Topics

RPC vs. REST: What's the difference and why does it matter?	What makes an API “truly” RESTful? (Constraints and interface)	REST best practices (common problems and their solutions)
Planning and developing an API (project life cycle)	Drupal services (Services 3, WSCCI)	Questions?

Protocol vs. Architecture

- Remote Procedure Call (RPC)
- Representational State Transfer (REST)

- XML
- JSON
- CSV

Characteristics of a robust API

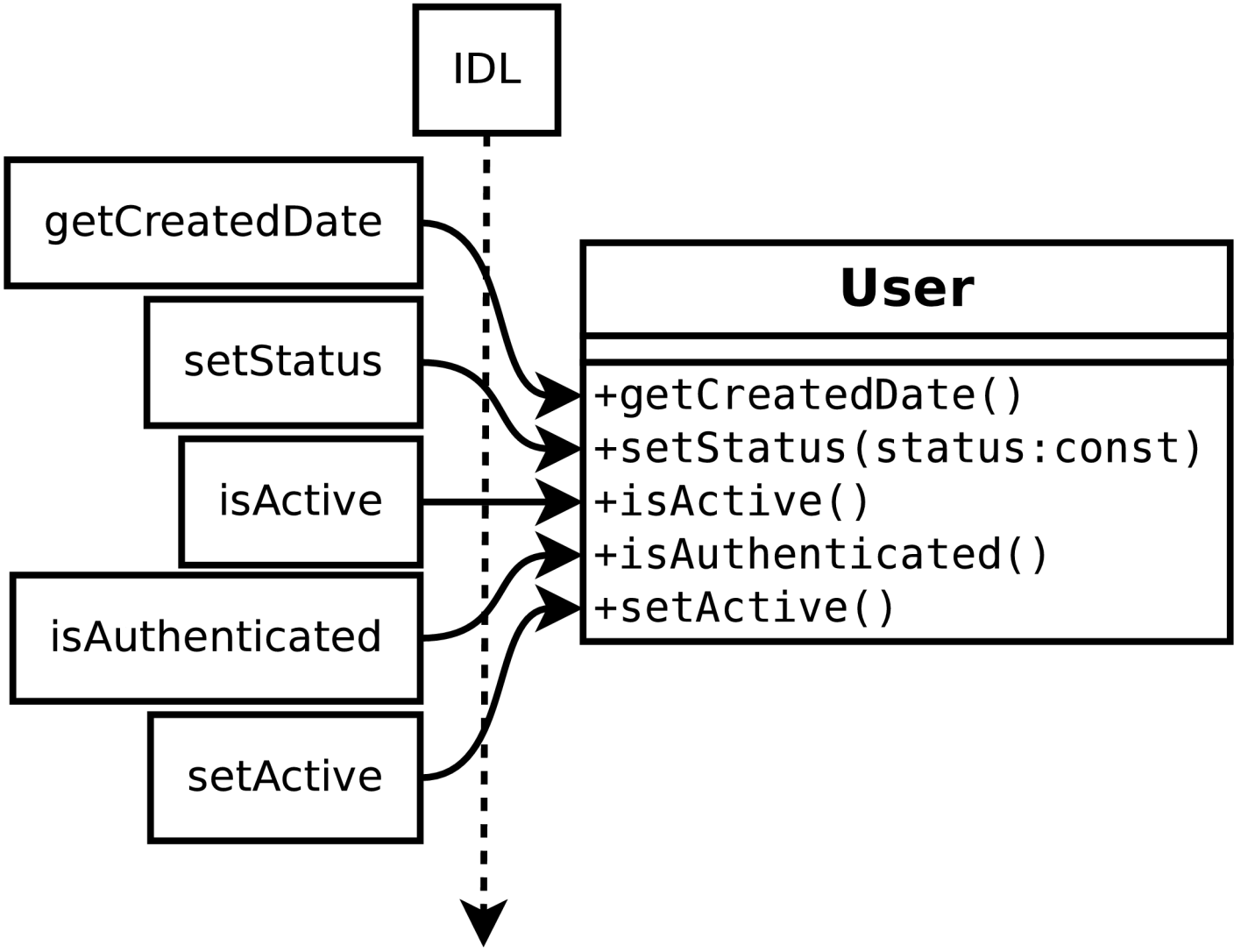
- Easy to evolve
- Interoperable
- Simple
- Discoverable

Affordance: The design itself communicates how it is meant to be used

Project Workflow

- Gather requirements and use cases.
- Write a broad (one page) spec.
- Gather stakeholder feedback.
- Iteratively identify and apply constraints
  - Write to the API early and often.
  - Mock response sets.
  - Prototype client code.

