Lecture 6 – Web Services, ReST Services and Twitter demo

Professor Richard O. Sinnott, Farzad Khodadadi
University of Melbourne
rsinnott@unimelb.edu.au

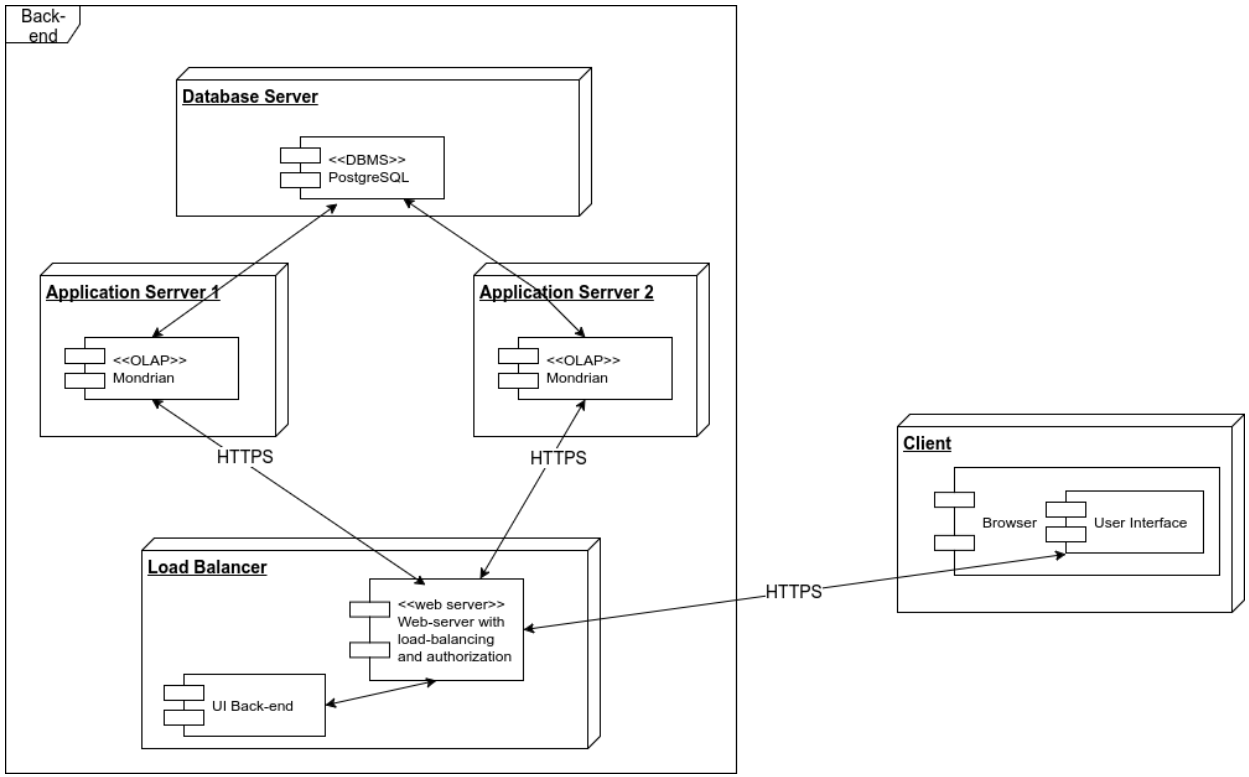
Lecture 6.1 – Web Services

Professor Richard O. Sinnott
Director, eResearch
University of Melbourne
rsinnott@unimelb.edu.au

What's in an Architecture?

A (system) architecture, is just the way different software components are distributed on computers, and the way in which they interact with each other

Architectures are often difficult to describe in words, hence diagrams are often used. Many ways to document them, e.g. UML, …



Service-oriented Architectures?

When an architecture is completely contained within the same machine, components can communicate directly, e.g. through *function calls* or *object instantiations.*

However, when components are distributed such a direct approach typically cannot be used (e.g. Assignment 2!)

