REST - Part 1

Syed Gillani
Laboratoire Hubert Curien St-Etienne, France

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

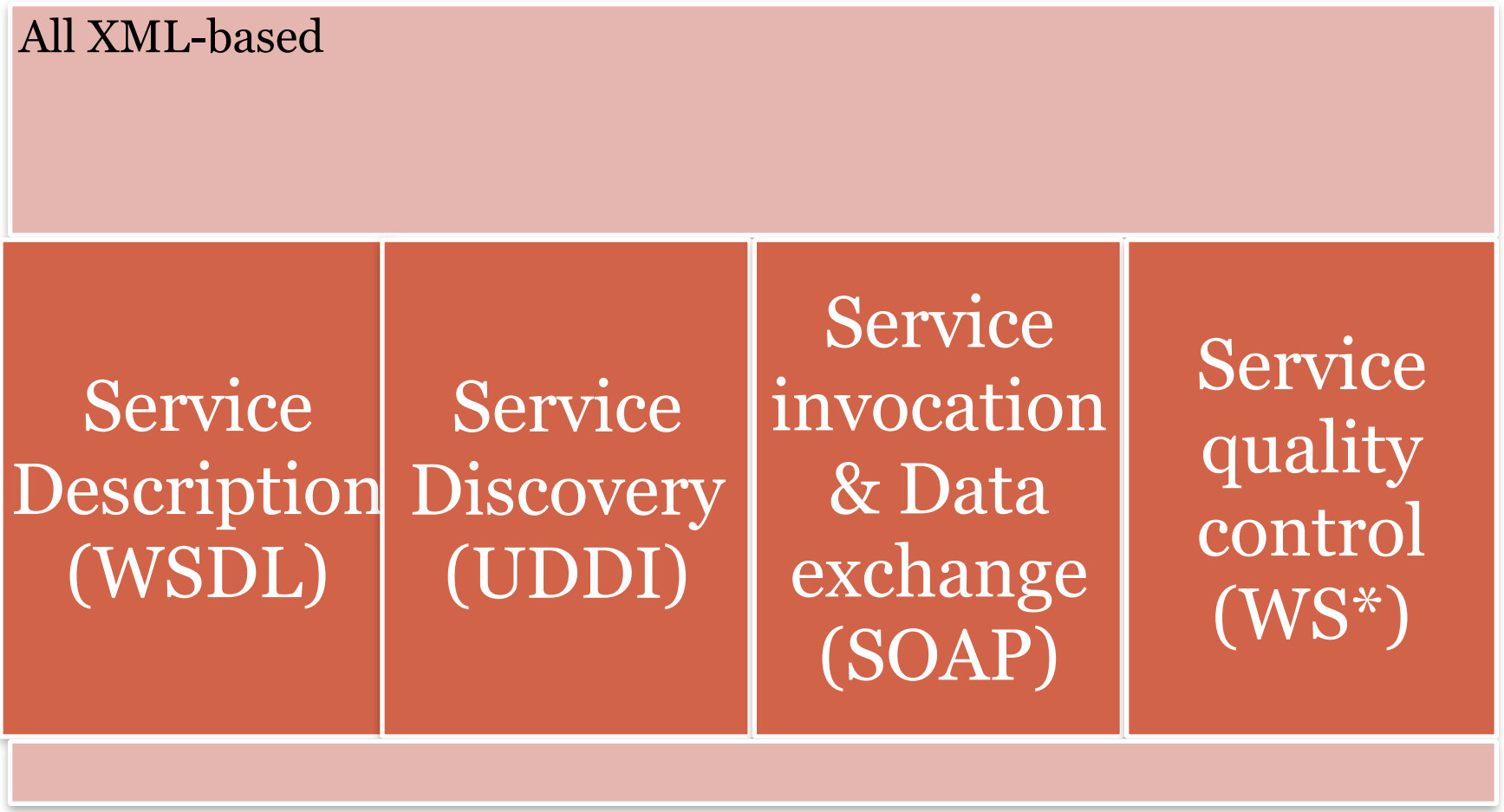
MINES Saint-Étienne

[WS Evolution]

