

SysML v2

Model Interoperability & Standard API Requirements

Axel Reichwein
Consultant, Koneksys
September 13, 2016

Systems Modeling Environment Conceptual Architecture

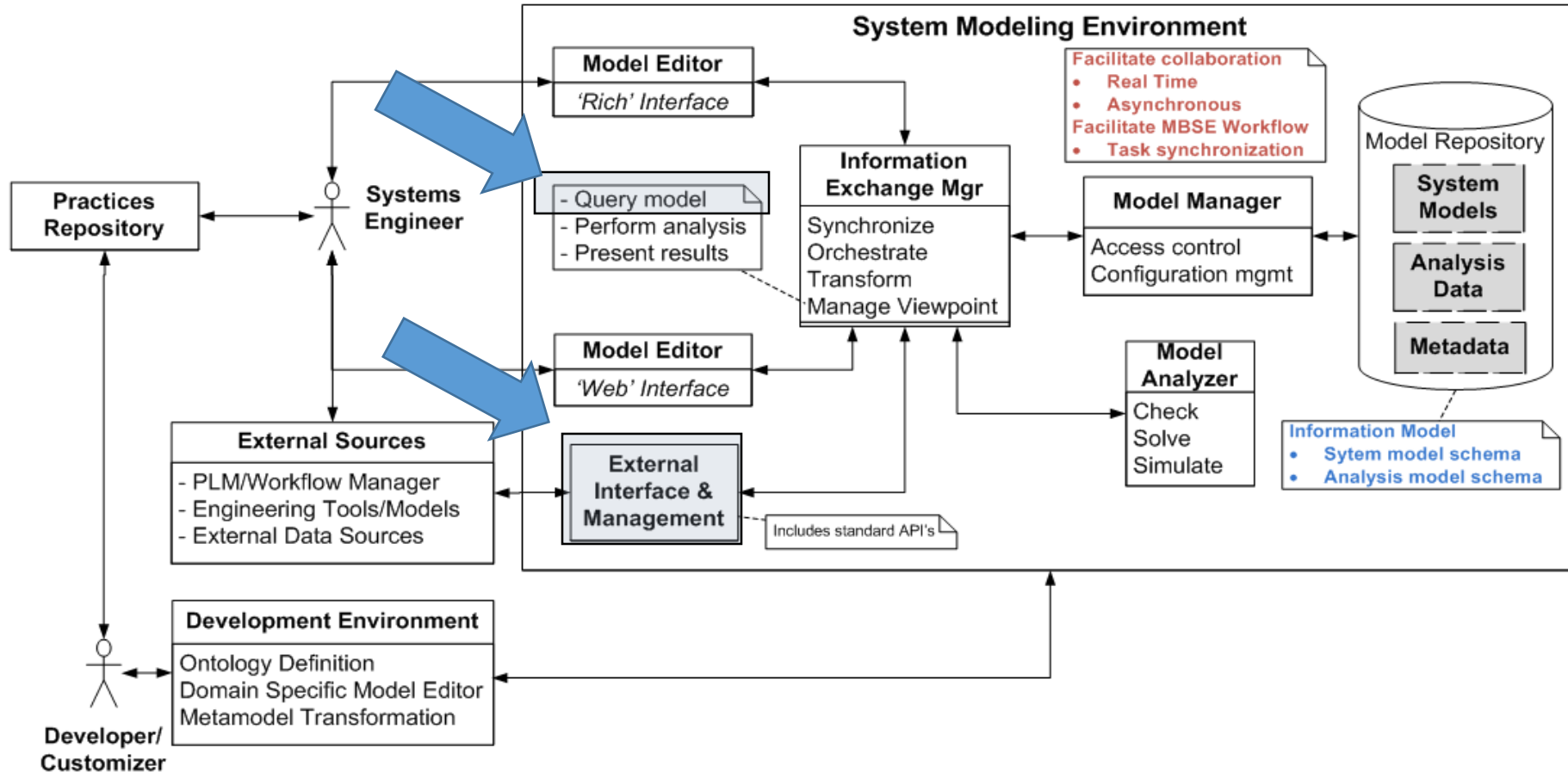


Image from "Status of SysML v2, Planning & Requirements, Berlin, Germany June 16, 2015, Sandy Friedenthal & Eldad Palachi, http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-roadmap:sysml_assessment_and_roadmap_working_group"

Goals of the SysML v2 Interoperability WG

- **Update template for specifying service requirements**
- Coordinate with other Concept Leads to capture integrated service requirements based on elaborated Hybrid SUV scenario
- Coordinate with other vendors on the team to define the general requirements for the standard API
- Consider limitations of OSLC and the need for event services (per Chris Delp)
- Confirm that the proposed exchange format is an XML serialization of RDF
- Clarify the concept for migrating SysML v1 model to SysML v2 models

