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

Service Architectures

Outline

Component-based development (CBD)

基于组件的开发(CBD)作为一种基于重用的软件系统开发方法出现于20世纪90年代后期。
出于沮丧,OO开发并没有导致广泛的重用作为最初的建议。
组件比对象类更抽象，可以被认为是独立的服务提供者。
组件是由它们的接口定义和一般可以认为是有两个相关的接口:提供和要求。

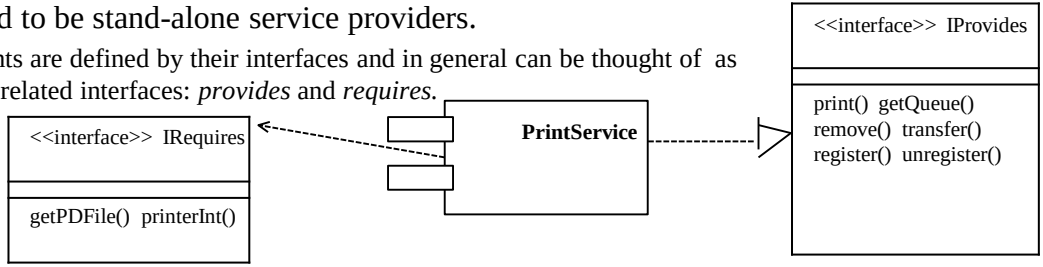
Overview

-Component-based development (CBD) emerged in the late 1990s as a reuse-based approach to software systems development.

÷It was motivated by the frustration that OO development had not led to extensive reuse as originally suggested.

-Components are more abstract than object classes and can be considered to be stand-alone service providers.

÷Components are defined by their interfaces and in general can be thought of as having two related interfaces: *provides* and *requires.*



CBD主要目的：解放程序员，实现可重用
component的运行依赖容器，利用CBD开发时，需要实现component和contract（配置文件），容器管理component的生命周期，system service（例如安全、并发等）由容器管理，程序员只需关注业务逻辑

Component Categories and Abstraction

-Software components provide a vehicle for software artifacts ***reuse***, and thereby may be used at all the levels of the software life cycle: analysis, design, implementation, and deployment.

-Hence, there are various kinds of software components:

÷*Conceptual components:* components at the analysis and design level.

÷*Implementation components:* development work product components such as source code files, data files etc.

÷*Deployment components*: involved in an executable system, such as dynamic libraries and executables.

软件组件为软件构件的重用提供了载体，因此可以在软件生命周期的所有级别上使用:分析、设计、实现和部署。
-因此，有各种各样的软件组件:
概念性组件:组件分析和设计水平。
实现组件:开发工作产品组件如源代码文件、数据文件等。
部署组件:参与一个可执行的系统,如动态库和可执行文件

-Components may also exist at different levels of abstraction.

÷*Functional abstraction*: the component implements a single function such as a mathematical function. The *provides* interface is the function.

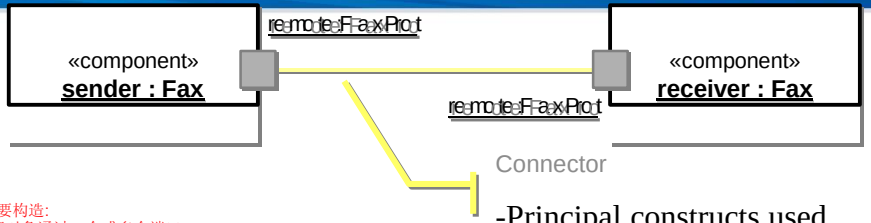
÷*Casual groupings*: the component is a collection of loosely related entities that might be data declarations, functions etc.

÷*Data abstractions*: the component represents a data abstraction or class in an OO language; the *provides* interface consists of operations to create, modify and access the data.

÷*Cluster abstractions*: the component is a group of related classes that work together (called framework); the *provides* interface is the composition of the *provides* interfaces of the objects involved.

