MONASH University

Information Technology

FIT5183: Mobile and Distributed Computing Systems (MDCS)

Lecture 3B
REST Architecture & RESTful Web Services

OUTLINE

- Understanding REST Architecture
- RESTful Web Services
- Using JAX-RS to create RESTful WS

Slides are adapted from below sources:

Roy Fielding's PhD thesis
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

The Java EE 6 Tutorial at
http://docs.oracle.com/javaee/6/tutorial/doc/gjbji.html

A SOAP Sample

POST /StockQuote HTTP/1.1

SOAPAction: “Some-URI”

```
<soapenv:Envelope
   xmlns:soapenv=".....">
   <soapenv:Header/>
   <soapenv:Body>
      <it:GetEmployeeName>
         <it:ID>4118830901957010</it:ID>
      </it:GetEmployeeName>
   </soapenv:Body>
</soapenv:Envelope>
```

With the use of corresponding WSDL…

```
<definitions>

  <types>
    definition of types.....
  </types>

  <message>
    definition of a message....
  </message>

  <portType>
    definition of a port.....
  </portType>

  <binding>
    definition of a binding....
  </binding>

</definitions>
```

You may have wondered..

Why is there is so much overhead in designing a simple web service like ‘GetEmployeeName’ using SOAP?

Understanding REST Architecture

REST (REpresentational State Transfer)

- **REST is an Architecture**
  - NOT a technology or standards specification or a protocol
  - REST was introduced in Roy Fielding's PhD thesis
  - It consists of elements and relationships between elements
  - Includes **architectural and interface constraints** for controlling the roles/features of these elements and also their allowed relationships

- While REST is not a standard, it **uses standards**:
  - HTTP
  - URL
  - XML/HTML/JSON/etc
  - MIME Types or Media types such as text/xml, image/gif, application/json, audio/mpeg

REST Definition

REST was introduced in Roy Fielding's PhD thesis and defined as:

> "**RE**presentational **S**tate **T**ransfer 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 PhD, 2000

REST Concepts

- Emphasis is on Resources (**nouns**)

Identified by a URI, For example: http://www.parts-depot.com/parts

- Uniform interface (**verbs**)
  - Small fixed set: Create, Read, Update, Delete (CRUD)

- State Representations
  - data and state transferred between client and server XML, JSON, Atom, XHTML, ...

Create, Read, Update and Destroy (CRUD)

- Four basic functions of persistent storage.
- Correspond to main actions (verbs) acting upon RESTful resources, which are all identifiable by URIs (nouns)
- Associated actions in HTTP, SQL (and DDS):

Operation	SQL	HTTP	DDS
Create	INSERT	PUT / POST	write
Read (Retrieve)	SELECT	GET	read / take
Update (Modify)	UPDATE	PUT / POST / PATCH	write
Delete (Destroy)	DELETE	DELETE	dispose