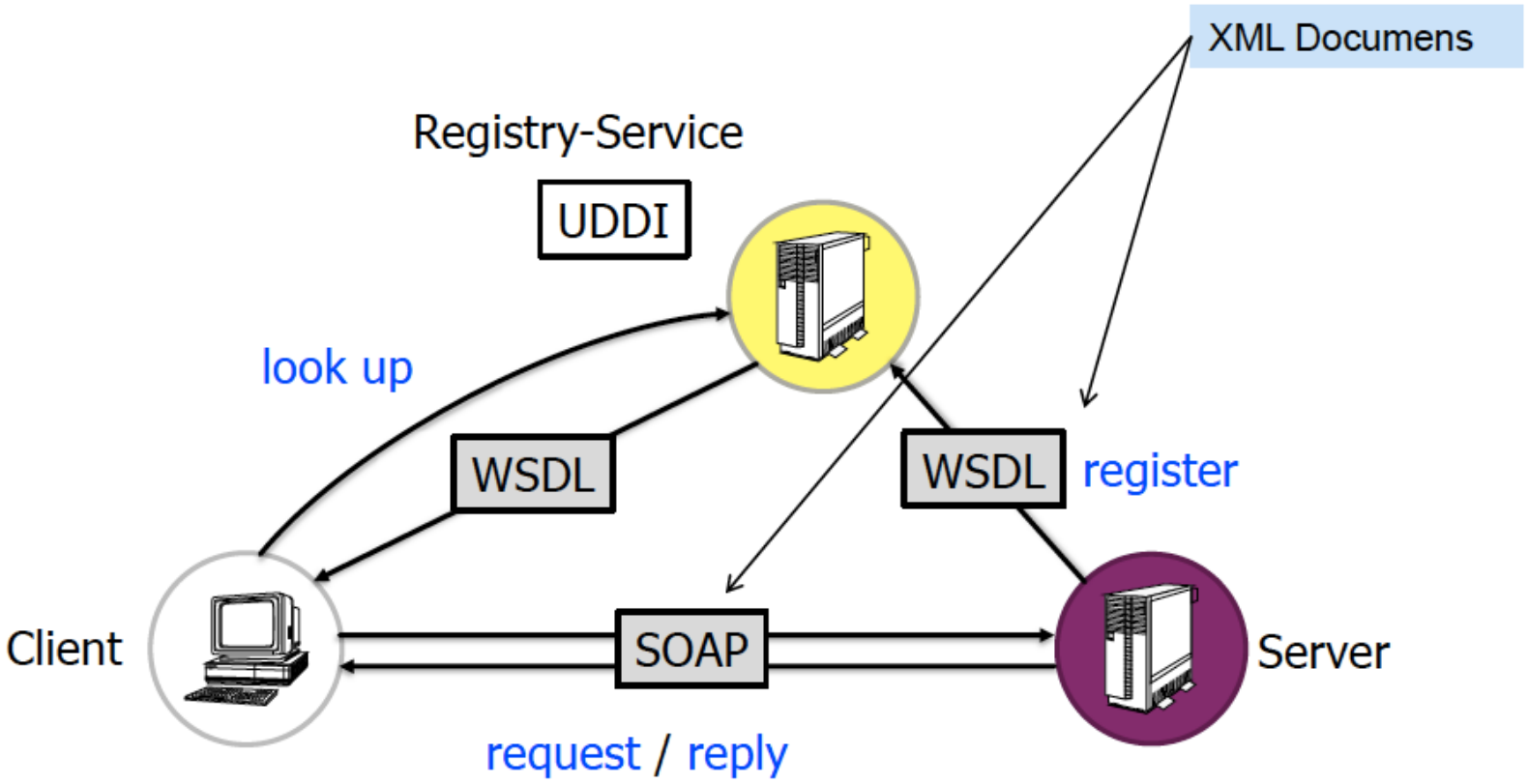
XML RPC was too Simple

We Created WS* (SOAP et co)

[WS* World Revisited: SOAP et co]



[WS* World Revisited: SOAP et co]

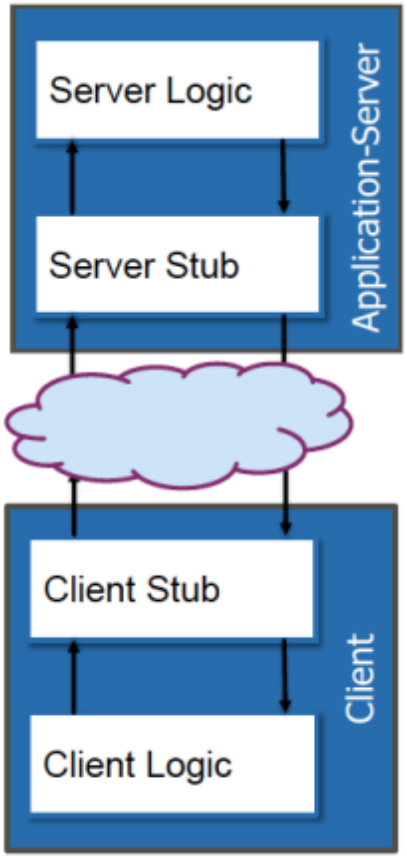


Material adapted from: S. Mayer. Service integration in the Web of Things, École d'Éte Web Intelligence 2013 – Web des Objets, 2013.

[WS-* in Practice]

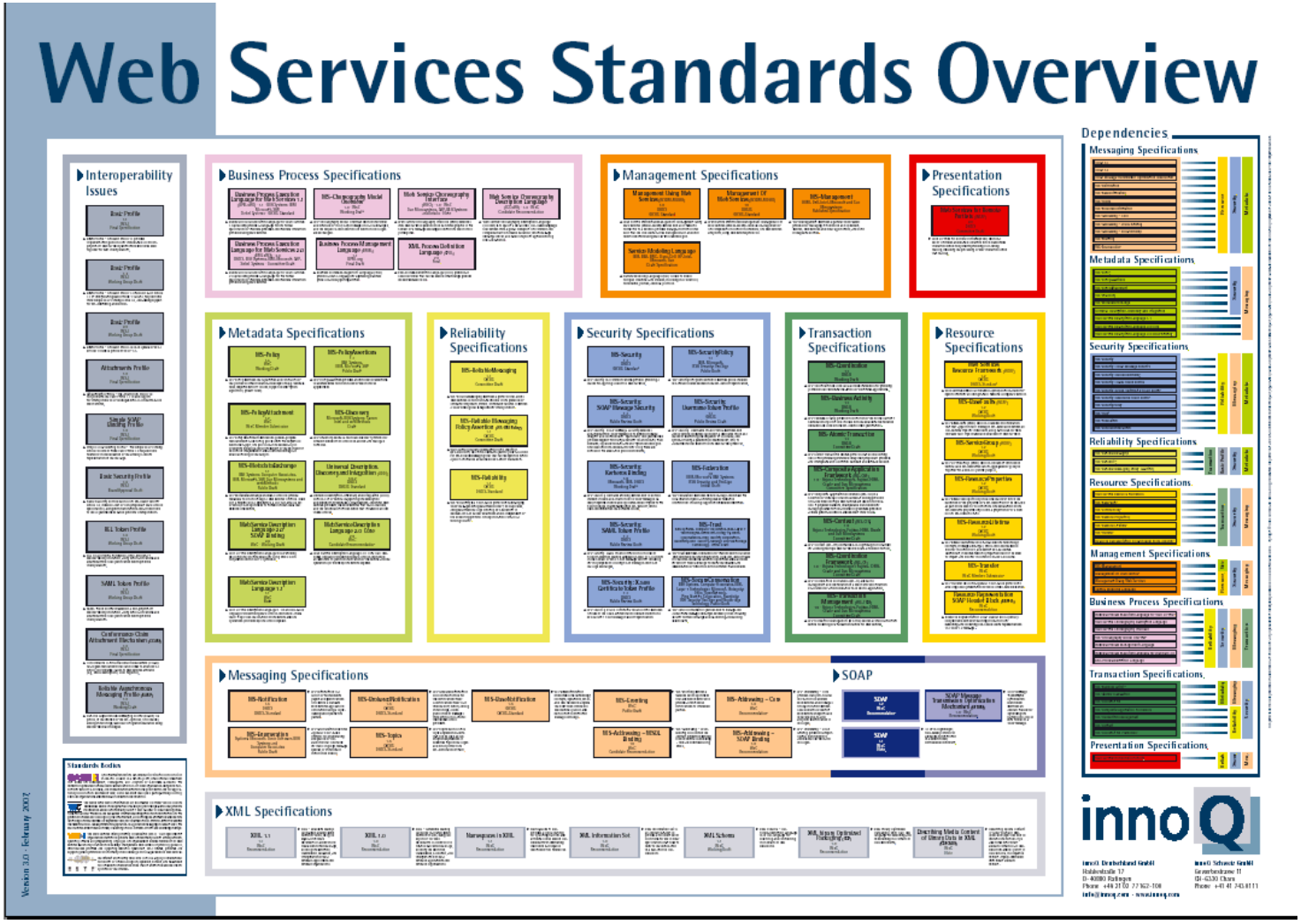
- **Stubs** handle communication issues, such as **message packing and unpacking ("marshalling")**
- The server stub is also responsible for publishing the **WSDL document**