÷*System abstraction*: the component is an entire self-contained system (also called COTS product); the *provides* interface is an API defined to allow programs to access the system commands and operations.

Constructs



在软件组件建模中使用的主要构造:
组件:与环境交互的复杂,物理对象通过一个或多个端口。
端口:边界对象实现的接口,通过该接口组件进行交互
协议:定义了有效的序列信息与周围环境之间的连接端口。

-Principal constructs used in software component modeling:

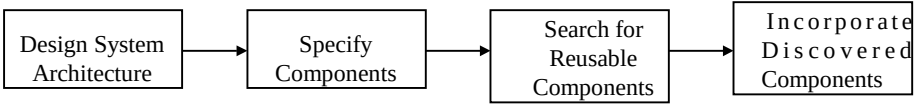
÷*Component:* complex, and physical objects that interact with their environments through one or more ports.

÷*Port:* boundary object that implements some of the interfaces through which a component interacts with its surroundings.

÷*Protocol:* defines the valid sequence of messages between connected ports

-Component-oriented development can be integrated into a system development process in one of two ways: *opportunistic reuse* and *development with reuse*. 面向组件的开发可以通过两种方式之一集成到系统开发过程中:机会重用和利用重用进行开发

Opportunistic Reuse

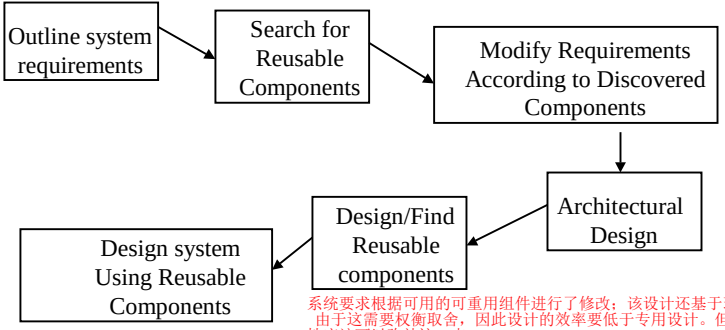


-The specifications are used to find reusable components which are then incorporated in the architecture. 规范用于寻找可重用的组件，然后将这些组件合并到体系结构中。

尽管这种方法可能导致显著的重用，但它与其他工程规程中采用的方法形成了对比

÷Although this approach may result in significant reuse, it contrasts with the approach adopted in other engineering disciplines.

Development with Reuse



系统要求根据可用的可重用组件进行了修改；该设计还基于现有组件。
由于这需要权衡取舍，因此设计的效率要低于专用设计。但是，较低的开发成本，快速交付和增强的系统可靠性应该可以弥补这一点。

-The system requirements are modified according to the reusable components available; the design is also based around existing components.

÷Since this requires some tradeoff, the design is less efficient than a special purpose design; however, lower costs of development, rapid delivery, and increased system reliability should compensate for that.

3. Component Models

软件组件符合组件模型，可以独立部署和组合，无需根据组合标准进行修改

-A software *component* conforms to a ***component model*** and can be independently deployed and composed without modification according to a ***composition*** standard.

Component Model

-A component model defines a set of standards for component development, deployment, and evolution.

-The main competing component models currently available include:

÷OMG’s CORBA Component Model (CCM),
÷Microsoft’s Distributed Component Object Model (DCOM)
÷Microsoft DotNET Framework
÷SUN Microsystems JavaBeans and Enterprise JavaBeans (EJB)

组件模型定义了一组用于组件开发、部署和发展的标准。
-目前可提供的主要竞争组件模型包括:
OMG的CORBA组件模型(CCM),
微软的分布式组件对象模型(DCOM)
微软DotNET框架
SUN Microsystems javabean和Enterprise javabean (EJB)

Basic Elements of a Component Model

组件模型的基本元素
-组件模型的基本元素包括接口标准、命名标准、元数据标准、定制标准、组合标准、演化标准和部署标准

-Basic elements of a component model include standards for interfaces, naming, meta data, customization, composition, evolution, and deployment.