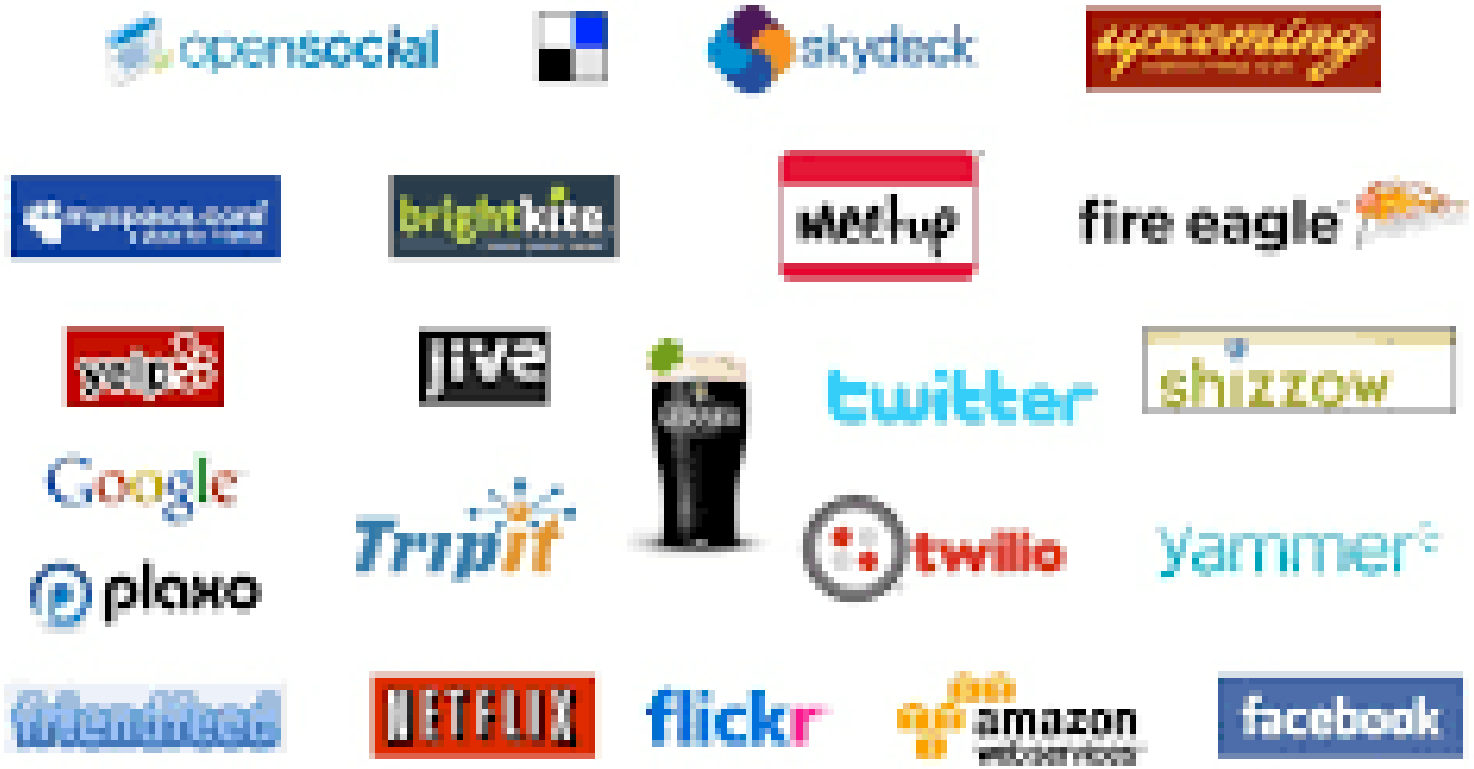
Source: Wikipedia

HTTP Example

Examples on the web

- Google AJAX Search API
  - http://code.google.com/apis/ajaxsearch/
- Amazon S3
  - http://aws.amazon.com/s3
- Services exposing Atom Publishing Protocol or GData
  - i.e. Google apps like Google Calendar
- Accidentally RESTful
  - Flickr, Del.icio.us API

REST web service API's



Architectural Constraints

1. Client/Server
2. Stateless
3. Cacheable
4. Uniform Interface
5. Layered Systems
6. Code-On-Demand

Architectural Constraints - 1

Client-Server

- The classic client-server architectural style
- ‘Separation of concerns’:

  - Separating the user interface concerns (e.g. of clients) from the data storage concerns (e.g. of servers)
  - Allows the components to **evolve independently**
  - Improves **scalability** by simplifying the server components
  - Improve the **portability** of the user interface across multiple platforms

Architectural Constraints - 2

Stateless

- "Communication must be stateless in nature"
  - Each request from client to server must contain all of the information necessary to understand the request
  - It cannot take advantage of any stored context on the server
  - This constraint induces **visibility**, **reliability**, and **scalability**:
    - **Visibility** is improved because a monitoring system does not have to look beyond a single request
    - **Reliability** is improved because it eases the task of recovering from partial failures
    - **Scalability** is improved because not having to store state between requests allows the server component to quickly free resources

Architectural Constraints - 3

Cacheable

- "The data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable."
  - "If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests."
  - Improves network **efficiency** and **performance** by "reducing the average latency of a series of interactions"
  - However, **Decreases reliability** if stale data within cache differs significantly from data that would have been obtained had request been sent directly to the server"

Architectural Constraints - 4

Uniform Interface

- All resources are accessed with a generic interface.
- Four (main) verbs: HTTP **GET, POST, PUT, DELETE**
  - **PATCH** is a method that enables updating part of the resources
  - **HEAD** is similar to **GET** but with no response body (header only)
  - **OPTIONS** a request for information about communication options
- The overall system architecture is greatly **simplified**
- But.. **degrades efficiency**, since information is transferred in a standardized form rather than one which is specific to an application's needs
- Well suited to coarse-grained hypermedia interactions typical of WWW

