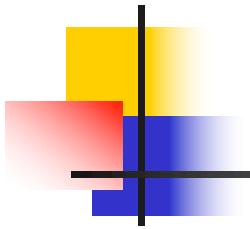
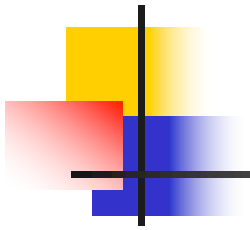
RESTful Web Services



RESTFUL Web Services

Problems with SOAP-based WS:

- SOAP-based WS uses GET/POST and "tunneled" RPC calls
- Operation call is "encoded" in XML, not an integral part of the protocol
- Uses HTTP (application-level protocol) as a *transport* protocol
- "Does not fit" within the basic WWW architecture

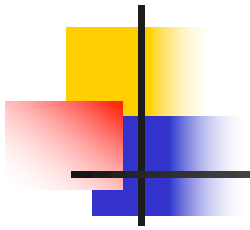


RESTFUL Web Services

- Enter REST…
- **REST** is

  REpresentational **S**tate **T**ransfer
- Coined by **Roy Fielding** in his Ph.D. dissertation[1] to describe a design pattern for implementing networked systems.

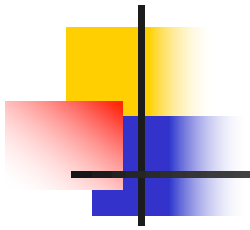
1. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm



RESTFUL Web Services

The basics: WWW

- World-Wide Web is centered around:
  - URI
  - HTML and XML (and now JSON)
  - HTTP



RESTFUL Web Services

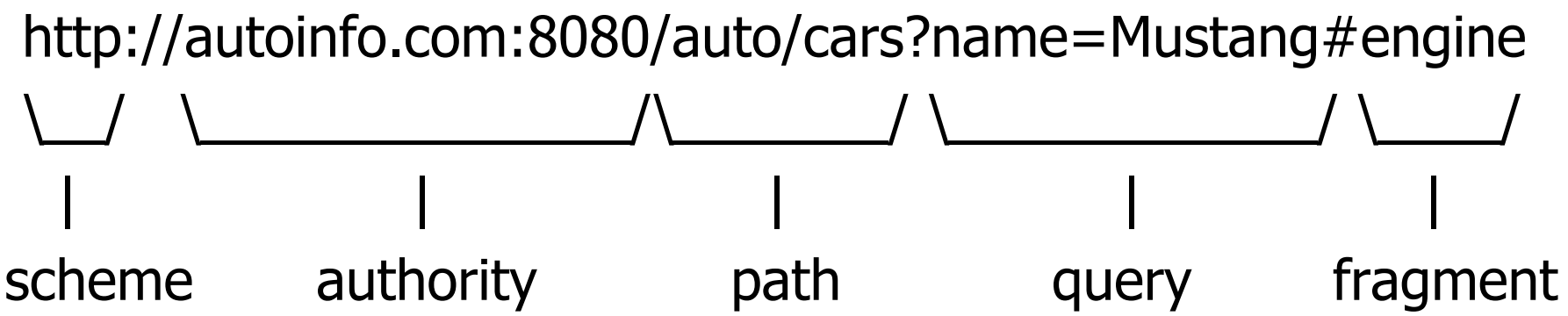
Uniform Resource Identifiers (URI)

- Essential for implementing a Resource Identification
- URIs are human-readable universal identifiers for available information
- Making everything a universally unique identified resource is important
- Today, we use IRIs (Internationalized Resource Identifiers)

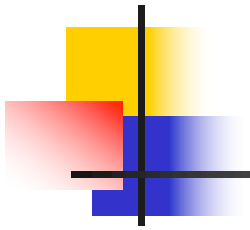
RESTFUL Web Services

- There are 2 types of IRIs:
  - URNs (Uniform Resource Names)

    e.g., urn:isbn:0405999832
  - URLs (Uniform Resource Locators)



Dozens of other schemes exist: https, ftp, telnet, mailto, ...



RESTFUL Web Services

A **URI** (usually a URL) represents a resource, which is usually some type of data. URI's path specifies which data. For example:

- A course offered in Fall 2020, e.g., CSCI 4060

  http://www.uga.edu/course/fall20/csci4060

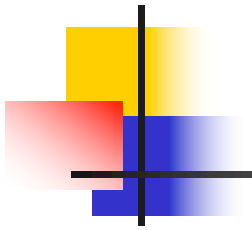
- A UGA student

  http://www.uga.edu/student/joe811334567

- Students enrolled in CSCI 4060, Fall 2020

  http://www.uga.edu/course/fall20/csci4060/enrollment

RESTFUL Web Services



- Car manufacturer, e.g., Ford

  http://car.info/manufacturer/ford

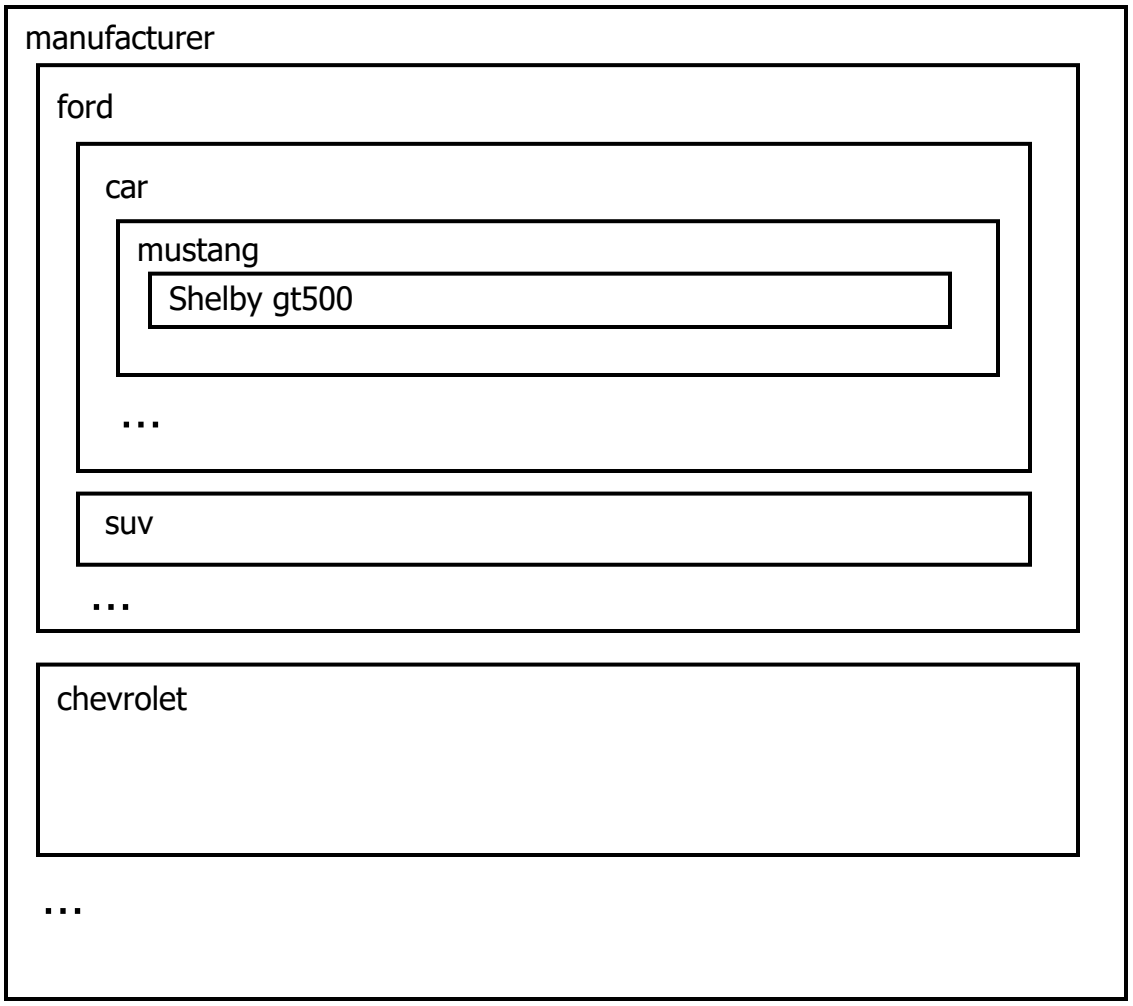
- A car model made by Ford, e.g., Mustang

  http://car.info/manufacturer/ford/car/mustang

- A Mustang's trim, e.g., Shelby GT500

  http://car.info/manufacturer/ford/car/mustang/shelbygt500

RESTFUL Web Services



Each rectangle is a resource

RESTFUL Web Services

Consider the following Java classes:

```
class Manufacturer { public List<Car> car;
                     public List<SUV> suv;
                     public List<Truck> truck; … }
class Car { public String model;  public List<Trim> trim; … }
class Trim { public String name; … }

List<Manufacturer> manufacturers;
Manufacturer manufacturer = manufacturers.get(0); //get Ford
Car car = manufacturer.car.get(0); // get Mustang
Trim trim = car.get(0); // get Shelby GT500
trim.getHorsePower();
trim.setHorsePower(760);
```

RESTFUL Web Services

Contrast this to URI-style of identifying data

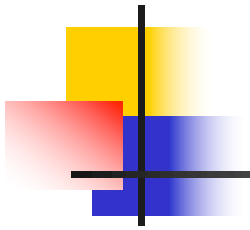
- A Mustang's trim, e.g., Shelby GT500:

http://car.info/manufacturer/ford/car/mustang/shelbygt500

_____/

|

path



RESTFUL Web Services

A resource (data) can be represented using:

- HTML
- XML
- JSON (Java Script Object Notation)
- RDF (Resource Description Framework)
- ATOM (A language for representing syndication feeds)

RESTFUL Web Services

For example:

http://cars.info/manufacturers/ford/cars/mustang/shelbygt500

XML representation:

```
<car>
   <name>Shelby GT500</name>
   <year>2020</year>
   <engine>5.2L Supercharged Cross Plane Crank V8</engine>
   <transmission>
      TREMEC 7-Speed Dual Clutch Transmission
   </transmission>
   ...
   ...
</car>
```

RESTFUL Web Services

HTTP (Hyper Text Transfer Protocol)

- Essential for implementing a Uniform Interface
- HTTP defines a small set of methods (the *Web's API)* for acting on URI-identified resources; amazingly, all Web exchanges are done using this simple HTTP API:
  - GET = "give me some info" (Read)
  - POST = "here's some new info" (Create)
  - PUT = "here's some updated info" (Update)
  - DELETE = "delete some info" (Delete)

  (additional methods exist but will not be discussed here)

RESTFUL Web Services

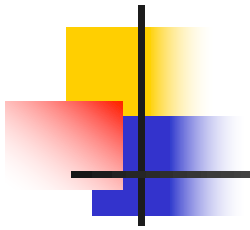
- The HTTP API follows CRUD, commonly used abbreviation of database operations:

  Create, Read, Update, and Delete, as they correspond to the basic SQL operations on a database table:

  - Create (add rows),
  - Read (retrieve rows),
  - Update (modify rows), and
  - Delete (remove rows)

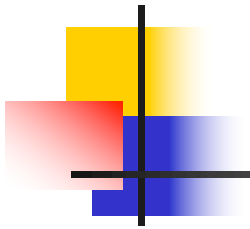
RESTFUL Web Services

- Safe (HTTP) methods can be repeated (without side-effects)
  - arithmetically safe; think of: `System.out.print(x);`
  - in practice, “without side-effects” means “without relevant side-effects”
- Idempotent methods can be repeated (with side-effects)
  - arithmetically idempotent: think about assignment: `x = 41;`
  - in practice, “with side-effects” means “the side-effects are repeatable and have the same resulting state”
- Non-idempotent (unsafe) methods should be treated with care. Think about an increment statement: `x++;`
  - HTTP has two main safe methods: GET HEAD
  - HTTP has two main idempotent methods: PUT DELETE
  - HTTP has one main non-idempotent method: POST



RESTFUL Web Services

"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 Felding



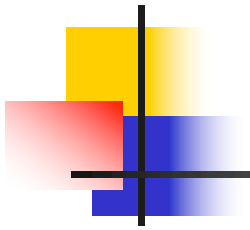
RESTFUL Web Services

State Transfers: Example

- The client requests a Web resource using some URI;
- A **representation** of the resource is returned (in this case as, e.g., an HTML document);
- The representation (*e.g.,* flightDL2231.html) places the client in a new **state**;
- When the client selects a hyperlink (e.g., delta.com) in flightDL2231.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.

RESTFUL Web Services

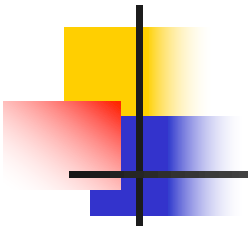
- REST is **not a standard**
  - You will not see the W3C putting out a REST specification.
  - You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.
- REST is a **design pattern**
  - You can't create a library with an implemented pattern.
  - You can only understand it and design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



RESTFUL Web Services

Designing RESTful Services

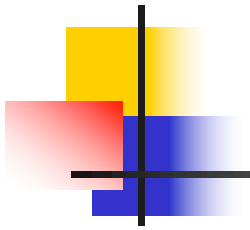
- Implementing a style, **not a protocol**
  - *Resources* rather than *operations*
  - *Nouns* rather than *verbs*
- *Resource-oriented design*
- What the data *is* instead of what you can *do* with the data



RESTFUL Web Services

Accessing RESTful Services

- REST is meant for interoperability
- Built on the same technology as WWW
- Can be "consumed" in many ways
  - Web browser
  - Web services
  - Lightweight clients
  - Command line (curl, wget)
- May be used by components of an SOA (Service Oriented Architecture), but need not to be



RESTFUL Web Services

Example

- Simple on-line banking service
  - Authenticated
  - Gives user-specific information about a customer, customer's profile, accounts, other resources, etc.
  - Provide resources represented as HTML and/or XML

RESTFUL Web Services

RESTful design: use standard methods, and consider their meaning and applicability w.r.t. resources (URIs)

