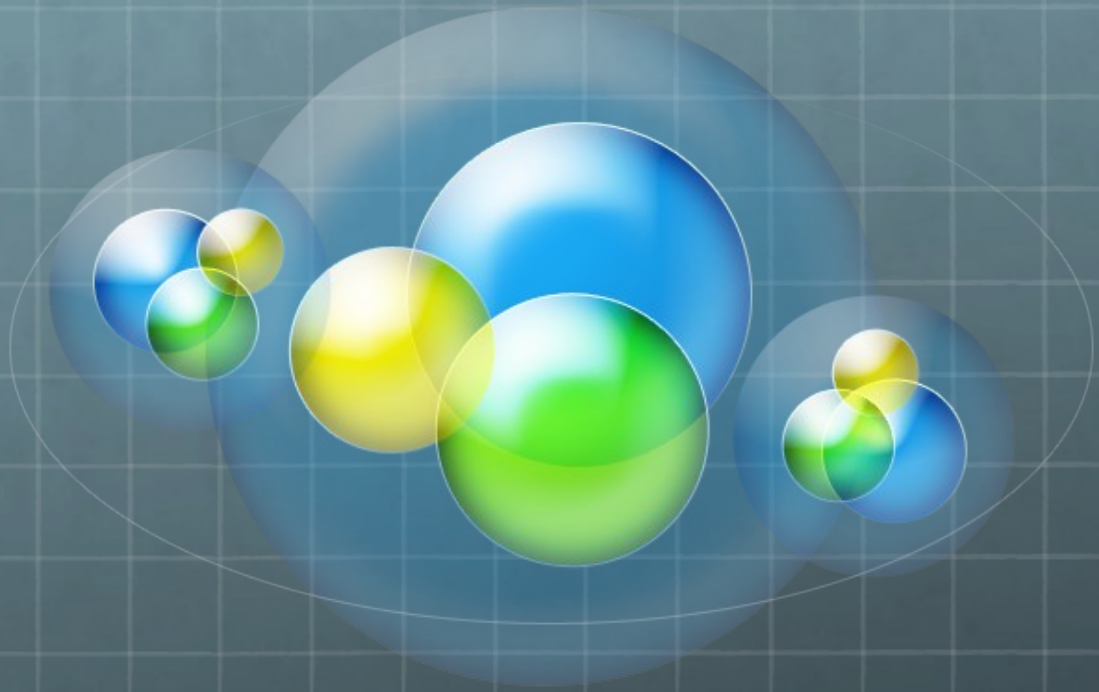


API Standards

Michael Petychakis
@mpetyx

A long time ago in a galaxy far, far away....

Vague but exciting ...

