior
  - 交互
    Interaction
  - 非功能性属性
    Non-functional properties

- System Structure (e.g., central component)
- Functional behaviour (e.g., sequence of opeations)
- Interactions (e.g., event notifications)
- Non-functional properties (e.g., no single point of failure) 无单点故障
- System’s Implementation (e.g., Using Java Swing toolkit)

“主要的 ” 意味着授予设计决策“架构状态 ” 的重要程度

- “Principal” implies a degree of importance that grants a design decision “architectural status”

  这意味着并非所有的设计决策都是架构性的

  - It implies that not all design decisions are architectural

  也就是说，它们不一定会影响系统的体系结构

  - That is, they do not necessarily impact a system’s architecture

如何定义“主要的 ” 将取决于甲方将什么定义为系统目标

- How one defines “principal” will depend on what the stakeholders define as the system goals

设计决策是在系统的生命周期中做出的和未做出的--体系结构有一个暂时的方面

- Design decisions are and unmade over a system's lifetime → Architecture has a temporal aspect

在任何给定的时间点，系统只有一个架构

- At any given point in time the system has only one architecture

一个系统的架构会随着时间而改变

- A system's architecture will change over time
  架构可能分支、会聚等
  - Architectures can be forked, converge etc.
  通常，许多相关的架构都在发挥作用
  - Typically many related architectures are in play

What is “good” Architecture

中国科学技术大学
University of Science and Technology of China

- 该体系结构适合于使用的背景
  The architecture is appropriate for the context of use.

E.g. a 3-Tier e-commerce architecture is not appropriate for an avionics project 三层电子商务体系结构不适合航空电子项目

- 关于“好的架构”的指导集中于
  Guidance on “good architecture” focusses on:
  - Process
  - Structure
- 体系结构应该捕获关于系统的主要设计决策
  Architecture should capture the **principal** design decisions about the system.
- 该蓝图--关注于结构、组件行为、组件交互以及这些如何影响系统的质量属性
  The blueprint – focussing on Structure, Component Behaviour, Component Interaction and how that influences Quality Attributes of Systems

Process

- 架构师团队很小，并且维护架构的完整性
  The architect team is small and maintains the integrity of the architecture
- 体系结构与需要管理的质量属性的优先列表相关
  The architecture is justified in relation to a prioritized list of quality attributes that need to be managed
- 使用反映甲方兴趣的视角编写文档
  Document using views that reflect stakeholder interest
- 根据架构交付质量属性的好坏来评估架构
  Evaluate the architecture in terms of how well it delivers the quality attributes
- 选择允许增量实现的架构
  Choose architectures that allow incremental implementation

Structure

- 使用良好的模块化结构:隐藏信息、独立的关注点、不太可能更改的良好健壮的接口
  Use good modular structure: hide information, separate concerns, good robust interfaces that are unlikely to change
- 结构取决于视角/视点:至少是静态的(捕获例如代码中的依赖),动态的(捕获例如数据流中的依赖),部署(捕获例如对资源的依赖)
  Structure depends on perspective/viewpoint: at least **static** (captures e.g. dependency in code), **dynamic** (captures e.g. dependency in data flow), **deployment** (captures e.g. dependency on resources)
- 使用众所周知的模式和策略(稍后见)来实现质量属性
  Use well known patterns and tactics (see later) to achieve quality attributes
- 不要依赖于特定版本的工具
  Don’t depend on particular versions of tools
- 产生数据的模块应该与使用数据的模块分开
  Modules producing data should be separate from those consuming data

Structure

不要期望模块(静态结构)和组件(动态结构)之间有简单的映射

- Don’t expect simple mapping between modules (static structure) and components (dynamic structure)

除非必要，否则不要依赖于部署环境中的特殊功能

- Don’t depend on special features in the deployment environment unless essential

体系结构应该在组件之间使用少量的交互方式

- Architecture should use a small number of ways of interaction between components

应该清楚地确定资源争用问题

- Should be clearly identified resource contention issues

e.g. if network capacity is a potential issue the architect should budget capacity across components or some other management approach

如果网络容量是一个潜在的问题，那么架构师应该对跨组件的容量进行预算，或者采用其他一些管理方法

Importance of Architecture

中国科学技术大学
University of Science and Technology of China

- Software Architecture:
  - 使我们能够管理系统的关键属性
    Enables us to manage the key attributes of a system
  - 允许对变更进行推理和管理
    Allows reasoning about and managing change
  - 允许预测关键质量属性
    Allows prediction of the key quality attributes
  - 允许甲方之间更好的沟通
    Allows better communication among stakeholders
  - 携带最早的(最基本的)设计决策
    Carries the earliest (most fundamental) design decisions
  - 定义实现的约束
    Defines constraints on implementation

Importance of Architecture

中国科学技术大学
University of Science and Technology of China

- Software Architecture:
  - Reflects the structure of an organisation
  - Provides the basis for evolutionary prototyping
  - Is the key artifact in reasoning about cost and scheduling
  - Can be used as the transferrable, reuseable model at the heart of a product line
  - Focusses on the assembly of components rather than on the creation of the components
  - Restricts design alternatives and channels developer effort in a coordinated way
  - Provides the basis for training new team members.

-反映一个组织的结构
-为进化原型提供基础
-是成本和进度推理的关键工件
-可在产品线的核心作为可转移、可重复使用的模型
-专注于组件的组装，而不是组件的创建
-限制设计替代方案，协调开发人员的工作
-为培训新团队成员提供基础

Structure

结构难以描述

- Structure is slippery to describe

有时我们想要说明性(它应该是这样的)--通常太整洁了

- Sometimes we want to be **prescriptive** (this is how it should be) – often too tidy

有时我们想要描述性(事实就是如此)--常常是一团糟

- Sometimes we want to be **descriptive** (this is how it is) – often a mess.

Prescriptive vs. Descriptive Architecture

中国科学技术大学 University of Science and Technology of China

系统的说明性架构捕获了在系统构建之前所做的设计决策

- A system’s *prescriptive architecture* captures the design decisions made prior to the system’s construction

