

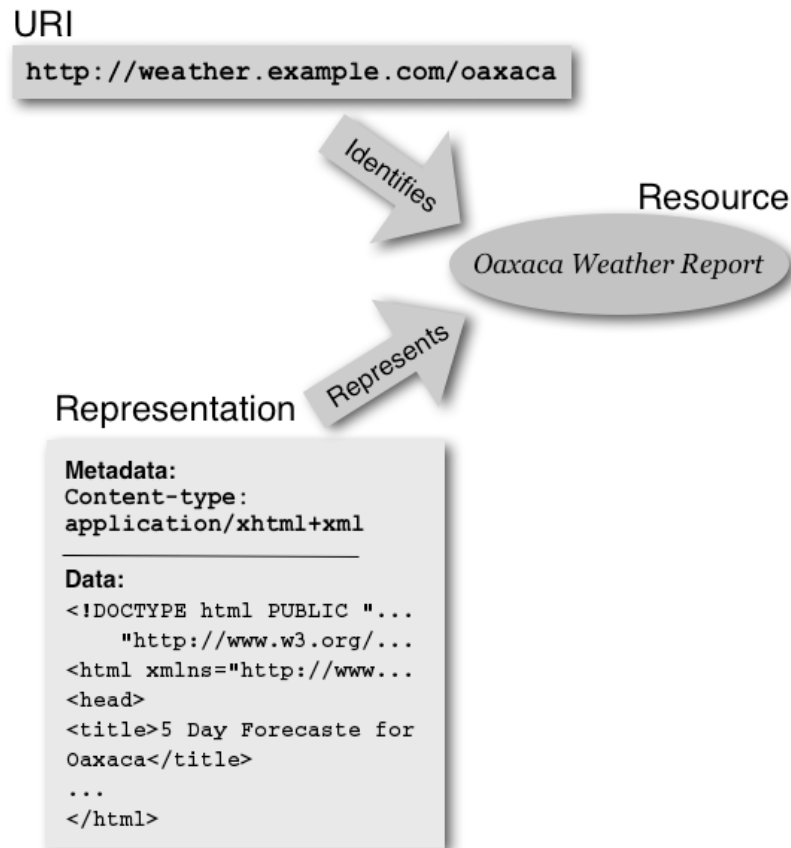
REST

Eleni Stroulia

Web-Architecture aspects

- The Web is a networked information system consisting of agents that exchange information.
- There are three important aspects to this system:
 - **Identification:** Objects in the networked information system called resources are identified by Uniform Resource Identifiers (URIs).
 - **Representation:** Agents (such as servers, browsers and multimedia players) communicate resource state through a non-exclusive set of data formats.
 - **Interaction:** Agents exchange representations via a non-exclusive set of protocols.

Two intuitions: client-server
hypertext



11-07

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." - R. T. Fielding

Fielding's homepage: <http://www.ics.uci.edu/~fielding>

REST - not a Standard

- It is just a prescription for how to use a suite of standards
 - HTTP
 - URL
 - XML, HTML, GIF, JPEG, ... (Resource Representations)
 - text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)

11-07

5

The REST style

- **Client-Server:** Separation of the user-interface concerns from the data-storage concerns
 - The two can evolve independently

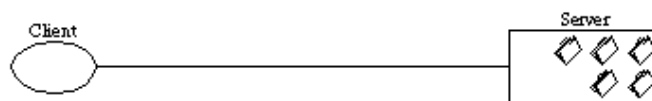


Figure 5-2. Client-Server

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

11-07

6

The REST style

- **Stateless:** Each client request must contain all of the information necessary to understand the request; session state is kept on the client.
 - Visibility: each request is self explanatory for an in-between monitor
 - Reliability: a failed request can be reissued
 - Scalability: simple and efficient server implementation with easily reallocatable resources
 - Performance: suffers due to data-heavy communication
 - Consistency: suffers due to inconsistent client implementations

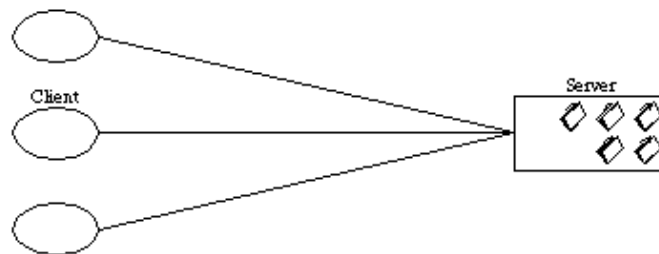


Figure 5-3. Client-Stateless-Server

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

11-07

7

The REST style

- **Cache:** Data in a response must be labeled as cacheable or non-cacheable; if cacheable, then a client cache may reuse it for later, equivalent requests.
 - Improved efficiency, scalability, and user-perceived performance
 - Stale data may be served