CERN DD/OC

Information Management: A Proposal

Tim Berners-Lee, CERN/DD

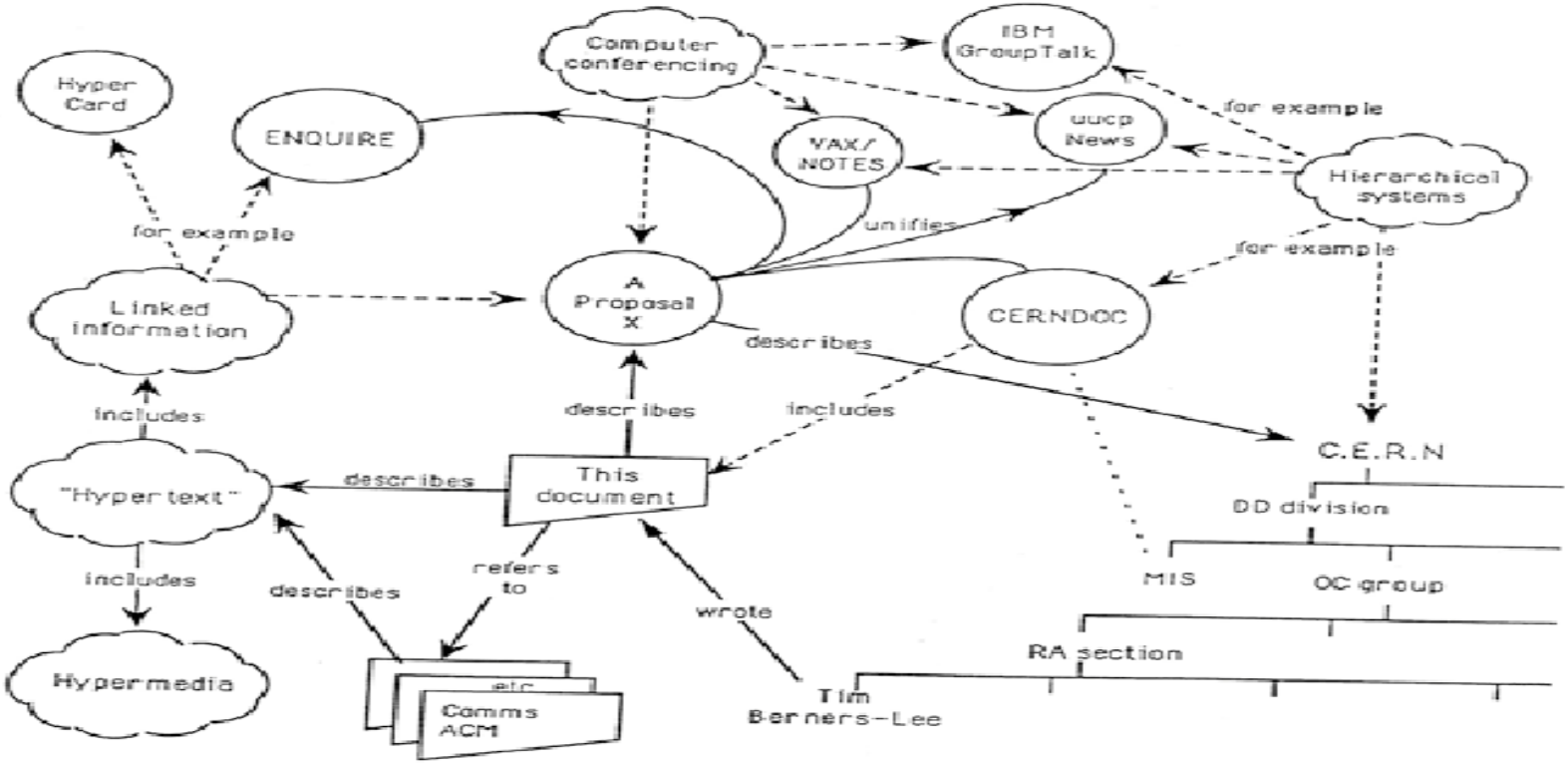
March 1989

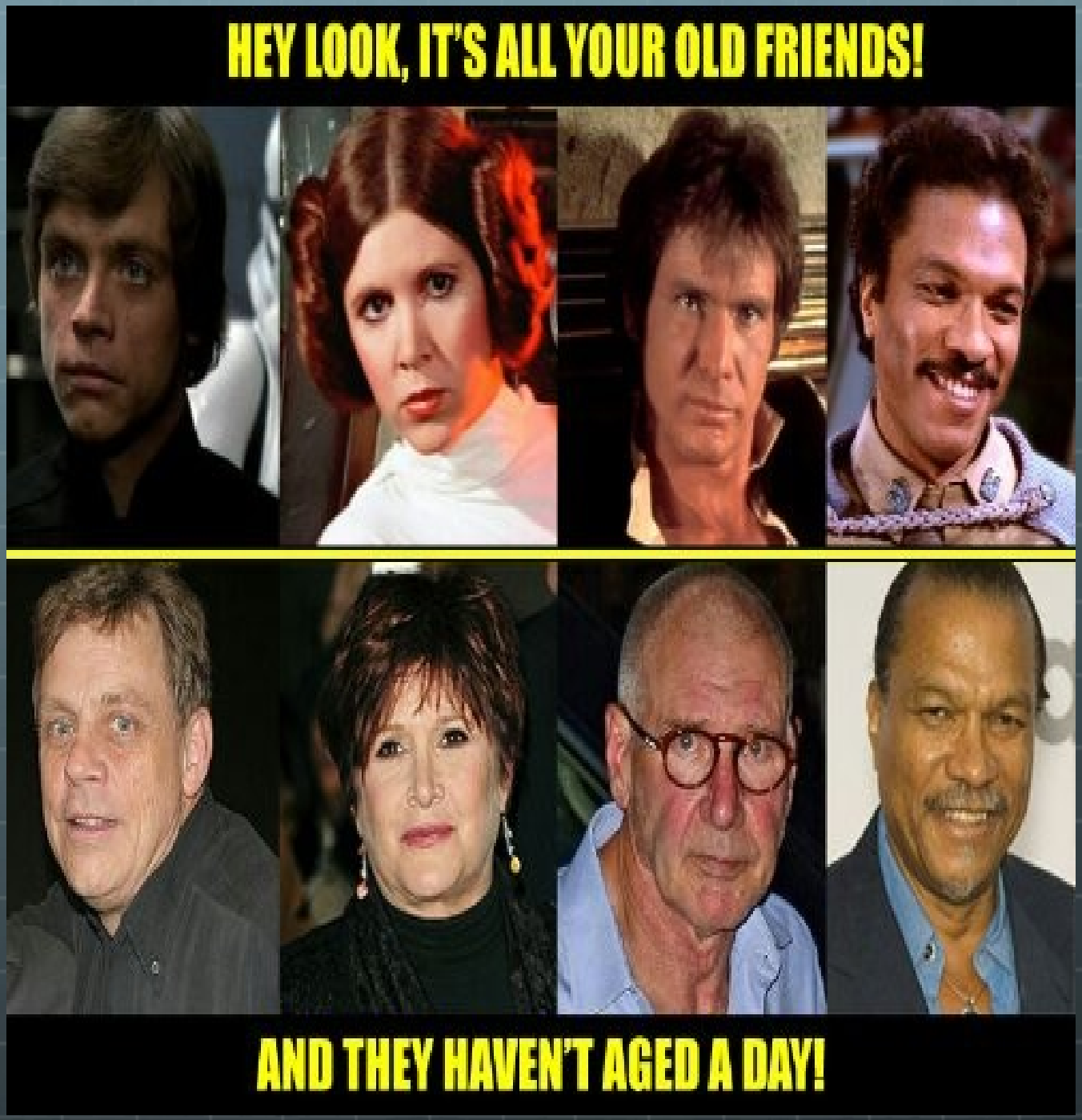
Information Management: A Proposal

Abstract

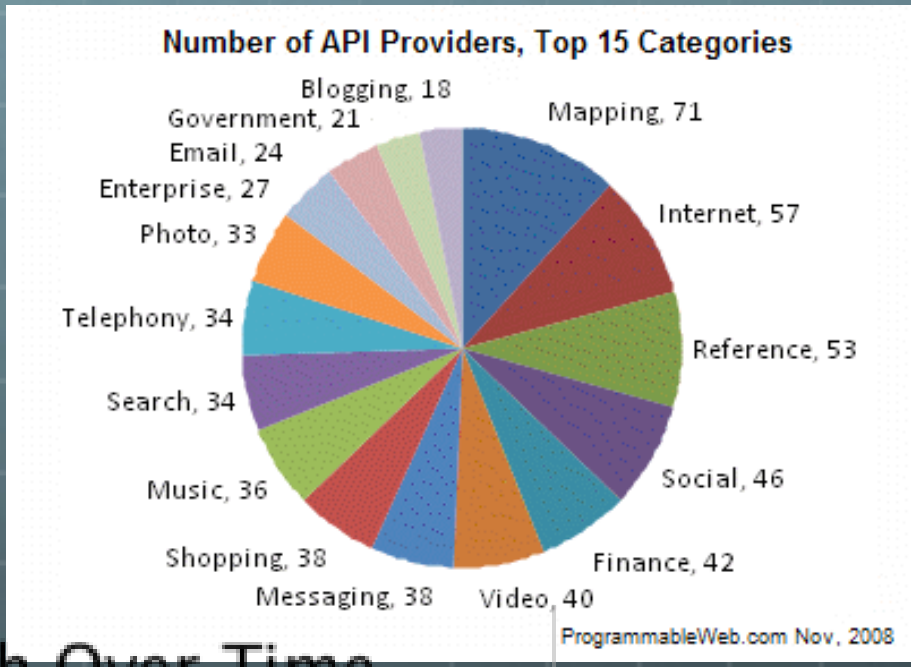
This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext sytstem.

Keywords: Hypertext, Computer conferencing, Document retrieval, Information management, Project control





@mpetyx


Number of API Providers, Top 15 Categories
Blogging, 18
Government, 21
Email, 24
Enterprise, 27
Photo, 33
Telephony, 34
Search, 34
Music, 36
Shopping, 38
Messaging, 38
Video, 40
Finance, 42
Social, 46
Reference, 53
Internet, 57
Mapping, 71
ProgrammableWeb.com Nov, 2008


API Growth Over Time
7000
6000
5000
4000
3000
2000
1000
0
2006
2007
2008
2009
2010
2011
2012

