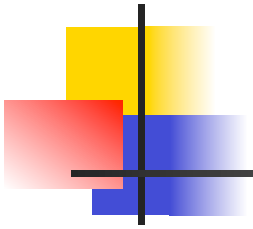
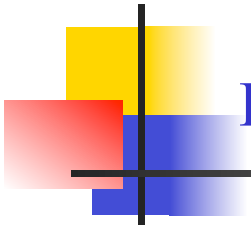
REST - Representational State Transfer



What is REST ?

REST is a term coined by Roy Fielding to describe an **architecture style** of networked systems. REST is an acronym standing for Representational State Transfer.



Rest – An architectural Style

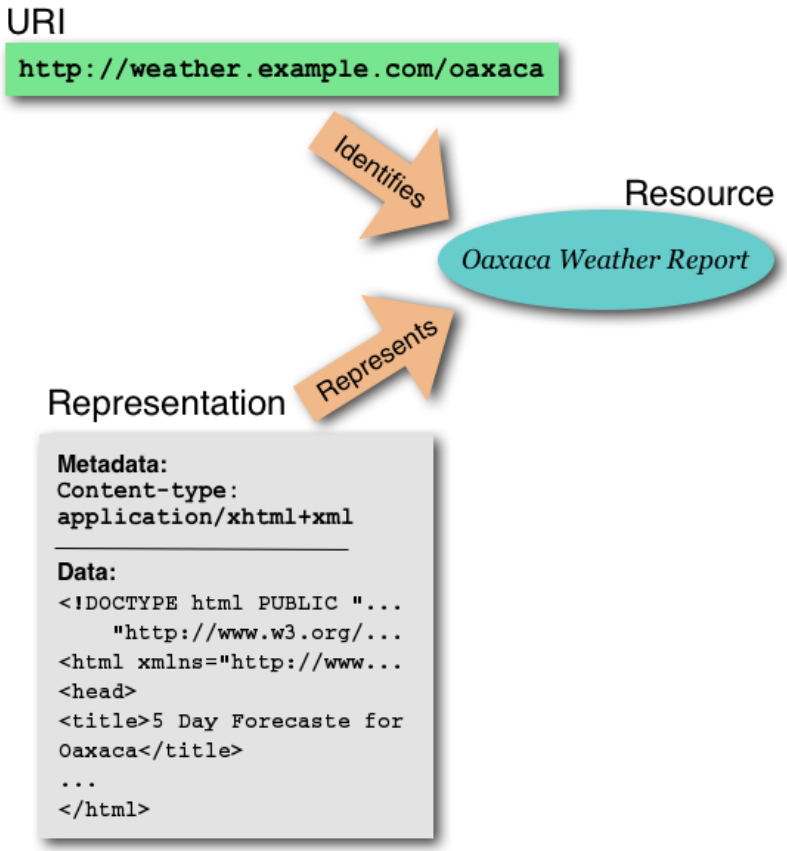
Elements

- Components – Proxy , gateway etc
- Connectors – client , server etc
- Data – resource , representation etc