Therefore, components (more properly, *systems*) have to interact in more loosely-coupled ways.

Services are often used for this. Typically *combinations* and *commonality* of services can be used to form a Service-oriented Architecture (SoA).

SOA Core Goals

A Service-oriented Architecture can serve many goals:

- A set of externally facing services that a business wants to provide to external collaborators
- An architectural pattern based on service providers, one or more brokers, and service requestors based on agreed service descriptions
- A set of architectural principles, patterns and criteria that support modularity, encapsulation, loose coupling, separation of concerns, reuse and composability
- A programming model complete with standards, tools and technologies that supports development and support of services (note that there can be many flavours of services)
- A middleware solution optimized for service assembly, orchestration, monitoring, and management, e.g. as workflows.

SOA Design Principles

- **Standardized service contract**: *Services adhere to a communications agreement, as defined collectively by one or more service-description documents.*
- **Service loose coupling**: *Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.*
- **Service abstraction**: *Beyond descriptions in the service contract, services hide logic from the outside world.*
- **Service reusability**: *Logic is divided into services with the intention of promoting reuse.*
- **Service autonomy**: *Services have control over the logic they encapsulate.*
- **Service statelessness**: *Services minimize resource consumption by deferring the management of state information when necessary.*
- **Service discoverability**: *Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.*
- **Service composability**: *Services are effective composition participants, regardless of the size and complexity of the composition.*

SOA Design Principles…ctd

- **Service granularity**: *a design consideration to provide optimal scope at the right granular level of the business functionality in a service operation.*
- **Service normalization**: *services are decomposed and/or consolidated to a level that minimizes redundancy, for performance optimization, access, and aggregation.*
- **Service optimization**: *high-quality services that serve specific functions are generally preferable to general purpose low-quality ones.*
- **Service relevance**: *functionality is presented at a level of granularity recognized by the user as a meaningful service.*
- **Service encapsulation**: *many services are consolidated for use under a SOA and their inner workings hidden.*
- **Service location transparency**: *the ability of a service consumer to invoke a service regardless of its actual location in the network.*

SOA for the Web: Web Services

- **Web** services used to implement service-oriented architectures
- Two main flavours
  - SOAP-based Web Services
  - ReST-based Web Services

- Both use HTTP, hence can run over the web
  - (although SOAP/WS often run over other protocols as well)

- There are MANY other flavours of web service:
  - Geospatial services (WFS, WMS, WPS…)
  - Health services (HL7)
  - SDMX (Statistical Data Markup eXchange)

SOAP/WS vs ReST

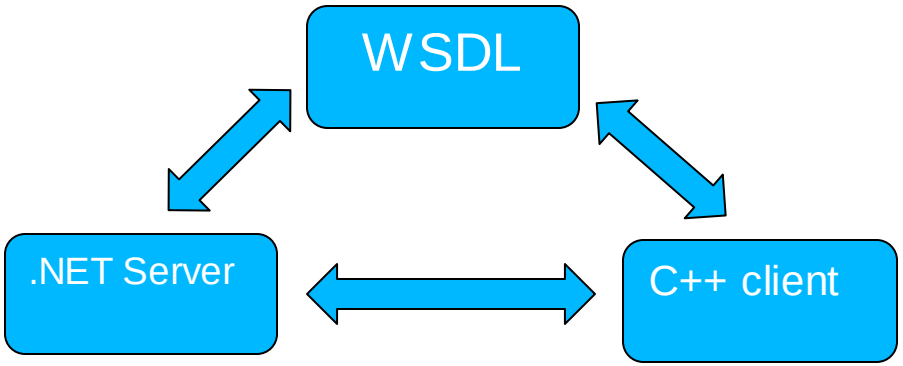
- Two patterns to call services over HTTP
  - SOAP/WS is built upon the *Remote Procedure Call paradigm*;
    - a language independent function call that spans another system
  - ReST is centered around *resources*, and the way they can be manipulated (added, deleted, etc.) remotely
    - (Examples later)
  - Actually ReST is more of a style to use HTTP than a separate protocol
  - ...while SOAP/WS is a stack of protocols that covers every aspect of using a remote service, from service discovery, to service description, to the actual request/response

WSDL

The *Web Services Description Language* (WSDL) is an XML-based interface description language that describes the functionality offered by a web service.