/profiles	GET – list all user profiles PUT – unused POST – add a new user profile DELETE – unused
/profiles/id	GET – get user profile details PUT – update user profile POST – unused DELETE – delete user profile
/resources	GET – list all resources PUT – unused POST – add resource DELETE – unused
/resources/id	GET – get resource details PUT – update resource information POST - unused DELETE – delete resource
/accounts	GET – list all accounts PUT – unused POST – add account DELETE - unused
/accounts/id	GET – get account details PUT – update account details POST - unused DELETE – delete account

RESTFUL Web Services

- A resource and sub-resources have unique URIs
- Sample resources may be:

/profile	/customer/patel
/profile/smith	/account
/customer	/account/1144
/customer/smith	/account/1144/balance

- Here, these are human-readable, but no real need for them to be so.

RESTFUL Web Services

The fixed HTTP operations on these URIs would:

GET on /customer – return data on all customers

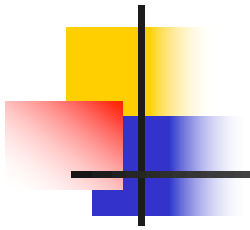
POST on /customer + XML data on a new customer
– create a new customer
returns new cust's URI

GET on /customer/smith – return data on cust. Smith

PUT on /customer/smith + XML data w/ updates – update data on customer Smith

DELETE on /customer/smith – delete cust. Smith

RESTFUL Web Services



GET on /account – return data on all accounts

POST on /account + XML data on a new account
– create a new account
returns new acct's URI, e.g.,
/account/1144

RESTFUL Web Services

GET on /account/1144 – return data on acct 1144

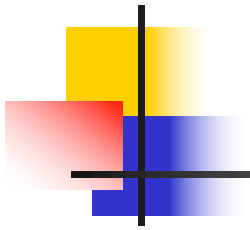
PUT on /account/1144 + XML data on a new new data – update data on acct 1144

GET on /account/1144/balance – return balance of acct 1144

PUT on /account/1144/balance – update balance of acct 1144

DELETE on /account/1144 – delete acct 1144

RESTFUL Web Services



- HTTP is the API (operations)
- How to combine this API with resources?
  - operations are “fixed”, so
  - create many URIs to represent many parts (aspects) of data, then
  - use a specific URI to read, create, update, delete that specific part of data
  - Note: not all operations will make sense on all URIs

RESTFUL Web Services

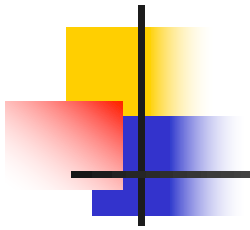
Assume we want to maintain a section of students (roll list):

```
class Section {
    Section(...) {...}
    void              addStudent( Student s ) {...}
    void              updateStudent( int id, Student s ) {...}
    List<Student> getStudents() {...}
    void              delStudent( int id ) {...}
    String            toString() {…}
}
```

Java code	**RESTful HTTP method invocations**
s = new Section(…)	/section **POST** + section data return: new URI: /section/123
s.toString()	/section/123 **GET** return: representation of section 123

RESTFUL Web Services

Java code	**RESTful HTTP method invocations**
s.addStudent(stud)	/section/123/student **POST** + stud. data return: /section/123/student/983
s.getStudent(983)	/section/123/student/983 **GET** return: represention of student 983
s.getStudents()	/section/123/student **GET** return: representation of all students in 123
s.updateStudent(983, stud)	/section/123/student/983 **PUT** + stud. data return: OK
s.deleteStudent(983)	/section/123/student/983 **DELETE** return: OK
s.getStudent(983)	/section/123/student/983 **GET** return: 404



RESTFUL Web Services

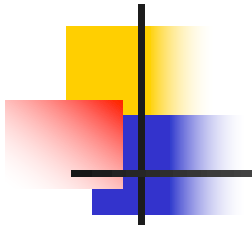
- The data (resource representation) that a web service returns should link to other related data (web of resources).
- Thus, design your data as a *network of information*.

RESTFUL Web Services

GET /parts

```
<?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.

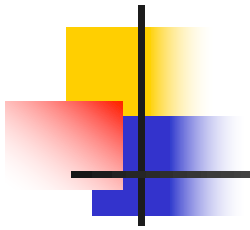


RESTFUL Web Services

GET /parts/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>
```

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

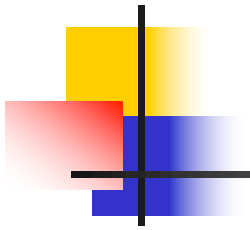


RESTFUL Web Services

REST Recap:

Resources: things (data), entities

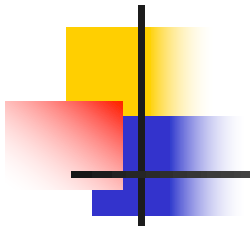
- Products
- Categories of products
- Customers
- Shopping carts
- University courses
- Students
- …