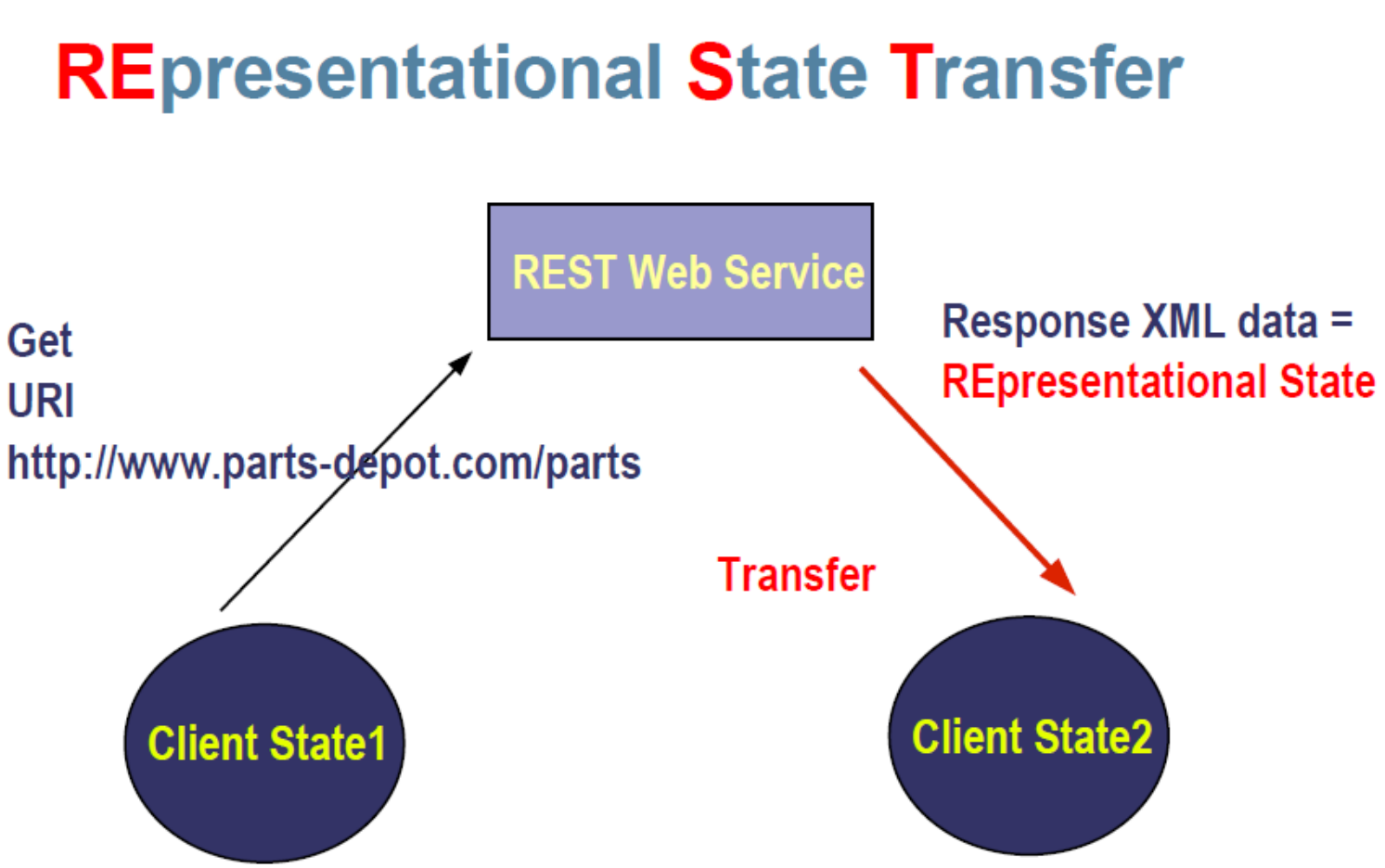
REST

- Ignores component implementation details.
- Focus on roles of components,their interactions and their interpretation of data elements.

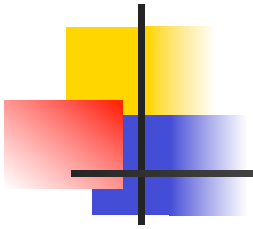
- **Resource**
- **URI-Uniform Resource Identifier (or URL)**
- **Web Page (HTML Page)**



What is REST?

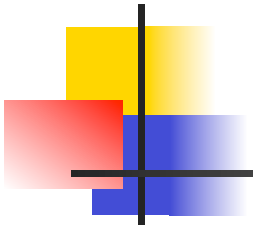

REpresentational State Transfer
REST Web Service
Get
URI
http://www.parts-depot.com/parts
Response XML data =
REpresentational State
Transfer
Client State1
Client State2

HTTP Request/Response As REST



"Representational State Transfer 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.



REST - An Architectural Style of Networked System

- Underlying Architectural model of the world wide web.
- Guiding framework for Web protocol standards.