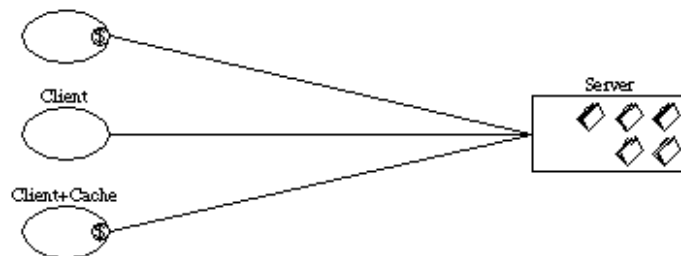


Figure 5-4. Client-Cache-Stateless-Server

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

11-07

8

The REST style

- **Uniform interface** between components
 1. identification of resources with URIs
 2. manipulation of resources through representations
 3. self-descriptive messages through HTTP
 4. hypermedia as the engine of application state

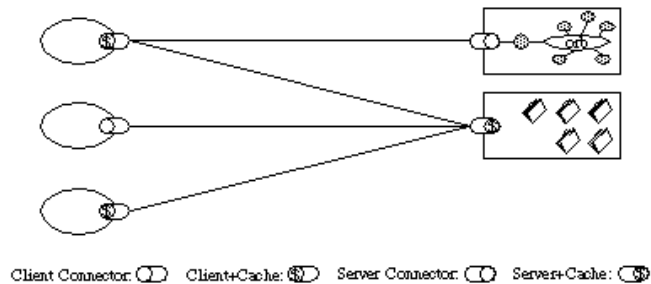


Figure 5-6. Uniform-Client-Cache-Stateless-Server

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

11-07

9

The REST style

- **Layered** system: Each component cannot "see" beyond the immediate layer with which they are interacting.
 - Independence of each layer
 - Components are simplified by moving infrequently used functionality to a shared intermediary
 - Intermediaries can also improve system scalability by load balancing
 - Additional overhead (offset by caching)

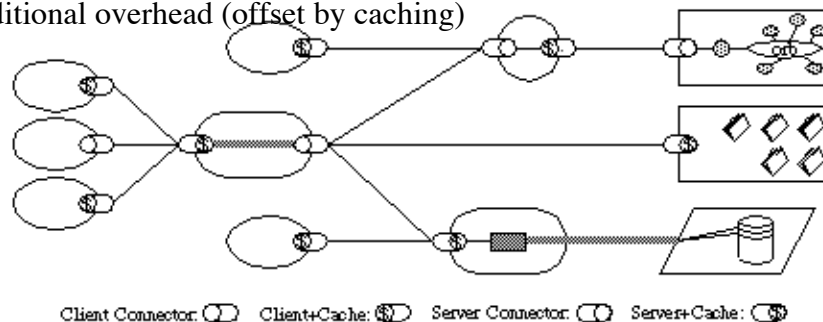


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

11-07

10

The REST style

- **Code-On-Demand:** Client functionality can be extended by downloading and executing code in applets or scripts.
 - reduces the number of features required to be implemented at the time of deployment
 - improves system extensibility
 - reduces visibility,

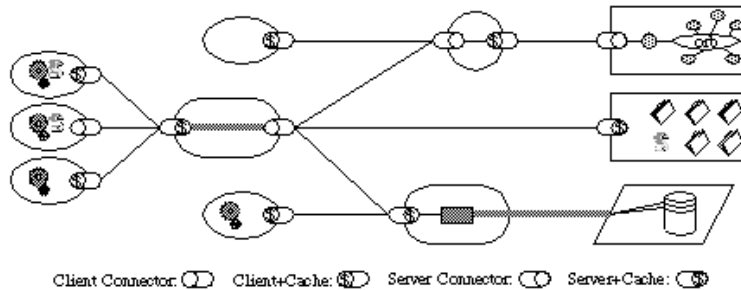


Figure 5-8. REST

Fielding's thesis (ch5) http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

