REST-based Web Services (I)

Introduction to Service Design and Engineering 2013/2014.

Lab session #5

University of Trento

Guiding notes based on **Helen Paik's** slides for the School of Computer Science and Engineering University of New South Wales

Outline

- REST principles recap
- Building REST Services Servlet Example
- Building REST Services with Jersey

What's REST?

- The term Representational State Transfer (REST) was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation
- It is an **architectural style** of networked systems (not a protocol - not a specification), by which **resources** are exposed through out the system.
- REST is a client-server architecture.
- Only representations of **resources** are exposed to the client
- The representation of resources places the client application in a **state**.
- Client state may evolve by **traversing hyperlinks** and obtaining **new representations**

REST Principles

- Resource identification through **URI**
- **Uniform interface:** resources are manipulated using a fixed set operations (HTTP GET, POST, PUT, DELETE methods)
- Self-descriptive messages: resources are decoupled from their representation

REST Principles: RESTful Flavor

- **Stateful interactions through hyperlinks:** every interaction with a resource is stateless (i.e., request messages are self-contained)

REST principles: resources

- Resource: Any *thing* (noun) that is worthy of being given a unique ID (URI) and be accessible via client
- Resources are something the server is responsible for managing Resources must have representations to be 'transmitted' to client

e.g., resources in the starbucks example: order, payment (represented in XML)

REST principles: uniform interface

Uniform Interface: Uniform 'verbs' that go with the resources (noun)

- Given a resource (coffee order): a representation in XML

```
<order xmlns="urn:starbucks">
   <drink>latte</drink>
</order>
```

- POST /starbucks/orders (to create an order)
  - returns: location: /starbucks/orders/order?id=1234
- GET /starbucks/orders/order?id=1234 (to read an order)
- PUT /starbucks/orders/order?id=1234 (to update an existing order)
- DELETE /starbucks/orders/order?id=1234 (to delete an existing order)

REST principles: hypermedia

Connectedness/Links: Resources may contain links to other resources e.g., Order resource is linked to Payment resource

In response to POSTing an order

```
201 Created
Location: /starbucks/orders/order?id=1234
Content-Type: application/xml
Content-Length: ...
<order xmlns="urn:starbucks">
  <drink>latte</drink>
  <link rel="payment" href="/starbucks/payments/order?id=1234"
          type="application/xml"/>
</order>
```