Architectural Constraints - 5

Layered System

- Allows an architecture to be composed of hierarchical layers
  - Each component cannot "see" beyond the immediate layer with which they are interacting
  - Clients have no knowledge that services they invoke may also invoke other services
- Layered systems improve system **scalability** by enabling load balancing of services across multiple networks and processors
- Their primary disadvantage is that they add overhead and **latency** to the processing of data, reducing user-perceived **performance**

Architectural Constraints - 6

Code-On-Demand

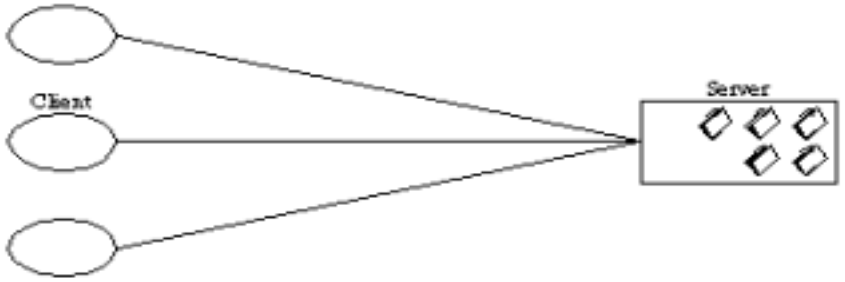
- An optional constraint
- “Allows client functionality to be extended by downloading and executing code in the form of applets or scripts.”

“The notion of an optional constraint may seem like an oxymoron. However, it does have a purpose in the architectural design of a system that encompasses multiple organizational boundaries. It means that the architecture only gains the benefit (and suffers the disadvantages) of the optional constraints when they are known to be in effect for some realm of the overall system. For example, if all of the client software within an organization is known to support Java applets [45], then services within that organization can be constructed such that they gain the benefit of enhanced functionality via downloadable Java classes. At the same time, however, the organization's firewall may prevent the transfer of Java applets from external sources, and thus to the rest of the Web it will appear as if those clients do not support code-on-demand. An optional constraint allows us to design an architecture that supports the desired behavior in the general case, but with the understanding that it may be disabled within some contexts.”

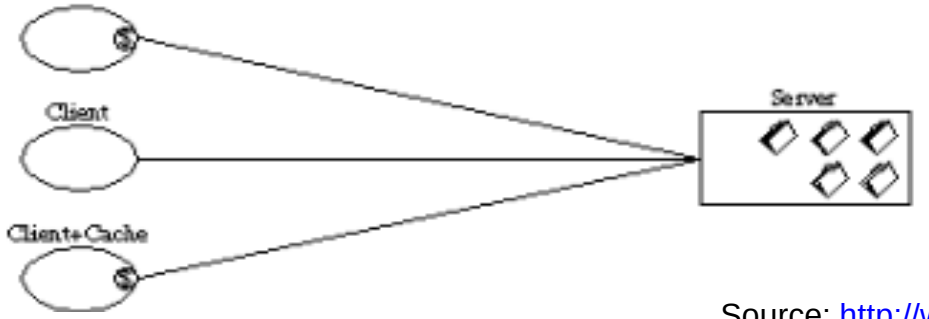
- Roy Fielding PhD, 2000

Evolution of REST Architecture

Client-Server



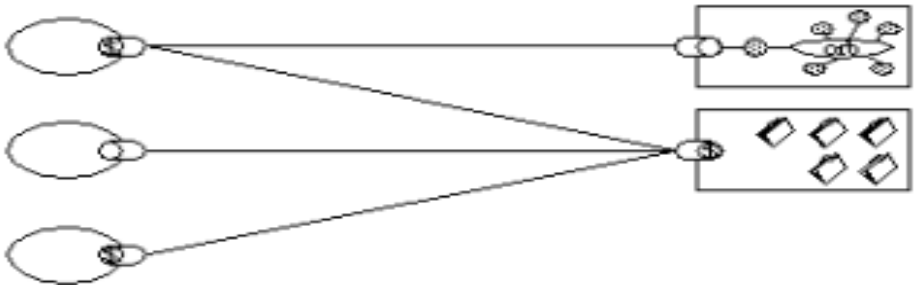
Client-Stateless-Server



Client-Cache-Stateless-Server

Source: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Evolution of REST Architecture