The Elements of REST Style

Data

- When a link is selected, information moves from where it is stored to human reader
- Distributed hypermedia options:
 1. Render the data where it is located and send a fixed-format image to the recipient: Heavy on the server; simple but functionally limited on the client
 2. Encapsulate the data with a rendering engine and send both to the recipient: Heavy on the network; functionally limited on the client
 3. Send the raw data to the recipient along with metadata that describes the data type, so that the recipient can choose their own rendering engine: No information hiding; both client and server need to understand the data
- REST hybrid
 - Shared understanding of data types with metadata: The data representation is chosen through negotiation; there is no assumption that this is the natural representation of the data in the server
 - Limited scope of what is revealed through a standardized interface: The metadata constitute “instructions to a rendering engine”

11-07

13

Table 5-1: REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

11-07

14

Connectors

- REST interactions are stateless: each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it.
 - no need for the connectors to retain application state between requests, thus reducing consumption of physical resources and improving scalability;
 - interactions can be processed in parallel without requiring that the processing mechanism understand the interaction semantics;
 - an intermediary can view and understand a request in isolation, which may be necessary when services are dynamically rearranged;
 - all of the information that might factor into the reusability of a cached response to be present in each request.
- Like~ procedural invocation
 - in-parameters consist of request control data, a resource identifier indicating the target of the request, and an optional representation.
 - out-parameters consist of response control data, optional resource metadata, and an optional representation.
 - processing can be invoked before the value of the parameters is completely known, thus avoiding the latency of batch processing large data transfers.

11-07

15

Table 5-2: REST Connectors

Connector	Modern Web Examples
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

Connectors

- Client and server
 - a component may include both client and server connectors.
- Cache to save cacheable responses to current interactions so that they can be reused for later requested interactions.
 - At the server or client side
 - Personal or shareable
- Resolver translates partial or complete resource identifiers into the network address information needed to establish an inter-component connection.
- Tunnel relays communication across a connection boundary, such as a firewall or lower-level network gateway.

11-07

17

Components

- **Origin server:** Apache httpd, Microsoft IIS
 - uses a server connector to govern the namespace for a requested resource
 - the definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources
 - Intermediary components act as both a client and a server in order to forward, with possible translation, requests and responses.
- **Gateway:** Squid, CGI, Reverse Proxy
 - an intermediary imposed by the network or origin server to provide an interface encapsulation of other services, for data translation, performance enhancement, or security enforcement
- **Proxy:** CERN Proxy, Netscape Proxy, Gauntlet
 - an intermediary selected by a client, to provide interface encapsulation of other services, data translation, performance enhancement, or security protection
- **User agent:** Netscape Navigator, Lynx, MOMspider
 - uses a client connector to initiate a request and becomes the ultimate recipient of the response

11-07

18

Table 5-3: REST Components

Component	Modern Web Examples
origin server	Apache httpd, Microsoft IIS
gateway	Squid, CGI, Reverse Proxy
proxy	CERN Proxy, Netscape Proxy, Gauntlet
user agent	Netscape Navigator, Lynx, MOMspider