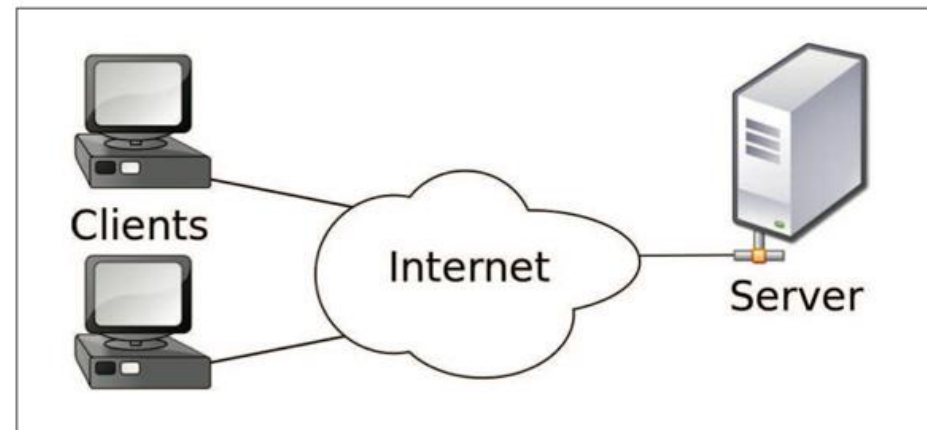
Previous Activities

- Overview of Linked Data capabilities for SysML v2 interoperability (Presentation to OMG meeting in December 2015 by Axel)
- Overview of upcoming OSLC 3.0 for capabilities for SysML v2 interoperability (Presentation to OMG meeting in June 2015 by Chas)
- **Consensus on using standards for SysML v2 interoperability based on RESTful web services, OSLC, Linked Data, and W3C standards**

Web APIs

- Different approaches for Web APIs
 - SOAP APIs
 - RESTful APIs
 - Hypermedia APIs
 - Linked Data
 - Linked Data Platform
 - OSLC v2 core
 - OSLC v3 core?

- Template for defining SysML v2 services depends on the type of Web API



http://www.dbguide.net/publishing/img/knowledge/tech_img1040.jpg

Why do (Web) APIs matter?

- Existing APIs are all unique
- You need to read the documentation for each API
- Tightly coupled and brittle solutions
- When dealing with many APIs, you will constantly be running against APIs that have changed and clients that have broken

Presentation Goals

- Provide an overview of different Web APIs
- Propose Web API for SysML v2
- No template yet for SysML v2 web services (possibility to reuse existing standard to define SysML v2 web services)

Yesterday's Most common Web API: SOAP API

- SOAP (Simple Object Access Protocol)
- W3C standard
- Latest version: v1.2 (2007)
- XML for information exchange
- Commonly using HTTP as transport protocol
- slow parsing speed of XML, and lack of a standardized interaction model

```
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"

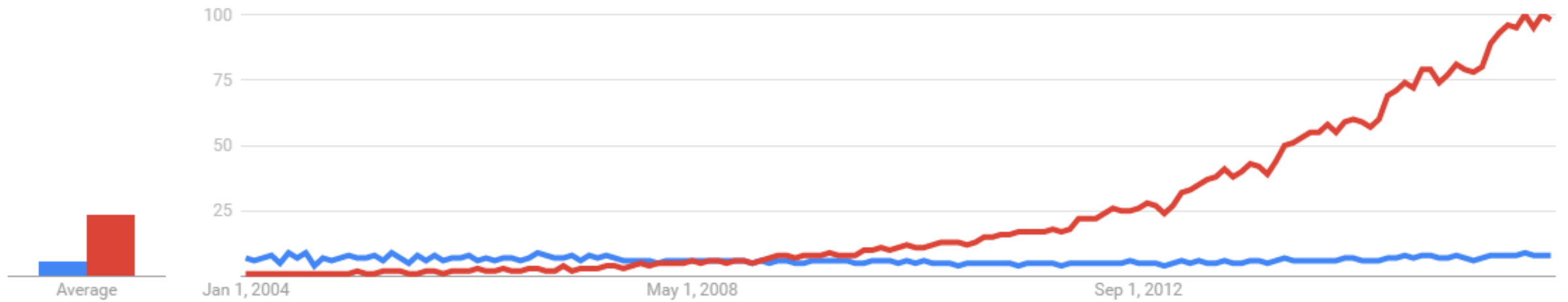
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock/Surya">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

<https://en.wikipedia.org/wiki/SOAP>

● SOAP API
Search term

● REST API
Search term

Interest over time ?



Today's Most common Web API: REST API

- REST (Representational state transfer)
- Not standard, just architectural style
- Formal REST constraints as to how a client should interact with a server defined in PhD thesis of Roy Fielding (2000)
- REST constraints used to define W3C standard HTTP

Example: Twitter API