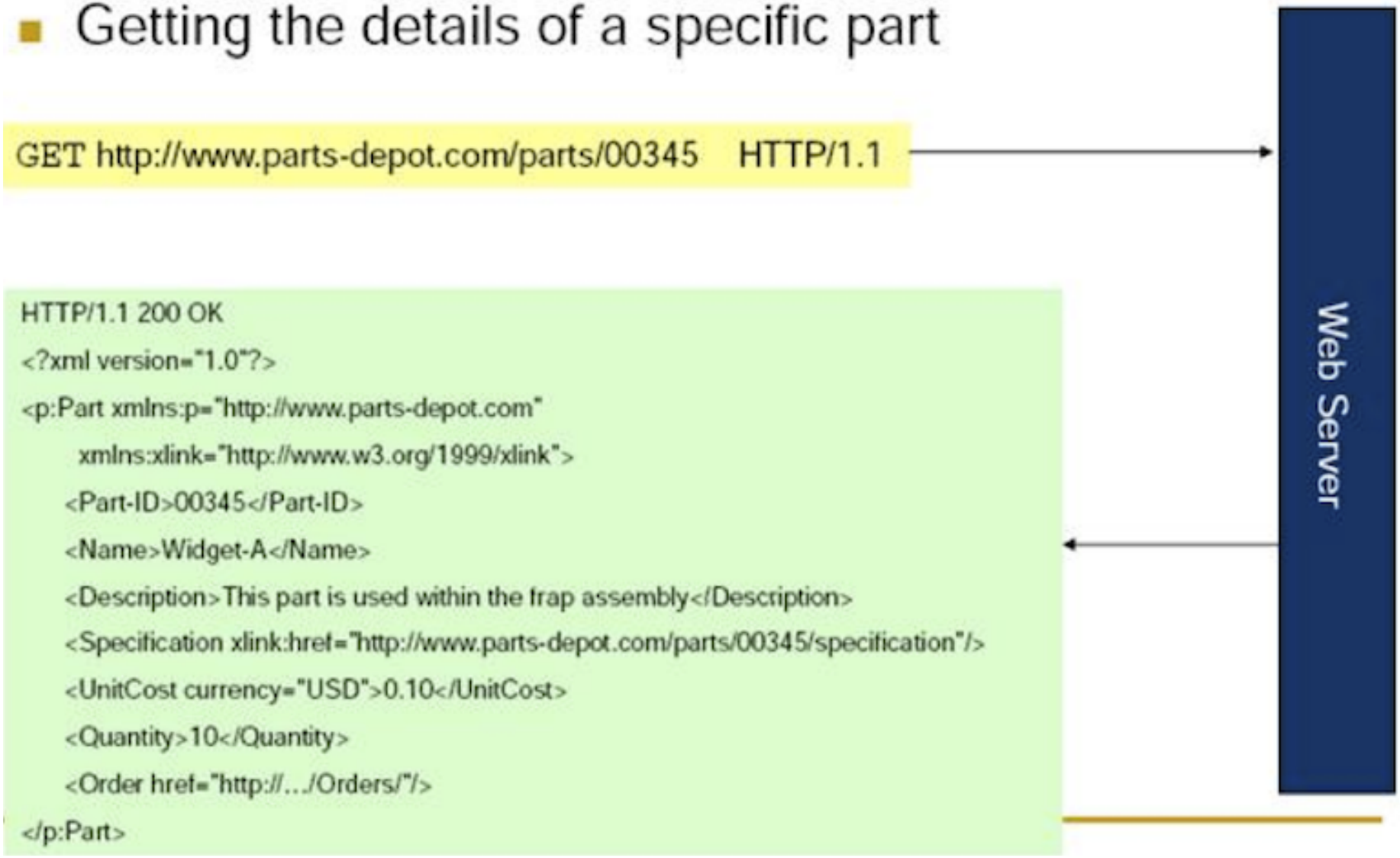
Both forward/backward links, when possible (e.g., order having 'cancel/delete' link)