Uniform-Client-Cache-Stateless-Server

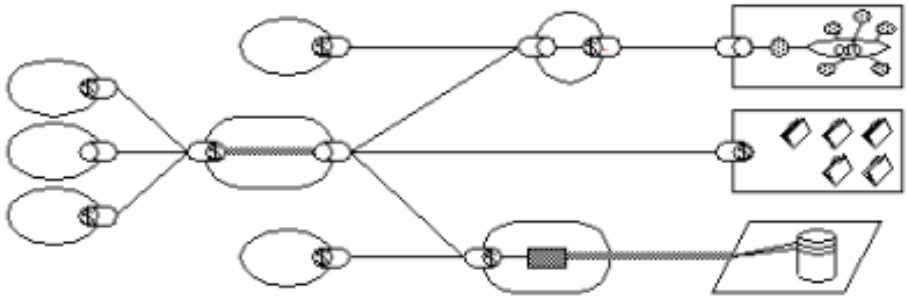


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Uniform-Layered-Client-Cache-Stateless-Server

Source: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST- Interface Constraints

1. Identification of **resources**
2. Manipulation of resources through **representations**
3. **Self-descriptive messages**
4. **Hypermedia** as the engine of application state (HATEOAS)

Interface Constraints – 1

Identification of Resources

REST is based on these notions:

- A **resource**
  - Any information that can be named can be a resource
  - A document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person)
  - Nouns instead of verbs
- A **resource identifier**
  - Each resource accessible by a URI (with or without host domain name)
  - A resource locator (URL) identifies a particular resource
  - 'Nice' URIs

Example

customers/ (6)

ID	URI
1	customers/1/ (http://localhost:8080/CustomerDB/resources/customers/1/)
2	customers/1/discountCode/ (http://localhost:8080/CustomerDB/resources/customers/1/discountCode/)
3	customers/2/ (http://localhost:8080/CustomerDB/resources/customers/2/)
4	customers/2/discountCode/ (http://localhost:8080/CustomerDB/resources/customers/2/discountCode/)
5	customers/25/ (http://localhost:8080/CustomerDB/resources/customers/25/)
6	customers/25/discountCode/ (http://localhost:8080/CustomerDB/resources/customers/25/discountCode/)

REST Resources

- Each resource addressable by a URI
  - A bucket e.g. /itemsDB/books
- GET /itemsDB/books
  - Get all the items in the bucket
- GET /itemsDB/books/{id}
  - Read the item by its id using GET
- POST /itemsDB/books
  - Create a new item by posting it to the bucket
- PUT /itemsDB/books/ {a new id}*
  - Creates a new item using the user provided new id
- PUT /itemsDB/books/ {existing-id}
  - Update an existing item using its id
- DELETE /itemsDB/books/ {existing-id}

Interface Constraints – 2

Manipulation of resources through **representations**

- A **representation** of a resource
  - A document capturing current state of a resource
  - A resource can have different representations
  - Client can specify which representation can accept (e.g. in http accept header)
  - A resource representation provides links for navigation and accessing further resources

- **State transition**: selecting a hyperlink, the client application changes to another state
  - With each resource representation the state of the client application is transferred

Resource Representation

- A resource can have multiple representations
  - Like XML, HTML, JSON, etc

- JSON
  - JSON stands for JavaScript Object Notation
  - uses JavaScript syntax for describing data objects
  - JSON is a lightweight text-data interchange format
  - JSON is "self-describing" and easy to understand

JSON Example

- Data is in name/value pairs , followed by a colon “**:**”
- Data is separated by commas “**,**”
- Curly braces hold objects “**{ }**”
- Square brackets hold arrays “**[ ]**”

```
{   "firstName": "John",    "lastName": "Smith",   "age": 25,    "address": {
        "streetAddress": "21 2nd Street",  "city": "New York",
        "state": "NY",  "postalCode": 10021
    },

    "phoneNumbers": [
        {
            "type": "home",  "number": "212 555-1234"
        },
        {
            "type": "fax",  "number": "646 555-4567"
        }   ]
}
```

JSON Parsing

- JSON parsing online
  - http://json.parser.online.fr/
  - http://www.jsoneditoronline.org/

{ "firstName": "John", "lastName": "Smith", "age": 25, "address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY", "postalCode": 10021 },"phoneNumbers": [{"type": "home", "number": "212 555-1234" }, {"type": "fax", "number": "646 555-4567" }] }

- Examples: Jackson, Jettison, and Google Gson libraries
  - Sample code using Gson:

```
JsonParser parser = new JsonParser();
JsonObject jsonObject = (JsonObject) parser.parse(json-Returned-data-String);
```