Standards for	Description
Interfaces	Specification of component behavior and interfaces; definition of an Interface Definition Language (IDL)
Naming	Global unique names for interfaces and components.
Meta data	Information about components and interfaces.
Interoperability	Communication among components from different vendors, and/or implemented in different languages.
Customization	Interfaces for customizing components.
Composition	Interfaces and rules for combining components.
Evolution Support	Rules and services for evolving components.
Packaging and deployment	Packaging implementation and resources needed for installing and configuring a component.

Component Model Implementation

支持符合模型的组件的执行所需的可执行软件元素的专用集合

-Dedicated set of executable software elements required to support the execution of components that conform to the model.

-Provide:

÷A run-time environment

÷Basic Services

÷Horizontal services that are useful across multiple domains

÷Vertical services providing functionality for a particular domain for software components.

Overview of the CCM

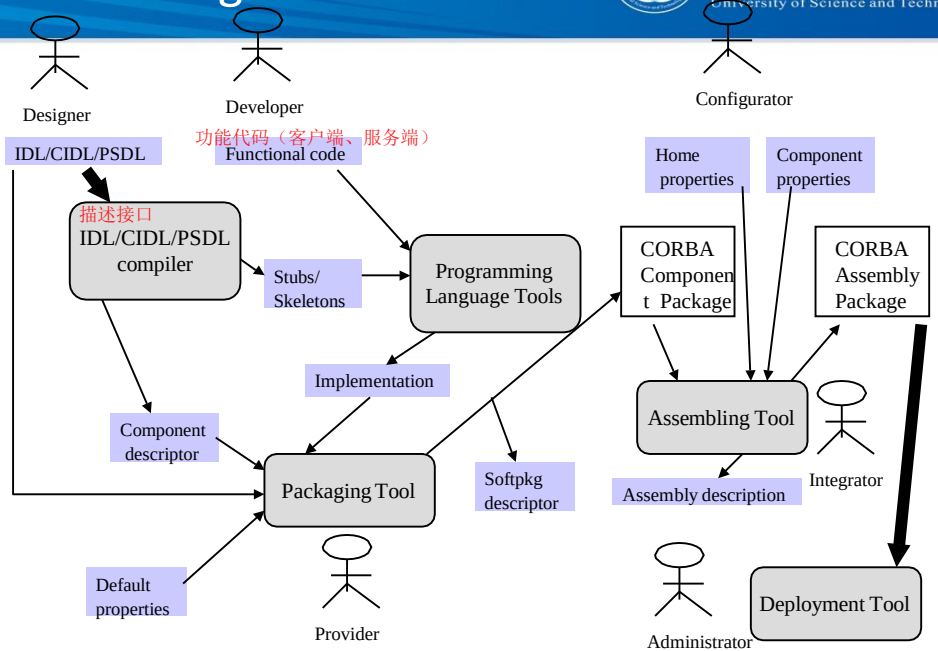
-The goals of the CORBA Component Model (CCM), like any other component model (e.g., DCOM, EJB etc.) is to facilitate reuse of CORBA applications.

-The CCM extends the standard CORBA Interface Definition Language (IDL) by including specific features for component description.

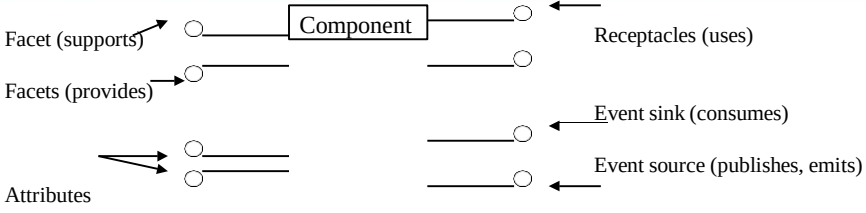
-The CCM also introduces a new declarative language, named the ***Component Implementation Definition Language (CIDL)***, which is used by code generators to generate code needed to deploy the components (in containers).