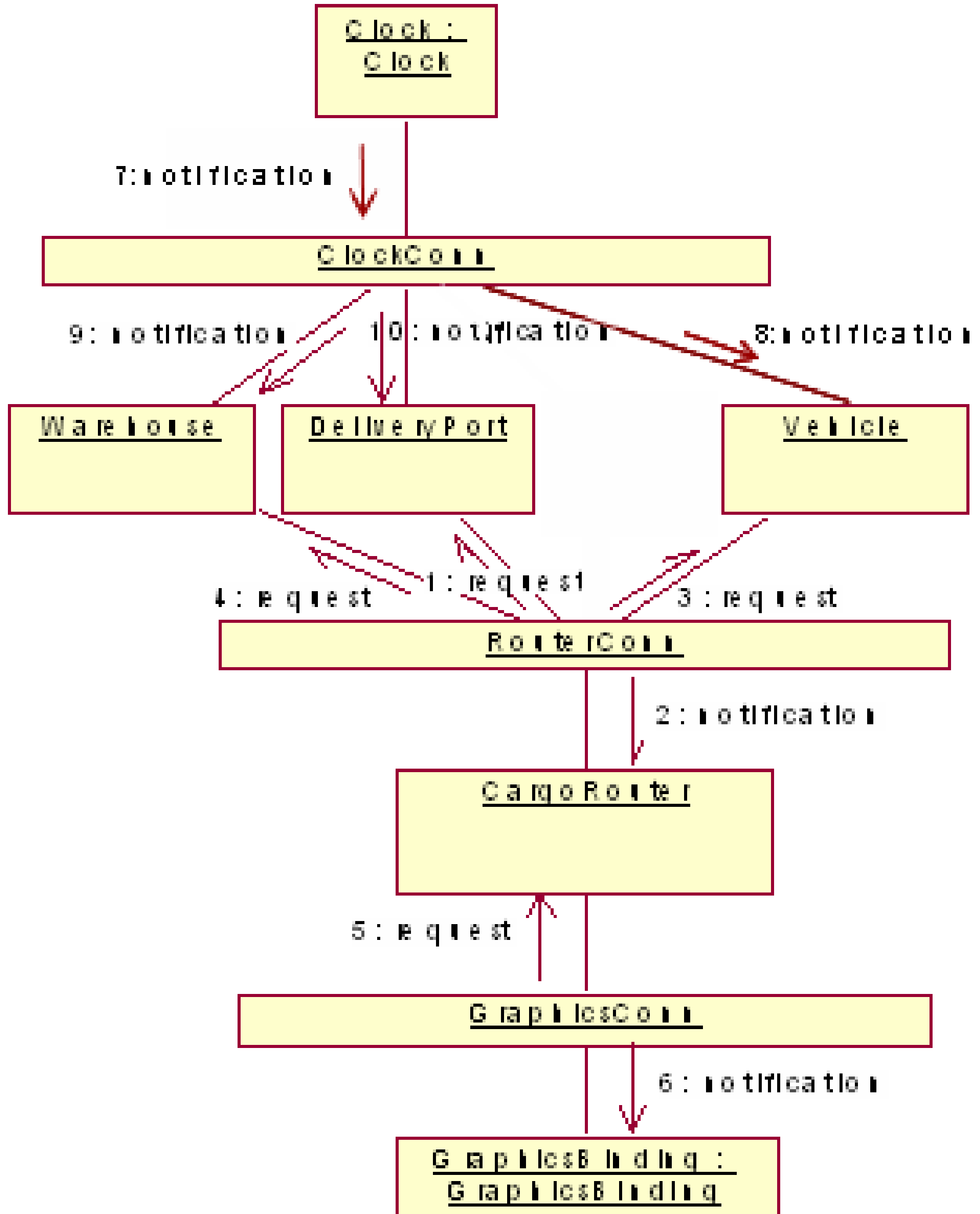
  构想、设计的架构

  - It is the *as-conceived* or *as-intended* architecture
- A system’s *descriptive architecture* describes how the system has been built