RPC


IDL
getCreatedDate
setStatus
isActive
isAuthenticated
setActive
User
+getCreatedDate()
+setStatus(status:const)
+isActive()
+isAuthenticated()
+setActive()

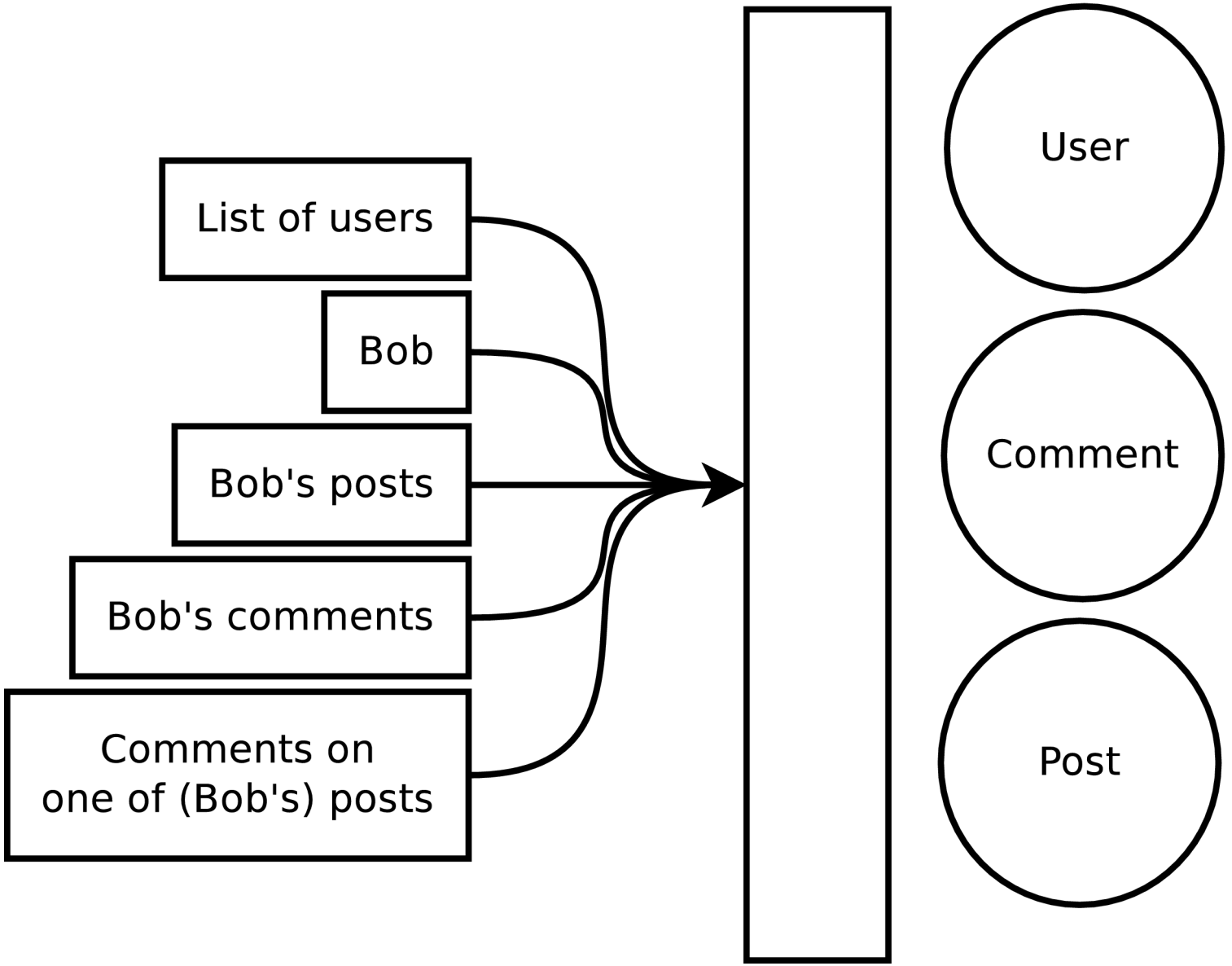
Example Soap call

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
<?xml version="1.0"?>
  <soap:Envelope xmlns:soap="...">
    <soap:Header>
    </soap:Header>
    <soap:Body>
      <m:GetStockPrice xmlns:m="">
        <m:StockName>IBM</m:StockName>
      </m:GetStockPrice>
    </soap:Body>
  </soap:Envelope>
</xml>
```

RPC

- Implementation leak
  - Implementation leaks into the interface
- Stateful
  - Latency can lead to (partial) failure
- Concurrency (caching)
- Impedence mismatch
  - IDLs add complexity
  - Some data types are difficult or impossible to map

REST


List of users
Bob
Bob's posts
Bob's comments
Comments on
one of (Bob's) posts
User
Comment
Post

REST: Constraints

- Client – server (separation of concerns)
- Stateless
- Cacheable
- Layered
- Uniform interface

REST: Interface Characteristics

- Identification of resources
- Manipulation through representations
- Self descriptive messages
- Hypermedia as the engine of application state

Rest: Client - Server

- Uniform interface
- Clear separation of concerns

<insert graphic here>

REST: Stateless

- Responses contain all the information necessary for servicing the request
- State (e.g., session) held by client
- Server side state is addressable by resource

REST: Cacheable

- Clients can cache responses
- Resources must implicitly or explicitly indicate whether and how they are cacheable

REST: Layered

- A client cannot tell whether it is directly connected to the server or to an intermediary
- Allows the use of caching proxys, etc.

  <insert graphic here>

REST: Uniform Interface

- Simplify and decouple architectural layers
- Separate concerns can evolve independently of one another

REST: Nouns and Verbs

Collections		
/v1/users	POST	Create a new user
/v1/users	GET	List users
/v1/users	PUT	Replace users with users
/v1/users	DELETE	Delete all users
Entities		
/v1/users/123	POST	Create or update Joe
/v1/users/123	GET	Show Joe
/v1/users/123	PUT	Create or update Joe
/v1/users/123	DELETE	Delete Joe

REST: Nouns and Verbs (2)

Create	PUT	iff you are sending the full content of the specified resource (URL)
Create	POST	if you are sending a command to the server to create a subordinate of the specified resource, using some server-side algorithm.
Retrieve	GET	
Update	PUT	iff you are updating the full content of the specified resource.
Update	POST	if you are requesting the server to update one or more subordinates of the specified resource.
Delete	DELETE	

REST: Resource addressing

- Refer variations to the query string
  - http://www.example.com/v1/users?disabled=true
- Associations build on existing URLs
  - http://www.example.com/v1/users/123/posts gets the posts belonging to a user
  - http://www.example.com/v1/posts/321/users gets the users belonging to a post

REST: Resources addressing

- Allow partial responses