11-07

19

A process view

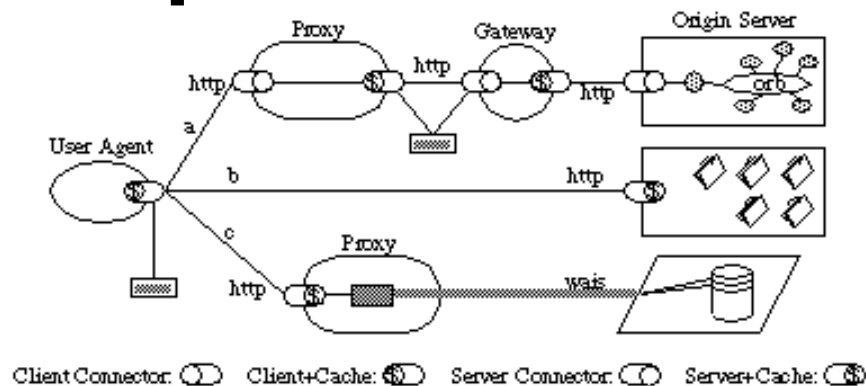


Figure 5-10. Process View of a REST-based Architecture

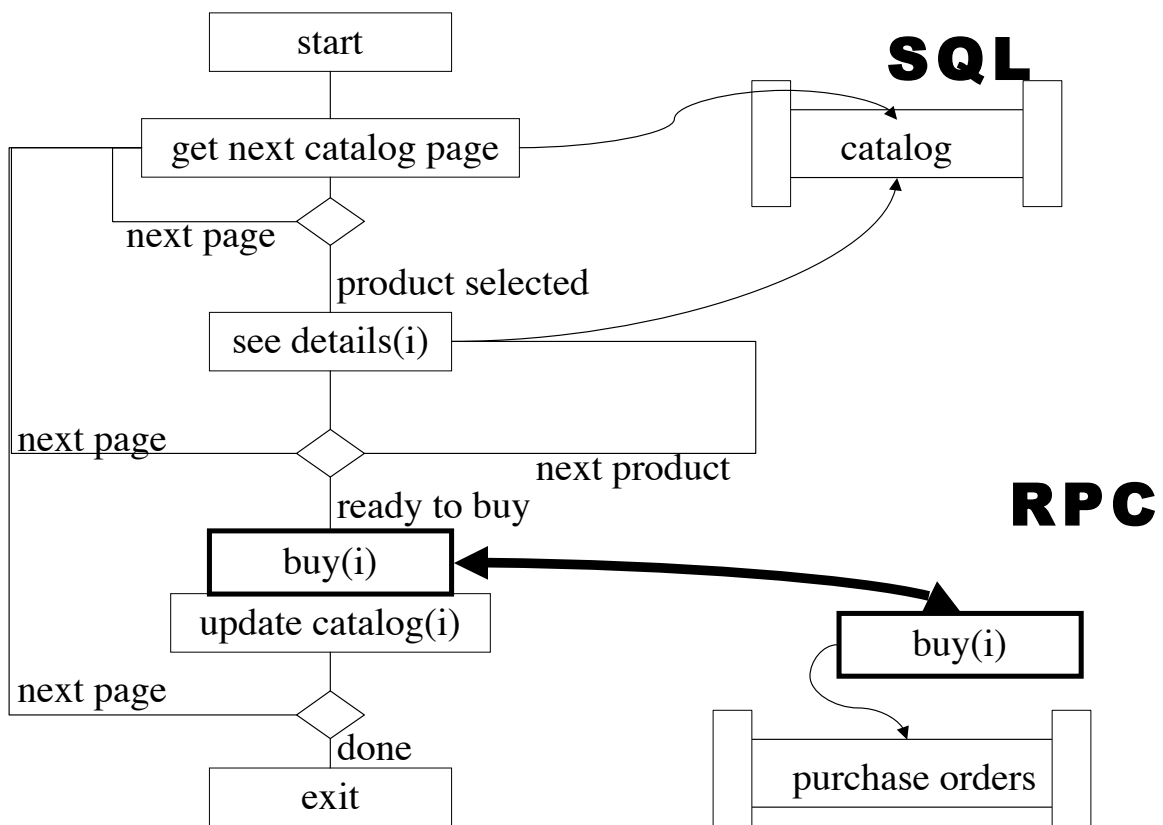
A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