-Developers have to deal only with the development of the components and their inherent logic and functionality.

CBD Process using the CCM



Component Model



-The CCM defines a *component* type to represent component instances.

-Component type definitions consist of a collection of *ports* definitions. The CCM defines 2 kinds of ports: ***facets*** and ***configuration ports***.

÷***Facets:*** consist of a set of interfaces that define the functionality supported or provided by the component.

÷***Configuration ports:*** correspond to a set of interfaces that specify how a component may interconnect and communicate with other components.

Several kinds of ports supported by the CCM, namely receptacle, attribute, emitter, publisher, and consumer.

-*Receptacles:* specify the external dependencies of the component, by describing the interfaces used by the component.

-*Attributes:* describe the properties of the component, and thereby serve as medium for their configuration and customization.

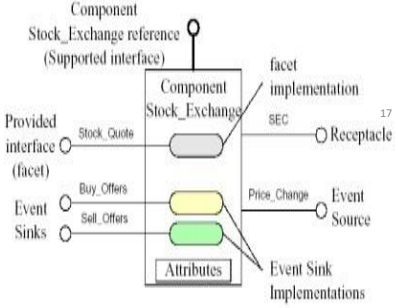
-*Event sources*: specify the events published by the component; two forms of events can be generated by the component:

÷*Publisher:* events for which the component is exclusive provider

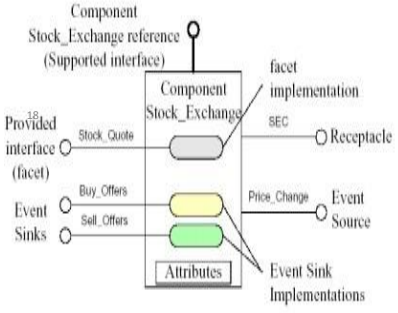
÷*Emitters:* events that share event channels with other event sources

-*Event sinks:* specify the events consumed by the component.

An example CCM Component



An example CCM Component With IDL Specification



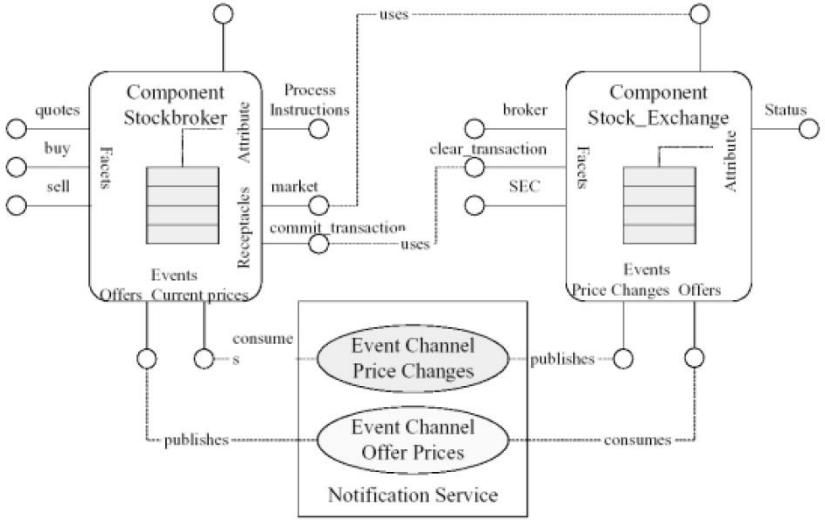

```
interface Sell, Buy;

// Define an equivalent, supported interfaces
component Stock_Exchange supports Sell, Buy {
  provides Stock_Quote;     // Facet

  consumes Buy_Offers;      // Event Sinks
  consumes Sell_Offers;

  publishes Price_Change;   // Event Source
  uses SEC;                 // Receptacle
    ...                     // Other definitions
};
```

Example of CCM Components Interactions



Note: CCM components interact through port mechanisms

-Represents the run-time environment of component instances.

