

Web Architecture & REST

1er cuatrimestre 2023

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

Agenda

Estilos que conocemos hasta ahora

La Web

¿Arquitectura de la Web?

Un poco de su historia

Requerimientos y Atributos de Calidad

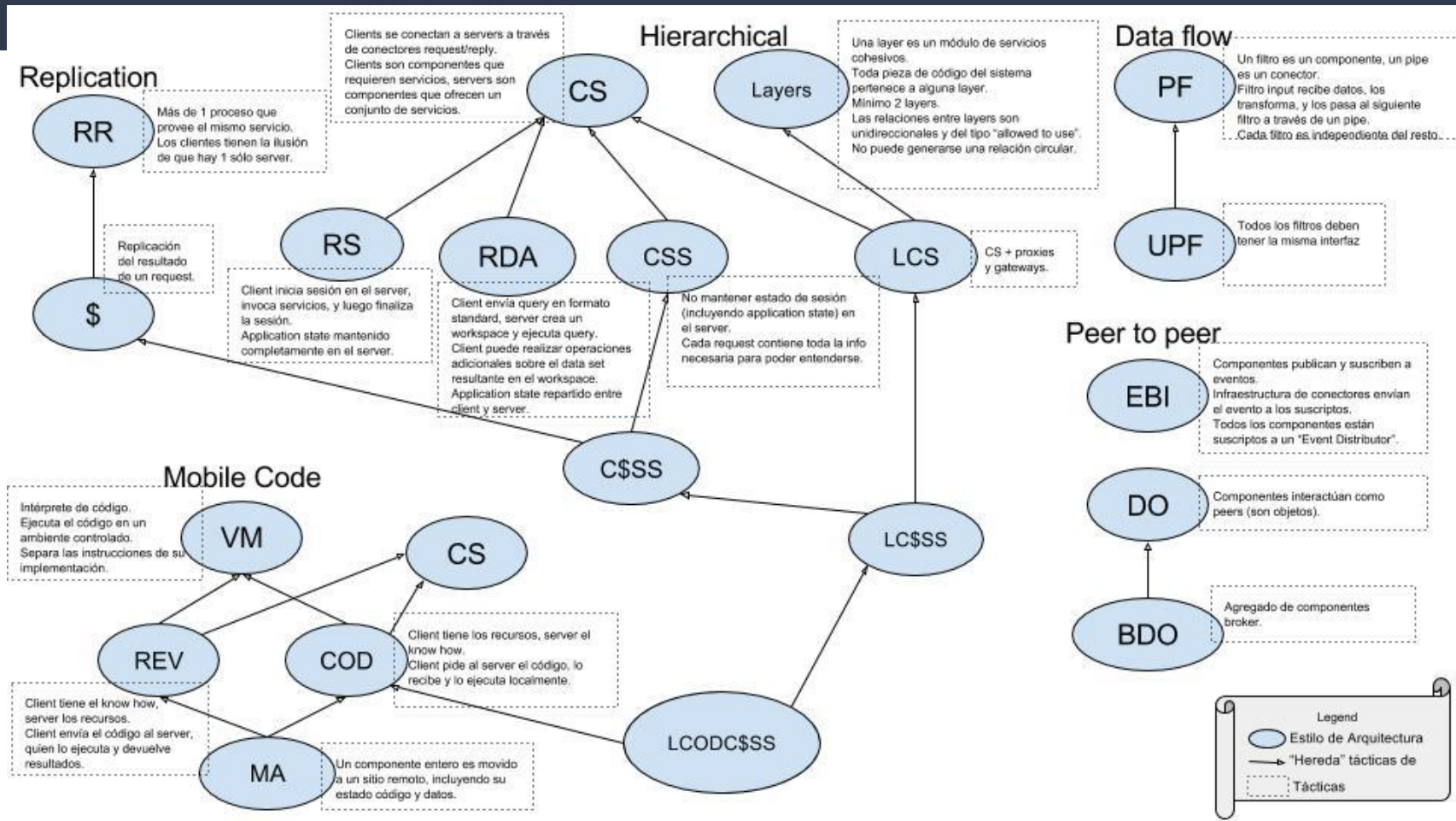
El camino de Fielding

REST

Hypermedia

REST APIs

¿Estilo de Arquitectura de la Web?