11-07

A distributed application

11-07

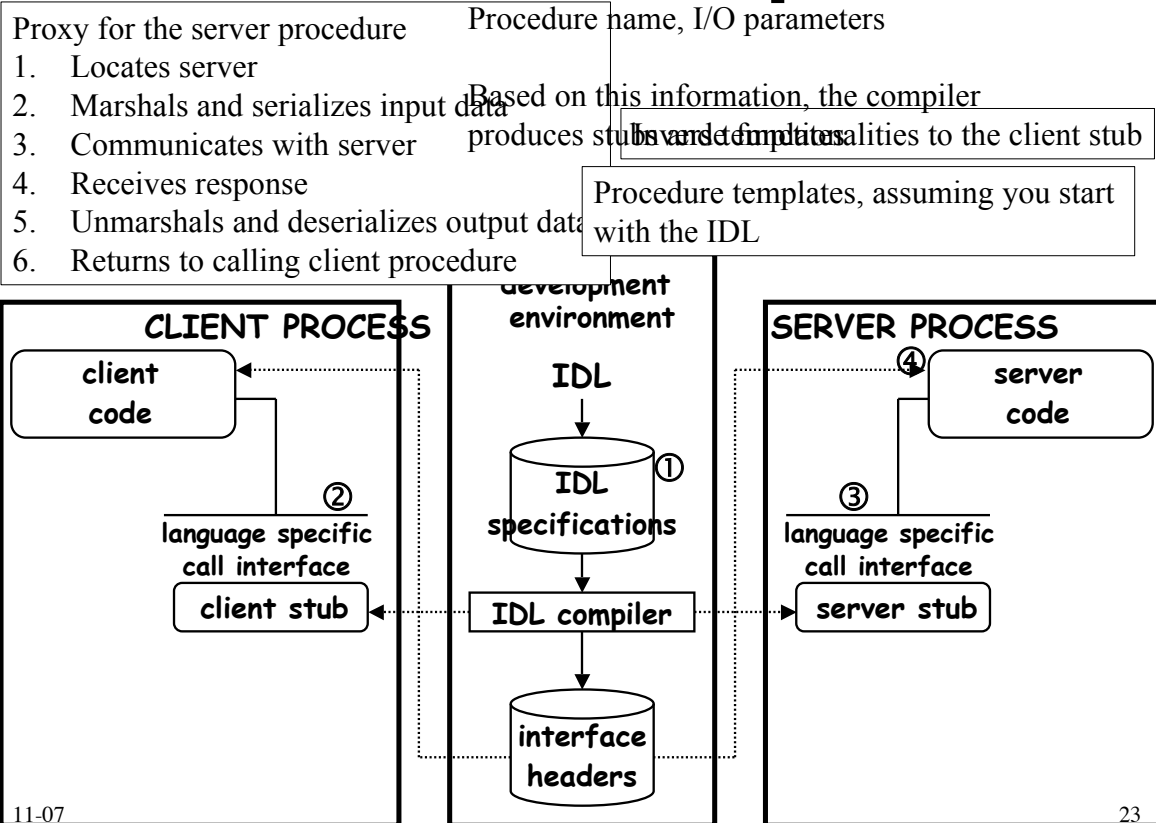
21



11-07

22

Middleware at compile time

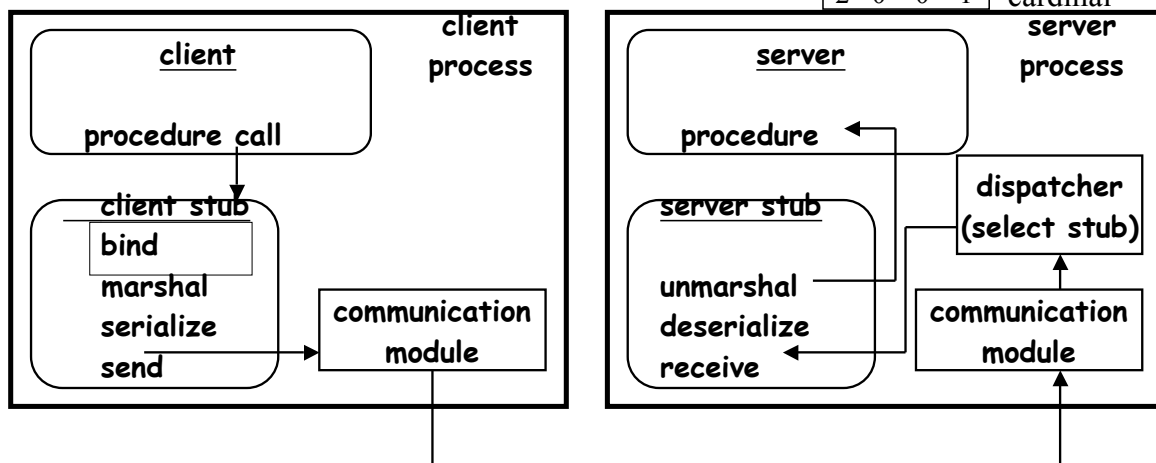


Middleware at run time

SUN XDR

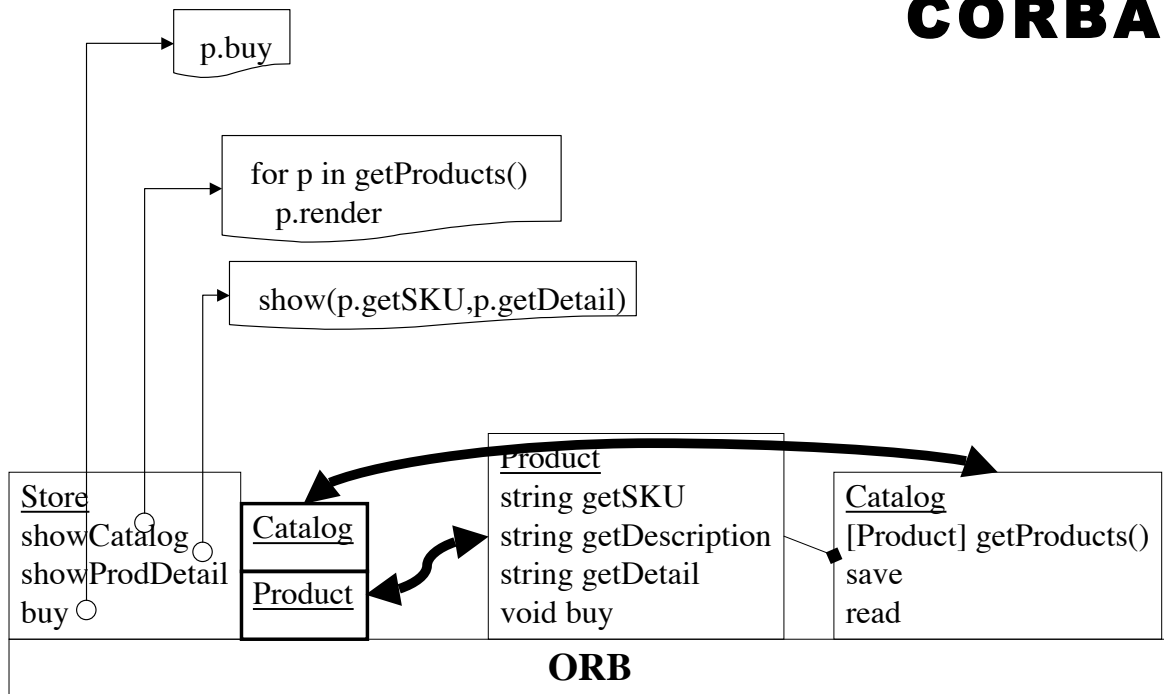
- messages are transformed into a sequence of 4 byte objects, each byte being in ASCII code
- it defines how to pack different data types into these objects, which end of an object is the most significant, and which byte of an object comes first
- simplifies computation at the expense of bandwidth

6	String length			
A	l	o	n	
s	o			
4	String length			
E	T	H	Z	
2	0	0	1	cardinal



Picture from "Web Services: Concepts, Architecture and Applications"
G. Alonso, F. Casati, H. Kuno, V. Machiraju
Copyright Springer Verlag Berlin Heidelberg 2004