RESTFUL Web Services

Uniform interface

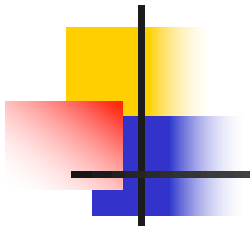
- The *same set of operations* applies to everything
- Small set of *verbs* applies to a large set of *nouns*
- *Verbs are universal* and do not differ per application



RESTFUL Web Services

Uniform interface

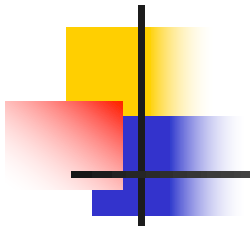
- GET, HEAD are *safe*
  - no change to a resource state
- PUT, DELETE are *idempotent*
  - possible change of a resource state, but can be repeated with the same resulting state
- POST is *unsafe* and non-idempotent
  - can have side-effects; each invocation will yield a different resource state



RESTFUL Web Services

Resource access via representations

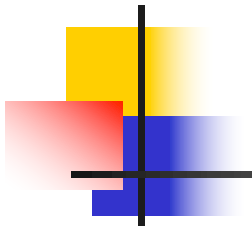
- Resource representation is sufficient
- Format can be negotiated
- Whatever representation is used, it must support links to other resources



RESTFUL Web Services

State

- Represented in the client
- Abstracted in resource representations w/ embedded links
- Navigable – possible state changes.



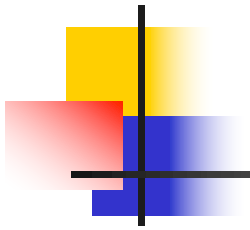
RESTFUL Web Services

RESTFul Design Methodology (Pautasso)

- Identify resources to be exposed as services, e.g.:
  - yearly sales report
  - book catalog
  - purchase order
  - open bugs
  - polls and votes, etc.
- Model relationships (e.g., containment, reference, state transitions) between resources with hyperlinks that can be followed to get more details (or perform state transitions)

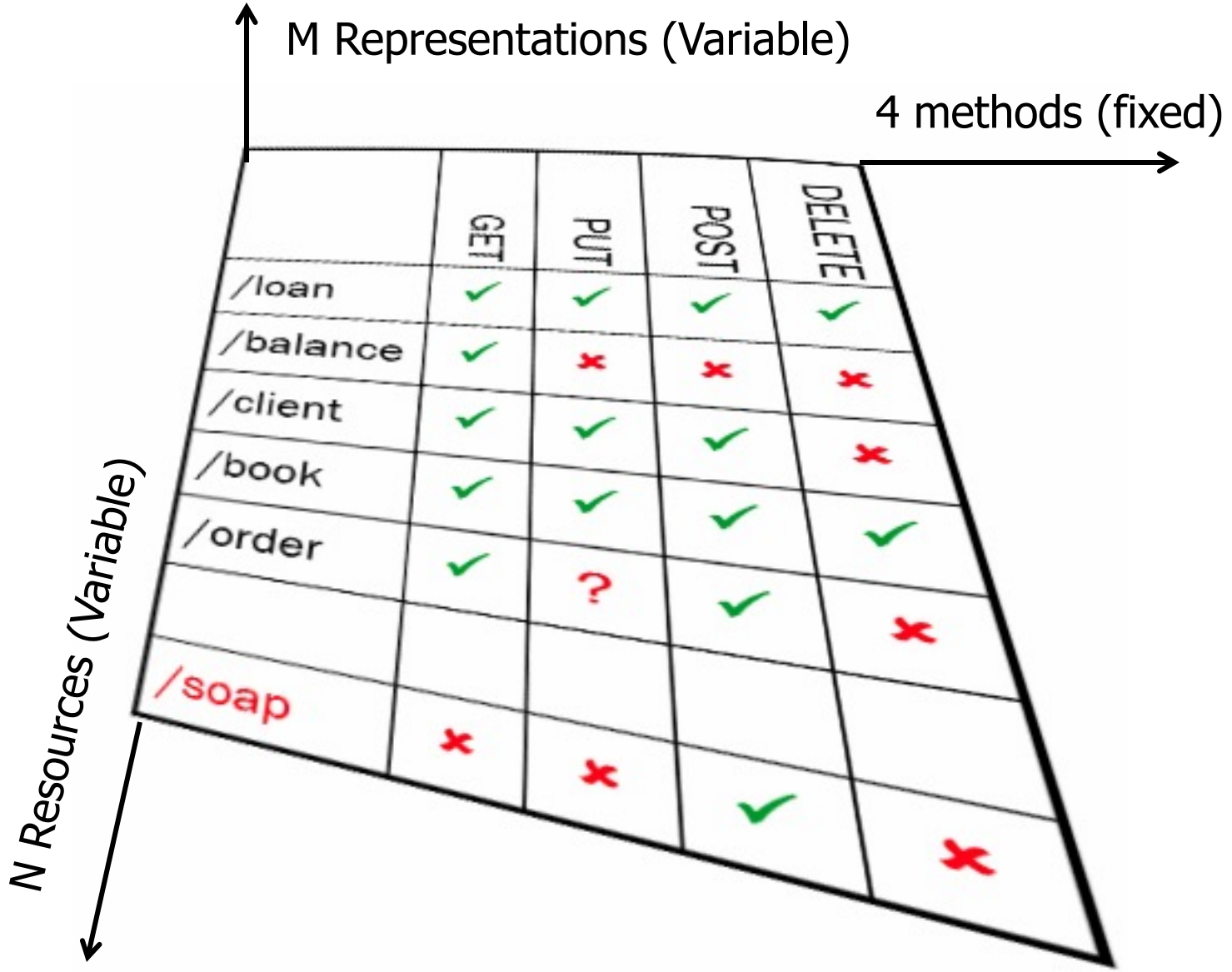
T. Erl, B. Carlyle, C. Pautasso, and R. Balasubramanian, *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST,* Prentice Hall, 2012.