WSDL provides a *machine-readable* description of how the service can be called, what parameters it expects, and what results/data structures it returns:

Definition – what it does
Target Namespace – context for naming things
Data Types – simple/complex data structures inputs/outputs
Messages – messages and structures exchanged between client and server
Port Type - encapsulate input/output messages into one logical operation
Bindings - bind the operation to the particular port type
Service - name given to the web service itself
Language independence and location transparency…



Simple WSDL Example

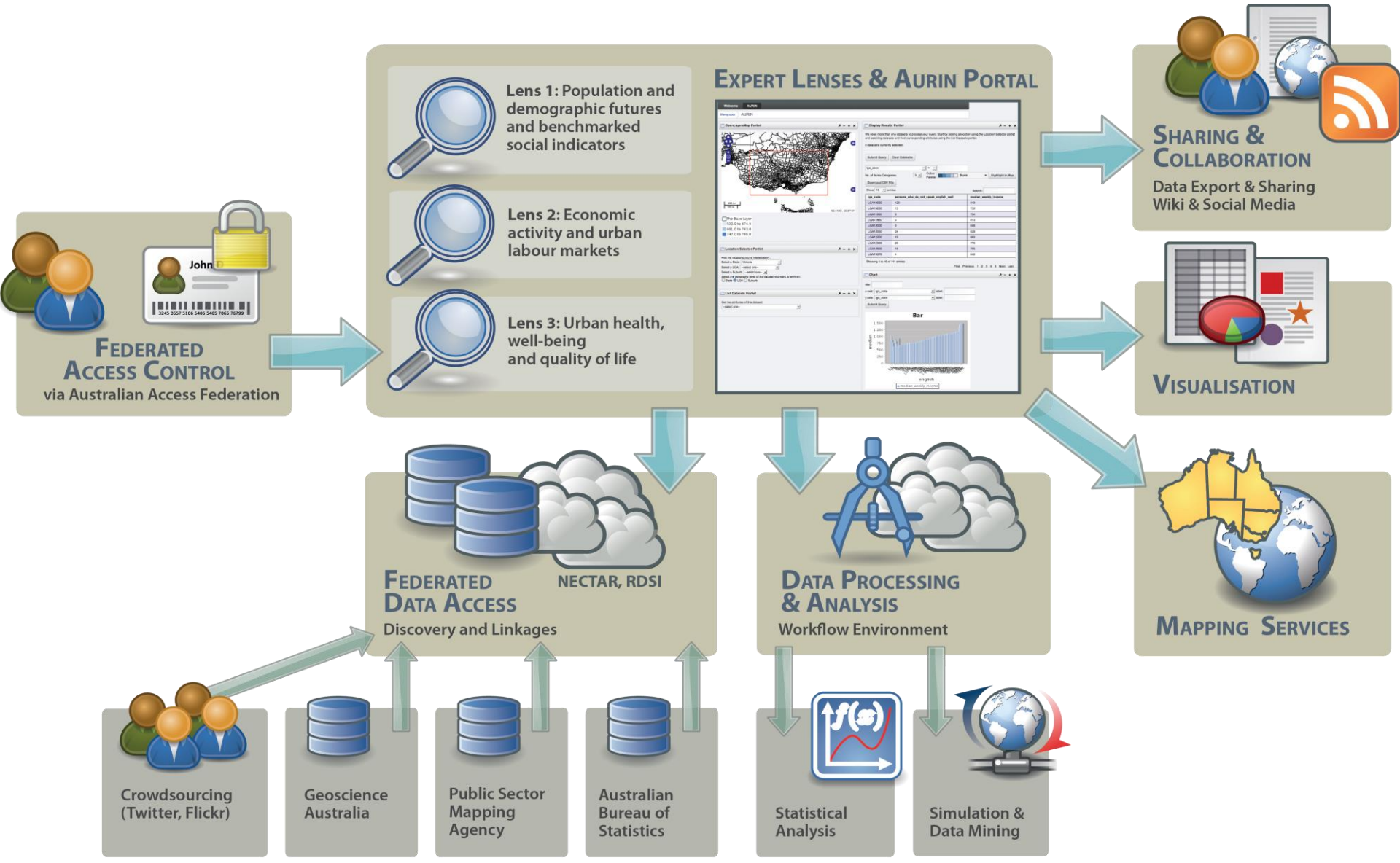
```
<definitions name = "HelloService"
   targetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns = "http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <message name = "SayHelloRequest">
      <part name = "firstName" type = "xsd:string"/>
   </message>
   <message name = "SayHelloResponse">
      <part name = "greeting" type = "xsd:string"/>
   </message>
   <portType name = "Hello_PortType">
      <operation name = "sayHello">
         <input message = "tns:SayHelloRequest"/>
         <output message = "tns:SayHelloResponse"/>
      </operation>
   </portType>
```

```
<binding name = "Hello_Binding" type = "tns:Hello_PortType">
      <soap:binding style = "rpc"
         transport = "http://schemas.xmlsoap.org/soap/http"/>
      <operation name = "sayHello">
         <soap:operation soapAction = "sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use = "encoded"/>
         </input>
         <output>
            <soap:body
               encodingStyle=http://schemas.xmlsoap.org/soap/encoding/"
               namespace = "urn:examples:helloservice"
               use = "encoded"/>
         </output>
      </operation>
   </binding>
   <service name = "Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding = "tns:Hello_Binding" name = "Hello_Port">
         <soap:address
            location = "http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

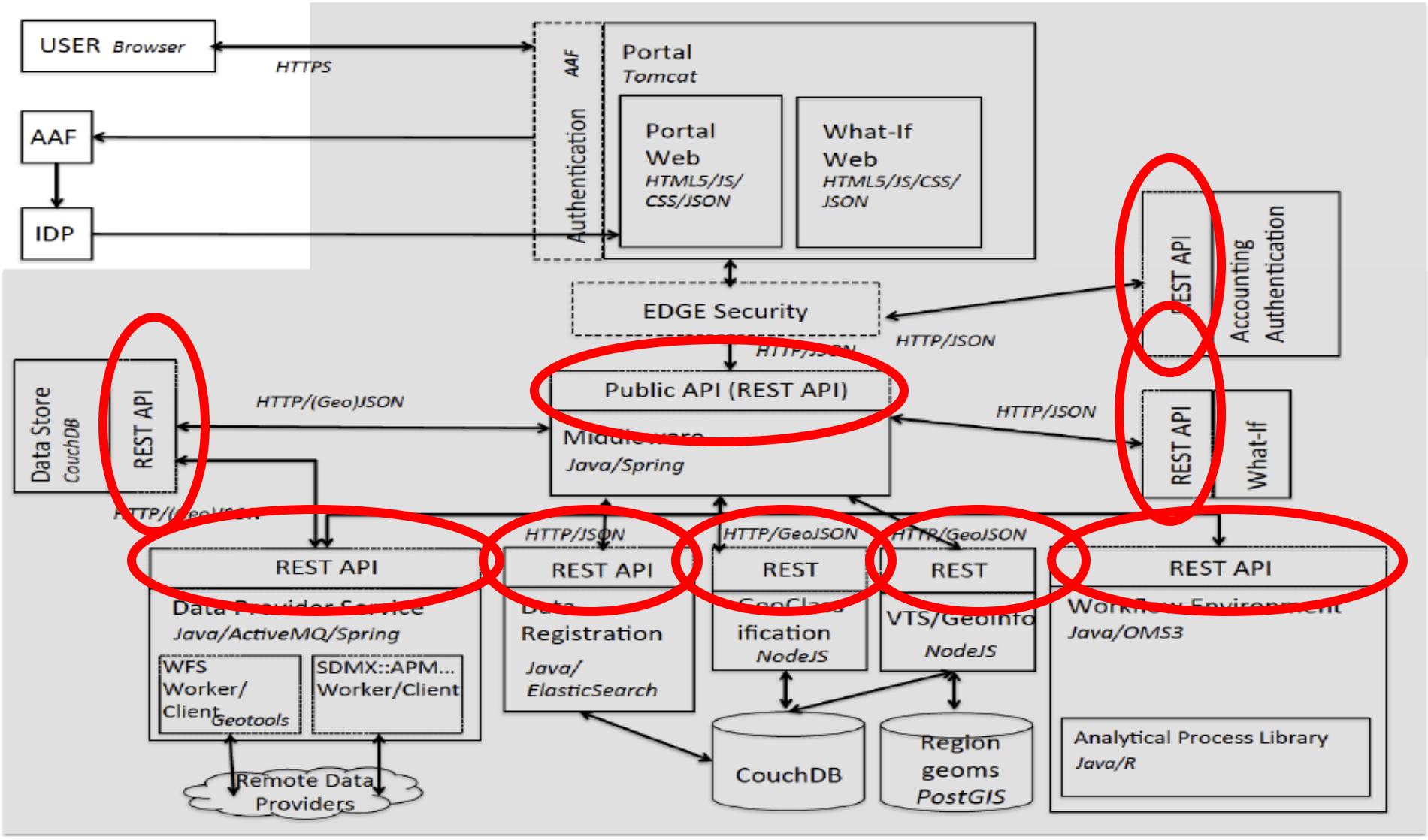
Note

- SOAP for exchange of structured information
- Specific names for operations