- Creating stubs:
  - ✓ Bottom-up (code-first): **stub from implementation** (JAX-WS)
  - ✓ Top-down (contract-first): **stub from WSDL**



Stub is just a name of program/software routine

[WS* World Revisited: SOAP et co]



[WS-* Discussion]

- Extremely powerful and lots of available tools
  - ✓ Huge complexity
- WSDL is quite verbose for simple functionality
- SOAP over HTTP uses only POST
  - ✓ The power of HTTP (the web) is not exploited

REST (Representational state transfer)

[What REST is not]

REST is not a

framework

REST is not a

technology

REST is not a

standard specification

REST is

an architectural style

which specifies a set of architectural constraints to use HTTP for web services

An **architectural style** is characterised by the features that make a building or other structure notable or historically identifiable (wikipedia)

[What is REST]

- An architectural style of networked systems
  - ✓ It is not a protocol and not a specification
  - ✓ On the contrary, its main goal was NOT to introduce new protocols and specifications
  - ✓ It is not a standard
  - ✓ It is not a recommendation
  - ✓ It is a design guideline
- **Objectives :** expose resources on the web
  - ✓ A resource : web page, a video etc.

Any computation that returns a result can be transformed into a URI that identifies the result

✓ x = sum(2,3) might become http://example.com/sum/a=2&b=3

[What is REST]

Intuition

Every thing you need to implement a remote web service is already there. You only need to rely on the web and exploit it well

REST

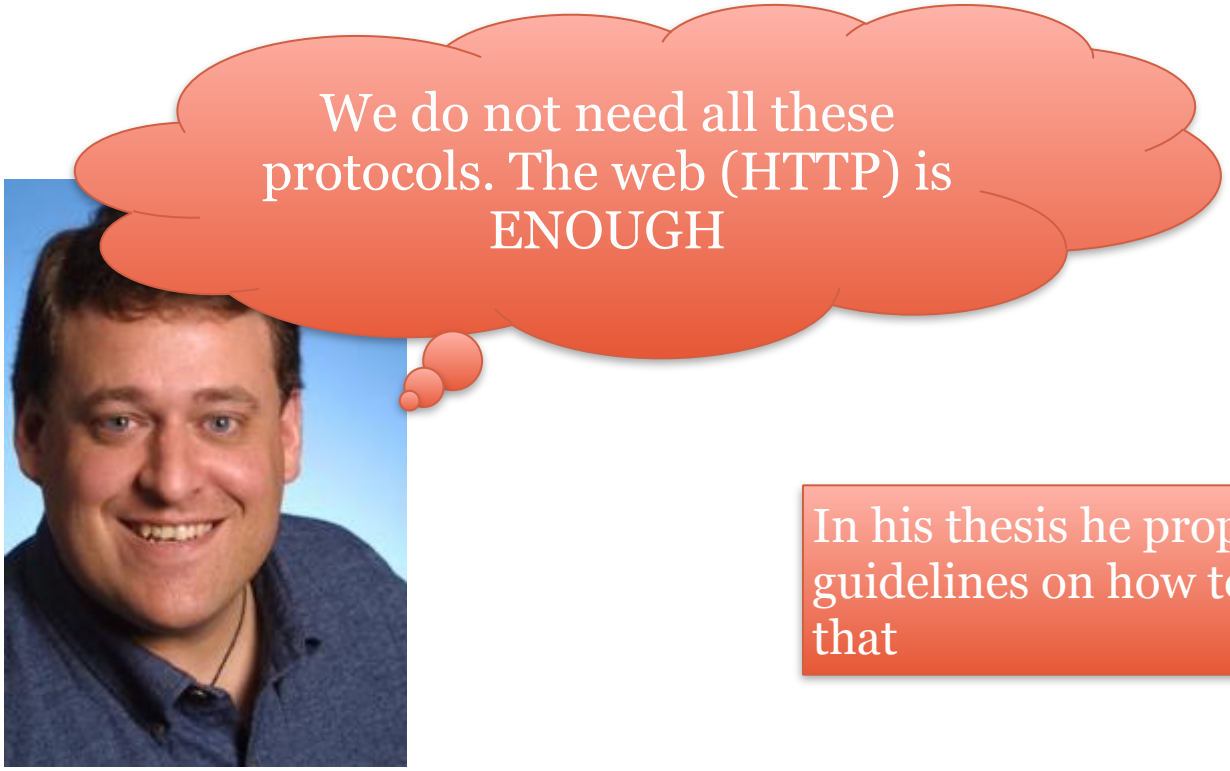
- It is a way of implementing a remote web service using standard technology that is already there
- **These technologies are HTTP, HTML, XML etc.**

[REST: Origins]

- REST : Representational State Transfer
- Roy Fielding did a PhD on the topic in UC Irvine in 2000
- He proposed the architecture style and the design guidelines
- His PhD Dissertation is a useful reference on REST and the motivation behind it
- He was considered by MIT as one of the top 100 innovators under 35

[REST: Origins]

In his thesis, he did not develop new tools or standards . He just said:



In his thesis he proposed guidelines on how to do that

[REST : Overview]

Client-Server Architecture

- Like the web, it is a client server architecture

Based on Representations

- Only representations are exposed to the client
- The representation places the client in a state

State evolution

- Client state may evolve when obtaining new representations
  - ✓ e.g. logging to Amazon

[Example: Amazon]

State Transfer

- These steps are the states. They tell you what operations you can do
- The client state evolves by navigation:
  - ✓ New operations available
- In REST, the state should be stored on the client. The server does not do state tracking

[Human vs Application-centric Web]

Human-Centric Web:humans are the primary actors initiating most web requests

The Application-Centric Web: conversations can take place directly between applications

[Remark]

REST is Application-centric

- Remember REST is a guide for web services not for web sites

- It is oriented to applications:
  ✓The goal is not to create nice web pages and nice graphics. Applications do not care about that
  ✓Instead of them, we use XML to communicate the information

[Resources: Definition]

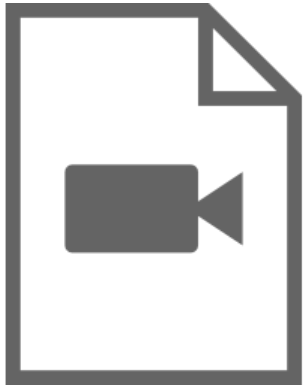
- A RESTful Architecture (e.g. the Web) is also called Resource-Oriented Architecture
  - ✓ Resource is the basic element
- A Resource (e.g. Book in Amazon's example):
  - ✓ is unique (Has a unique identifiers ISBN)
  - ✓ Has at least one representation (e.g. Paper or Kindle)
  - ✓ Has attributes
  - ✓ Has a schema or a definition
  - ✓ Can provide context
- In SOAP we have methods & objects, with REST we have resources

[Resource: Example]

- Web Site
- Your CV
- A song (e.g. in a MP3 web server)
  - ✓ Attributes: singer, year etc.,
- An application
- A printer

[Resource: Multiple Representations]

Resource: Sally's CV



[Resource VS Representation]

- The resource is not given to the user
  - ✓ Think of the resource as a record of the CV stored in The DB
- Instead we send representations of the Data:
  - ✓ XML
  - ✓ Text
  - ✓ Photo
  - ✓ Etc.

