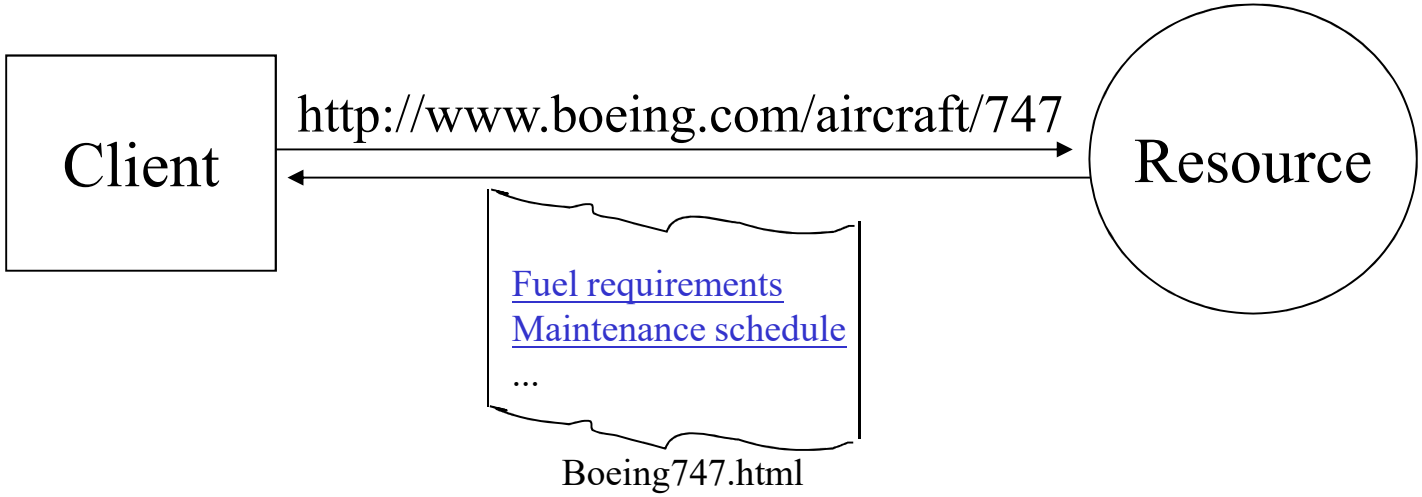
What is REST?

" REST " was coined by Roy Fielding in his Ph.D. dissertation [1] to describe a design pattern for implementing networked systems.

Why is it called "Representational State Transfer? "



The Client references a Web resource using a URL.
A **representation** of the resource is returned (in this case as an HTML document).
The representation (*e.g.,* Boeing747.html) places the client in a new **state**.
When the client selects a hyperlink in Boeing747.html, it accesses another resource.
The new representation places the client application into yet another state.
Thus, the client application **transfers** state with each resource representation.

Representational State Transfer

"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

Motivation for REST

The motivation for developing REST was to create a design pattern for how the Web should work, such that it could serve as the guiding framework for the Web standards and designing Web services.

REST - Not a Standard

- REST is not a standard
  - You will not see the W3C putting out a REST specification.
  - You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.
- REST is just a **design pattern**
  - You can't bottle up a pattern.
  - You can only understand it and design your Web services to it.
- REST does prescribe the **use** of standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/*etc.* **(Resource Representations)**
  - text/xml, text/html, image/gif, image/jpeg, *etc.* (Resource Types, MIME Types)