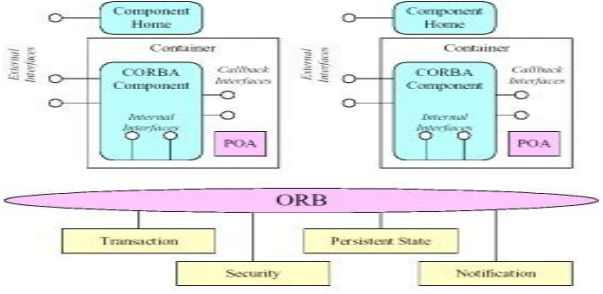
÷The CORBA component container implements component access to global system services such as transactions, security, events, and persistence.

÷The container reuses the existing CORBA infrastructure. In doing so, the inherent complexity of CORBA is hidden both to the developer and to the container.

-Container and component instances interact through two kinds of interfaces:

÷*Internal API:* a set of interfaces provided by the container to component implementations.

÷*Callback Interfaces:* a set of interfaces provided by component implementations to the container.



Web Services

Service Architecture for the Web

Web as a database

- The Web we use is full of data
- Book information, opinions, prices, arrival times, blogs, tags, tweets, etc.
- The data is organized around a simple data model: node-link model Each node is a data item that has a unique address and a
- representation
- Representation formats are e.g. HTML, PDF,... for humans, or e.g XML, JSON for programs
- Nodes can be interlinked using their unique addresses

我们使用的网络充满了数据
•预订信息、意见、价格、到达时间、博客、标签、推文等。
•数据是围绕一个简单的数据模型组织的:节点-链接模型，每个节点是一个数据项，有一个唯一的地址和一个表示
•表示格式如HTML，PDF，…用于人类，或者用于程序的XML、JSON
•节点可以使用其唯一的地址进行互连

Web的理念与CORBA的远程过程调用相同，只是标准不同

Web as a platform for distributed systems

- The Web is full of services that allow humans and programs to use the Web data
- Services also have unique addresses
- They use a particular representation for data exchange, e.g. XML, SOAP, WSDL
- Services follow a particular architecture that defines how services are used
- Programmers combine a number of services to achieve a desired functionality and create a distributed system, e.g. mashups

Web service关注B2B，程序与程序之间的交互，不同程序语言开发的程序之间的交互

Web中充满了允许人类和程序使用Web数据的服务
•服务也有唯一的地址
•它们使用特定的表示来进行数据交换，例如XML、SOAP、WSDL
•服务遵循特定的架构，定义了如何使用服务
•程序员结合大量的服务来实现期望的功能，并创建一个分布式系统，例如mashu

Types of services

- What is the Google search engine?
  - It is a **service** for querying a massive database (Web search index)
- What is a given Web application?
  - It is a **service** offering (remotely) a specific functionality
- What is a Web site?
  - It is a **service** offering specific human consumable information

谷歌搜索引擎是什么?
•它是一个用于查询海量数据库(Web搜索索引)的服务
•什么是给定的Web应用程序?
•它是一种提供(远程)特定功能的服务
•什么是网站?
•它是一种提供特定人类可消费信息的服务

Types of services

- All of these services are for users
- However, we are interested in services for programmers Such services provide an API
- Programmers use the API, unique addresses, representations of services
- Programmers follow the arch. style to integrate and combine services to achieve a desired functionality
- We will call this part of the Web: the programmable Web

所有这些服务都是为用户提供的
•然而，我们感兴趣的是为程序员提供的服务，这些服务提供API
•程序员使用API、唯一地址和服务表示
•程序员遵循拱门。样式来集成和组合服务以实现所需的功能
•我们将把这部分Web称为:可编程Web

Kind of Things on the Programmable Web

- There are numerous approaches to web services in all areas
- The programmable Web is based on HTTP for data transport and in most cases XML or JSON for data representation
- However, some services serve HTML, plain text, binary data, etc. Also, other things such as addressability or APIs are different
- We need a classification!