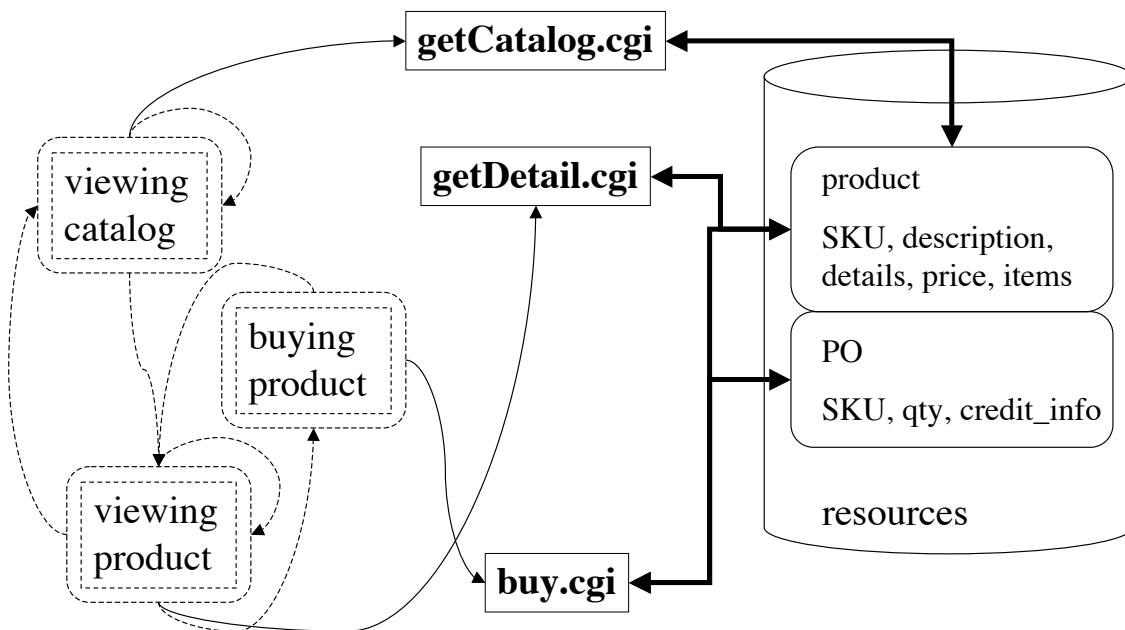
CORBA



11-07

25

web: REST



11-07

26

A REST web application

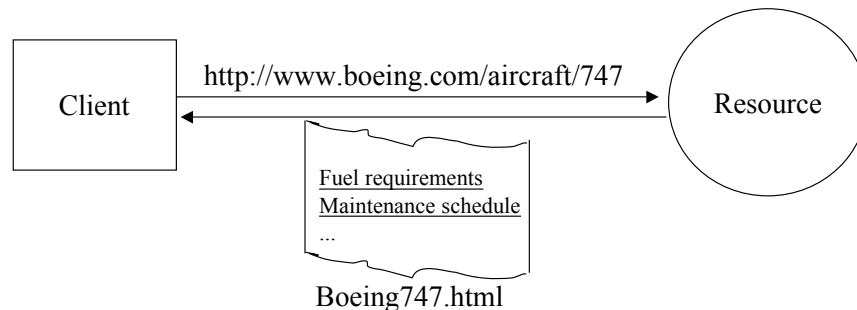
Based on

- <http://www.xfront.com/REST-Web-Services.html> and
- <http://www.xfront.com/sld001.htm>

11-07

27

Why it is called "Representation 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 application in a **state**. The result of the client traversing a hyperlink in Boeing747.html is the fact that another resource is accessed.
The new representation places the client application into yet another state.

The client changes (**transfers**) state with each resource representation
⇒ Representation State Transfer!

11-07

28

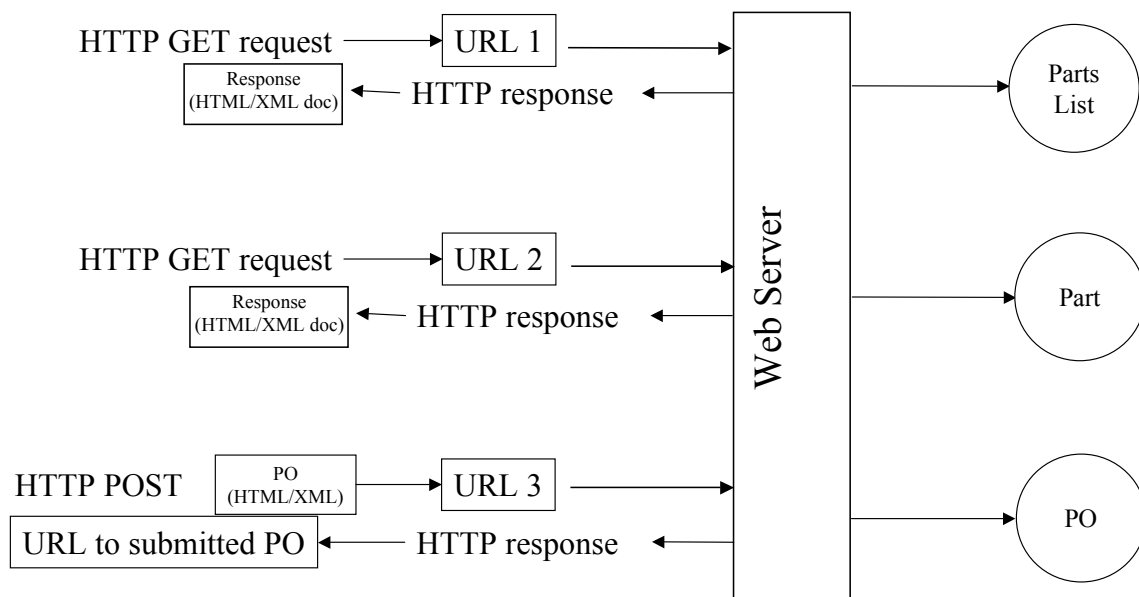
The Parts Depot

- 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)

11-07

29

The REST way of Implementing the Web Services



11-07

30

REST 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
 - How the parts list is generated is completely transparent to the client (loose coupling).
- There may be alternative representations of the same resource (HTML, XML)
- The representation forwarded to the client includes links to detailed info about each part.
 - The client can transfer to the next state by examining and choosing from among the alternative URLs in the response document.