```
GET
https://api.twitter.com/1.1/followers/list.
json?cursor=-
1&screen_name=twitterdev&skip_status=true&i
nclude_user_entities=false
```

Example Result

```
{
  "users": [
    {
      "id": 2960784075,
      "id_str": "2960784075",
      "name": "Jesus Rafael Abreu M",
      "screen_name": "chuomaraver",
      "location": "",
      "profile_location": null,
```

<https://dev.twitter.com/rest/reference/get/followers/list>

Elements of HTTP Request/Response

- Verb
- URL
- Query parameters
- Body

Example: Twitter API

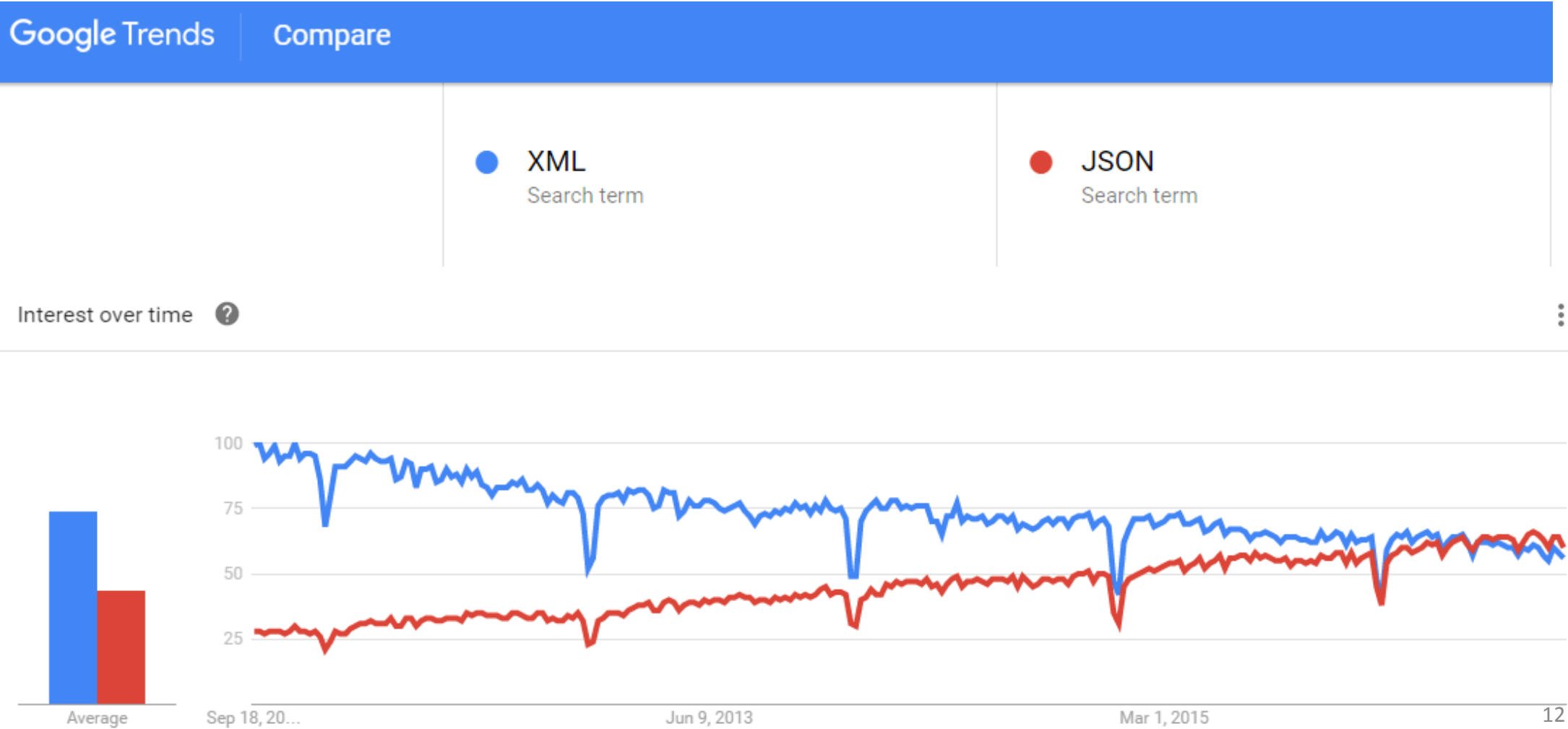
```
GET
https://api.twitter.com/1.1/followers/list.
json?cursor=-
1&screen_name=twitterdev&skip_status=true&i
nclude_user_entities=false
```

Example Result