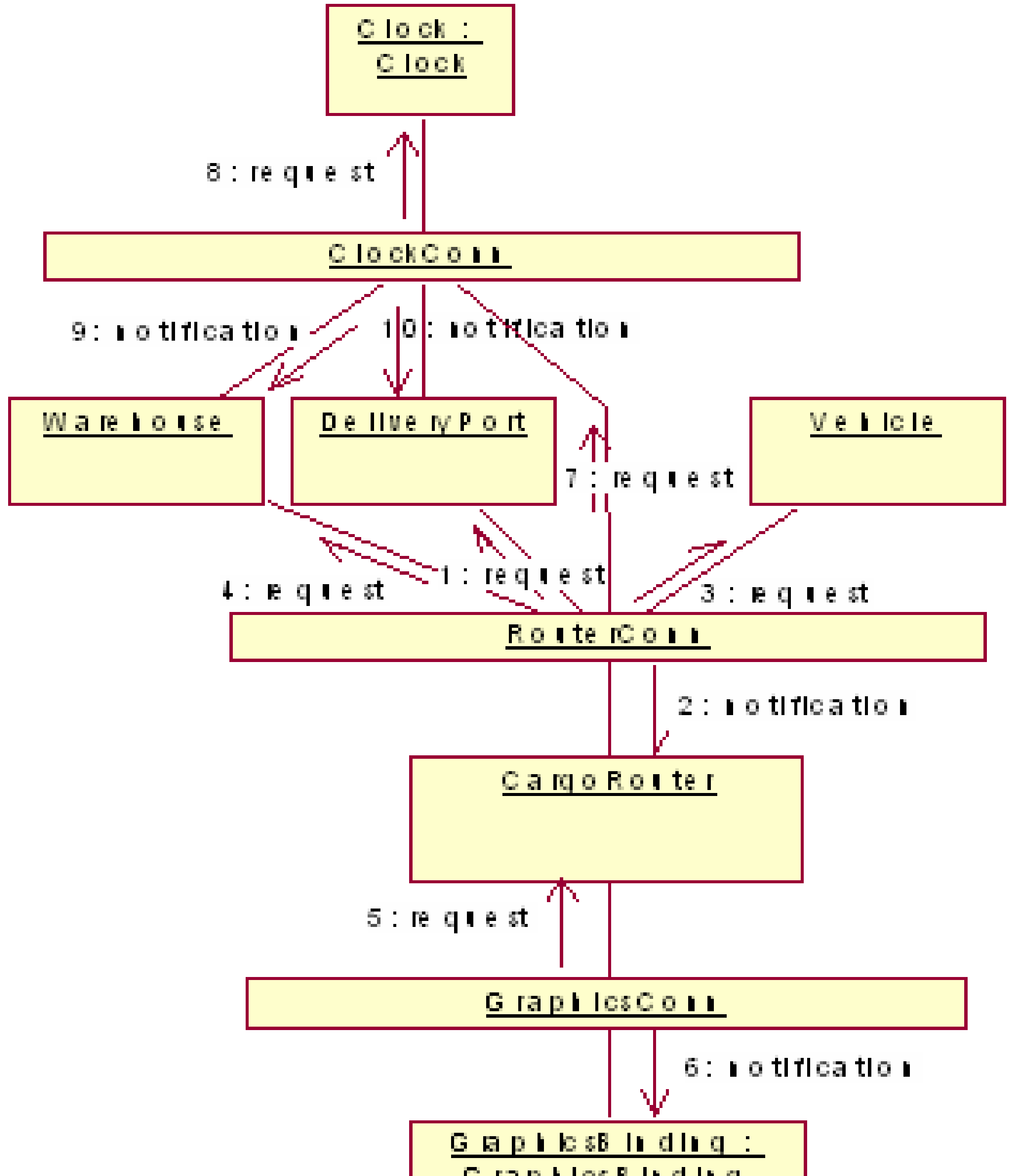
  系统的描述性体系结构描述了系统是如何构建的

  实现的架构

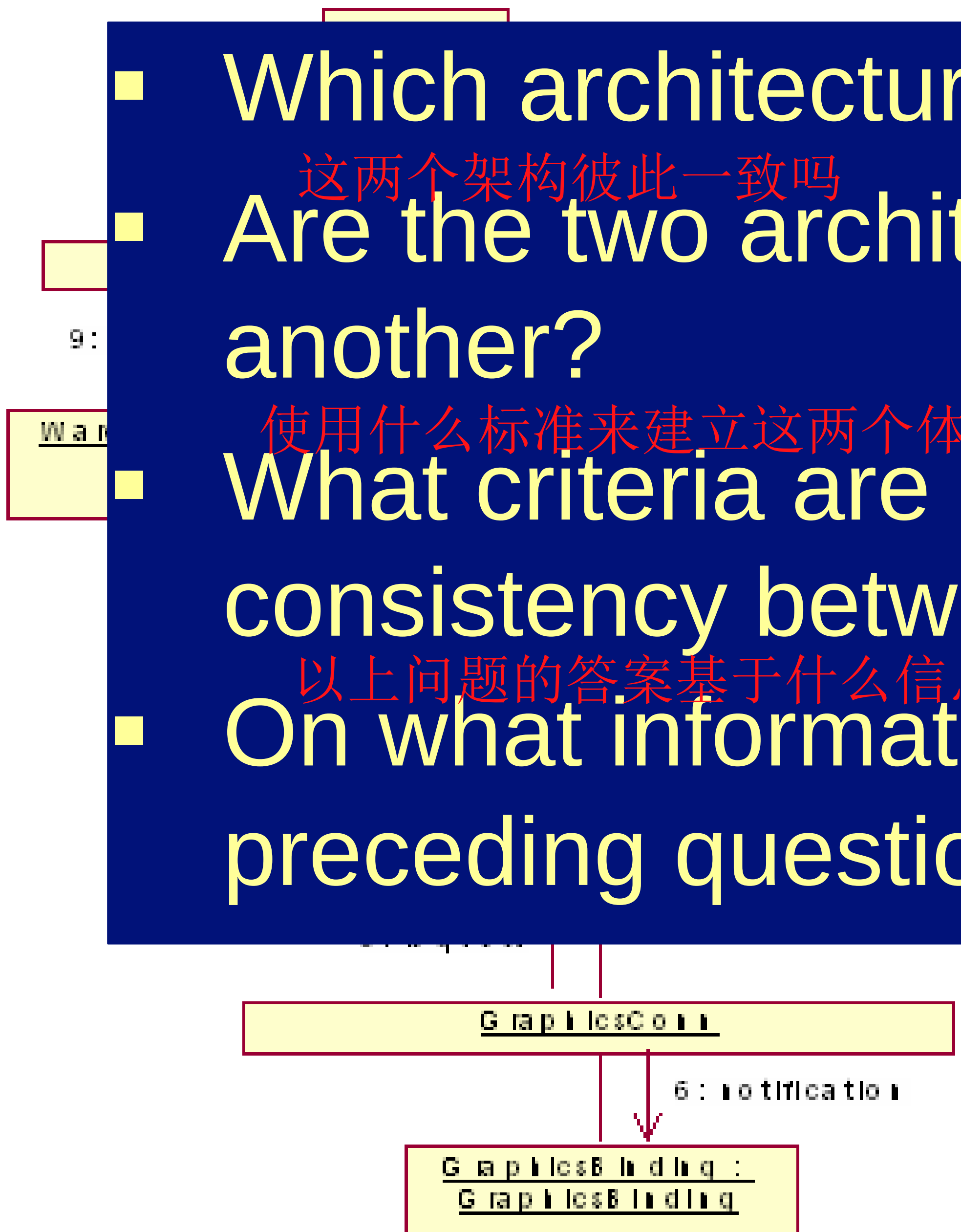
  - It is the *as-implemented* or *as-realized* architecture

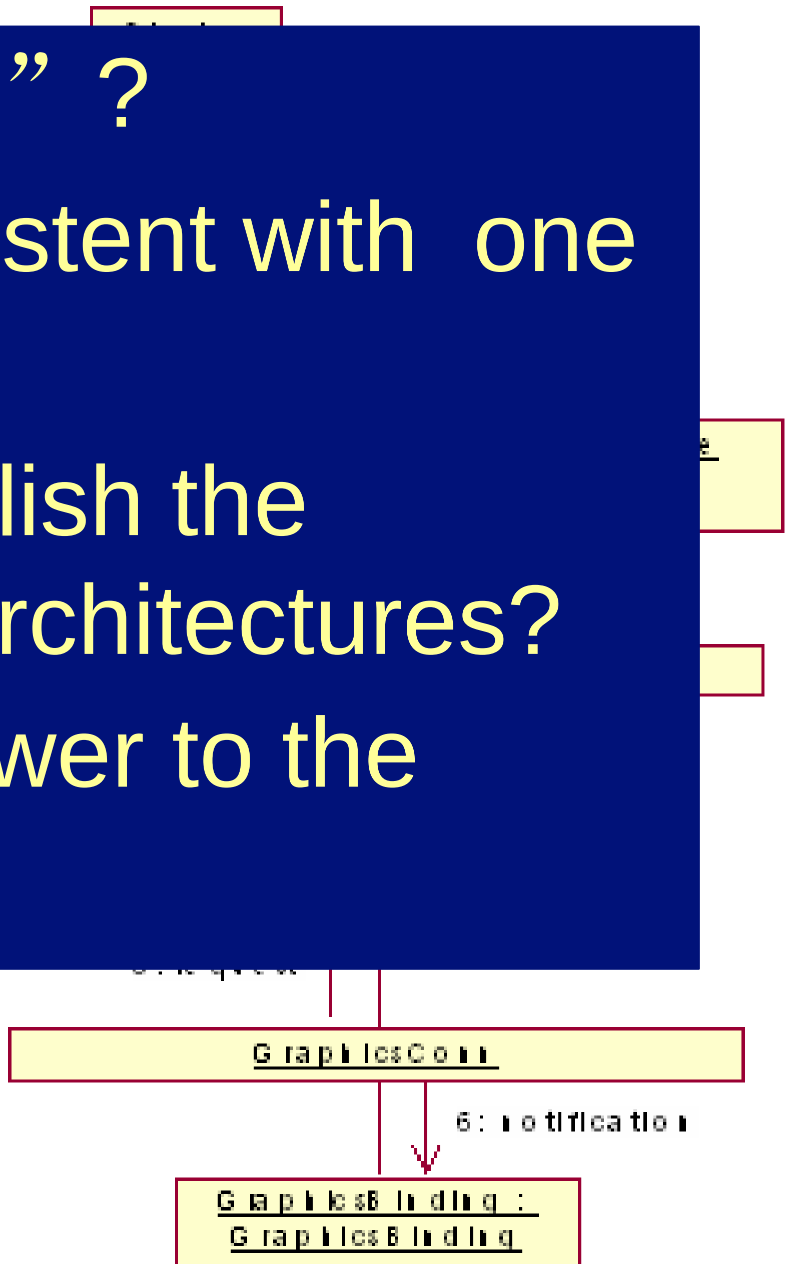


Clock : Clock
7: notification
ClockComm
9: notification
10: notification
8: notification
Warehouse
DeliveryPort
Vehicle
4: request
1: request
3: request
RouterComm
2: notification
CargoRouter
5: request
GraphicsComm
6: notification
GraphicsBinding : GraphicsBinding



Clock : Clock
8: request
ClockComm
9: notification
10: notification
Warehouse
DeliveryPort
Vehicle
7: request
4: request
1: request
3: request
RouterComm
2: notification
CargoRouter
5: request
GraphicsComm
6: notification
GraphicsBinding :





- Which architecture is “correct”？

这两个架构彼此一致吗

- Are the two architectures consistent with one another?