@m

958 million websites

60 trillion

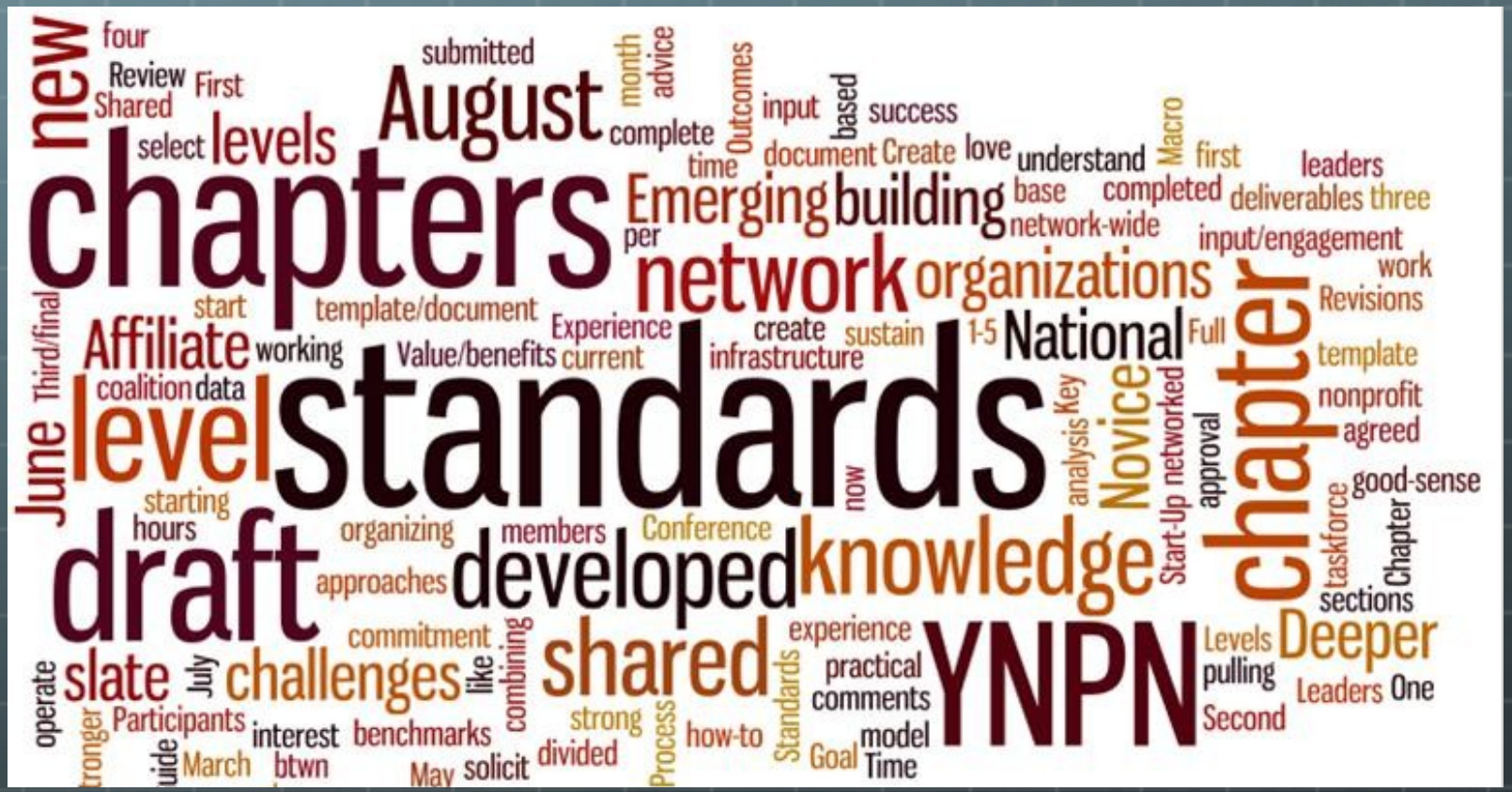
web pages

Each API is Unique

- Documentation?
- Fragile Systems
- Different Practices
- Different cycles

Think it’s Fun?!