```
{
  "users": [
    {
      "id": 2960784075,
      "id_str": "2960784075",
      "name": "Jesus Rafael Abreu M",
      "screen_name": "chuomaraver",
      "location": "",
      "profile_location": null,
```

<https://dev.twitter.com/rest/reference/get/followers/list>

Today's Most common Web API Media Type: JSON



Today's Most common Web API Media Type: JSON

- Typical body format: JSON
- No semantics associated with plain JSON
- Meaning of JSON body needs to be discovered through human-readable documentation
- No complex reasoning/analytics possible without knowing the meaning of data

Example Result

```
{
  "users": [
    {
      "id": 2960784075,
      "id_str": "2960784075",
      "name": "Jesus Rafael Abreu M",
      "screen_name": "chuomaraver",
      "location": "",
      "profile_location": null
    }
  ]
}
```

<https://dev.twitter.com/rest/reference/get/followers/list>

Tomorrow's Most common Web API Media Type: JSON-LD (?)

- JavaScript Object Notation for Linked Data
- JSON-LD = one possible serialization of RDF
- Additional identifiers (URIs) can be used to specify the meaning of JSON properties, and the type of JSON objects

Example of Google Knowledge Graph REST API

Response body

A response message contains a list of entities, presented in [JSON-LD](#) format and compatible with [schema.org](#) schemas (with limited [external extensions](#)).

The following JSON-LD example shows how the response body is structured:

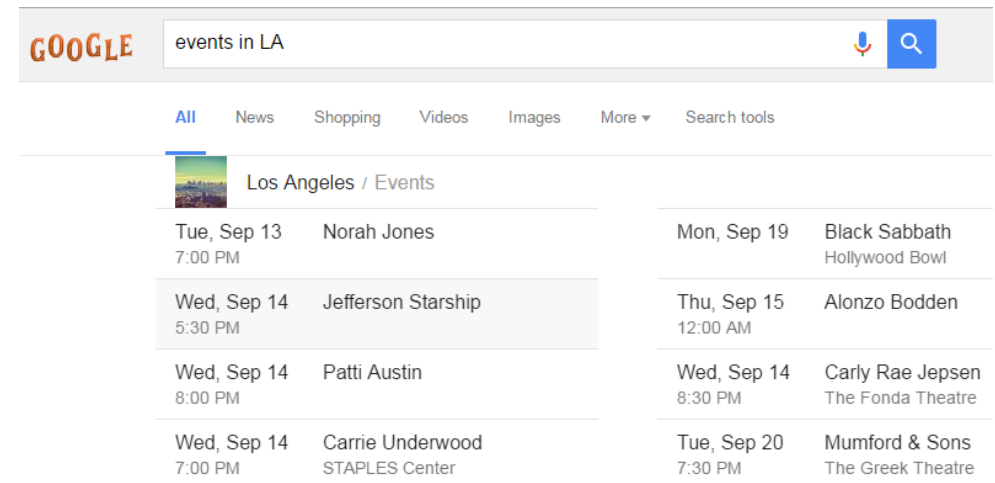
```
{
  "@context": {
    "@vocab": "http://schema.org/",
    "goog": "http://schema.googleapis.com/",
    "resultScore": "goog:resultScore",
    "detailedDescription": "goog:detailedDescription",
    "EntitySearchResult": "goog:EntitySearchResult",
    "kg": "http://g.co/kg"
  },
  "@type": "ItemList",
  "itemListElement": [
    {
      "@type": "EntitySearchResult",
      "result": {
        "@id": "kg:/m/0d1567",
        "name": "Taylor Swift",
        "@type": [
          "Thing",
          "Person"
        ],
        "description": "Singer-songwriter",
        "image": {
          "contentUrl": "https://t1.gstatic.com/images?q=tbn:ANd9GcQmVDAhjhWnN20Wys2ZM03PGAhupp5tN2Lw",
          "url": "https://en.wikipedia.org/wiki/Taylor_Swift",
          "license": "http://creativecommons.org/licenses/by-sa/2.0"
        }
      }
    }
  ]
}
```

Example of JSON-LD used to improve web search results

- Web admins describe events in JSON-LD conforming to schema.org RDF vocabulary
- Data on the Web in JSON-LD conforming to schema.org gets merged into Google Knowledge Graph providing better search results to the user (search results with meaning!)

<https://schema.org/Event>