所有领域都有许多web服务的方法
•可编程Web基于HTTP进行数据传输，在大多数情况下基于XML或JSON进行数据表示
•然而，一些服务提供HTML、纯文本、二进制数据等。同样，其他的东西，如寻址性或api也是不同的
•我们需要分类！

Classification based on architectural design

- Which operation should a service execute? This is **method information**
- What data should be manipulated? This is **scoping information**

服务应该执行哪个操作?这是方法信息
•应该操纵哪些数据?这是范围信息

Method information

- Question: how the client conveys its intention to the server?
- How does a server know a certain request is a request to retrieve some data?
- Instead of a request to delete the same data?
- Why should the server do **this** instead of doing **that**

问:客户机如何向服务器传达其意图?
•服务器如何知道一个请求是一个检索数据的请求?
•而不是请求删除相同的数据?
•为什么服务器应该这样做而不是那样做

Web Service Architectures

REST & Web-Services

Competing Architectures

- Resource-Oriented Architectures (RESTful) 面向资源的架构
- RPC-Style Architectures (SOA)
- REST-RPC Hybrid Architectures

Resource-Oriented Architectures

- **Resource-Oriented Architectures**
- Descriptive URLs
- URLs reflect the application state Inherit semantic from HTTP methods Should be stateless
- Limited by its simplicity (limited HTTP methods, ...)

面向资源的架构
•描述url
URL反映应用程序的状态
从HTTP方法继承的语义应该是无状态的
•受其简单性的限制(有限的HTTP方法，…)

RPC-Style Architectures

- **RPC - Remote Procedure Call**
- An RPC style service receives an envelope full of data from the client
- The service answers with a similar envelope again full of data to the client
- Both method and scoping information are inside of the envelope

HTTP methods typically POST but sometimes also GET

RPC传送方法和数据，REST只传送数据

远程过程调用
•RPC样式的服务从客户端接收一个装满数据的信封
•服务用一个同样装满数据的信封回复客户
•方法和范围信息都在信封内
HTTP方法通常是POST，但有时也会GET

RPC-Style Architectures

- The best example of the envelope format is SOAP There exist other envelope formats like XML-RPC
- Every RPC-style service defines a completely new vocabulary
- E.g. the way how method information and scoping information are represented
- You need another language to define the representation: e.g. WSDL

信封格式的最好例子是SOAP，还有其他的信封格式，如XML-RPC
•每个rpc样式的服务都定义了一个全新的词汇表
•例如方法信息和范围信息是如何表示的
•您需要另一种语言来定义表示:例如WSDL

Problems of RPC-Style Architectures

- RPC implies an API
- APIs tend to enforce tight coupling of modules and systems We use declarative XML to describe APIs
- This introduces processing overhead

RPC意味着一个API
api倾向于加强模块和系统的紧密耦合
我们使用声明性XML来描述api
•这会引入处理开销

REST-RPC Hybrid Architectures

- **REST-RPC Hybrid Architectures**
- Inherit parts from REST and RPC style architectures
- Used by many Web Site for their API (e.g. Flickr, del.icio.us, …)

REST-RPC混合架构
•从REST和RPC风格架构中继承部分
•用于许多网站的API(例如。
Flickr，…)

• Resource Oriented Architectures

- Theory Behind REST

出现比Web Service晚

Resource Oriented Architectures

- 4 defining features of ROA
- **Addressability**: the scoping information is kept in the URL
- **Uniform interface**: the method information is kept in the HTTP method
- **Statelessness**: every HTTP request is isolated from other requests
- **Connectedness**: you link resources into the Web of resources

利用HTTP的方法

ROA定义的4个特征
•可寻址性：范围信息保存在URL中
统一接口：方法信息保存在HTTP方法中
•无状态：每个HTTP请求都与其他请求隔离
•连通性：你将资源连接到资源网络中

Addressability

- Resources are exposed through URLs - an application exposes a number of URLs
- When you have URLs you bookmark, cache responses, chain URLs,
- ...
- Many Web applications do not work this way, i.e. they are not addressable, e.g. GMail