¿La Web = Internet? NO

Internet



- Sistema de redes de computadoras interconectadas
 - ◆ Internet Protocol suite ("TCP/IP")

Internet protocol suite	
Application layer	
BGP · DHCP · DNS · FTP · HTTP · IMAP · LDAP · MGCP · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · more...	
Transport layer	
TCP · UDP · DCCP · SCTP · RSVP · more...	
Internet layer	
IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · more...	
Link layer	
ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · DSL · ISDN · FDDI) · more...	

World Wide Web



- Sistema de documentos **hipertexto** interrelacionados, accedido a través de internet
- Podría verse a la web como a una aplicación "corriendo" sobre Internet
- Tecnologías
 - ◆ HTTP
 - ◆ HTML
 - ◆ URI
 - ◆ JavaScript

La Web

1945: Vannevar Bush - idea de links entre documentos

1965: Ted Nelson utiliza el término “hypertext”

1974: TCP spec

1978: Internet Protocol (IP) spec

1984: Primer implementación de DNS (BIND)

1988: Primer conexión IP entre Europa y EEUU

1989: Tim Berners-Lee inventa la Web



TED Talk 2009

https://www.ted.com/talks/tim_berniers_lee_on_the_next_web

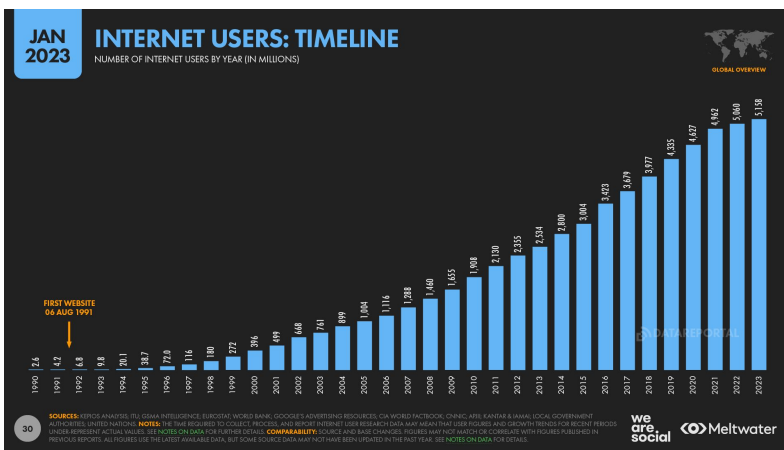
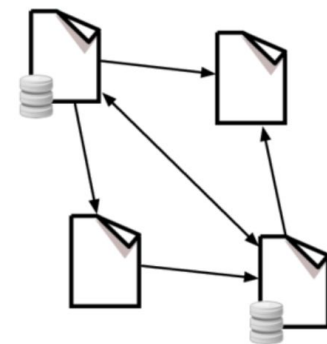
El objetivo de la Web



*“Web’s major goal was to be a **shared information space** through which **people and machines could communicate**”*

Requerimientos de la Web

- Proveer a los usuarios una forma de **almacenar y estructurar** su **información**
 - ◆ Permanente o efímera
 - ◆ Puede ser usada por esa mismas personas y/o por otras
- Permitir **referenciar** la información almacenada por otros usuarios
 - ◆ No debe ser necesario que todos mantengan copias locales