```
<script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type": "Event",
  "name": "The Missoula Marathon",
  "sponsor": {
    "@type": "Organization",
    "name": "Run Wild Missoula",
    "url": "http://www.runwildmissoula.org/"
  },
  "startDate": "2016-07-10T06:00",
  "offers": [
```



GOOGLE events in LA

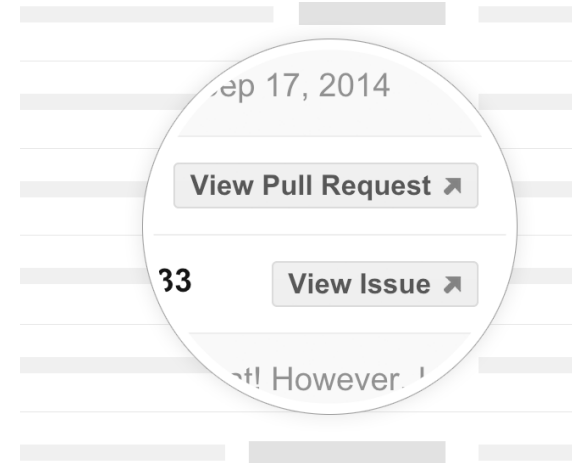
All News Shopping Videos Images More Search tools

Los Angeles / Events

Tue, Sep 13 7:00 PM	Norah Jones	Mon, Sep 19	Black Sabbath Hollywood Bowl
Wed, Sep 14 5:30 PM	Jefferson Starship	Thu, Sep 15 12:00 AM	Alonzo Bodden
Wed, Sep 14 8:00 PM	Patti Austin	Wed, Sep 14 8:30 PM	Carly Rae Jepsen The Fonda Theatre
Wed, Sep 14 7:00 PM	Carrie Underwood STAPLES Center	Tue, Sep 20 7:30 PM	Mumford & Sons The Greek Theatre

JSON-LD support for data integration across applications

- Github [added JSON-LD and schema.org support](#). Now if you get a pull request via email, and your mail client supports it (like Gmail does), you'll see an action button shown in the display without having to open the email, like so
- When you get an email from Github, it will now include markup that looks like this:



```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "EmailMessage",
  "description": "View this Pull Request on GitHub",
  "action": {
    "@type": "ViewAction",
    "url": "https://github.com/perma-id/w3id.org/pull/47",
    "name": "View Pull Request"
  }
}
</script>
```


Bad vs Good URLs

- Bad URLs contain information irrelevant to the user (such as internal numeric identifiers for values in a database, illegibly-encoded data, session IDs, implementation details)

Non-semantic URL	Semantic URL
http://example.com/index.php?page=name	http://example.com/name
http://example.com/index.php?page=consulting/marketing	http://example.com/consulting/marketing
http://example.com/products?category=2&pid=25	http://example.com/products/2/25
http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss	http://example.com/news.rss
http://example.com/services/index.jsp?category=legal&id=patents	http://example.com/services/legal/patents
http://example.com/kb/index.php?cat=8&id=41	http://example.com/kb/8/41
http://example.com/index.php?mod=profiles&id=193	http://example.com/profiles/193
http://en.wikipedia.org/w/index.php?title=Semantic_URL	http://en.wikipedia.org/wiki/Semantic_URL

https://en.wikipedia.org/wiki/Semantic_URL

Different Possibilities to Construct a Web API

- One service endpoint URL vs many service endpoint URLs
- Usage of a single HTTP verb vs Usage of a multiple HTTP verbs
- Body composed of XML vs JSON vs RDF
- Autonomous services only discoverable through human-readable API documentation vs interconnected services whereby each service refers to other available services

Example: Twitter API

```
GET
https://api.twitter.com/1.1/followers/list.
json?cursor=-
1&screen_name=twitterdev&skip_status=true&i
nclude_user_entities=false
```

Example Result

```
{
  "users": [
    {
      "id": 2960784075,
      "id_str": "2960784075",
      "name": "Jesus Rafael Abreu M",
      "screen_name": "chuomaraver",
      "location": "",
      "profile_location": null,
```

<https://dev.twitter.com/rest/reference/get/followers/list>

Different Levels of REST: Richardson Maturity Model

Level	Major feature	Analogy
Level 0	Same URL for all services	One single function accepting many arguments
Level 1	Breaking a large service endpoint down into multiple resources .	Object-oriented approach (getter/setter applied to a specific object)
Level 2	Introducing a standard set of verbs for handling similar situations in the same way	Common naming convention (getter/setter methods)
Level 3	Introducing discoverability , providing a way of making a protocol more self-documenting	Somewhat similar to reflection but more powerful (only possible through hyperlinks)

Example of Level 3: REST API of Paypal

- One of the key features of the [PayPal REST Payments API](#) is Hypermedia As The Engine Of Application State ([HATEOAS](#)).
- HATEOAS enables you to interact with the Payments API entirely through hyperlinks. Each Payments API request returns an array of links that enable you to request more information about and further interact with Payments API resources.
- <https://developer.paypal.com/docs/integration/direct/paypal-rest-payment-hateoas-links/>

When you make an initial payment request through the REST API, the response contains a `links` array of HATEOAS `link` objects:

```
{
  "links": [
    {
      "href": "https://api.paypal.com/v1/payments/payment/PAY-1B56960729604235TKQQIYVY",
      "rel": "self",
      "method": "GET"
    },
    {
      "href": "https://api.paypal.com/v1/payments/cgi-bin/webscr?cmd=_express-checkout&token=EC-60385559L1062554J",
      "rel": "approval_url",
      "method": "REDIRECT"
    },
    {
      "href": "https://api.paypal.com/v1/payments/payment/PAY-1B56960729604235TKQQIYVY/execute",
      "rel": "execute",
      "method": "POST"
    }
  ]
}
```

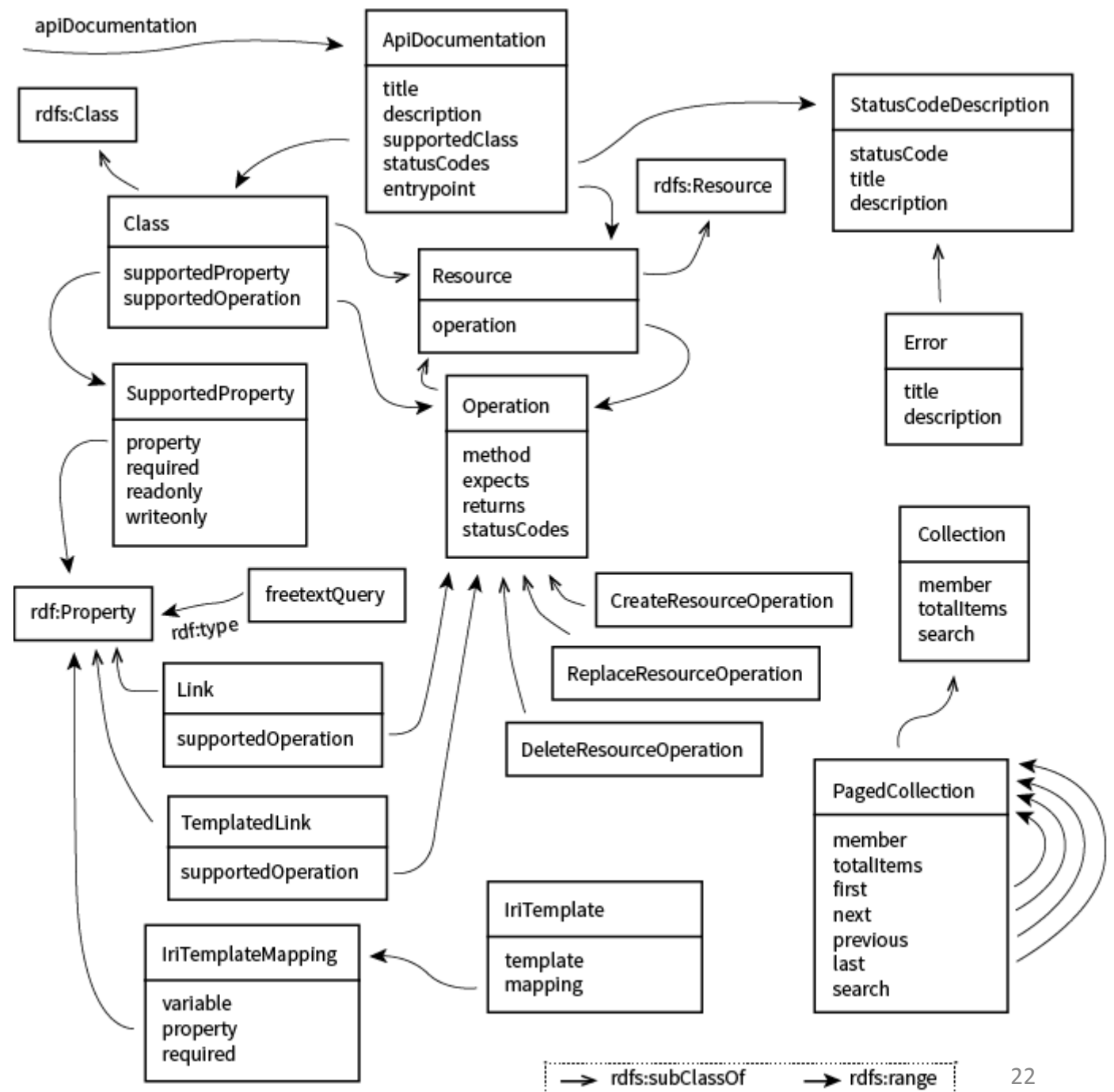
Level 3 REST APIs = Hypermedia APIs

- Most "RESTful APIs" make it as far as to the last point but the majority [fail the hypermedia constraint](#).
- However this is so innate to the architecture that it has its own abbreviation — [HATEOAS](#). It is also what makes RESTful design stand out and be as powerful as it is.

Hydra: W3C Standard for Hypermedia APIs

- Hydra provides a vocabulary to define a Hypermedia API
- Unofficial draft for now

<https://www.hydra-cg.com/spec/latest/core/>



Hydra Example

← → ↻ 🏠 www.markus-lanthaler.com/hydra/console/?url=http://www.markus-lanthaler.com/hydra/api-demo/

Hydra Console

Enter an URL:

↻ Load

Response