Uniform interface

- Standardized HTTP methods: CRUD operations (**C**reate, **R**etreive, **U**pdate & **D**elete) 简单，效率高
- Two principles
- **Safety**: GET only reads data
- **Idempotence**: the same operation has the same effect whether you apply it once or multiple times

标准化的HTTP方法:CRUD操作(创建，检索，更新和删除)
•两个原则
安全性:GET只读取数据
•幂等:相同的操作无论应用一次或多次都有相同的效果

Statelessness

- Every request contains all necessary information
- There is no state managed on the server side
- In fact, there are two kinds of state
- You should distinguish between the application state and resource state
- Application state lives on the client Resource state lives on the server

Statelessness

- When you use a search engine your current query and your current page belongs to application state
- They are different for every client
- Resource state is same for every client, i.e. search index A crawler can update the search index

Links and connectedness

- Representations of resources, i.e. HTML or XML might have links to other resources
- Axiom for ROA services: Hypermedia as the engine of application state
- The current application state is not stored on the server as a resource state
- It is tracked by the client as an application state and created by the path that client takes through the Web

Links and connectedness

- For example
  http://www.google.com/search?q=jellyfish
- The first page is the starting application state You have links to other application states Obvious for the human Web

Links and connectedness


```
<Buckets>
<Bucket>
<Name>crummy.com</Name>
<URL>https://s3.amazonaws.com/crummy.com</URL>
<CreationDate>...</CreationDate>
</Bucket>
...
</Buckets>
```

- Following the link in the URL element takes the client to a new application state
- Use links, links, and then use more links, ...

Designing ROA

- Figure out data set
- Split the data set into resources Then, for each resource
- Name the resources with URLs
- Expose a subset of the uniform interface

Designing ROA

- Design representations accepted from the client
- Design representations served to the client
- Integrate this resource into other resources using links
- Consider possible application states
- Consider possible error states

Example

- Application similar google maps: maps of cities, streets, planets,... Data set: Maps, points, cities, planets, ...
- Resources: list of resources, individual resources, results of algorithms applied to the data set
- Example resources: the list of planets, Mars, Earth, San Francisco, Inffeldgasse, ...
- An algorithmic resource: a list of places that match certain criteria - all cities with more than 1 million of people

Example

- Name the resources: create meaningful URLs
- http://maps.example.com/Earth
- http://maps.example.com/Earth/France/Paris
- http://maps.example.com/Earth/Austria/Cities
- http://maps.example.com/Earth/Germany/Cities?pop=1000000

Example

- Design representations
- A representation talks about resource state
- A representation links to other (application and resource) states
- http://maps.example.com/Earth/Austria →
- http://maps.example.com/Earth/Austria/Cities
- http://maps.example.com/Earth/Austria/Cities → http://maps.example.com/Earth/Austria/Vienna

Read/Write ROA systems

- User accounts should be resources
- To access these resources you need to use HTTP authentication
  - https://maps.example.com/user/rkern
- Connect with the previous resources: e.g. custom places on a map
  - https://maps.example.com/user/rkern/Earth/Graz/Inffeldgasse/office

Method Semantic in ROA systems

- The HTTP method implies the semantics Use GET to retrieve (read the content)
- `GET https://maps.example.com/user/rkern`
- Use PUT to write (modify the content)
- `PUT https://maps.example.com/user/rkern`
- + Content
- Use DELETE to write (remove the resource)
- `DELETE https://maps.example.com/user/rkern`
- ⇒ limited to the methods supported by HTTP

REST

- **REST** (Representational State Transfer) No strict typing
- Typically XML as data format
- Concepts (RESTful) describes constrains: client-server, stateless, cacheable, layered system, code on demand, uniformity
- Nowadays, REST is often used a umbrella terms for related architectures

RESTful frameworks

- Ruby on Rails with a plugin
- Django in Python
- Restlet in Java (http://www.restlet.org/)
- Jersy