使用什么标准来建立这两个体系结构之间的一致性

- What criteria are used to establish the consistency between the two architectures?

以上问题的答案基于什么信息

- On what information is the answer to the preceding questions based?

- When a system evolves, ideally its prescriptive architecture is modified first 当系统发展时，理想情况下首先修改它的说明性架构

在实践中，系统—以及它的描述架构—经常是直接修改的

- In practice, the system – and thus its descriptive architecture – is often directly modified
- This happens because of
  - Developer sloppiness
  - Perception of short deadlines which prevent thinking through and documenting
  - Lack of documented prescriptive architecture
  - Need or desire for code optimizations
  - Inadequate techniques or tool support

这是因为
· 开发人员马虎
· 对短期截止日期的看法阻碍了思考和记录
· 缺少文件化的说明性架构
· 需要或渴望代码优化
· 不适当的技术或工具支持

- Two related concepts 两个相关的概念
  - Architectural drift
  - Architectural erosion

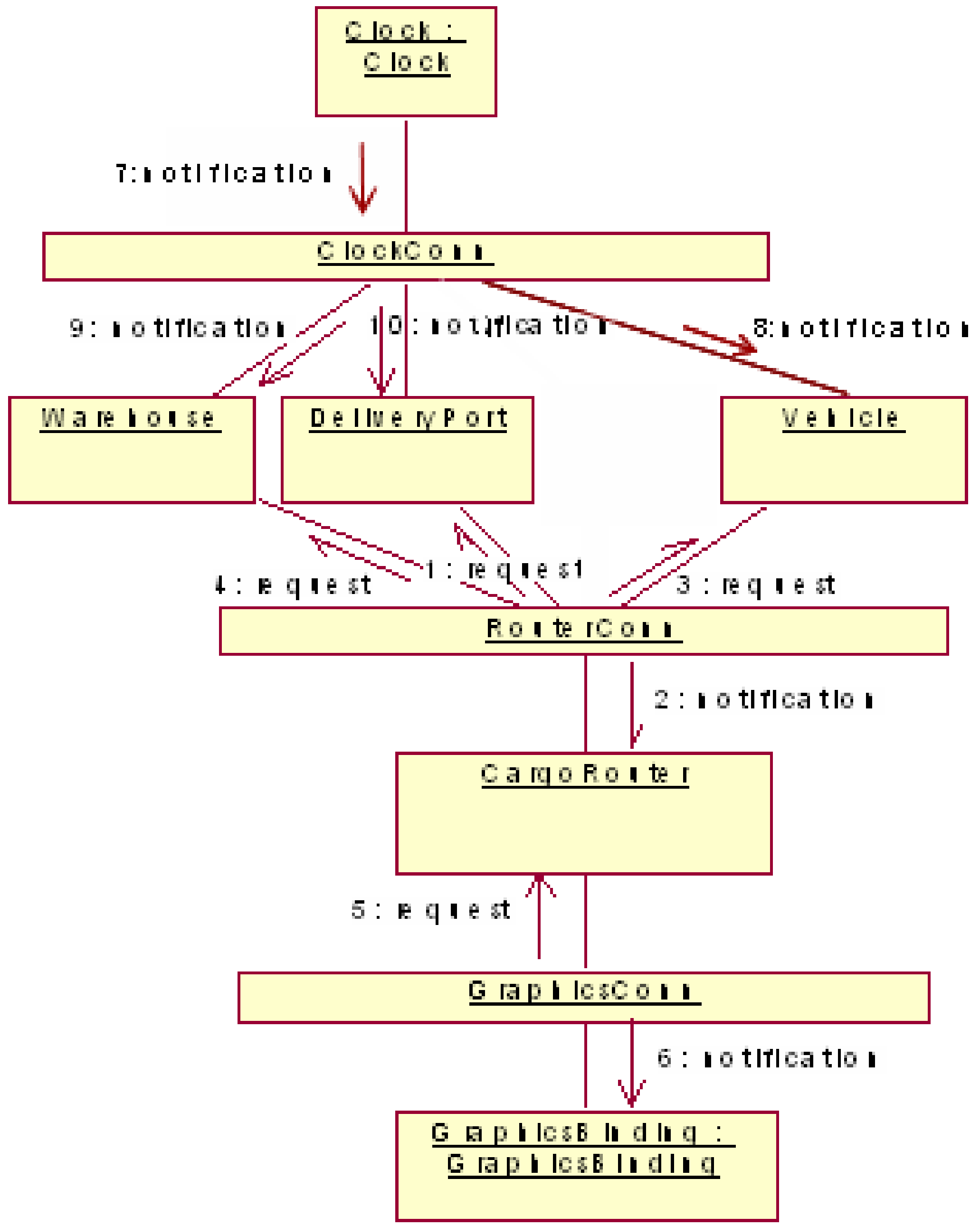
◆架构漂移
◆架构侵蚀

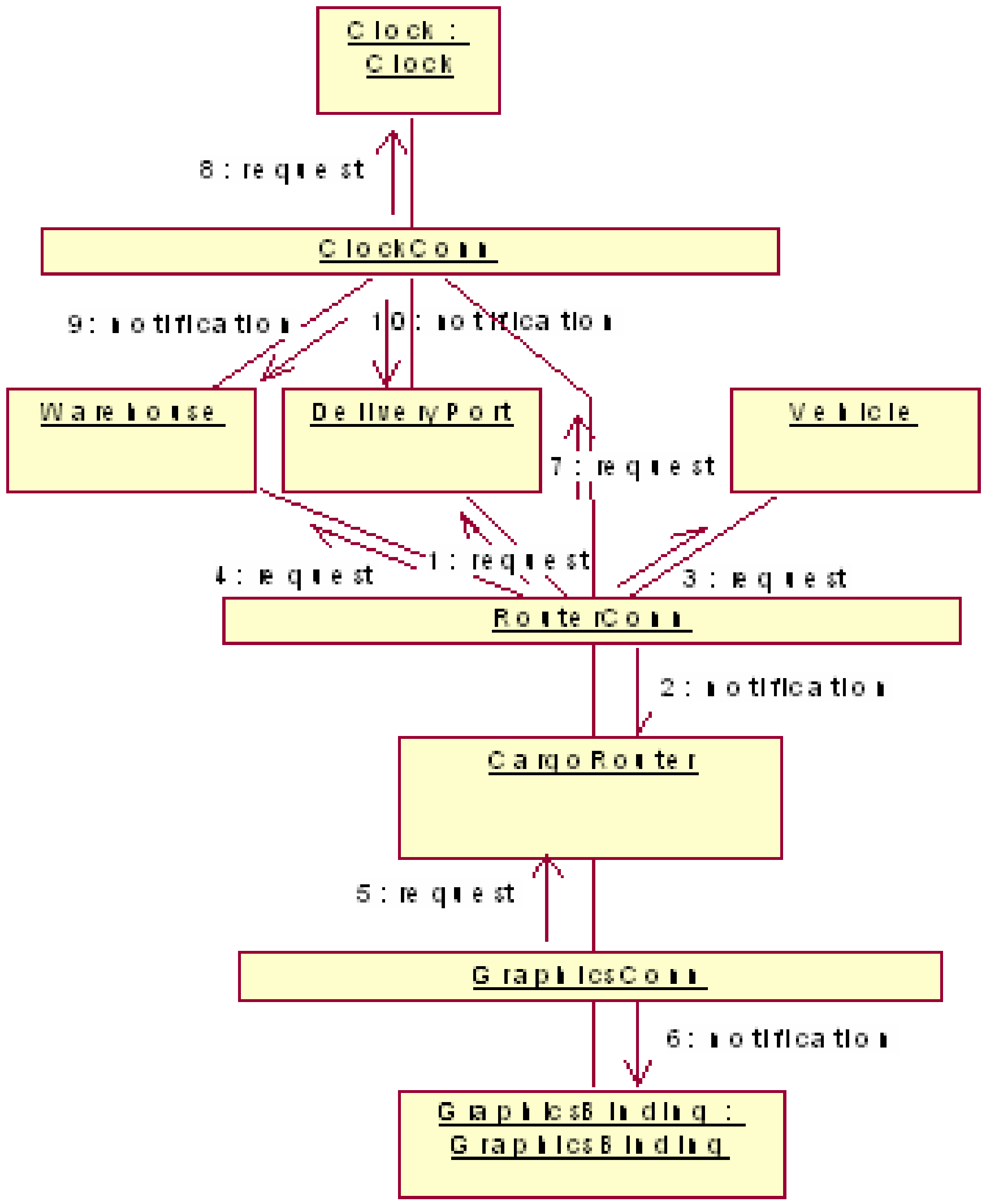
- *Architectural drift* is introduction of principal design decisions into a system's descriptive architecture that
  - are not included in, encompassed by, or implied by the prescriptive architecture
  - but which do not violate any of the prescriptive architecture' s design decisions