Check the Google response structure here:
https://developers.google.com/custom-search/json-api/v1/reference/cse/list#response

Interface Constraints – 3

Self-Descriptive Messages

- “REST constrains messages between components to be self-descriptive in order to support intermediate processing of interactions. ”
- Each message contains all the information necessary to complete the task

Interface Constraints – 4

Hypermedia As The Engine Of Application State (HATEOAS)

- "HyperMedia is a term used for hypertext which is not constrained to be text: it can include graphics, video and sound"
- **Hypermedia** links control the client choices that can lead to change of **application state** therefore it acts **as an engine**
- Client can select the link by examining the link metadata

RESTful Web Services

REST Web Services

- “A restful service is a web of resources that programs can navigate.”
- In REST, services are replaced by resources.
- In REST the emphasis is on point-to-point communication
- REST is mainly restricted to http methods (GET, POST, PUT, DELETE, PATCH) but in SOAP web services, methods are defined by yourself.

REST Web Services Characteristics

- **Client-Server**: a pull-based interaction style: consuming components pull representations.
- **Stateless**: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- **Cache**: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- **Uniform interface**: all resources are accessed with a generic interface (e.g., HTTP GET, POST, PUT, DELETE).
- **Named resources** – the system is comprised of resources identified using URIs
- **Interconnected resource representations** - the representations of the resources are interconnected using URIs/URLs, thereby enabling a client to progress from one state to another.
- **Layered components** - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

REST Web Service HTTP Methods

Method	Meaning	Idempotent	Safe
GET	Retrieve a copy of a Resource	YES	YES
DELETE	Remove a Resource	YES	NO
POST	Create or sometimes update a Resource without a known ID	NO	NO
PUT	Update a Resource or sometimes create with a known ID	YES	NO
PATCH	Update only a part of Resource	YES	NO

The difference between PUT and POST

- POST is not idempotent but PUT is
- The client uses PUT when it's in charge of deciding which URI the new resource should have.
- The client uses POST when the server is in charge of deciding which URI the new resource should have.
- More details: @http://restcookbook.com/HTTP%20Methods/put-vs-post/

Principles of REST Web Service Design

1) Identify all of the conceptual entities that you wish to expose as services.
2) Create a URL to each resource. The resources should be nouns, not verbs.
3) Categorise your resources according to whether clients can just receive a representation of the resource, or whether clients can modify (add to) the resource.
4) All resources accessible via HTTP GET should be side-effect free/safe/ idempotent. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.
5) Put hyperlinks within resource representations to enable clients to drill down for more information, and/or to obtain related information.
6) Design to reveal data gradually. Don't reveal everything in a single response document. Provide hyperlinks to obtain more details.
7) Specify the format of response data using a schema.
8) Describe how your services are to be invoked using either a WSDL/WADL document, or simply an HTML document.

Mashups

- A mashup, in web development, is a web page, or web application, that uses content from more than one source to create a single new service displayed in a single graphical interface.
  - The term implies easy, fast **integration**, frequently using open application programming interfaces (API) and data sources to produce enriched results that were not necessarily the original reason for producing the raw source data.

- Mashup examples:
  - http://ukdataexplorer.com/european-translator/
  - http://www.deitel.com/Books/Web2eBook/WebServicesMashupsWidgetsGadgets/tabid/2494/Default.aspx

UK Data Explorer

European word translator

Enter one or two lower-case English words to see translations from Google Translate.

hello × Translate it!

(All input is converted to lower case)

Examples: banana the cat she runs

Random words: confirm dish

A few things to keep in mind:

- Translations are generated by Google Translate. Some may be inaccurate or use non-European (e.g. Brazilian Portuguese) words.
- Just one translation is provided for each word; watch out for words with multiple meanings!
- The Google Translate API does not yet translate into all European languages.
- If Google Translate cannot find a translation, it simply shows the English word.

This page was inspired by the etymology maps by Bezbojnicul on reddit. It was built using D3, maps from Natural Earth, and the Google Translate API. Contact James Trimble.