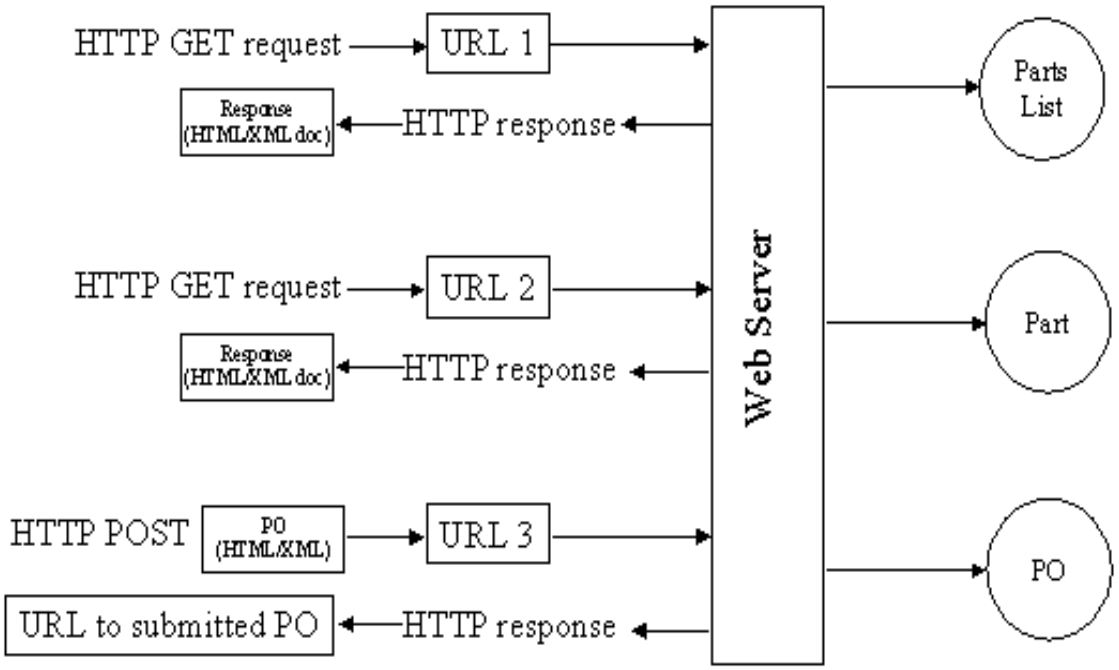
REST based web services

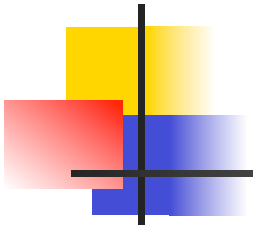
- Online shopping
- Search services
- Dictionary services

Parts Depot Web Services

- Parts Depot, Inc has deployed some web services to enable its customers to:
  - get a list of parts
  - get detailed information about a particular part
  - submit a Purchase Order (PO)

REST way of Implementing the web services





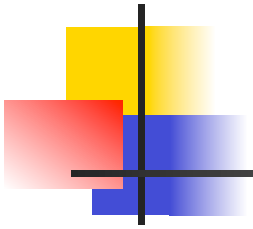
Service – Get parts list

The web service makes available a URL to a parts list resource

Client uses : http://www.parts-depot.com/parts

Document Client receives :

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com" xmlns:xlink="http://www.w3.org/1999/xlink">
        <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
        <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
        <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
        <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```



Service – Get detailed part data

The web service makes available a URL to each part resource.

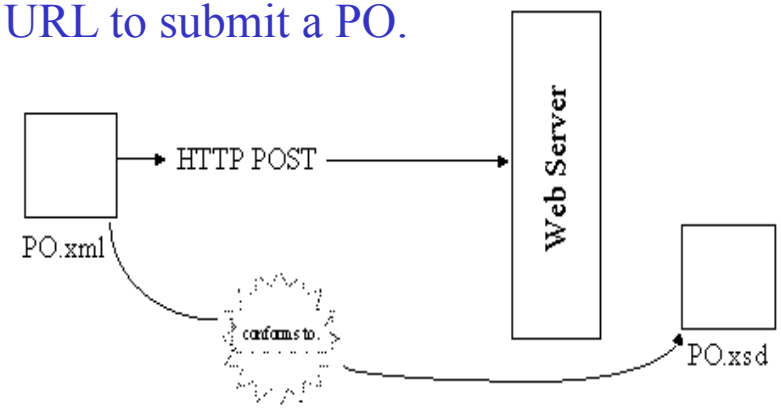
Client uses : http://www.parts-depot.com/parts/00345

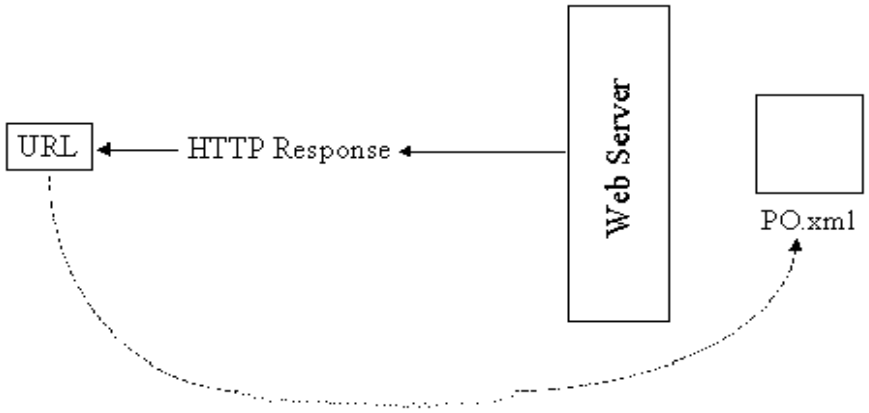
Document Client receives :