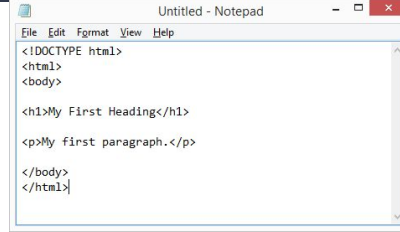


- Los usuarios pueden estar localizados a lo largo y ancho del **planeta**
- Los equipamientos de los usuarios:
 - ◆ están conectados a través de Internet
 - ◆ son **heterogéneos** (terminales, workstations, servers, supercomputers, etc)
- Posibilitar el **deployment incremental** a medida que nuevas personas y organizaciones se sumen al proyecto

Atributos de Calidad de la Web

→ Low Entry-barrier (Usability)

- ◆ creación de info es voluntaria
- ◆ posibilitar adopción
- ◆ lectores, autores, desarrolladores



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

→ Extensibility

- ◆ un sistema de larga vida debe estar preparado para el cambio

→ Availability

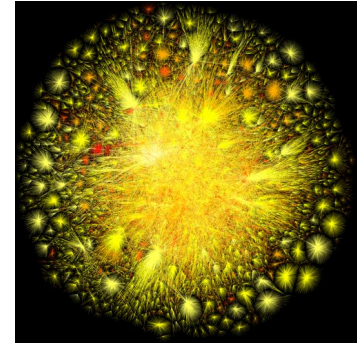
- ◆ la red puede presentar condiciones adversas y las mejores aplicaciones construidas sobre la web deben poder resistir a ellas

→ Internet-Scale

- ◆ **Anarchic Scalability:** no hay relaciones a largo plazo ni coordinación entre diferentes partes del sistema
 - No se puede pretender que los clientes mantengan conocimiento de todos los servers
 - No se puede pretender que los servers mantengan estado entre requests
- ◆ **Independent Deployment:** las partes del sistema evolucionan a diferente ritmo

→ Distributed Hypermedia

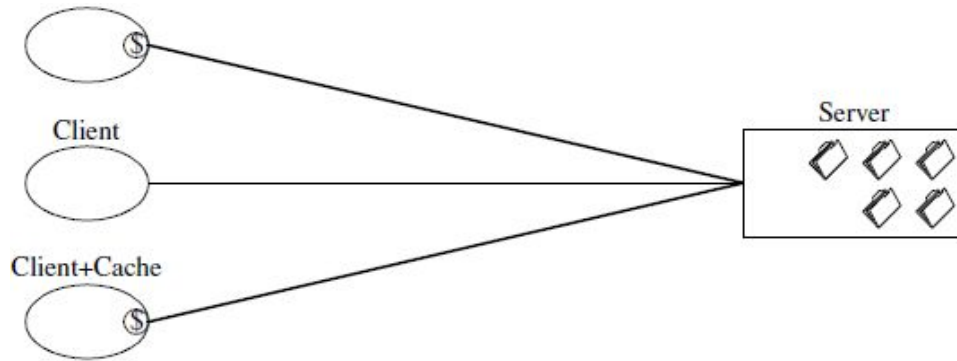
- ◆ Las **instrucciones sobre qué se puede hacer con los datos** se trata igual que a los datos
- ◆ Tanto los datos como sus instrucciones son manejadas por el server



Buscando la Arquitectura de la Web

El camino de Fielding

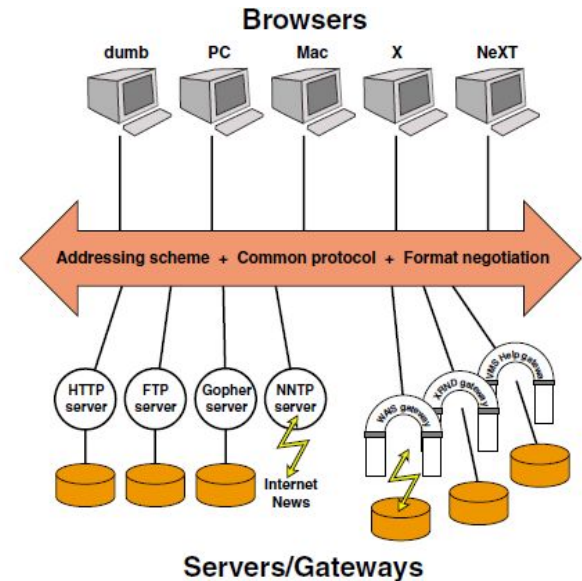
Web Architecture circa 1994



¿Estilo de Arquitectura?