Representation Negotiation

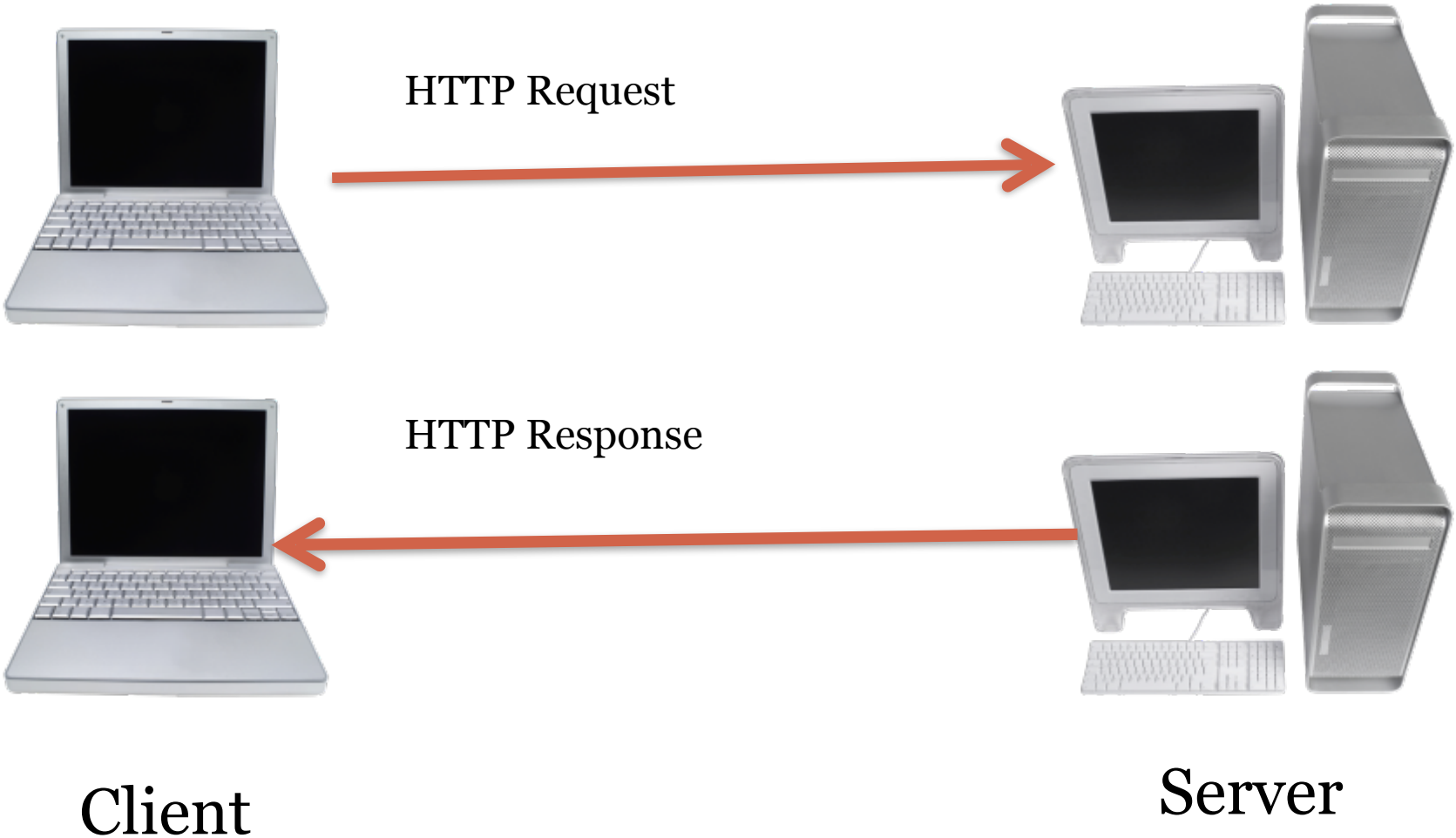
- Different applications might want different representations of the same resource
  - Application X prefers XML …

- REST guidelines advocate exploiting the full capacities of HTTP
- Lets recall HTTP

[Internet ≠ Web]

- Internet emerged in the late 1960's
- The web is an application over the internet
- The web was developed by Tim Berners Lee (@CERN) in late 1980s
- 1993: web browser (navigateur)
- 1994: W3C
- 1997: Recommendation

[HTTP Request & Response]



HTTP Request (1/2)

```
GET  http://google.fr/ HTTP/1.1
Host:www.google.fr
Accept: text/html
Accept-Language: text/html,text/xml..
Accept-Charset: ISO-8859-1,utf-8
Cookie: PHPSESSID=e532bd99681ebf06242a614c67f44572
```

- Host: The server hosting the required function
- Accept: list of MIME content types supported by the client (text, XML, JSON, RDF, Image, Video)
- Accept-Language: Specifies the supported languages
- Accept-Charset: Specifies the accepted character encodings

[HTTP Request (2/2)]

```
GET  http://google.fr/ HTTP/1.1
Host:www.google.fr
Accept: text/html
Accept-Language: fr,fr-fr
Accept-Charset: ISO-8859-1,utf-8
Cookie: PHPSESSID=e532bd99681ebf06242a614c67f44572
```

- GET is a method
  - ✓ A method is a command specifying a type of request
  - ✓ It ask the server to perform/execute a certain function
  - ✓ In general the action concerns a resource identified by the URL that follows the method name

[HTTP Methods: Example Get]

- The method is the most important part of the request
- It makes an action on a *resource*

Example:

The following URL presents **Télécom Saint-Etienne on Youtube:**

https://www.youtube.com/watch?v=OAYoOynYmgI

Whats happening after the question (?) mark

http://youtube/OAYoOynYmgI?t=4m41s

[HTTP Example: POST]

Submit form on the web

- Post does not put any parameters in the URL link
  - ✓ The URL does not change
  - ✓ The parameters are sent in the header (HTTP header)


```
POST http://www.exemple.fr/search.html
Host: www.exemple.fr
Accept: text/html
Accept-Language: fr,fr-fr
Accept-Charset: ISO-8859-1,utf-8
Cookie:
PHPSESSID=e532bd99681ebf06242a614c67f44572nom=dupont&mail=dupont
%40gmail.com&sujet=reclamation
```