架构漂移是将主要设计决策引入到系统的描述性架构中
· 是不包含或隐含在说明性体系结构中
· 但是不违反任何说明性架构的设计决策

- *Architectural erosion* is the introduction of architectural design decisions into a system's descriptive architecture that violate its prescriptive architecture

架构侵蚀是将体系结构设计决策引入到系统的描述性体系结构中，但违反了系统的说明性体系结构


Clock : Clock
7:notification
ClockComm
9: notification
10: notification
8:notification
Warehouse
DeliveryPort
Vehicle
4: request
1: request
3: request
RouterComm
2: notification
CargoRouter
5: request
GraphicsComm
6: notification
GraphicsBinding : GraphicsBinding


Clock : Clock
8: request
ClockComm
9: notification
10: notification
Warehouse
DeliveryPort
7: request
Vehicle
4: request
1: request
3: request
RouterComm
2: notification
CargoRouter
5: request
GraphicsComm
6: notification
GraphicsBinding : GraphicsBinding

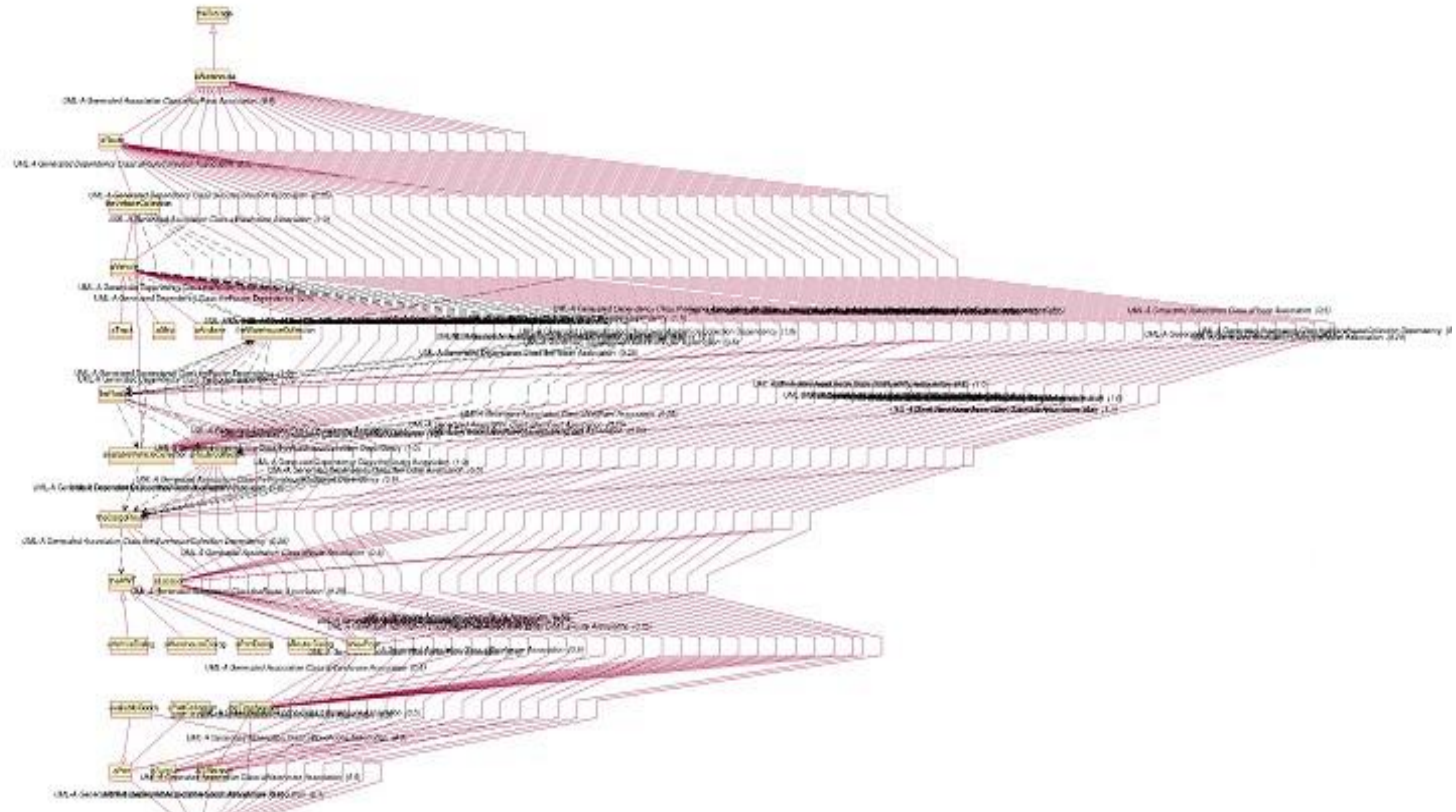
如果允许架构退化发生，那么迟早会被迫恢复系统的架构

- If architectural degradation is allowed to occur, one will be forced to *recover* the system’s architecture sooner or later

架构恢复是从实现级工件确定软件系统架构的过程

- *Architectural recovery* is the process of determining a software system’s architecture from its implementation- level artifacts
- Implementation-level artifacts can be
  - Source code
  - Executable files
  - Java .class files

实现级别的工件可以
◆源代码
◆可执行文件
◆Java.class文件



软件系统的体系结构通常不是(也不应该是)一个统一的整体

- A software system’s architecture typically is not (and should not be) a uniform monolith
- A software system’s architecture should be a composition and interplay of different elements