11-07

31

Data Returned: Parts List

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.parts-depot.com
    http://www.parts-depot.com/parts.xsd">
  <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>
```

11-07

32

REST Service: Get part details

- The web service makes available a URL to each part resource.
- A client requests a specific part by accessing <http://www.parts-depot.com/parts/00345?flavor=xml>
- The response data is again linked to still more data - the specification for this part may be found by traversing the hyperlink.

11-07

33

Data Returned: Part

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.parts-depot.com
    http://www.parts-depot.com/part.xsd">
  <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>
```

11-07

34

Issues

- 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>
- Session maintenance through URL rewriting reflects logical URLs, expressing what resource is desired.
 - Parts Depot stores all parts data in a database. Code at the Parts Depot web site receives a logical URL request and generates the part response document which is returned to the client.
 - 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 point to physical (HTML) pages (unlikely as implementation)

11-07

35

Issues

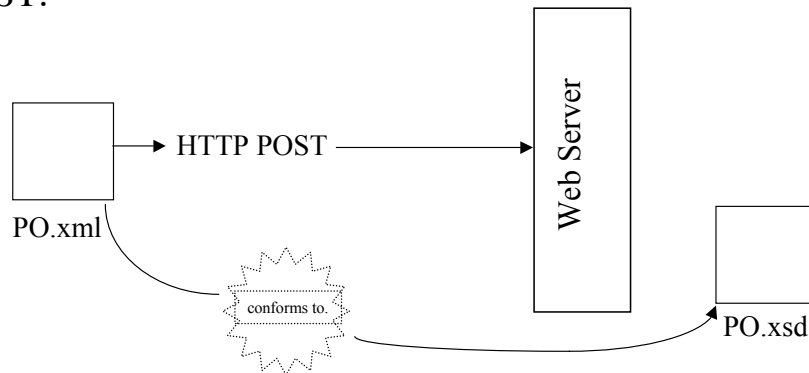
- What if I have a complex query?
 - For example: "show me all parts whose unit cost is under \$0.50 and for which the quantity is less than 10". How would you do that with a simple URL?
- Some responses are transient “views” of resources
 - For complex queries Parts Depot provides a service to allow a client to retrieve a form that would then be filled in by the client.
 - When the client hits "Go" the form gathers up the client's input and generates a URL based upon it. The client, in this case, generates a “recipe” for a URL

11-07

36

REST Service: Submit a PO

- The web service makes available a URL to submit a purchase order (PO)
 - The client creates a PO instance document which conforms to the PO schema that Parts Depot has designed (and publicized in a xsd document). The client submits PO.xml as the payload of an HTTP POST.

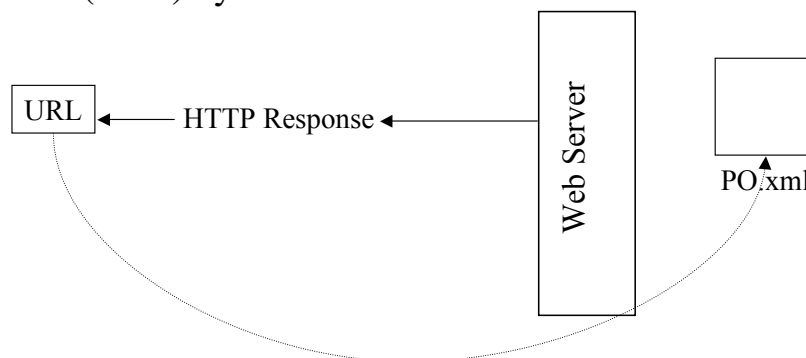


11-07

37

REST Service: Submit a PO

- The PO service responds to the HTTP POST with a URL to the submitted PO. Thus, the client can retrieve the PO any time thereafter.
 - The PO becomes a piece of information which is shared between the client and the server. The shared information (PO) is given an address (URL) by the server.



11-07

38

Resources

- Architecture of the World Wide Web, Volume One
 - <http://www.w3.org/TR/webarch/>
- Fielding's dissertation on REST
 - <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
 - http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- Building Web Services the REST Way, by Roger L. Costello
 - <http://www.xfront.com/REST-Web-Services.html> and
 - <http://www.xfront.com/sld001.htm>