C\$SS

- **Client-Server:** Portabilidad de UI + evolución independiente (Internet-scale)
- **Stateless:** Visibility + Reliability + Scalability
- **Cache:** Performance



© 1992 Tim Berners-Lee, Robert Cailliau, Jean-François Groff, C.E.R.N.

Agregando tácticas

*“The design rationale behind **the Web architecture can be described by an architectural style** (...).*

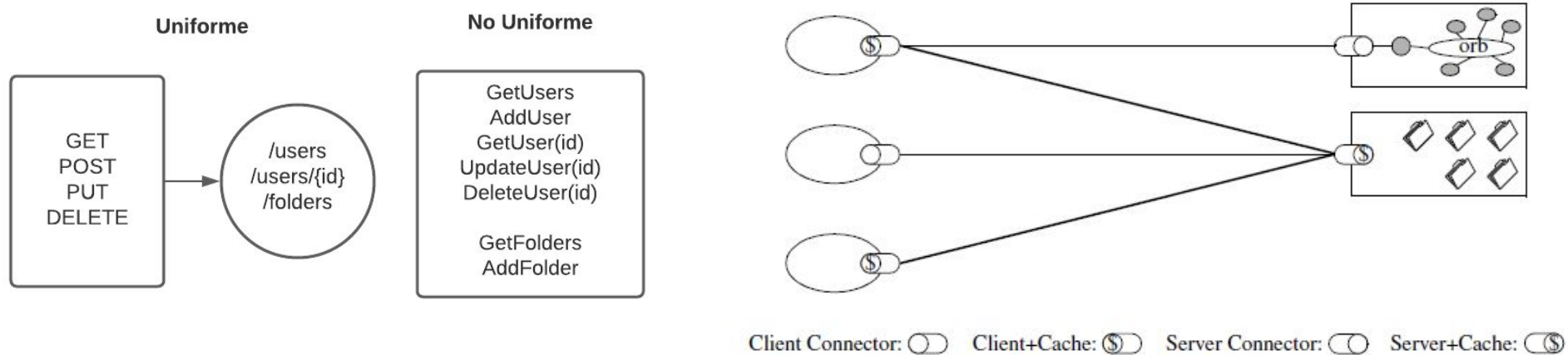
By examining the impact of each constraint as it is added to the evolving style, we can identify the properties induced by the Web's constraints.

Additional constraints can then be applied to form a new architectural style that better reflects the desired properties of a modern Web architecture.”



Roy Fielding

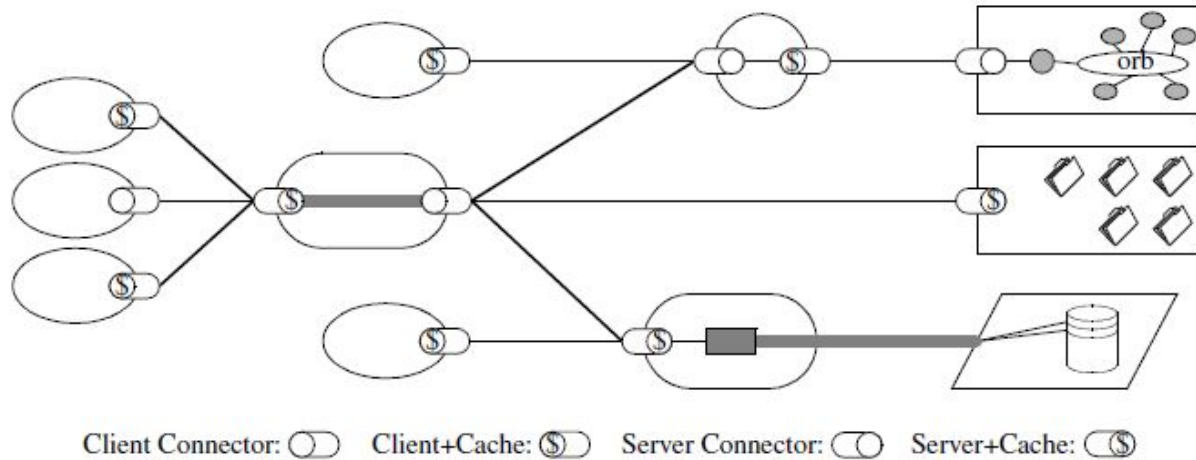
C\$SS + Uniform Interface (U)