Using JAX-RS to create RESTful WS

Two Java Technologies for Web Services: JAX-WS and JAX-RS

- **JAX-WS** (The Java API for XML Web Services)
  - A Java programming language API for creating SOAP (Big) web services
  - Its implementation is part of project **GlassFish**
  - Supporting the WS-* set of protocols, which provide standards for security and reliability, and interoperate with other WS-* conforming clients and servers
  - The developer does not need to generate or parse SOAP messages
  - The JAX-WS runtime system converts the API calls and responses to and from SOAP messages

From: The Java EE 6 Tutorial at http://docs.oracle.com/javaee/6/tutorial/doc/gjbji.html

JAX-RS

- **JAX-RS API**
  - A Java programming language API designed to make it easy to develop applications that use the REST architecture
  - To simplify development of RESTful Web services and their clients in Java in a standard and portable manner
  - Use **Annotations**

- **Jersey**
  - Is the reference implementation of JAX-RS (JSR311), implements support for the annotations
  - Jersey provides it's own API that extend the JAX-RS toolkit with additional features
  - Jersey Homepage https://jersey.java.net/

Code Sample

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
…
@Path("student.student")
Public class StudentFacadeREST extends AbstractFacade<Student>{
…
@GET
@Override
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public List<Student> findAll() {
     return super.findAll();
  }
@GET
@Path("{id}")
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public Student find(@PathParam("id")  Integer id) {
     return super.find(id);
  }
```

http://localhost:8080/TestStudent/webresources/**student.student**
http://localhost:8080/TestStudent/webresources/**student.student/{id}**

@Path

- The @Path annotation's value is a relative URI path
- It indicates **where the resource will be hosted**
  - E.g. @Path("/helloworld") is a static URI path
- You can also embed variables in the URIs to make a URI path template, substituted at runtime
- Variables are denoted by curly braces
  - E.g. /helloworld/{username}
    - @Path("/{name1}/{name2}/")
    - http://example.com/{name1}/{name2}/
    - http://example.com/james/gatz/
    - @Path("/maps/{location}")
    - http://example.com/maps/Main%20Street

Annotations

@Produces

- Specifies the MIME media types **a resource can produce** and send back to the client
  - E.g. In @Produces("text/plain"), the Java method will produce representations identified by the MIME media type "text/plain"

@Consumes

- Specifies the MIME media types **a resource can consume** that were sent by the client
  - E.g. @Consumes("text/plain")

HTTP Methods

```
@POST
@Consumes({"application/xml",  "application/
json"})
public void create(Book entity) {
    super.create(entity);
}
@PUT
@Path("{id}")
@Consumes({"application/xml", "application/
        json"})
public void edit(@PathParam("id") Integer id,
        Book entity) {
    super.edit(entity);
}
@DELETE
@Path("{id}")
public void remove(@PathParam("id") Integer  id)
        {
         super.remove(super.find(id));
         }
```

```
@GET
@Path("{id}")
@Produces({"application/xml",
        "application/json"})
public Book find(@PathParam("id")
        Integer id) {
        return super.find(id);
        }
```

```
@GET
@Produces({"application/xml",
        "application/json"})
public List<Book> findAll() {
    return super.findAll();
}
```

Sample RESTful Web service

```
@Path("/helloworld/{name}")
public class HelloworldResource {
    @Context
    private UriInfo context;
    /** Creates a new instance of HelloworldResource */
    public HelloworldResource() { }
    /**
     * Retrieves representation of an instance of jaxrstest.HelloworldResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/plain")
    public String sayHello(@PathParam("name") String name,
                                @DefaultValue("HR") @QueryParam("dep") String department) {
        return "Hi "+name+" Welcome to "+ department +" department"; }
    @GET
    @Path("/sunresource")
    public String testSubResource(){
        return "This is from Sub Resource"; }
    /**
     * PUT method for updating or creating an instance of HelloworldResource
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT
    @Consumes("text/plain")
    public void putText(String content) {
        System.out.println("======"+content+"====="); }
}
```

Further Information about REST & JAX-RS

- For more information about RESTful web services and JAX-RS, see "RESTful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision": http://www2008.org/papers/pdf/p805-pautassoA.pdf
- "Fielding Dissertation: Chapter 5: Representational State Transfer (REST)": http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- *RESTful Web Services*, by Leonard Richardson and Sam Ruby, available from O'Reilly Media at http://oreilly.com/catalog/9780596529260/
- JSR 311: JAX-RS: The Java API for RESTful Web Services: http://jcp.org/en/jsr/detail?id=311
- JAX-RS project: http://jsr311.java.net/
- Jersey project: http://jersey.java.net/


please drop the
SOAP
mashery.com
get RESTful
developer.mashery.com


Gmail
I haven't used SOAP
for years