The Answer



@mpetyx

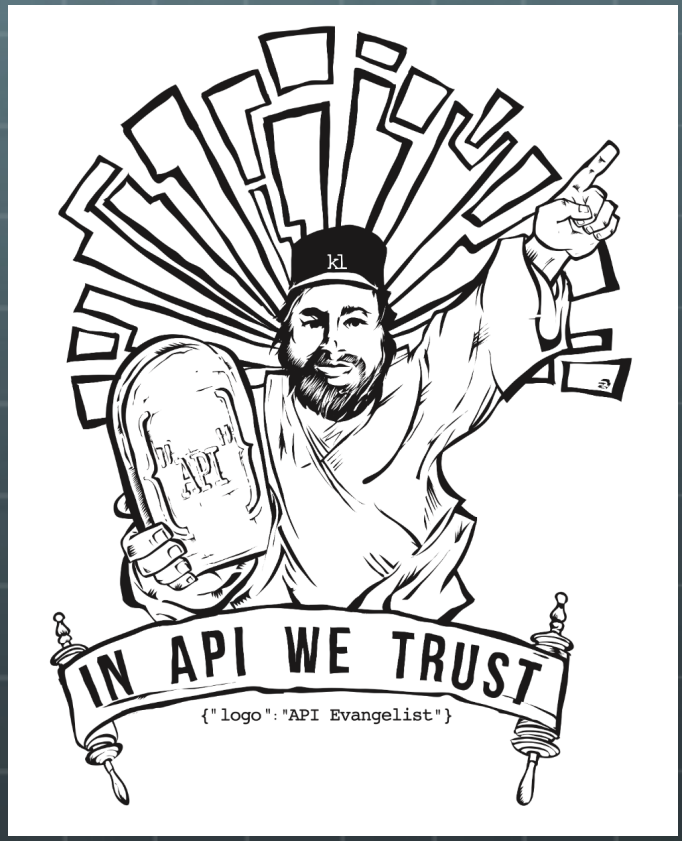
{A} API COMMONS

apiblueprint

Hydra

RAML

swagger



@mpetyx

Narrowing down the choices

- API-Blueprint
- RAML
- Swagger
- What about Hypermedia? ioDocs? WADL? WSDL 2.0?

So now we want to know…

- What's behind the name?
- How does it model REST?
- What tools are available?
- Is there a community / following?
- What's it gonna be?

What's behind the name?

	API-Blueprint	RAML	Swagger
Format	Markdown	YAML	JSON
Spec License	MIT	ASL 2.0 / TM	ASL 2.0
Available at	GitHub	GitHub	GitHub
Sponsored by	Apiary	Mulesoft	Reverb
Current Version	1A3	0.8	1.2
Initial commit	April, 2013	Sep, 2013	July, 2011
Commercial Offering	Yes	Yes	No
API Design Approach	Top-down	Top-down	**Bottom-up**

How does it model REST?

	API-Blueprint	RAML	Swagger
Resources	X	X	X (“api”)
Methods/Actions	X (“action”)	X (“method”)	X (“operation”)
Query Parameters	X	X	X
Path/URL Parameters	X	X	X
Header Parameters	X	X	X
Representations (status codes, mime-types)	X	X	X
Documentation	X	X	X

How does it model REST – part 2

	API-Blueprint	RAML	Swagger
Authentication		Basic, Digest, Oauth 1&2, (*)	Basic, API-Key, OAuth 2
Representation Metadata	<any> (inline)	<any> (inline/external)	JSON Schema (subset)
Nested Resources	X	X	
Composition/Inheritance	Resource Models	Traits, Resource Types	
File inclusions		X	
API Version metadata		X	X
Sample Representations	X	X	

What about tooling?

	API-Blueprint	RAML	Swagger
Authoring	apiary.io	API-Designer	(3rd party)
Ad-hoc testing	apiary.io	API-Console	Swagger-UI
Documentation	X	X	X
Mocking	X	X	(3rd party)
Server Code	(3rd party)	java	java,scala,ruby, node.js
Client Code		(3rd party)	java,groovy,scala,objc, android,c#,flash,php,python,ruby,js
Generate from code			java, 3rd party (go,python)
Validation	X	X (java)	X
Parsing	C++ (nodejs, c#)	Java, js	Java, js

Some Community Stats

	API-Blueprint	RAML	Swagger
Stackoverflow questions	37	18	596
Most Github Stars (on single project)	613	478	1859
Total Github projects (search on name)	72	52	340
Google search (name + “ rest”)	807K	64K	5M

So what gives!?

- **Swagger:**
  - **Pros: high adoption rate, good code-level tooling, large community**
  - **Cons: bottom-up, lacks advanced metadata constructs**
- **RAML**
  - **Pros: good online design tools and mature supporting infrastructure, seemingly useful advanced constructs**
  - **Cons: very new to the game, low adoption, lacks code-level tooling**
- **API-Blueprint**
  - **Pros: good online design tools, good community involvement**
  - **Cons: new to the game, low adoption, lacks advanced constructs and code level tooling**

How decide?

- What do your API consumers need/expect?
  - Documentation
  - Client code / SDKs / etc
  - Technologies
- What do you need?
  - Approach
  - Tooling
  - Technologies
- Do you need API management? Commercial Support?