- Specific data types
- Specific bindings
- …

Remembering AURIN Simplified



Example of AURIN SoA



Lecture 6.2 – ReST-based Services

What is ReST?

"*Representational State Transfer (ReST)* is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

- Roy Fielding

What's in a Name?

1) Clients requests Resource through Identifier (URL)
2) Server/proxy sends representation of Resource
3) This puts the client in a certain state.
4) Representation contains URLs allowing navigation.
5) Client follows URL to fetch another resource.
6) This transitions client into yet another state.
7) Representational **S**tate **T**ransfer!

Resource-Oriented Architecture (ROA)

- A ROA is a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web
- A resource is anything that's important enough to be referenced as a thing in itself.
- If your users might
  - want to create a hypertext link to it
  - make or refute assertions about it
  - retrieve or cache a representation of it
  - include all or part of it by reference into another representation
  - annotate it
  - or perform other operations on it

  ...then you should make it a resource.

Resource and URL Examples

Resource	Potential URL
Version 1.0.3 of the software release	http://www.example.com/software/releases/1.0.3.tar.gz
The latest version of the software release	http://www.example.com/software/releases/latest.tar.gz
The first weblog entry for October 24, 2006	http://www.example.com/weblog/2006/10/24/0
A road map of Little Rock, Arkansas	http://www.example.com/map/roads/USA/AR/Little_Rock
Some information about jellyfish	http://www.example.com/wiki/Jellyfish
A list of the open bugs in the bug database	http://www.example.com/bugs/by-state/open
The relationship between two acquaintances, Alice and Bob	http://www.example.com/relationships/Alice;Bob

Mapping Actions to HTTP Methods

ACTION	HTTP METHOD
Create Resource	PUT to a new URI POST to an existing URI
Retrieve Resource	GET
Update Resource	POST to an existing URI
Delete Resource	DELETE

- Common mistake: Always mapping PUT to Update and POST to create
- PUT should be used when target resource url is known by the client