- Principio de Generalidad en las interfaces: interfaz común entre componentes
- Datos transferidos en formato estándar
- Implementaciones desacopladas de servicios

- + **Simplicity**
- + **Visibility**
- + **Independent Evolvability**
- **Efficiency (Performance)**

U C\$SS + Layers

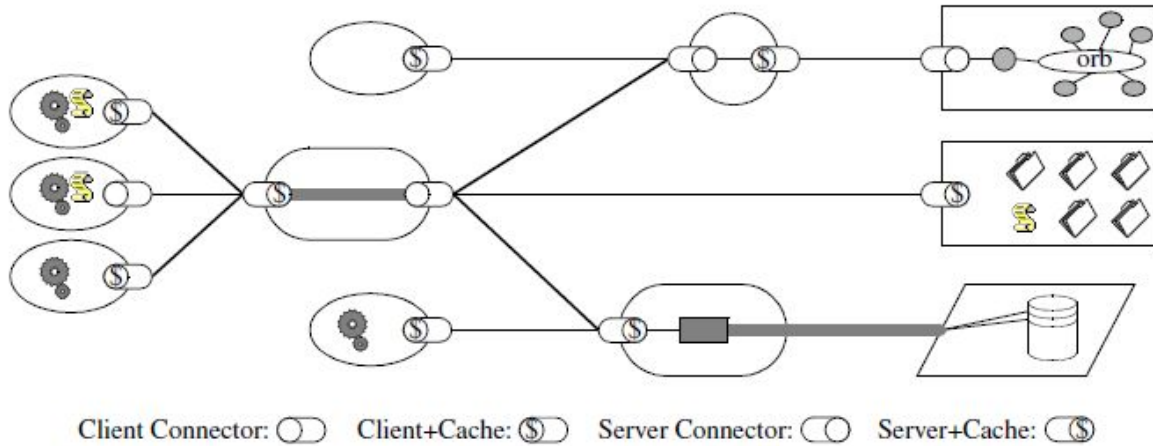


ULC\$SS

- Cada componente no puede “ver” más allá de su layer inmediata
- Load Balancing
- Shared cache en layers intermedias
- Componentes intermedios pueden transformar contenido