软件系统的体系结构应该是不同元素的组合和相互作用
· 处理
· 数据，也称为信息或状态
· 交互

  - Processing
  - Data, also referred as information or state
  - Interaction

在系统架构中封装处理的元素和数据被称为软件组件

- Elements that encapsulate processing and data in a system's architecture are referred to as *software components*

软件组件是一个体系结构实体
●封装系统功能和数据的一个子集
●通过显式定义的接口限制访问该子集
●显示定义了所需执行上下文的依赖关系

- **Definition**
  - A *software component* is an architectural entity that
    - encapsulates a subset of the system's functionality and/or data
    - restricts access to that subset via an explicitly defined interface
    - has explicitly defined dependencies on its required execution context

组件通常提供特定于应用程序的服务

- Components typically provide application-specific services

特定于应用程序的组件

- Application-specific components
  - Examples: Cargo, warehouse, vehicle

受限的重用组件

- Limited reuse components
  - Examples: Web servers, clocks, connections

可重用组件

- Reusable components
  - Examples: GUI components, class and math libraries

- In complex systems *interaction* may become more important and challenging than the functionality of the individual components
  在复杂系统中，交互可能比单个组件的功能更加重要和具有挑战性
- **Definition**
  软件连接器是一个架构构建块，其任务是影响和调节组件之间的交互
  - A *software connector* is an architectural building block tasked with effecting and regulating interactions among components
- In many software systems connectors are usually simple procedure calls or shared data accesses
  在许多软件系统中，连接器通常是简单的过程调用或共享数据访问
- Connectors typically provide application-independent interaction facilities 连接器通常提供与应用程序无关的交互设施
  . 可独立于部件进行描述
  - Can be described independent of the components

- Procedure call connectors
- Shared memory connectors
- Message passing connectors
- Streaming connectors
- Distribution connectors
- Wrapper/adaptor connectors

●过程调用连接器
●共享内存连接器
●消息传递连接器
●流连接器
●分布连接器
●包装/适配器连接器

组件和连接器在给定的系统架构中以特定的方式组合，以实现系统的目标

- Components and connectors are composed in a specific way in a given system's architecture to accomplish that system's objective
- **Definition**

  体系结构配置或拓扑，是软件系统体系结构的组件和连接器之间的一组特定关联

  - An *architectural configuration*, or topology, is a set of specific associations between the components and connectors of a software system's architecture

- Certain design choices regularly result in solutions with superior properties

某些设计选择通常会产生具有优异性能的解决方案
· 与其他可能的替代方案相比，这样的解决方案更优雅、有效、高效、可靠、可演化、可扩展，等等

  - Compared to other possible alternatives, solutions such as this are more elegant, effective, efficient, dependable, evolvable, scalable, and so on

体系结构风格是体系结构设计决策的命名集合
●是适用于一个给定的开发环境
●约束特定于该上下文中特定系统的架构设计决策
●在每个产生的系统中引出有益的品质

- **Definition**
  - An *architectural style* is a named collection of architectural design decisions that
    - are applicable in a given development context
    - constrain architectural design decisions that are specific to a particular system within that context
    - elicit beneficial qualities in each resulting system

中国科学技术大学
University of Science and Technology of China

- REST style (Representational State Transfer) – HTTP

REST风格(具象状态传输)

- 客户端和服务器之间的统一接口
- 无状态:各个请求之间客户端的上下文没有存储在服务器上。请求URL中携带所有状态。
- 客户端应该能够缓存对请求的响应
- 分层架构:客户端无法分辨自己是直接连接到服务器还是通过代理连接到服务器
- 按需编码(可选):服务器应该能够通过客户端脚本扩展客户端的功能

  - Uniform Interface between clients and servers
  - Stateless: No client context stored on server between requests. All state is carried in the request URL.
  - Clients should be able to cache responses to requests
  - Layered architecture: Clients cannot tell if they are connected directly to the server or thro’ a proxy
  - Code on demand (optional): Server should be able to extend the client’s functionality thro’ client-side scripts

- **Definition**

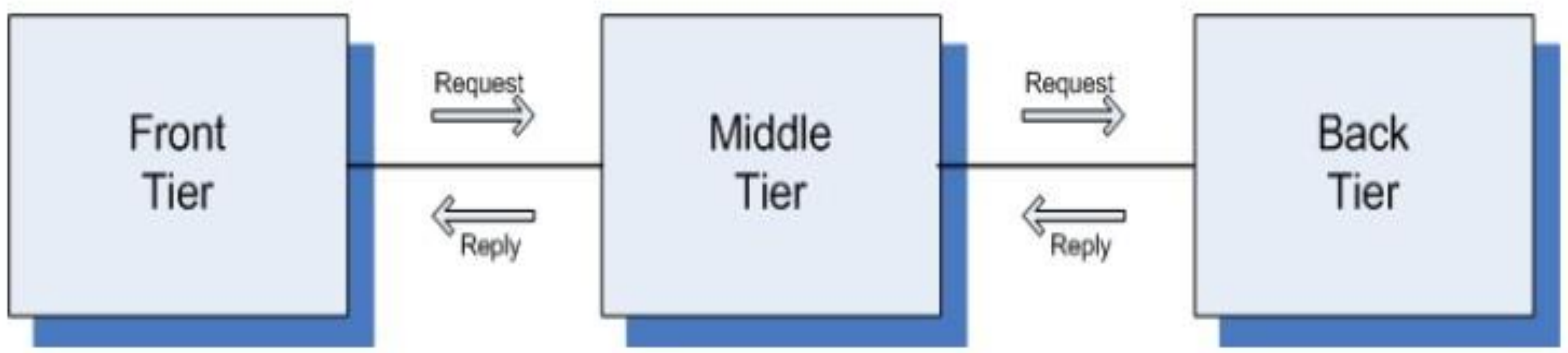
  体系结构模式是一组适用于重复出现的设计问题的体系结构设计决策，并且参数化以说明问题出现时的不同软件开发上下文

  - An *architectural pattern* is a set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears

在现代分布式系统中广泛使用的模式是三层系统模式

- A widely used pattern in modern distributed systems is the *three-tiered system* pattern
  - Science
  - Banking
  - E-commerce
  - Reservation systems

◆科学
◆银行
◆电子商务
◆预订系统



- Front Tier
  - 包含访问系统服务的用户界面功能
    Contains the user interface functionality to access the system's services
- Middle Tier
  - 包含应用程序的主要功能
    Contains the application's major functionality
- Back Tier
  - 包含应用程序的数据访问和存储功能
    Contains the application's data access and storage functionality

- **Style**

  · 提供一组采用解决方案的指导原则
  · 需要相当大的努力来应用
  架构师需要根据架构风格来证明设计选择

  - Provides a set of guiding principles in adopting solutions
  - Requires considerable effort to apply.

Architect needs to justify the design choices based on the architectural style.