- POST should be used when target resource URL is server generated.

A Generic ROA Procedure

1. Figure out the data set
2. Split the data set into resources and for each kind of resource
3. Name the resources with URIs
4. Expose a subset of the uniform interface
5. Design the representation(s) accepted from the client
6. Design the representation(s) served to the client
7. Integrate this resource into existing resources, using hypermedia links and forms
8. Consider the typical course of events: what's supposed to happen?
9. Consider error conditions: what might go wrong?

ReST Best Practices #1

1) Keep your URIs short – and create URIs that don't change.
2) URIs should be opaque identifiers that are meant to be discovered by following hyperlinks, not constructed by the client.
3) Use nouns, not verbs in URLs
4) Make all HTTP GETs side-effect free. Doing so makes the request "safe".
5) Use links in your responses to requests! Doing so connects your response with other data. It enables client applications to be "self-propelled". That is, the response itself contains info about "what's the next step to take". Contrast this to responses that do not contain links. Thus, the decision of "what's the next step to take" must be made out-of-band.

ReST Best Practices #2

6) Minimize the use of query strings. For example:
Prefer:
`http://www.amazon.com/products/AXFC`
Over:
`http://www.amazon.com/products?product-id=AXFC`

7) Use HTTP status codes to convey errors/success

200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Auth Required
408 Request Timeout
409 Conflict

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 Version Not Supported

ReST Best Practices #3

8) In general, keep the REST principles in mind.

In particular:

- Addressability
- Uniform Interface
- Resources and Representations instead of RPC
- HATEOAS

ReST – Uniform Interface

Uniform Interface has four more constraints:

- Identification of Resources

  All important resources are identified by one (uniform) resource identifier mechanism (e.g. HTTP URL)

- Manipulation of Resources through representations