```
{
  "@context": "/hydra/api-demo/contexts/EntryPoint.jsonld",
  "@id": "/hydra/api-demo/",
  "@type": "EntryPoint",
  "issues": "/hydra/api-demo/issues/",
  "register_user": "/hydra/api-demo/users/",
  "users": "/hydra/api-demo/users/"
}
```

<http://www.markus-lanthaler.com/hydra/console/?url=http://www.markus-lanthaler.com/hydra/api-demo/>

Documentation: **EntryPoint** ▾

The main entry point or homepage of the API.

@id	IRI readonly	The entity's IRI 🔗 Operations
issues	Collection readonly	The collection of all issues 🔗 Operations
register_user	Resource readonly	IRI to register a new user 🔗 Operations
my_account	User readonly	If logged in, a link to the user account 🔗 Operations
users	Collection readonly	The collection of all users (for debugging purposes) 🔗 Operations

Recommendations related to SysML v2 Web API based on developments in Web community

- SysML v2 Web API = Hypermedia API
- **Don't specify requirements related to**
 - Service endpoint URL structure (e.g. <https://sysmltool.com/projectA/blocks/>)
 - Specific media types (e.g. application/ld+json)
- **Specify requirements related to**
 - Usage of HTTP verbs (POST to create, GET to read, etc..)
 - Usage of media types supporting RDF data model
 - Usage of RDF describing SysML v2 data should conform to SysML v2 vocabulary
 - Usage of RDF W3C Hydra to describe SysML v2 Web API

What about W3C Linked Data standards?

- Linked Data Principles: instructions for servers on how to publish Linked Data such that **clients can read Linked Data**
- Linked Data Platform: instructions for servers on how to offer services such that **clients can also create, update, delete Linked Data**

LDP Vocabulary

<https://www.w3.org/TR/ldp/>

- 4. Linked Data Platform Resources
 - 4.1 Introduction
 - 4.2 Resource
 - 4.3 RDF Source
 - 4.4 Non-RDF Source
- 5. Linked Data Platform Containers
 - 5.1 Introduction
 - 5.2 Container
 - 5.3 Basic
 - 5.4 Direct
 - 5.5 Indirect

LDP Extension for paging

- For separating large resources into smaller chunks