  - **LinkedIn**: /people:(id,first-name,last-name,industry)
  - – Requires a documentation lookup (but you're going to have a hard time googling “:(“
  - **Google**: ? fields=title,media:group(media:thumbnail)
  - **Facebook**: /joe.smith/friends? fields=id,name,picture

REST: Resource Addressing

- Require versioning by including it in the URL
  - RESTafarian: send version back in the response
  - /v1/users (version is the highest level of scope)
  - Don't use minor version
- Account for incomplete HTTP implementations
  - Some clients do not support PUT and DELETE
  - Put the method in as an optional param: /dogs?method=put&location=park
  - Use “magic” methods?

REST: Responses

- Keep responses consistent, natural and pluggable
  - **Digg**: Accept: application/json, ?type=json (type overrides Accept)
    - Using the HTTP Accept header is more truly restful but can break mobile and flash applications
    - Providing default and fallback behaviors requires additional documentation
  - **Google**: ?alt=json
    - Results in a more verbose query string
    - Makes format optional, which implies a default behavior that needs to be documented
  - **Foursquare**: venue.json
    - Keeps the query string clean, natural, and semantic

REST: Responses

- All collections should include count and pagination
- Pagination should follow convention
- **Facebook**: page, rpp
  - “rpp” means “Records per page”. This isn't obvious and will result in a documentation lookup.
- **LinkedIn**: start, count
  - Meaning of “start” and “count” is clear and the math is straightforward.
- **Twitter**: offset, limit
  - Offset and limit are familiar from popular RDBMs. This is the clearest of the three implementations.

REST: Responses

- Attributes should be named according to the convenions of the output format (e.g., lower camel case for JSON)
  - **Twitter**: “created_at”: “Thu Nov 03 05:19:38 +0000 2011”
  - **Bing**: “DateTime”: “2011-10-29T09:35:00Z”
  - **Foursquare**: “createdAt”: 1320296464

REST: Responses

- USE URLs or URL templates

```
{"photos"[
"http://example.com/images/1.jpg",
"http://example.com/images/1.jpg"]}
```

OR

```
{"photos":{
"ids":[1,2],
"uritpl":"http://.../images/{id}",
}}
```

REST: Errors

- Give verbose messages in the response payload with as many hints as possible as to what might be going wrong
  - Error messages are not the place to save bandwidth
  - Informative messages save on documentation lookups
- Make rigorous use of HTTP response codes
  - POST should respond with Location and “201 Created”
- Provide for the suppression of response codes and HTTP errors
  - Support clients and apps (Flash) that don't handle response codes nicely
  - Move response codes and messages into the response payload
  - Make the suppression parameter impossible to miss
  - **Twitter**: ?suppress_response_codes=true HTTP Code 200 {”error” : “Could not authenticate you.” }

REST: Authentication

- Make authentication pluggable
- DO NOT invent your own authentication scheme.
  - OAuth
  - HTTP Auth
  - Key exchange

REST: Search

- Just use Solr