REST principles: hypermedia

- Getting the list of parts

REST principles: hypermedia

- Getting the details of a specific part



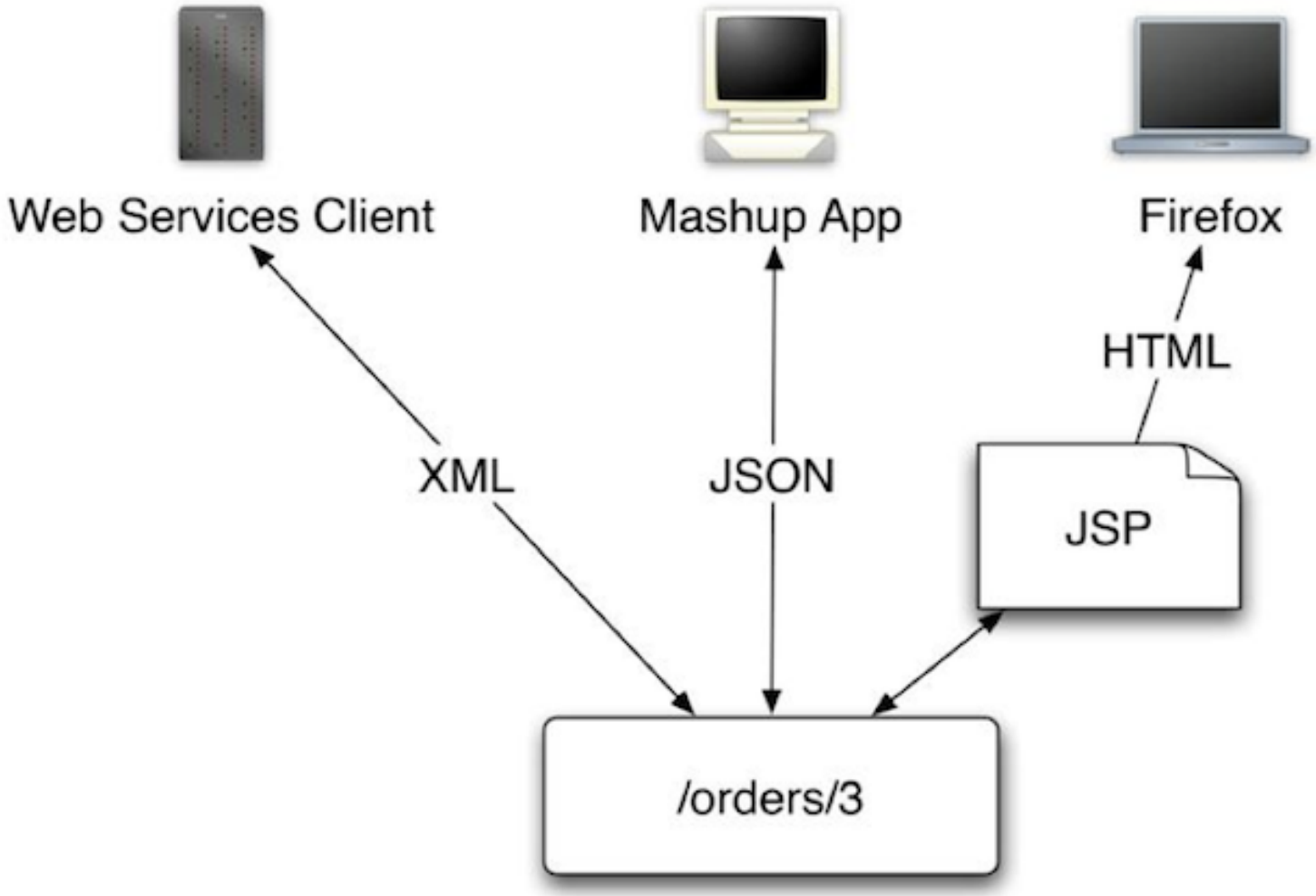
REST principles: satefy and idempotence

REST Uniform Interface, if properly followed, gives you two properties:

- **Safety (GET):** Read-only operations. The operations on a resource do not change any server state. The client can call the operations 10 times, 1000 times, it has no effect on the server state.
- **Idempotence (GET, PUT and DELETE):** Operations that have the same “effect" whether you apply them once or more than once. An effect here may well be a change of server state. An operation on a resource is idempotent if making a request once has the same effect as making the identical request multiple times.

REST principles: representations

One resource, many representations



REST principles: representations

- **XML**

```
<animals>
<dog>
  <name>Rufus</name>
  <breed>Labrador</breed>
</dog>
<dog>
  <name>Marty</name>
  <breed>whippet</breed>
</dog>
<cat name="Matilda" />
</animals>
```

- **JSON**

```
{ animals : {
dog: [  {   name:"Rufus",
        breed:"Labrador"
    },
    {
      name:"Marty",
        breed:"whippet"
    }
  ],
cat : {
    name:"Matilda"
  }
}
}
```