[And Back to REST]

Attention

- HTTP and its methods exist before REST
- However, REST is a guideline advocating the full use of HTTP power for web services
- In REST terminology, GET, POST etc. are used to handle representations

[Most Important HTTP Methods]

- **GET**: requests a representation of the specified resource
  - Parameters are passed in the URL
  - http://www.gooogle.fr is a get request
- **POST:** requests that the server accepts the entity enclosed in the request as a new subordinate of the web resource
- **PUT**: requests that the enclosed entity be stored under the supplied URL
- **DELTE**: deletes the specified resource

[Remarks about HTTP Methods]

- Enable reading, writing, modifying and getting meta-data about a resource
- A resource is always identified using URL
- Example
  - ✓ www.moneyconvertor/conversion?s=dolar&d=euro&amount=100
  - ✓ Is a GET request equivalent of invoking a method sum(int a, int b)

Service Invocation versus Resource Access

- In SOAP we invoke a service (method): myService()
- In REST, we access the resource page (URL) with the appropriate method

REST Principles

[REST Principles]

1. Resource Identification
   - ✓ you need to know what are the resources we model (e.g. a person, a movie)
2. Addressability
   - ✓ clients should be able to refer to the resources
3. Statelessness
   - ✓ the provider does not keep any information about the client profile
4. Resource Representations
   - ✓ e.g. XML, PDF, video, etc
5. Links & Connectedness
   - ✓ resources are connected
6. Uniform interface:
   - ✓ the HTTP methods

[Rest Principles: 1) Resource Identification]

- Resources are identified by a URI
  - www.example.com/services/myresource1
- A resource should be accessible at least @ one URI
- URI can be
  - URN: e.g. ISBN for books, DOI
  - URL
- A URI designates exactly one resources

[Rest Principles: 2) Addressability]

Addressable service

- A service is addressable if it exposes its data set as resources
  - ✓each one having a URI

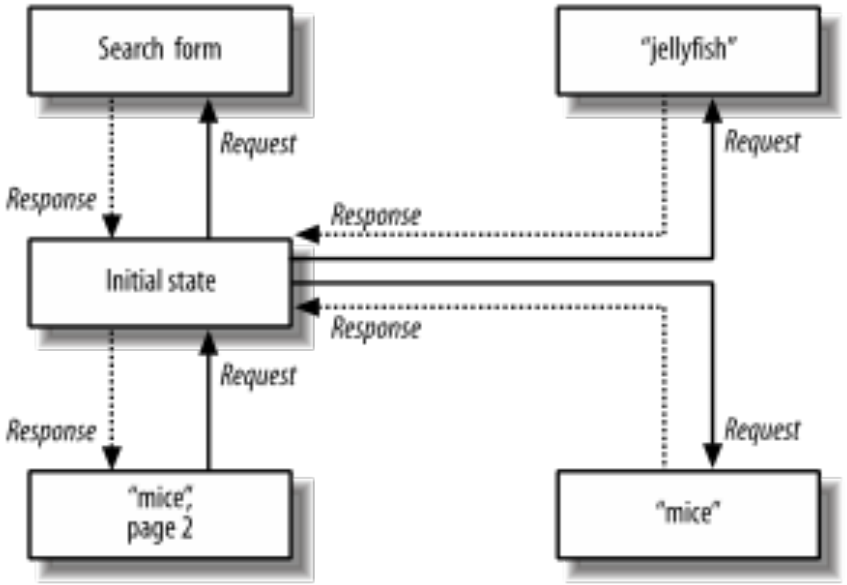
- ✓ The filesystem is addressable
- ✓ Google is addressable
  - http://www.google.fr/?q=hello
  - You can even bookmark it
- ✓ This seems natural in the web, but..

[REST Principles: 3) Statelessness]

Statelessness

[Statelessness : Google Example (1/2)]

- Statelessness : requests does not change google
- No order
  - ✓ Search for “jellyfish”
  - ✓ Then for “mice”
- It does not remember what you did
- Does not expect you to do anything
- When you make a search with “jellyfish” the results are not stored
  - ✓ Accessing page 2 is making a new query



[Statelessness : Google Example (2/2)]

- It is lightweight
  - ✓ Google does not need to track you behaviour which would be an enormous overhead
- You (the client) need to tack your state
  - ✓ e.g. when you want the second page of you search results, google does not sore the result, you need to retrieve it:
  - ✓ You rely on the parameters passed on the GET Method
    - http://www.google.fr/?q=hello&start=30
    - This access page 3 of the search result
    - Google forgets about the search you did on page 1 and performs a new one
- Google is scalable

[REST Principles: 3) Statelessness]

Why it is useful

- Scalable, easy to cache & addressable
  - ✓e.g. using the GET parameters in the URL
- Can be bookmarked
- HTTP is stateless

It is not always respected

- Broken when we use _SESSION variable (such as in PHP)
  - ✓Amazon example
- This is not very scalable

[REST Principles: 3) Statelessness]
What if a state is needed ?

Then the state of the client should be sent to the server in each request

Guidlines

- The state remains on the client side
- The client state passed as argument to the server
- The server does not keep the state of the client
  - ✓ Compare with Amazon

[REST Principles: 3) Statelessness]
Flicker Example

- Then do I need to send all my photos to flicker every time I make a request ?
  - ✓ No, of course not
- Photos are resources, when you upload one, it is stored on the server
- You need to distinguish between client & resource state

Service	Resource State	Client State
flickr	***Photos are resources and***	***You browser is a client and has***
Google	***Indexed web pages***	***Your query and your current***

[REST Principles: 3) Statelessness] Summary

- Statelessness applies to the client state
  - ✓ The resource state is the same for every client and it is managed on the server
- The Server should sleep, unless awaken by the client => It should rest
  - ✓ When it sleeps it forgets
- The client manages its own path in the application

[REST Principles: 4) Resource Representation]

A resource needs a representation for it to be sent to the client

A representation of a resource

- A representation is a data about the resource or a view of its current state
  - ✓Let us say that we have a program and we want to represent its bugs
    - XML file, web page, pdf etc.

[REST Principles: 4) Resource Representation]

- Might contain meta-data about the resource
  - ✓ E.g. meta-data of books:
    - Cover image
    - Reviews

Two-way traffic

- The client can also send representations
  - ✓ In Flicker, the client uploads his representation
  - ✓ The server creates a new resource for your picture
- Always remember that a server does not give access to a resource
  - ✓ A resource is a record in the DB
  - ✓ Only access to representation is given

[REST Principles: 4) Resource Representation]
How to get a representation ?