RESTFUL Web Services



- Define "nice" URIs to address the resources
- Understand what GET, POST, PUT, and DELETE mean when invoked on each resource (and whether it is allowed or not)
- Design and document resource representations
- Implement and deploy on an Application Server

RESTFUL Web Services



RESTFUL Web Services

- Prefer Nouns to Verbs
- Keep your URIs short
- Follow a "positional" parameter-passing scheme (instead of the key=value&p=v encoding)
- URI postfixes can be used to specify the content type
- Do not change URIs
- Use redirection if you really need to change them

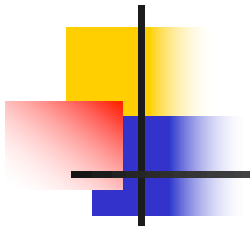
GET /book?isbn=245566&action=delete

instead, use:

DELETE /book/245566

Note: REST URIs are opaque identifiers that are meant to be discovered by following hyperlinks and not constructed by the client

Warning: URI Templates introduce coupling between client and server



RESTFUL Web Services

What is the right way of creating resources (initialize their state)?

POST /resource

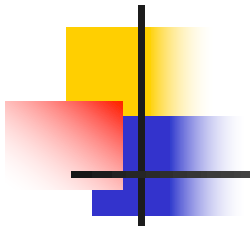
201 Created

Location: /resource/{id}

Let the server compute the unique id.

Problem: Duplicate instances may be created, if requests are repeated due to unreliable communication.

RESTFUL Web Services



- A server always responds with a status code, which indicates the basic outcome of a request.
- The most common status codes are:
  - 200 OK – indicating success, and
  - 404 Not Found – indicating that the requested resource was not found on the server

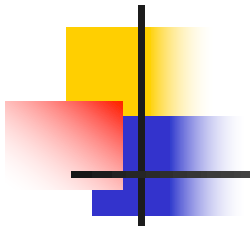
RESTFUL Web Services

Standard HTTP Status Codes

Information: 1xx
Success: 2xx
Redirection: 3xx
Client Error: 4xx
Server Error: 5xx