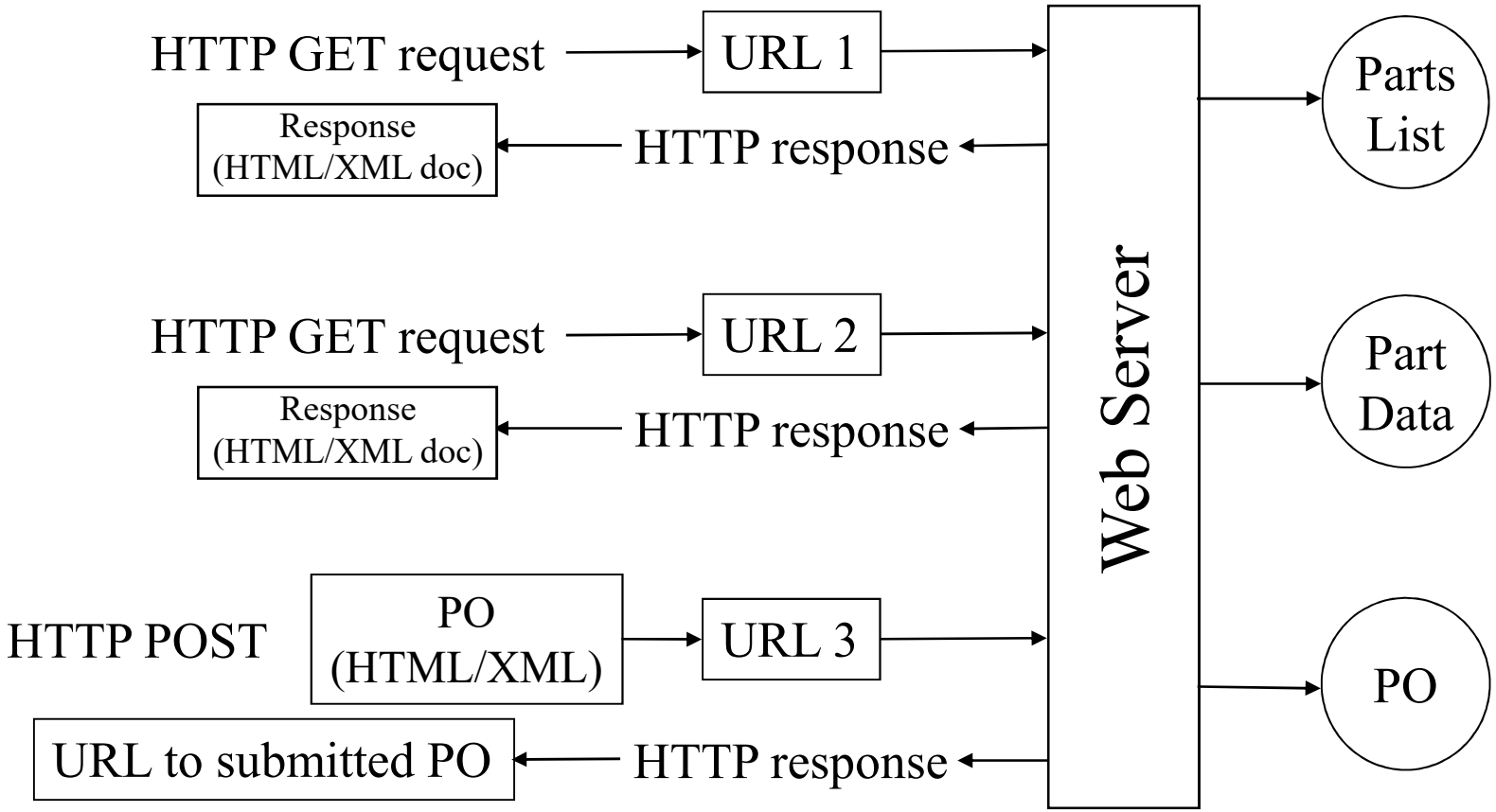
Learn by Example

- The REST design pattern is best explained with an example.
- I will present an example of a company deploying three Web services using the REST design pattern.

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)

The REST way of Designing the Web Services



Web Service for Clients to Retrieve a List of Parts

- Service: Get a list of parts
  - The web service makes available a URL to a parts list **resource**. A client uses this URL to get the parts list:
    - http://www.parts-depot.com/parts
    - Note that **how** the web service generates the parts list is completely transparent to the client. This is *loose coupling*.

REST Fundamentals

- Create a resource for every service.
- Identify each resource using a URL.

Data Returned - Parts List

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

Note that the parts list has links to get detailed info about each part. This is a key feature of the REST design pattern. The client transfers from one state to the next by examining and choosing from among the alternative URLs in the response document.

REST Fundamentals

- The data that a Web service returns should link to other data. Thus, design your data as a network of information.
- Contrast with OO design, which says to encapsulate information.

Web Service for Clients to Retrieve a Particular Part

- Service: Get detailed information about a particular part
  - The web service makes available a URL to each part resource. *For example*, here's how a client requests a specific part:
    - http://www.parts-depot.com/parts/00345

Data Returned - Part 00345

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

Again observe how this data is linked to still more data - the specification for this part may be found by traversing the hyperlink. Each response document allows the client to drill down to get more detailed information.

Questions and Answers

Question

What if Parts Depot has a million parts, will there be a million static pages? *For example*:

http://www.parts-depot/parts/000000
http://www.parts-depot/parts/000001
...
http://www.parts-depot/parts/999999

Answer

We need to distinguish between a logical and a physical entity.
The above URLs are **logical**. They express **what** resource is desired.

They do not identify a **physical** object. The advantage of using a logical identifier (URL) is that changes to the underlying implementation of the resource will be transparent to clients (that's loose coupling!).

Quite likely, Parts Depot will store all parts data in a database. Code at the Parts Depot web site will receive each logical URL request, parse it to determine which part is being requested, query the database, and generate the part response document to return to the client.

Answer (cont.)

Contrast the above logical URLs with these physical URLs:

http://www.parts-depot/parts/000000.html
http://www.parts-depot/parts/000001.html
...
http://www.parts-depot/parts/999999.html

These URLs are clearly pointing to physical (HTML) pages. If there are a million parts it will not be very attractive to have a million static pages. Furthermore, changes to how these parts data is represented will effect all clients that were using the old representation.

Question

What if I have a complex query?

For example:
Show me all parts whose unit cost is under $0.50 and for which the quantity is less than 10

How would you do that with a simple URL?

Answer

For complex queries, Parts Depot will provide a service (resource) for a client to retrieve a form that the client then fills in.

When the client hits "Submit" the browser will gather up the client's responses and generate a URL based on the responses.

Thus, oftentimes the client doesn't generate the URL (think about using Amazon - you start by entering the URL to amazon.com; from then on you simply fill in forms, and the URLs are automatically created for you).

Summary of the REST Design Pattern

The REST Design Pattern

- Create a resource for every service.
- Uniquely identify each resource with a logical URL.
- Design your information to link to other information. That is, the information that a resource returns to a client should link to other information in a network of related information.

The REST Design Pattern (cont.)

- All interactions between a client and a web service are done with simple operations. Most web interactions are done using HTTP and just four operations:
  - retrieve information (HTTP GET)
  - create information (HTTP PUT)
  - update information (HTTP POST)
  - delete information (HTTP DELETE)

The REST Design Pattern (cont.)

- Remember that Web components (firewalls, routers, caches) make their decisions based upon information in the HTTP Header. Consequently, the destination URL must be placed in the HTTP header for Web components to operate effectively.
  - Conversely, it is anti-REST if the HTTP header just identifies an intermediate destination and the payload identifies the final destination.