- + Internet-scale
- + Independent Evolvability
- Performance

U LC\$SS + CODopt



$$\text{LCODC\$SS} + \text{U} = \text{REST}$$

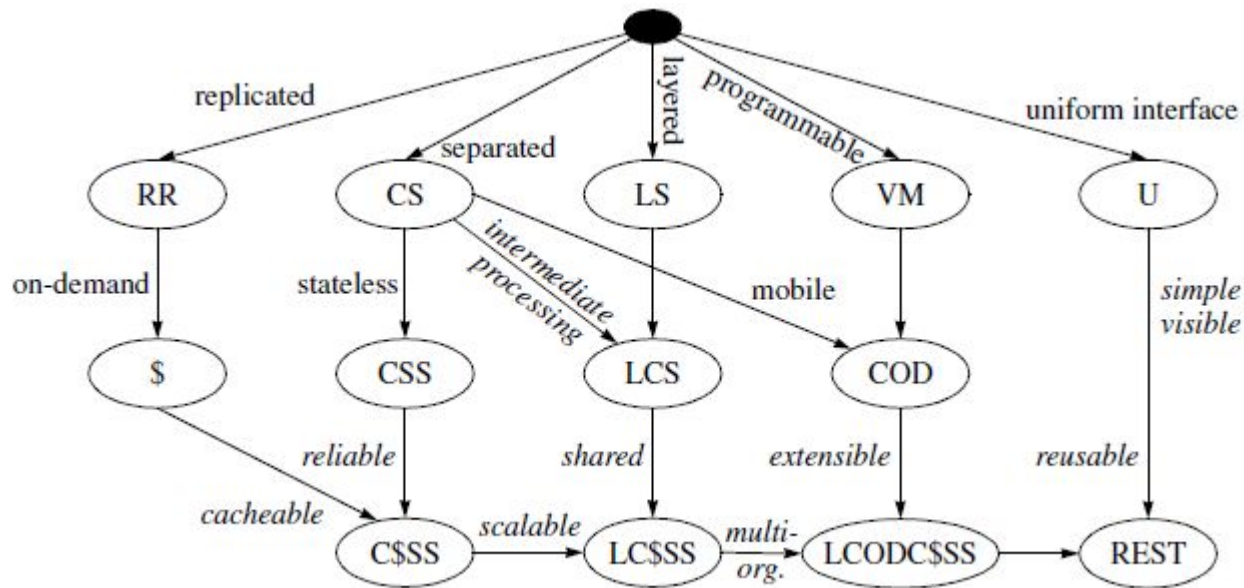
- Clients pueden descargar y ejecutar código (scripts, applets)
- Se reducen cantidad de features a pre-implementar

- + Extensibility
- Visibility

REST

REST

REpresentational State Transfer



Es un estilo de arquitectura

Captura la esencia de la arquitectura de la Web

Uniform Interface – 5 tácticas

1. Identification of Resources (“addressability”)
 - a. Todo recurso debe tener 1 ID
 - b. El ID puede cambiar

Simplicity, Visibility, Reusability
2. Access methods have the **same semantics** for all resources
 - a. Ej.: GET, PUT, POST, etc. en HTTP

Visibility, Scalability & Availability (posibilita LS y \$)
3. Manipulation of resources through **representations**
 - a. Representación: una secuencia de bytes y metadata que los describe
 - b. Capturan el estado actual ó deseado de un recurso
 - c. Se transfieren entre componentes

Simplicity, Visibility, Reusability, Evolvability (information hiding)
4. Self-descriptive messages
 - a. En particular, la interacción es stateless
 - b. Ej.: Content-Type header en HTTP
 - c. Las respuestas explicitan política de Cache

Visibility, Scalability & Availability (posibilita LS y \$), Evolvability (extensible communication)
5. **Hypermedia As The Engine Of Application State (HATEOAS)**
 - a. AKA “HATEOAS”, “The Hypermedia Constraint”, “Connectedness”
 - b. Próximo slide!