100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect

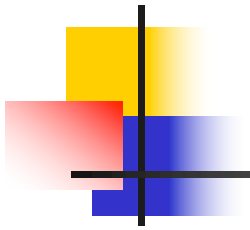
RESTFUL Web Services



4xx Client's fault

400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed

RESTFUL Web Services

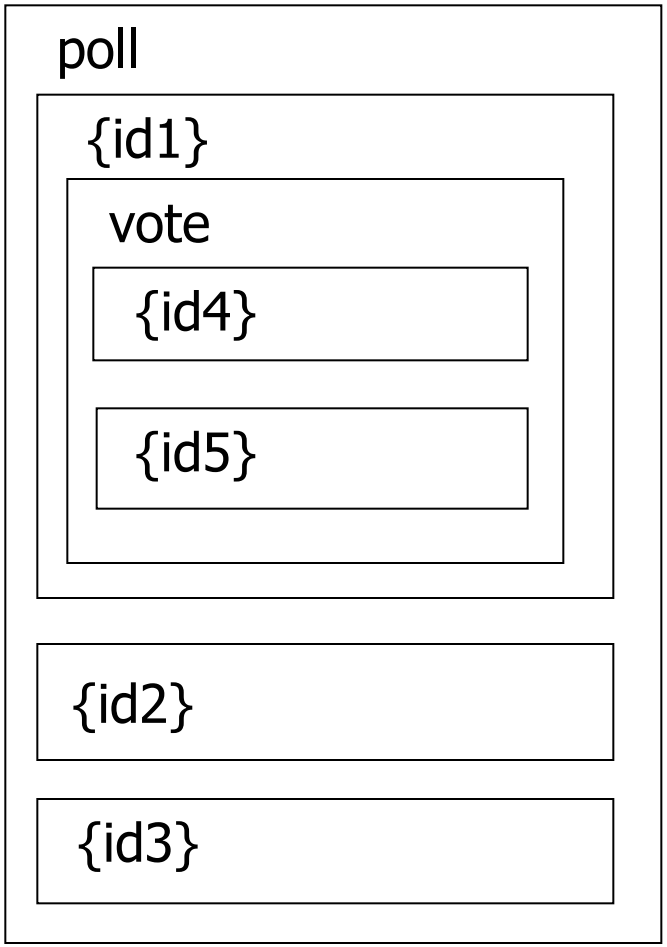


5xx Server's fault

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

RESTFUL Web Services

1. Resources: **polls** and **votes**
2. Containment Relationship:

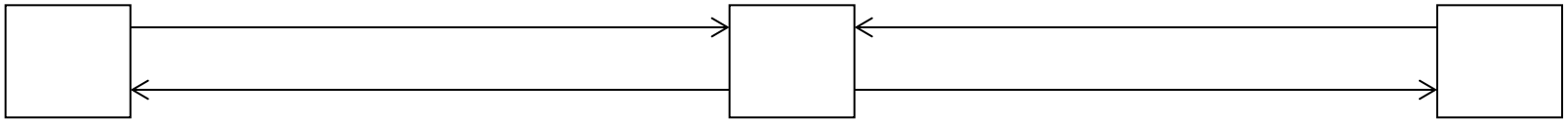


	GET	PUT	POST	DELETE
/poll	Y	N	Y	N
/poll/{id}	Y	Y	N	Y
/poll/{id}/vote	Y	N	Y	N
/poll/{id}/vote/{id}	Y	Y	N	?

3. URIs embed IDs of "child" instance resources
4. POST on the container is used to create child resources
5. PUT/DELETE for updating

RESTFUL Web Services

1. Creating and reading a poll on URIs:
   /poll
   /poll/090331x
   /poll/090331x/vote -- available URI



POST /poll
<options>A,B,C</options>

201 Created
Location: /poll/090331x

GET /poll/090331x

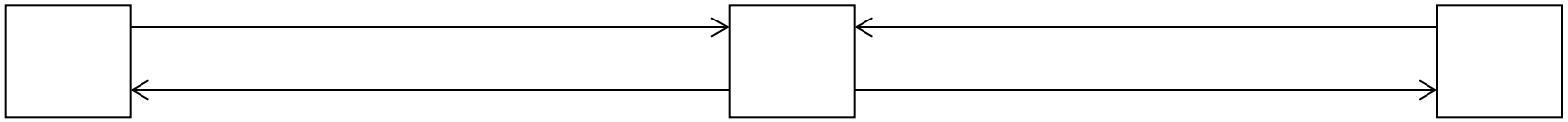
200 OK
<options>A,B,C</options>
<votes href="/vote"/>

From: Cesare Pautasso, Erik Wilde

RESTFUL Web Services

2. Creating a poll by calling POST on a vote sub-resource

/poll/090331x/vote



POST /poll/090331x/vote
<name>Question 1</name>
<choice>B</choice>

201 Created
Location: /poll/090331x/vote/1

GET /poll/090331x

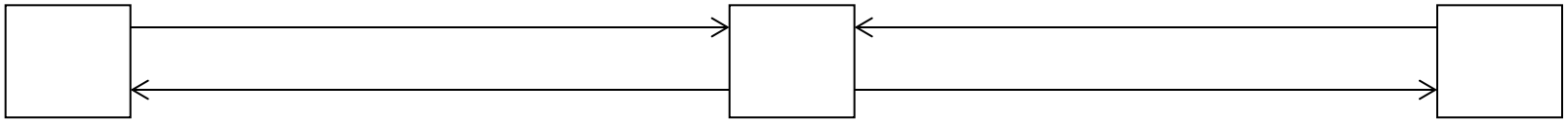
200 OK
<options>A,B,C</options>
<votes>
  <vote id="1">
    <name>Question 1</name>
    <choice>B</choice>
  </vote>
</votes>