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com" xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly</Description>
    <Specification xlink:href="http://www.parts-depot.com/parts/00345/specification"/> <UnitCost
     currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</p:Part>
```

Service – Submit purchase order (PO)

The web service makes
available a URL to submit a PO.

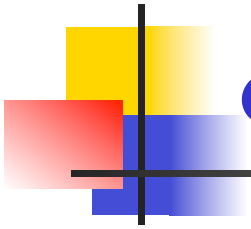




1)The client creates a PO instance document (PO.xml)

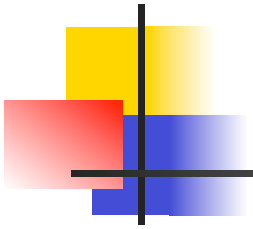
2)Submits the PO.xml(HTTP POST)

3)PO service reponds with a URL to the submitted PO.



Characteristics of a REST based network

- Client-Server: a pull-based interaction style(Client request data from servers as and when needed).
- 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 which are named using a URL.
- Interconnected resource representations - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.



Principles of REST web service design

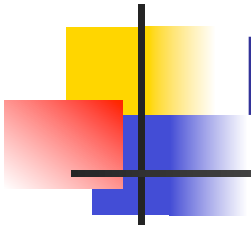
- 1.Identify all the conceptual entities that we wish to expose as services. (Examples we saw include resources such as : parts list, detailed part data, purchase order)
- 2. Create a URL to each resource.
- 3. Categorize our resources according to whether clients can just receive a representation of the resource (using an HTTP GET), or whether clients can modify (add to) the resource using HTTP POST, PUT, and/or DELETE).
- 4. All resources accessible via HTTP GET should be side-effect free. 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 (DTD, W3C Schema, RelaxNG, or Schematron). For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.
- 8. Describe how our services are to be invoked using either a WSDL document, or simply an HTML document.

REST Vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user

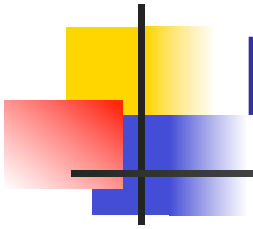
- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
          <soap:body pb="http://www.acme.com/phonebook">
                    <pb:GetUserDetails>
                              <pb:UserID>12345</pb:UserID>
                    </pb:GetUserDetails>
          </soap:Body>
</soap:Envelope>
```



REST Vs SOAP

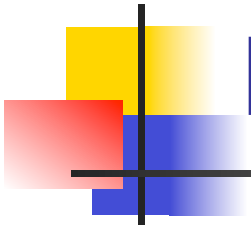
- Simple web service as an example: querying a phonebook application for the details of a given user
- And with REST? The query will probably look like this:
  http://www.acme.com/phonebook/UserDetails/12345
- GET /phonebook/UserDetails/12345 HTTP/1.1
  Host: www.acme.com
  Accept: application/xml
- Complex query:
  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe



REST Vs SOAP

SOAP is definitely the heavyweight choice for Web service access. It provides the following advantages when compared to REST:

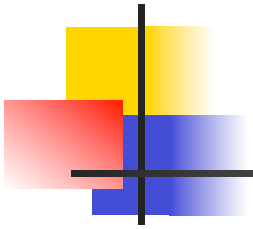
- Language, platform, and transport independent (REST requires use of HTTP)
- Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
- Standardized
- Provides significant pre-build extensibility in the form of the WS* standards
- Built-in error handling
- Automation when used with certain language products



REST Vs SOAP

REST is easier to use for the most part and is more flexible. It has the following advantages when compared to SOAP:

- No expensive tools require to interact with the Web service
- Smaller learning curve
- Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
- Fast (no extensive processing required)
- Closer to other Web technologies in design philosophy



Summary

- **REST – Is an architectural style.**
- **It is the architectural style of the WEB**

- **Resource**

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