1. You should have the distinct URI of each representation
   1. www.example.com/release /1.04_en (english)
   2. www.example.com/release /1.04_fr (french)
   3. www.example.com/release /1.04_es (spanish)
2. you use the GET with the URI

✔ Addressability

Each representation is accessible

✔ Statelessness

The server receives from the client all the information necessary

[REST Principles: 4) Resource Representation]
How to get a representation ?

2. Use the HTTP method HEAD to negotiation the content
   - ✓ Use HEAD on www.example.com/release/
   - ✓ The server sends the client a list of available representations:
     - English, French, Spanish
     - The client can choose
   - ✓ Representations
     - e.g. Filetype: pdf, XML, JSON etc.

GET vs HEAD

- The first method: You need to know the URI of the representation beforehand. It is easier to implement from client perspective
- The second method: you only need to know the server address
  - ✓ Better since the service might change
  - ✓ Loosely coupling

[Let's look at puppies]

hackett

hackett

hackett

hackett

[A Dog Store: REST Example]

```
...
/getAllDogs
/locationVerify
/foodNeeded
/createRecurringDogWalk
/giveDirectOrder
/healthCheck
/getRecurringDogWalkSchedule
/getLocation
/getDog
/massDogParty
/getNewDogsSince
/getRedDogs
/getSittingDogs
/dogStateChangesSearch
/replaceSittingDogsWithRunningDogs
/saveDog
...
```

[A Dog Store: REST Example]

We only Need two based URLs per resource in Dogs databased

The first one is for collection:

```
/dogs
```

The first one is for an element:

```
/dogs/1234
```

[REST Calls (HTTP Methods)]

```
POST
GET
PUT
DELETE
```

```
CREATE
READ
UPDATE
DELETE
```

[REST Calls (HTTP Methods)]

Resource	**POST** create	**GET** read	**PUT** update	**DELETE** delete
/dogs	create a new dog	list dogs	replace dogs with new dogs	delete all dogs
/dogs/1234	treat as a collection create new dog in it	show Bo	if exists update Bo if not create Bo	delete Bo

[REST Calls (HTTP Methods)]

Resource	**POST** create	**GET** read	**PUT** update	**DELETE** delete
/dogs	create a new dog	list dogs	**bulk update dogs**	delete all dogs
/dogs/1234	**error**	show Bo	if exists update Bo **if not error**	delete Bo

[REST Calls (HTTP Methods)]

What about Complex Variations"?"

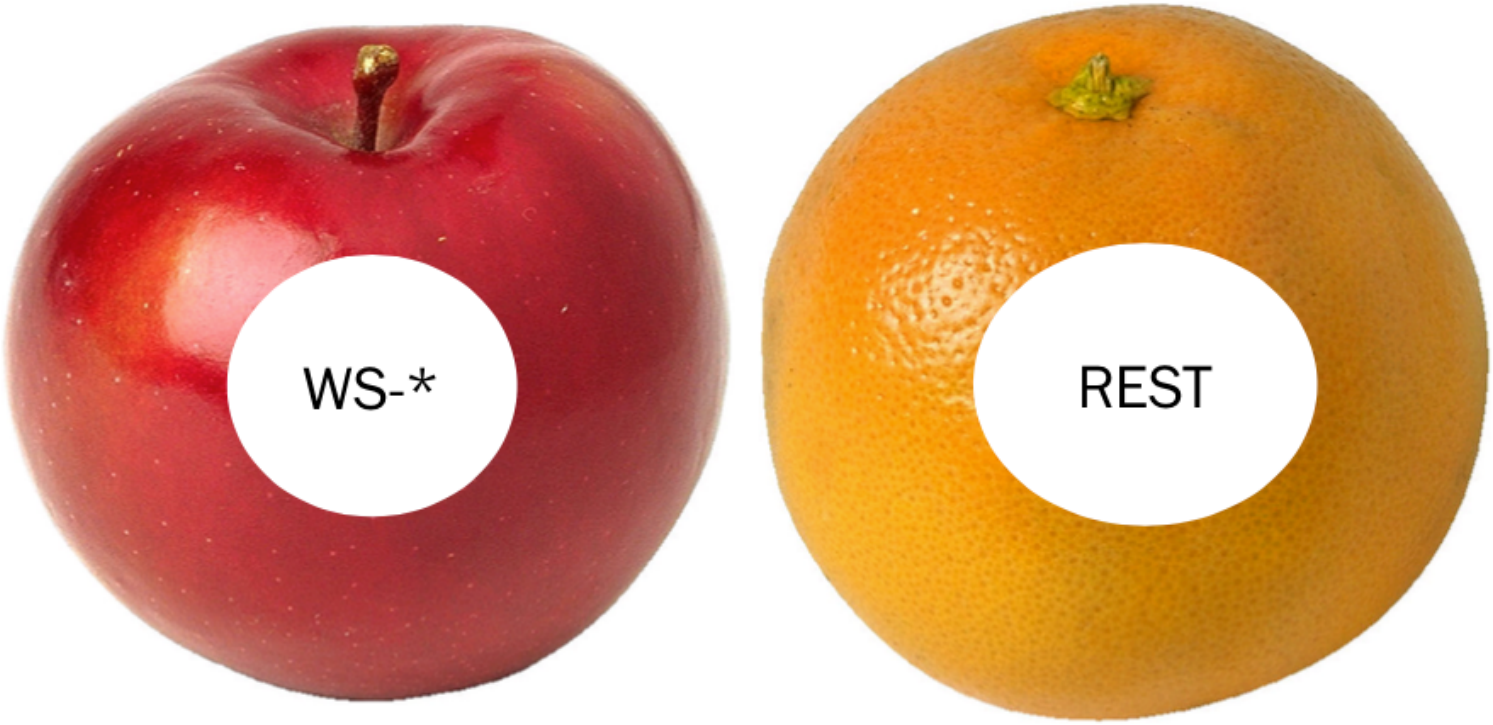
- Sweep variations under the '?'

```
/dogs?color=red&state=running&location=park
```