  Each resource can have one or more representations. Such as application/xml, application/json, text/html, etc. Clients and servers negotiate to select representation.

- Self-descriptive messages

  Requests and responses contain not only data but additional headers describing how the content should be handled. Such as if it should be cached, authentication requirements, etc. Access methods (actions) mean the same for all resources (universal semantics)

(HTTP GET, HEAD, OPTIONS, PUT, POST, DELETE, CONNECTION, TRACE, PATCH)

ReST – Uniform Interface - HATEOAS

- HATEOAS – Hyper Media as the Engine of Application State
- Resource representations contain links to identified resources
- Resources and state can be used by navigating links
  - links make interconnected resources navigable
  - without navigation, identifying new resources is service-specific
- RESTful applications *navigate* instead of *calling*
  - representations contain information about possible traversals
  - application navigates to the next resource depending on link semantics
  - navigation can be delegated since all links use identifiers

Making Resources Navigable

- Essential for using Hypermedia Driven Application State
- RPC-oriented systems need to expose the available functions
  - functions are essential for interacting with a service
  - introspection or interface descriptions make functions discoverable
- ReSTful systems use a Uniform Interface
  - no need to learn about functions
  - but how to find resources?
    - find them by following links from other resources
    - learn about them by using URI Templates
    - understand them by recognizing representations

HTTP Methods

- HTTP methods can be
  - Safe
  - Idempotent
  - Neither
- Safe methods
  - Do not change repeating a call is equivalent to not making a call at all.
- Idempotent methods
  - Effect of repeating a call is equivalent to making a single call
- GET, OPTIONS, HEAD – Safe
- PUT, DELETE – Idempotent
- POST – Neither safe nor idempotent

Demonstration

References

- *Restful Web Services*, Leonard Richardson and Sam Ruby, 2007, O'Reilly.

- *Architectural Styles and the Design of Network-based Software Architectures*. Fielding, Roy Thomas, UC - Irvine, 2000.