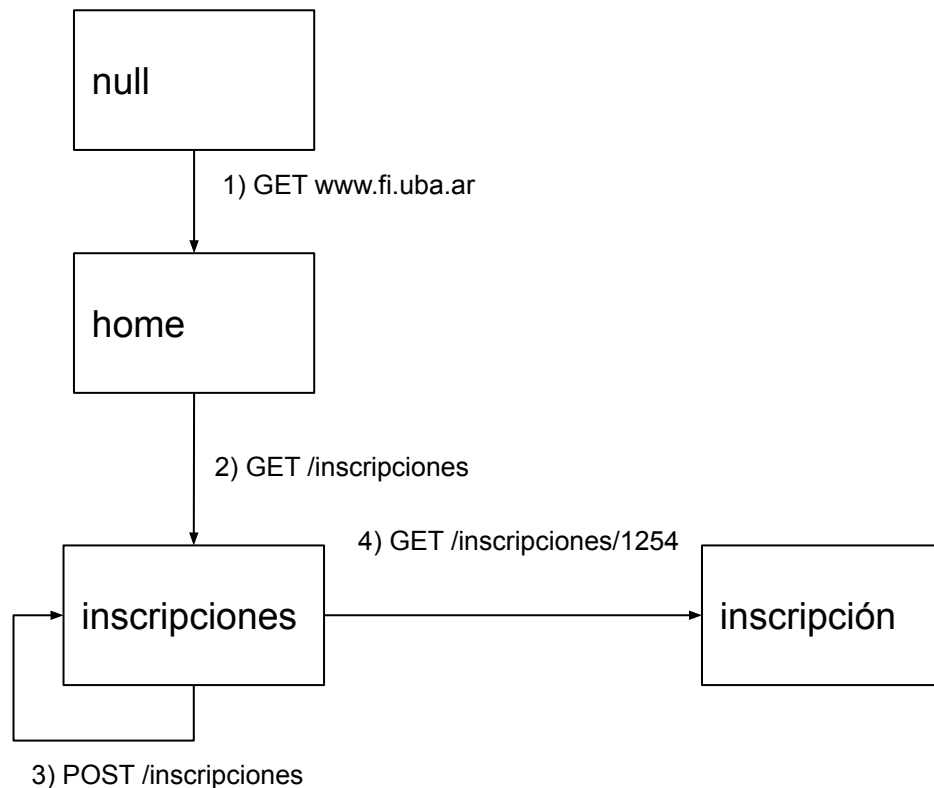
Simplicity, Visibility, Reusability, Evolvability (loose coupling), Adaptable (late binding of application transitions)

Application State

Application State: información acerca de “dónde está” (el camino) un client en la interacción con el server

Resource State: estado de un recurso, almacenado por el server, que puede ser modificado a través de enviar representaciones al server

En REST, el Application State es mantenido en el cliente



Hypermedia

- “Presence of **application control information** embedded within, or as a layer above, the **presentation of information**”
 - ◆ “application control information” = instrucciones sobre qué se puede hacer con la información
- Conecta recursos y describe sus habilidades en lenguaje legible por máquinas
- Es la forma que tiene el server de presentar el **menú de opciones** al client
 - ◆ El server informa qué puede pasar
 - ◆ El client decide qué es lo que va a pasar
- Relaciones hypermedia: links
 - ◆ Permiten estructuración ilimitada de info

HTML <a>

```
<a href = " http://www.fi.uba.ar/es/biblioteca">Biblioteca</a>
```

Significa que el cliente puede hacer un request:

```
GET /es/biblioteca HTTP/1.1
Host: www.fi.uba.ar
```

HTML <form>

```
<form action="/es/biblioteca" method="post">
  <input type="text" name="terminoBusqueda" value=""/>
  <input type="submit" value="Buscar"/>
</form>
```

```
POST /es/biblioteca HTTP/1.1
Host: www.fi.uba.ar
Content-Type: application/x-www-form-urlencoded

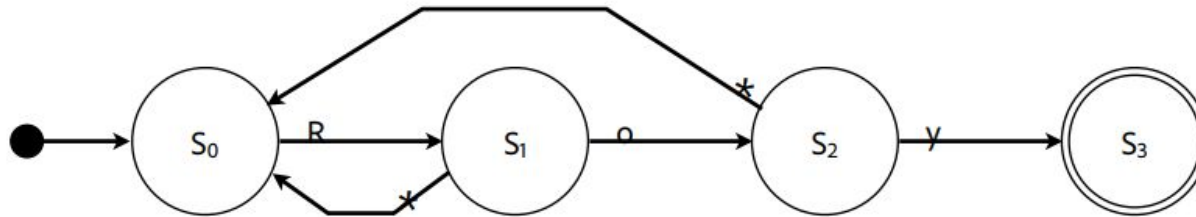
terminoBusqueda=Hola&submit=Buscar
```