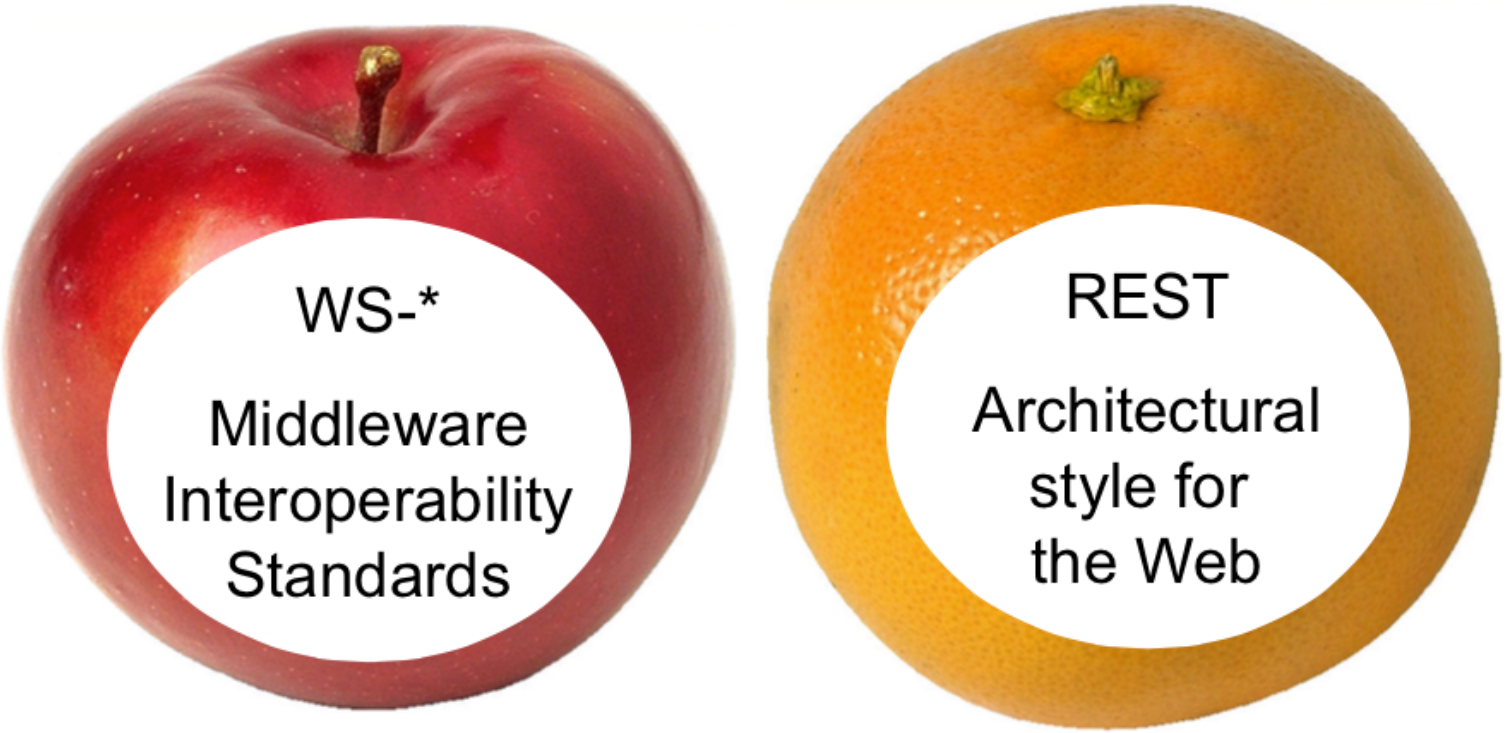
https://www.youtube.com/watch?v=OAYoOynYmgI

I want to watch video
?v=OAYoOynYmgI

[REST Vs WS-* (SOAP)]



[REST Vs WS-* (SOAP)]

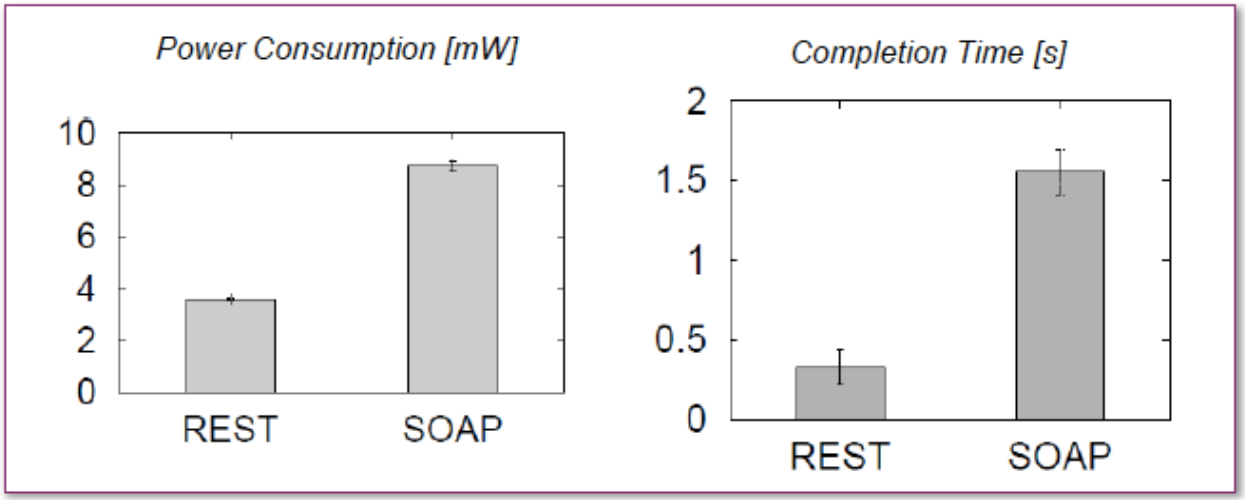


[WS-* and REST]

- WS-* uses HTTP as a **transport** protocol
  - ✓ SOAP uses only POST
    - HTTP-based caching not possible
    - The operation itself is pushed in the SOAP message
  - ✓ Applications remain **outside of the Web**
- REST uses HTTP as an **application** protocol
  - ✓ Applications are **part of the Web**

[WS-* VS REST]

- REST vs. SOAP on resource-constrained devices



D. Yazar and A. Dunkels: Efficient Application Integration in IP-based Sensor Networks, 2009.

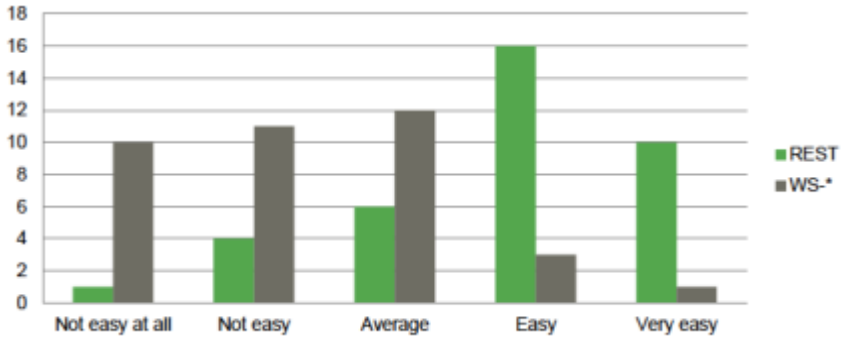
Material adapted from: S. Mayer. Service integration in the Web of Things, École d'Éte Web Intelligence 2013 – Web des Objets, 2013.