- **Pattern**

  · 提供具体的解决方案，尽管参数化了具体的问题
  · 只需很少的人力或理由来应用
  · 通常适用于特定的系统(例如，基于gui的系统)

  - Provides concrete solutions, although parameterized to the specific problem.
  - Requires very little manual effort or justification to apply.
  - Usually applies to specific systems (e.g., GUI-based systems)

Architectural Models, Views, and Visualizations

架构模型

- **Architecture Model**

  记录有关系统的部分或全部架构设计决策的工件

  - An artifact documenting some or all of the architectural design decisions about a system

架构可视化

- **Architecture Visualization**

  一种向甲方描述有关系统的部分或全部架构设计决策的方法

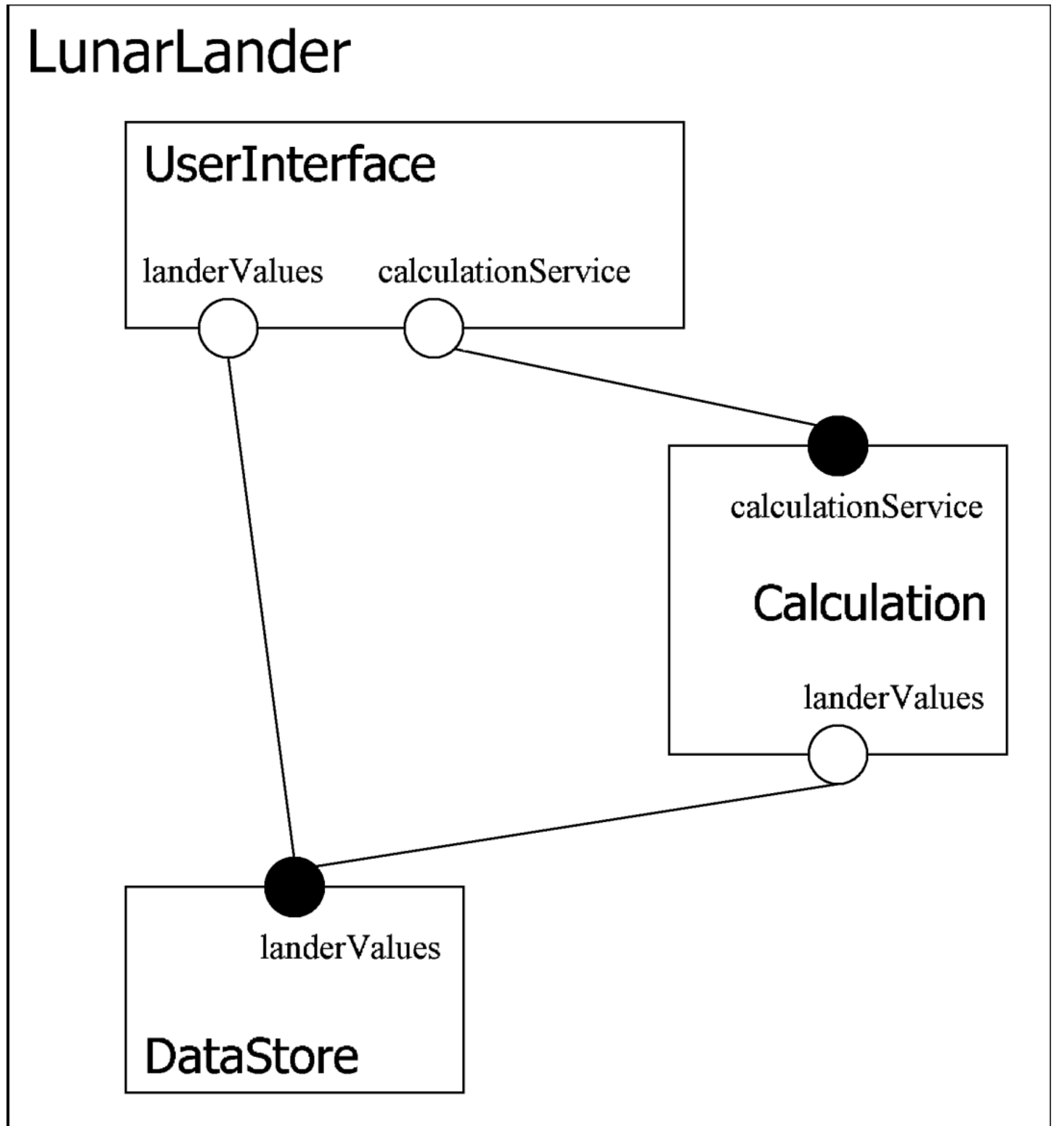
  - A way of depicting some or all of the architectural design decisions about a system to a stakeholder

架构视图

- **Architecture View/Perspective** 相关架构设计决策的子集 通常属于横切功能
  - A subset of related architectural design decisions
  - Typically pertain to a cross-cutting functionality