From: Cesare Pautasso, Erik Wilde

RESTFUL Web Services

3. Existing votes can be updated (access control headers not shown)

/poll/090331x/vote/1



PUT /poll/090331x/vote/1
<name>Question 1</name>
<choice>C</choice>

200 OK

GET /poll/090331x

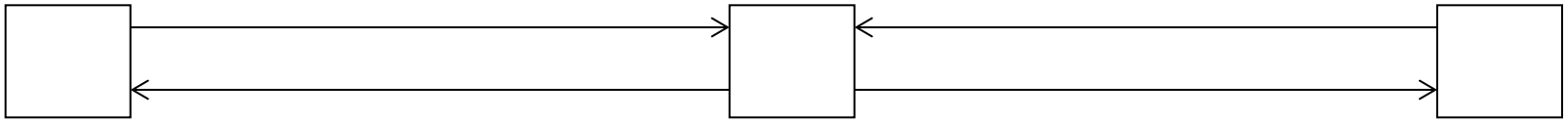
200 OK
<options>A,B,C</options>
<votes>
  <vote id="1">
    <name>Question 1</name>
    <choice>C</choice>
  </vote>
</votes>

From: Cesare Pautasso, Erik Wilde

RESTFUL Web Services

4. Polls can be deleted once a decision has been made

/poll/090331x



DELETE /poll/090331x

200 OK

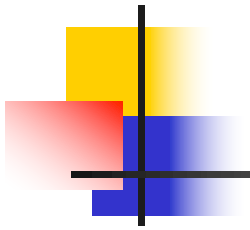
GET /poll/090331x

404 Not Found

They used to publish their API at http://doodle.com/xsd1/RESTfulDoodle.pdf but not anymore.

From: Cesare Pautasso, Erik Wilde

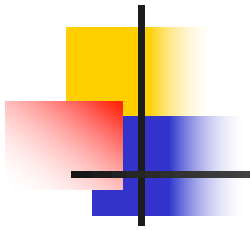
IDL for RESTFUL Services



- Describing RESTful services has been a problem for some time; no WSDL equivalent notation for REST
- Recently, Swagger and its successor OpenAPI (v3.1.0) have been gaining popularity:

  https://spec.openapis.org/oas/latest.html

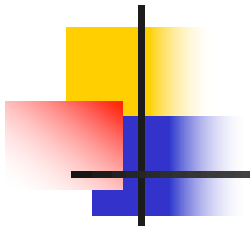
- It is integrated with JSON schema, a recent addition to JSON format



RESTFul WS: JAX-RS

Implementations

- Jersey
- Restlet
- **JBoss RESTEasy**
- Jakarta REST
- And many others...

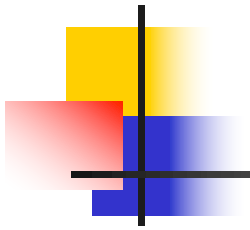


RESTFul WS: JAX-RS

- JAX-RS is an acronym for Java API for RESTful Web Services
- Java classes and methods are annotated as processors of REST requests
- In JAX-RS, a Resource is a POJO

  (POJO = Plain Old Java Object)
  - No interface to implement
  - Just define the matching URI

RESTFul WS: JAX-RS



- Resource (class) methods are annotated to respond to a combination of:
  - Method type (GET, POST, etc.)
  - Requested URI (as a string or a regex)
  - Requested (or allowed) content type (XML, JSON, etc.)

RESTFul WS: JAX-RS

- For example: GET /phonebook/11234

```
@Path( "/phonebook" )
public class PhoneBookResource {
   ...
   @GET
   @Path( "{id}" )
   @Produces( MediaType.APPLICATION_XML )
   public Person getEntryXML( @PathParam("id") Integer id )
```

RESTFul WS: JAX-RS

- For example: GET /phonebook/11234

```
@Path( "/phonebook" )
public class PhoneBookResource {
   ...
   @GET
   @Path( "{id}" )
   @Produces( MediaType.APPLICATION_XML )
   public Person getEntryXML( @PathParam("id") Integer id )
```

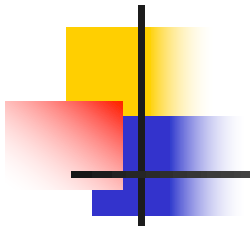
Base path for the URI

Method to be handled

Additional path element

Request URI: /phonebook/{id} Response: XML

Resource representation

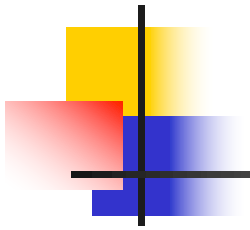


RESTFul WS: JAX-RS

- If your method returns `void`, the request processing method should return a 204 (successfully processed, no message body)

- Automatic URL encoding is provided
  - `@Path("product list")` is identical to `@Path("product%20list)`

RESTFul WS: JAX-RS



- Annotate methods with:
  - **`@GET`**
  - **`@PUT`**
  - **`@POST`**
  - **`@DELETE`**
  - Other HTTP methods available, as well
- JAX-RS forwards to the correct method, based on the request method

RESTFul WS: JAX-RS

- Annotate classes and/or methods with:

```
@Path( "path" )
```

  to create a method responding to a given path in the URI.

- It is possible to include a fragment `{id}`, for example:

```
@Path( ”/account/{number}" )
```

  to obtain a variable parts of the URI, e.g., some resource identifier included in the URI.