Examples

WADL

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application xmlns="http://research.sun.com/wadl/2006/10">
    <doc xmlns:jersey="http://jersey.dev.java.net/" jersey:generatedBy="Jersey: 1.0-ea-SNAPSHOT 10/02/2008 12:17 PM"/>
    <resources base="http://localhost:9998/storage/">
        <resource path="/containers">
            <method name="GET" id="getContainers">
                <response>
                    <representation mediaType="application/xml"/>
                </response>
            </method>
            <resource path="{container}">
                <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="template" name="container"/>
                <method name="PUT" id="putContainer">
                    <response>
                        <representation mediaType="application/xml"/>
                    </response>
                </method>
                <method name="DELETE" id="deleteContainer"/>
                <method name="GET" id="getContainer">
                    <request>
                        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="query" name="search"/>
                    </request>
                    <response>
                        <representation mediaType="application/xml"/>
                    </response>
                </method>
                <resource path="{item: .+}">
                    <param xmlns:xs="http://www.w3.org/2001/XMLSchema" type="xs:string" style="template" name="item"/>
                    <method name="PUT" id="putItem">
                        <request>
                            <representation mediaType="*/*"/>
                        </request>
                        <response>
                            <representation mediaType="*/*"/>
                        </response>
                    </method>
                    <method name="DELETE" id="deleteItem"/>
                    <method name="GET" id="getItem">
                        <response>
                            <representation mediaType="*/*"/>
                        </response>
                    </method>
                </resource>
            </resource>
        </resource>
    </resources>
</application>
```

@mpetyx

Swagger


```
{
   apiVersion: "0.2",
   swaggerVersion: "1.1",
   basePath: "https://api.github.com",
   resourcePath: "/assignees",
 - apis: [
    - {
         path: "/repos/{owner}/{repo}/assignees",
         description: null,
       - operations: [
          - {
               httpMethod: "get",
               summary: "List assignees",
               notes: "This call lists all the available assignees (owner + collaborators) to which issues may be assigned.",
               responseClass: "void",
               nickname: "GetListAssignees",
             - parameters: [
                - {
                     name: "owner",
                     description: null,
                     paramType: "path",
                     dataType: "string",
                     required: "true"
                  },
                - {
                     name: "repo",
                     description: null,
                     paramType: "path",
                     dataType: "string",
                     required: "true"
                  }
               ]
            }
         ]
      },
    - {
         path: "/repos/{owner}/{repo}/assignees/{assignee}",
         description: null,
       - operations: [
          - {
               httpMethod: "get",
               summary: "Check assignee",
               notes: "You may also check to see if a particular user is an assignee for a repository.",
               responseClass: "void",
```

Mashery IOdocs


```
{
  - endpoints: [
      - {
            name: "Identity Methods",
          - methods: [
              - {
                    MethodName: "Get Identity via Twitter ID",
                    Synopsis: "This method allows you to retrieve a KloutID for a numeric Twitter ID",
                    HTTPMethod: "GET",
                    URI: "/identity.json/tw/:twitterId",
                    RequiresOAuth: "N",
                  - parameters: [
                      - {
                            Name: "twitterId",
                            Required: "Y",
                            Default: "",
                            Type: "string",
                            Description: "A Twitter ID (e.g. 500042487)"
                        }
                    ]
                },
              + {…},
              + {…},
              + {…},
              + {…}
            ]
        },
      - {
            name: "User Methods",
          - methods: [
              - {
                    MethodName: "Show User",
                    Synopsis: "This method allows you to retrieve a user object",
                    HTTPMethod: "GET",
                    URI: "/user.json/:kloutId",
                    RequiresOAuth: "N",
                  - parameters: [
                      - {
                            Name: "kloutId",
                            Required: "Y",
                            Default: "",
                            Type: "string",
                            Description: "A KloutId (e.g. 635263)"
```


@mpetyx

Google Discovery