符号图

Graphical Diagram

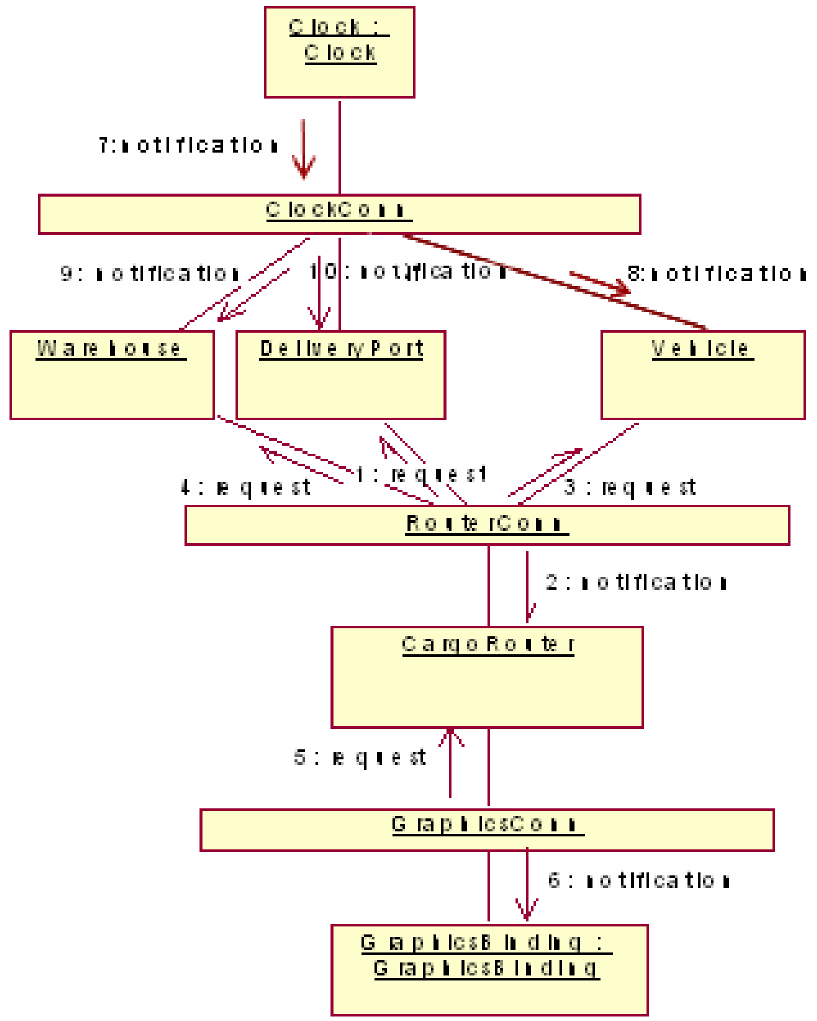


文字描述

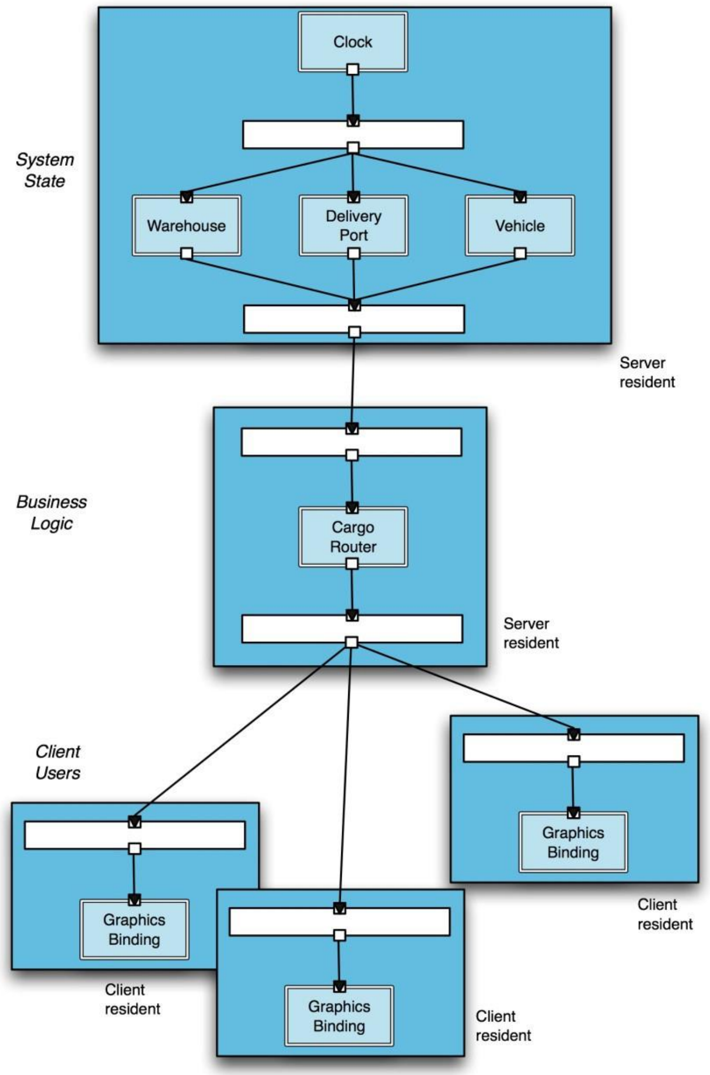
Textual descriptions

```
component DataStore{  provide landerValues;
}


 component Calculation{  require
    landerValues;
 provide calculationService;
 }

 component UserInterface{  require
    calculationService;  require
    landerValues;
 }

 component LunarLander{  inst
 U: UserInterface;  C: Calculation;  D:
    DataStore;
 bind
 C.landerValues -- D.landerValues;
    U.landerValues -- D.landerValues;
    U.calculationService --
 C.calculationService;
 }
```

Structural View

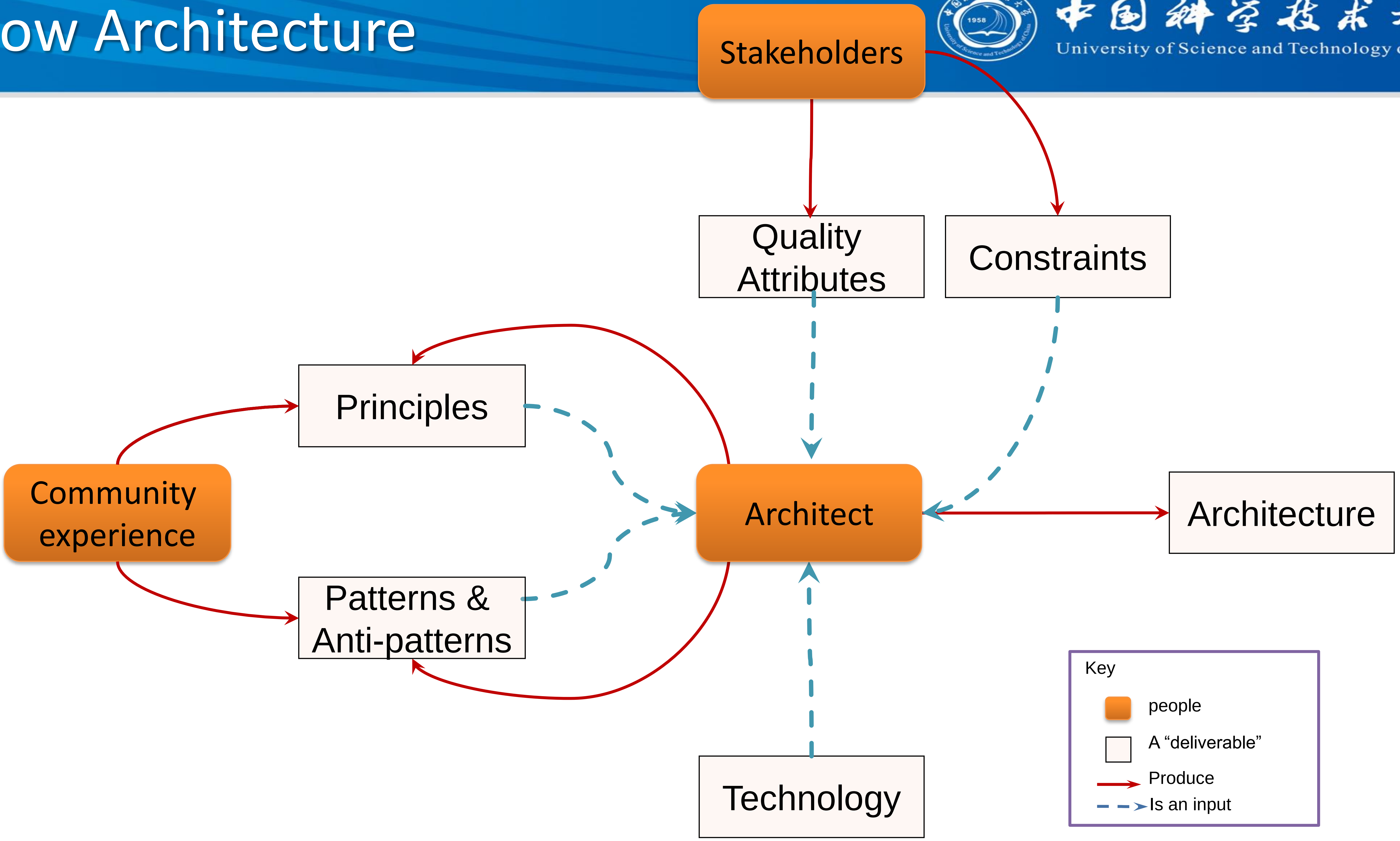


Deployment View



How Architecture

中国科学技术大学
University of Science and Technology of China



Stakeholders
Quality Attributes
Constraints
Principles
Community experience
Architect
Architecture
Patterns & Anti-patterns
Technology
Key
people
A “deliverable”
Produce
Is an input