[WS-* and REST]

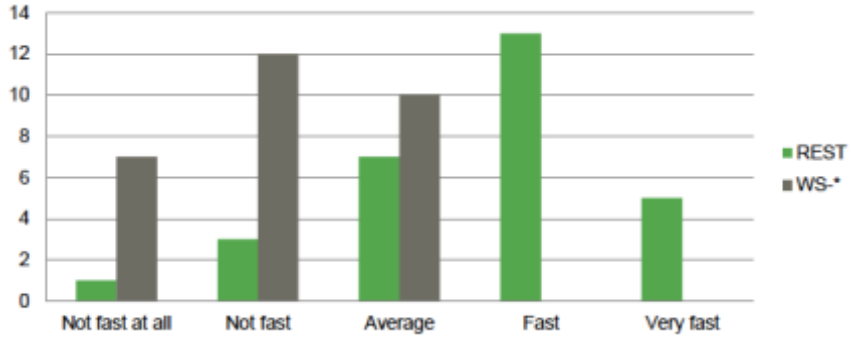
- A developer's perspective
  - ✓ ETH Zurich
  - ✓ 69 computer science students (3rd or 4th year of Bachelor studies)

REST:
- very **easy** to understand, learn and implement
- More **lightweight** and **scalable**

WS-*:
- WSDL enables **service contracts**
- Better **security** features
- Better level of **abstraction**



Ease of learning



Speed of learning

D. Guinard, I. Ion, and S. Mayer. REST or WS-*? A developers' perspective, 2011.

Material adapted from: S. Mayer. Service integration in the Web of Things, École d'Éte Web Intelligence 2013 – Web des Objets, 2013.

[WS-* and REST]

- REST is well suited for “Do-It-Yourself”, ad-hoc integration over the Web (mashups)

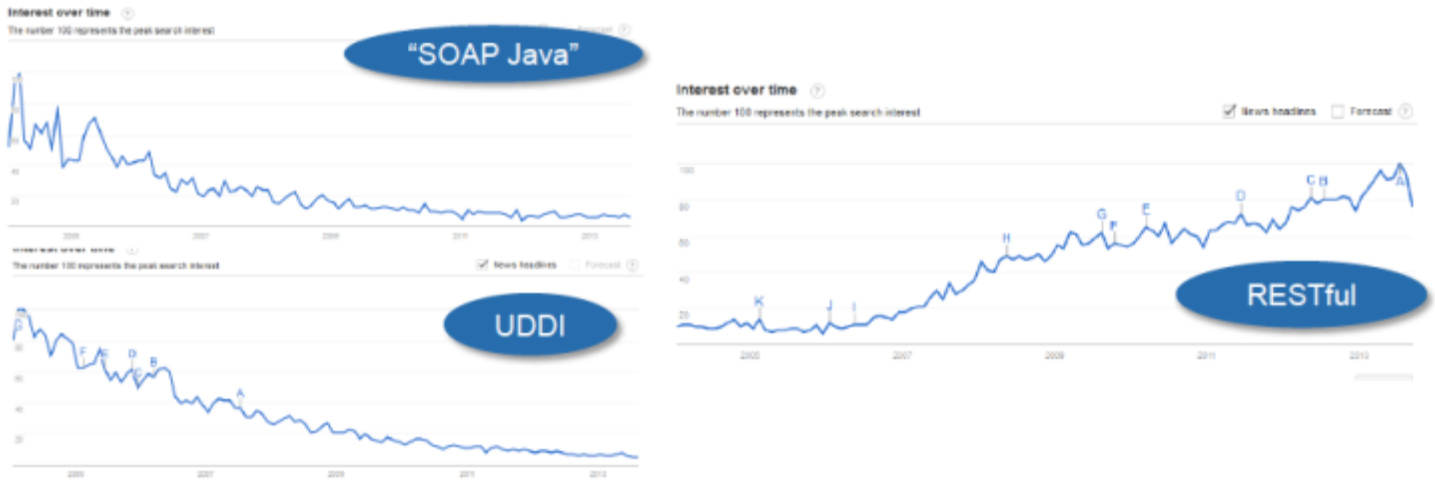
- WS-* is well suited in enterprise applications with advanced QoS requirements
  - ✓ However, REST gains more adoption here as well

C. Pautasso, O. Zimmermann, and F. Leymann: RESTful Web Services vs. “Big” Web Services. Making the Right Architectural Decision, 2008.

[WS-* and REST]

- UDDI **discontinued** by IBM, Microsoft, SAP in 2006
  - ✓ Functionality removed from Windows Server in 2010
- WS-* APIs **discontinued** by Google in 2011

Google Trends, 2004-2013



Material adapted from: S. Mayer. Service integration in the Web of Things, École d'Éte Web Intelligence 2013 – Web des Objets, 2013.

[REST vs WS* (SOAP): Technologies]

REST	SOAP
A STYLE	***A Standard***
Proper REST: Transport must be HTTP/ HTTPS	***Normally transport is HTTP/HTTPS but can be something else***
Response data is normally transmitted as XML, can be something else. ***❖On average the lighter of the two as does not have SOAP header overhead***	***Response data is transmitted as XML***
Request is transmitted as URI ***❖Exceptionally light compared to web services*** ***❖Limit on how long it can be*** ***❖Can use input fields***	***Request is transmitted as XML***
Analysis of method and URI indicate intent	***Must analyse message payload to understand intent***
…	***WS* initiatives to improve problems like compression or security***

[REST vs WS* (SOAP): Languages]

REST	SOAP
Easy to be called from JavaScript	*JavaScript can call SOAP but it is hard, and not very elegant.*
If JSON is returned it is very powerful (keep this in mind)	*JavaScript parsing XML is slow and the methods differ from browser to browser.*
C# (Visual Studio) parsing of REST means using HttpWebRequest and parsing the results (string/xml) or normal service consumption (.NET 3.5 SP 1 and later).	*C# (Visual Studio) makes consuming SOAP very easy and provides nice object models to work with.*
C# version 4 should make this easier thanks to new dynamic methods.	*...*
There are 3rd party add-on's for parsing JSON with C# so that may make it easier.	*...*

[Fin]

Acknowledgments:

- ✓ Leonard Richardon & Sam Ruby, **RESTful Web Services.** O'Reilly Media;
- ✓ Roy Fielding PhD dissertation: **Architectural Styles and the Design of Network-based Software Architectures**
- ✓ **Amro NAJJAR, Web Services**