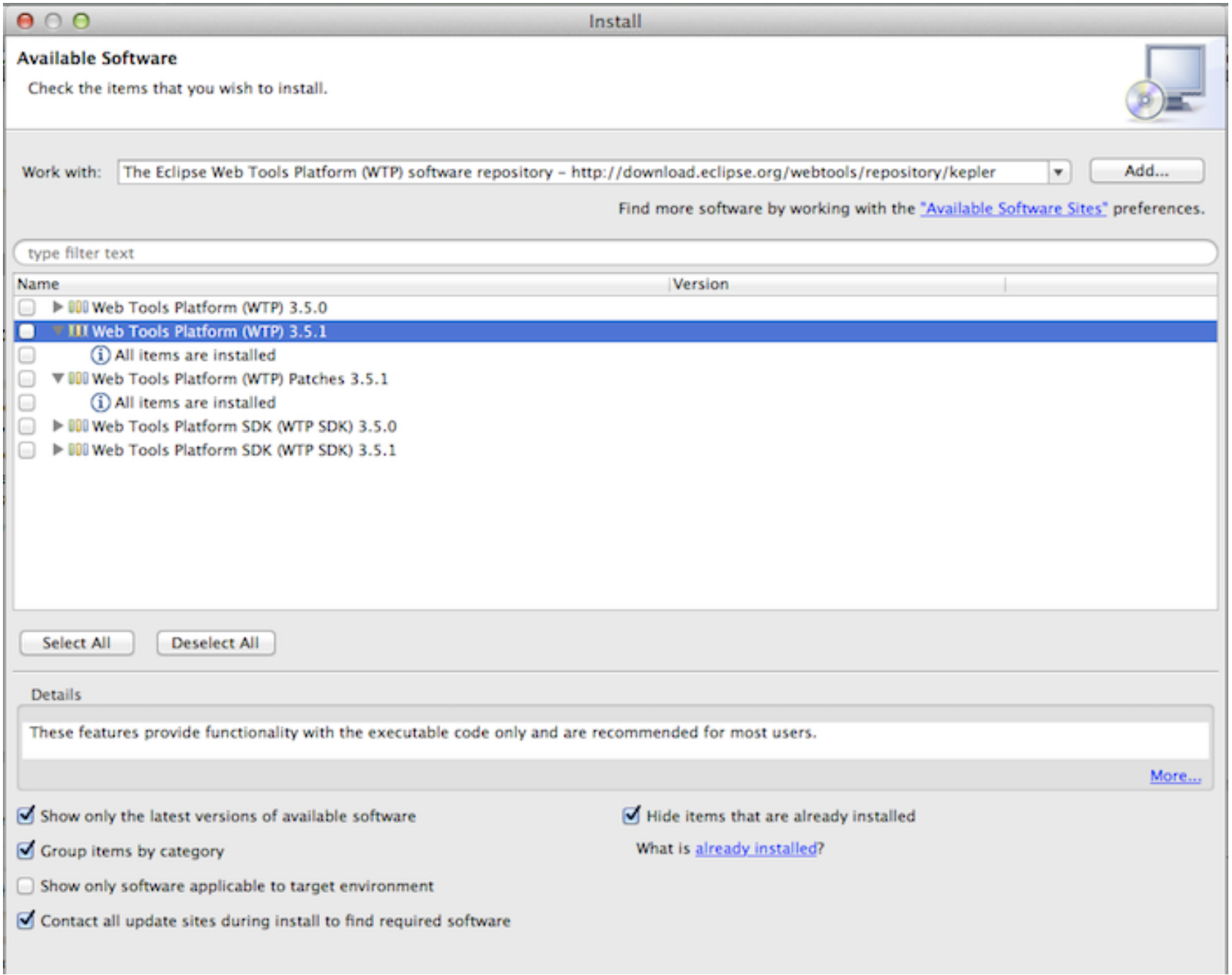
Building REST Services

- REST does not requires you to use a specific client or server-side framework in order to write your Web services. All you need is:
  - a client or server that supports the HTTP protocol (i.e., a web server, a browser).
  - choose a language of your choice
- **In Java:** You'd use servlets and override doGet(), doPost(), doPUT() and doDelete()
  - URLs contains: servlet path + path info (all you need to process a request in REST)
  - You could use a third-party library for generating specific content type (CSV, JSON or XML, etc.) or use Strings concatenations for simple responses.

Configuration - Eclipse WTP (1)

- For the lab, we will use **Eclipse WTP**, which provides tools for developing standard Java web applications and Java EE applications
- To install, use **Help -> Install new software -> All Available Sites**
  - You can also use only the WTP repository: **http://download.eclipse.org/webtools/repository/kepler** (might change according to your version of eclipse)
- Search for "**Web Tools Platform**" and install all what's inside that category (using the latest version)
- In old versions of eclipse, there might be a category "Web, XML, Java EE Development and OSGi Enterprise Development". Install all inside.