```
{
   kind: "discovery#restDescription",
   etag: ""DGgqtFnjgu83tuwvvVNNUhOiHWk/ovEOlJqyhA6jrpbW4GovtVpYUd8"",
   discoveryVersion: "v1",
   id: "adexchangebuyer:v1",
   name: "adexchangebuyer",
   version: "v1",
   title: "Ad Exchange Buyer API",
   description: "Lets you manage your Ad Exchange Buyer account.",
   ownerDomain: "google.com",
   ownerName: "Google",
 - icons: {
      x16: "http://www.google.com/images/icons/product/doubleclick-16.gif",
      x32: "http://www.google.com/images/icons/product/doubleclick-32.gif"
   },
   documentationLink: "https://developers.google.com/ad-exchange/buyer-rest",
   protocol: "rest",
   baseUrl: "https://content.googleapis.com/adexchangebuyer/v1/",
   basePath: "/adexchangebuyer/v1/",
   rootUrl: "https://content.googleapis.com/",
   servicePath: "adexchangebuyer/v1/",
   batchPath: "batch",
 - parameters: {
    - alt: {
         type: "string",
         description: "Data format for the response.",
         default: "json",
       - enum: [
            "json"
         ],
       - enumDescriptions: [
            "Responses with Content-Type of application/json"
         ],
         location: "query"
      },
    - fields: {
         type: "string",
         description: "Selector specifying which fields to include in a partial response.",
         location: "query"
      },
```

Apiary Blueprint

```
FORMAT: X-1A
HOST: https://alpha-api.app.net

# Real World API
This API Blueprint demonstrates a real world example documenting a portion of [App.net API](http://developers.app.net).

NOTE: This document is a **work in progress**.

# Group Posts
This section groups App.net post resources.

## Post [/stream/0/posts/{post_id}]
A Post is the other central object utilized by the App.net Stream API. It has rich text and annotations which comprise all of the content a users sees
in their feed. Posts are closely tied to the follow graph...

+ Parameters
    + post_id (string, `1`) ... The id of the Post.

+ Model (application/json)

    ```js
    {
        "data": {
            "id": "1", // note this is a string
            "user": {
                ...
            },
            "created_at": "2012-07-16T17:25:47Z",
            "text": "@berg FIRST post on this new site #newsocialnetwork",
            "html": "<span itemprop=\"mention\" data-mention-name=\"berg\" data-mention-id=\"2\">@berg</span> FIRST post on <a
href=\"https://join.app.net\" rel=\"nofollow\">this new site</a> <span itemprop=\"hashtag\" data-hashtag-
name=\"newsocialnetwork\">#newsocialnetwork</span>.",
            "source": {
                "client_id": "udxGzAVBdXwGtkHmvswR5MbMEeVnq6n4",
                "name": "Clientastic for iOS",
                "link": "http://app.net"
            },
            "machine_only": false,
            "reply_to": null,
            "thread_id": "1",
            "num_replies": 3,
            "num_reposts": 0,
            "num_stars": 0,
            "entities": {
                "mentions": [{
                    "name": "berg",
                    "id": "2",
                    "pos": 0,
                    "len": 5
                }],
```
```

RAML

```
#%RAML 0.8
---
title: User Management
version: v1
/users:
  get:
    description: Get a collection of users
  post:
    description: Create a new user in the collection
  /{userId}:
    get:
      description: Get a single user
    put:
      description: Update a single user
    delete:
      description: Delete a single user
```

A bit of Python

@mpetyx

```python
from pyswagger import App, Security
from pyswagger.contrib.client.requests import Client
from pyswagger.utils import jp_compose

# load Swagger resource file into App object
app = App._create_('http://petstore.swagger.io/v2/swagger.json')

auth = Security(app)
auth.update_with('api_key', '12312312312312312313q') # api key
auth.update_with('petstore_auth', '12334546556521123fsfss') # oauth2

# init swagger client
client = Client(auth)

# a dict is enough for representing a Model in Swagger
pet_Tom=dict(id=1, name='Tom', photoUrls=['http://test'])
# a request to create a new pet
client.request(app.op['addPet'](body=pet_Tom))

# - access an Operation object via App.op when operationId is defined
# - a request to get the pet back
pet = client.request(app.op['getPetById'](petId=1)).data
assert pet.id == 1
assert pet.name == 'Tom'

# new ways to get Operation object corresponding to 'getPetById'.
# 'jp_compose' stands for JSON-Pointer composition
pet = client.request(app.resolve(jp_compose('/pet/{petId}', base='#/paths')).get(petId=1)).data
assert pet.id == 1
```

@mpetyx

rest_api_demo/api/blog/endpoints/categories.py

```python
ns = api.namespace('blog/categories', description='Operations related to blog categories')


@ns.route('/')
class CategoryCollection(Resource):

    def get(self):
        """Returns list of blog categories."""
        return get_all_categories()

    @api.response(201, 'Category successfully created.')
    def post(self):
        """Creates a new blog category."""
        create_category(request.json)
        return None, 201


@ns.route('/<int:id>')
@api.response(404, 'Category not found.')
class CategoryItem(Resource):

    def get(self, id):
        """Returns details of a category."""
        return get_category(id)

    @api.response(204, 'Category successfully updated.')
    def put(self, id):
        """Updates a blog category."""
        update_category(id, request.json)
        return None, 204

    @api.response(204, 'Category successfully deleted.')
    def delete(self, id):
        """Deletes blog category."""
        delete_category(id)
        return None, 204
```