 - <https://www.w3.org/TR/ldp-paging/>
4. Example paging message exchanges
 - 4.1 Traditional flow without paging
 - 4.2 Simple paging flow using redirects
 - 4.3 Optional paging links
 5. Linked Data Platform Paging Clients
 - 5.1 General requirements
 - 5.2 Client preferences
 6. Linked Data Platform Resources
 - 6.1 Paging considerations
 - 6.2 HTTP GET
 7. Linked Data Platform Containers
 - 7.1 Requirements when paging LDP Containers
 - 7.2 Ordering of container members across pages
 - 7.3 HTTP GET requirements for member ordering across pages

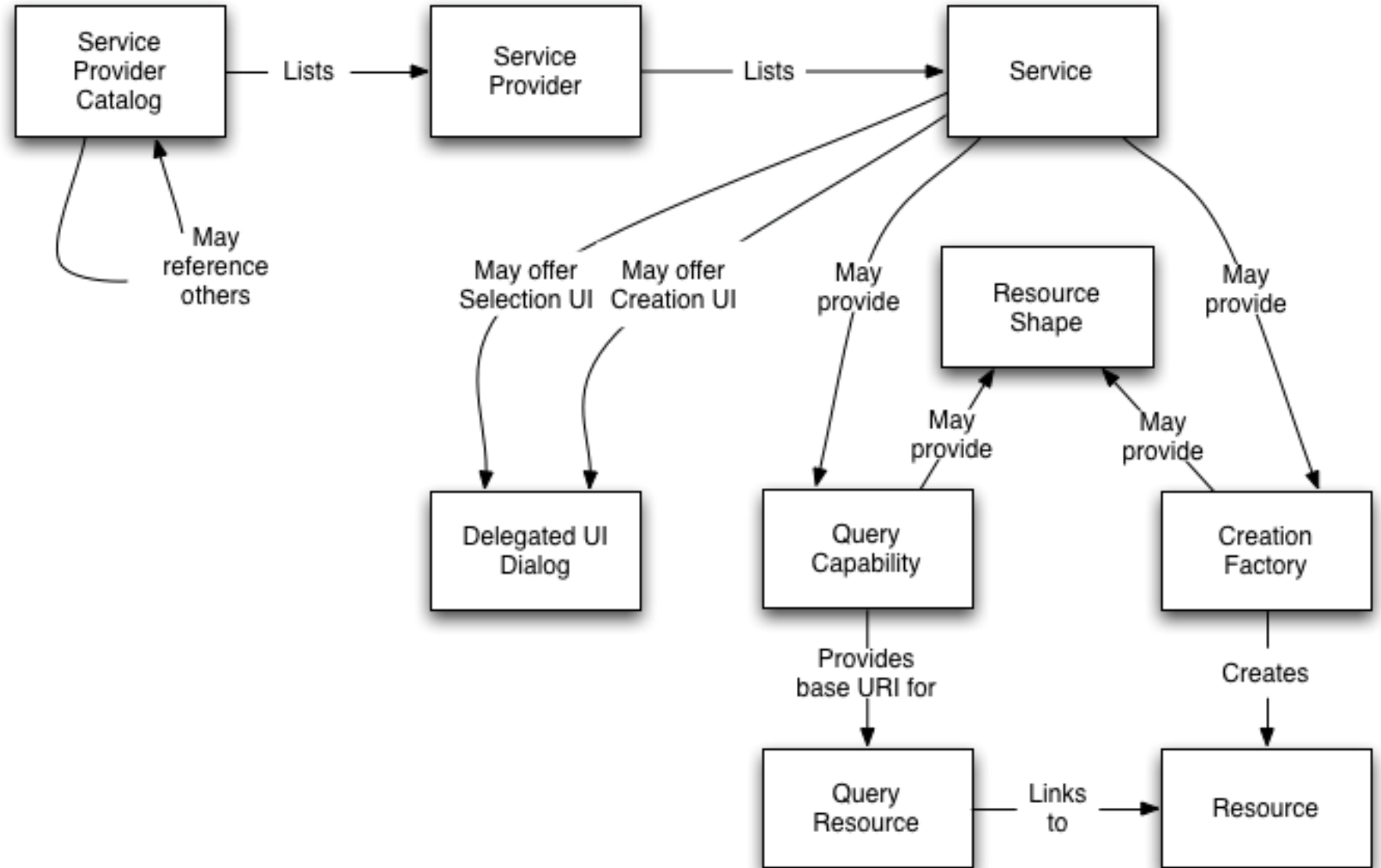
Recommendations related to SysML v2 Web API based on developments in Linked Data community

- Not many implementations of LDP (Apache Marmotta)
- Big overlap between LDP and W3C Hydra on Collections
- LDP provides rules for Level 2 REST API
- LDP and Hypermedia APIs (Level 3) are compatible
- **Requirement:**
 - SysML v2 Web API conforming to W3C LDP and W3C LDP Paging

What about OSLC Core Specification?

- **Discovery:** How OSLC servers publish the REST APIs for their provided services, and how clients discover and use them.
- **Resource Operations:** Defines OSLC resource representations and how OSLC client applications create, read, update and delete resource managed by OSLC servers through HTTP methods.

OSLC Core Spec



<http://open-services.net/bin/view/Main/OslcCoreSpecification>

Recommendations related to SysML v2 Web API based on developments in OSLC community

- Many implementations of OSLC
- Big overlap between LDP, and OSLC v2 Core
- OSLC v3 Core = LDP ?
- Or OSLC v3 Core = W3C Hydra on Collections?
- OSLC domain-specific vocabularies will remain
- **OSLC core spec will most likely become replaced by a W3C spec**
- **No requirements related to SysML v2 Web API based on OSLC Core spec**

Summary

- Personal Recommendations:
 - SysML v2 Web API = Hypermedia API
 - SysML v2 Hypermedia API defined according to W3C Hydra and conforming to W3C LDP and W3C LDP Paging
 - Definition of SysML v2 service template based on W3C Hydra