Configuration - Eclipse WTP (2)



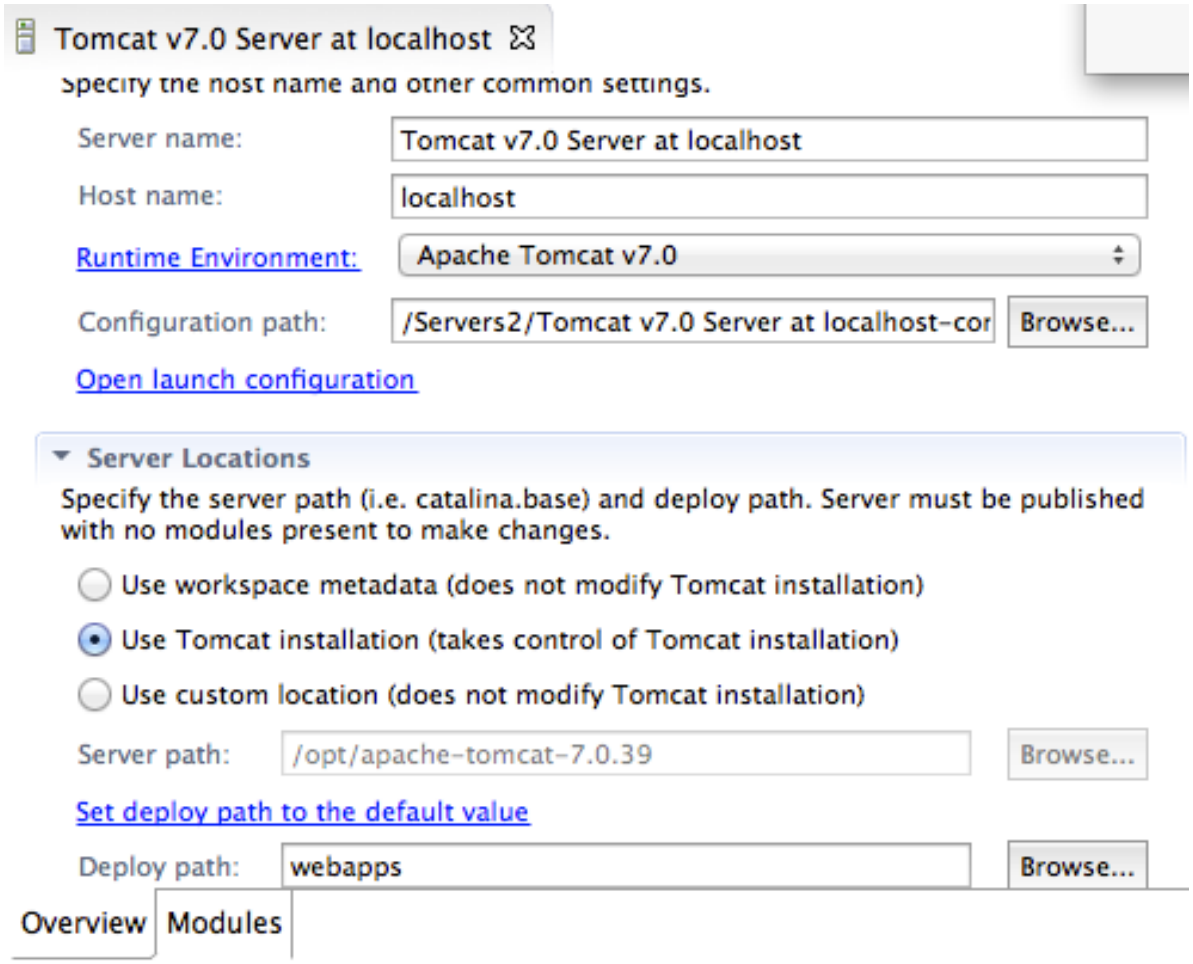
Configuration - Tomcat Server in Eclipse (1)

- The first step is to configure a Runtime Environment for Tomcat
- In eclipse, go to **Preferences -> Server -> Runtime Environments -> Add**
- Select your version of Tomcat.
- To compile the JSP into servlets you need to use the JDK. You can check your setup by clicking on the Installed JRE button.
- Press Finish and then OK. You are now ready to use Tomcat with WTP.

Configuration - Tomcat Server in Eclipse (2)

- Second step is to create a running **Tomcat Server**
- Open the Server view for this: **Window -> Show View -> Other -> Server -> Servers**
- If no server is available, you need to create a new one (just follow the steps of the wizard)
- Once the server is created, double click on it to open its configuration file
- I reccomend you mark "use tomcat installation"

Configuration - Tomcat Server in Eclipse (3)



The Simplest REST Example - A Servlet

- Check the code example in lab5/Example1-Servlet