@mpetyx

Automated API Provisioning

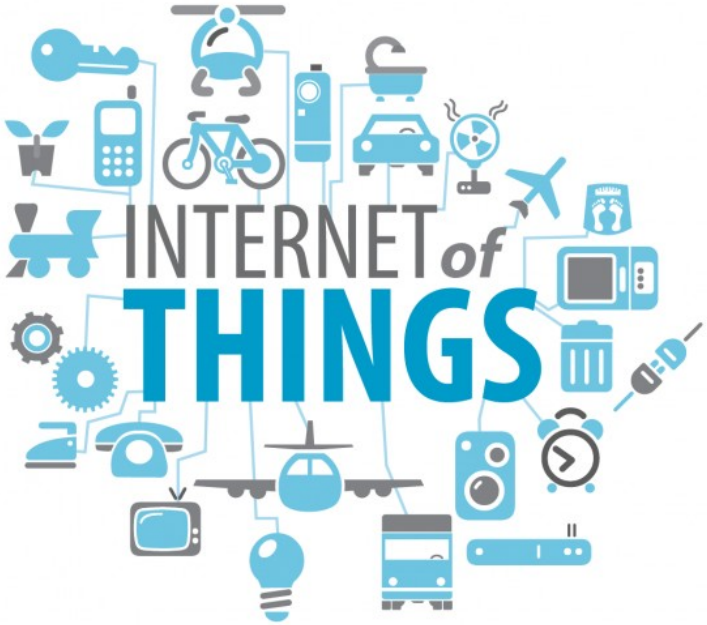
- Problem: How do I use an API without specifically coding for it?
- Answer: Use vocabulary to define operations on classes and properties
- Proposition: Annotations are the result of operations on entities or the relationships between entities.
  - Those results are also entities, which may be operated upon.

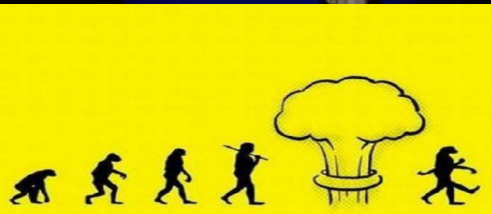
Links and Operations

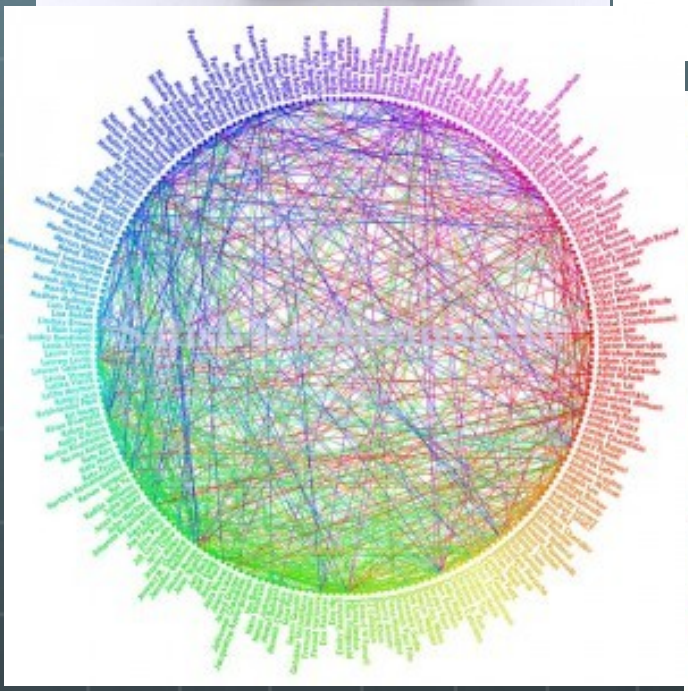
- Define operations on entities, types of entities, or their properties.
- When does a property link to an entity?
- How do you use pagination to reference and collect linked entities?
- What operations can I perform on an entity, or property of that entity?
- Where does authentication/authorization intersect with generic API interactions?

What is Hydra?

- W3C Community Group for Linked APIs
  - REST + Linked Data
  - “At the intersection of web schemas and RESTful web applications”







@mpetyx

Keep in touch!

- Me
  - Github mpetyx/
  - Twitter @mpetyx
  - Mail hello@apilama.com
  - Blog www.apilama.com