Volviendo a HATEOAS

- Application state
 - ◆ del lado del cliente
 - ◆ sólo puede ser modificado haciendo un request y procesando response
- El cliente sabe qué requests puede hacer a continuación examinando los controles hypermedia de la representación recibida.
 - ◆ ⇒ **hypermedia es el motor que permite cambios en el application state**

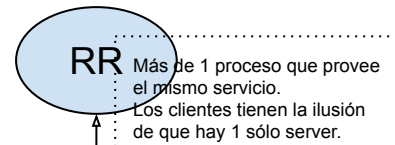
- El client sólo necesita saber 1 estado y sus transiciones
- Client se puede adaptar a cambios del server
- Server puede cambiar su implementación interna sin afectar clients

"induces simple, visible, reusable, and cacheable through data-oriented integration, evolvable via loose coupling, and adaptable through late binding of application transitions"

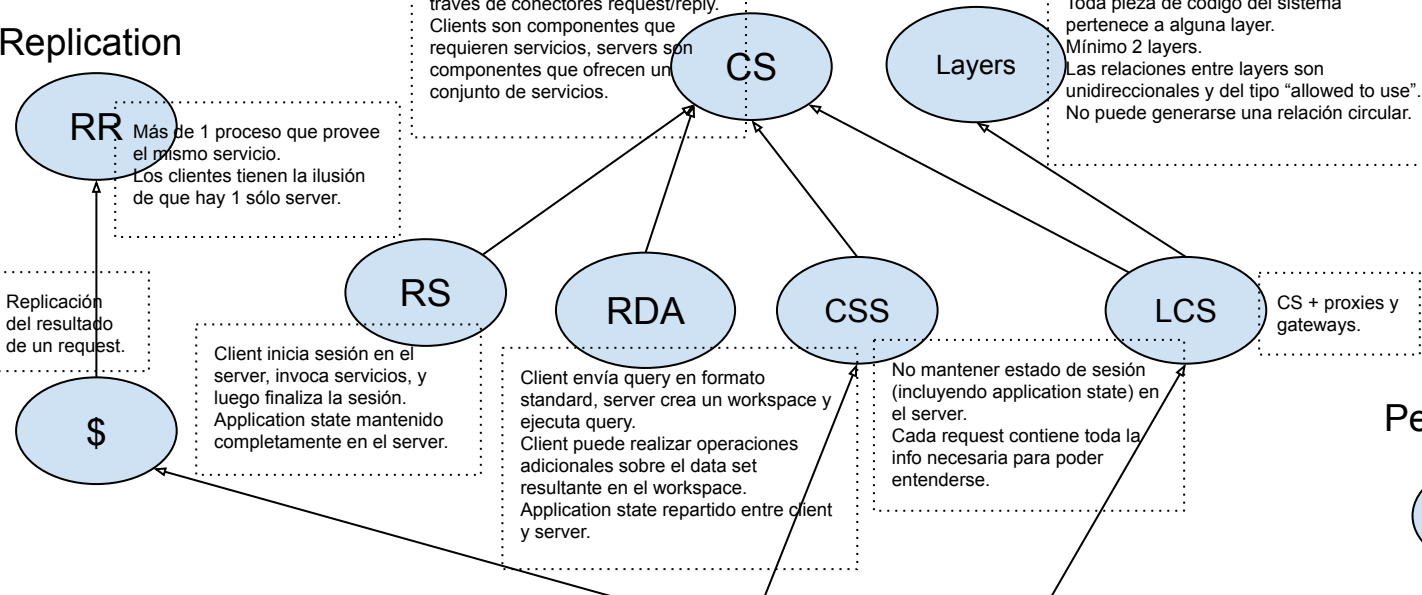


Los estilos de arquitectura,
incluyendo el de la Web

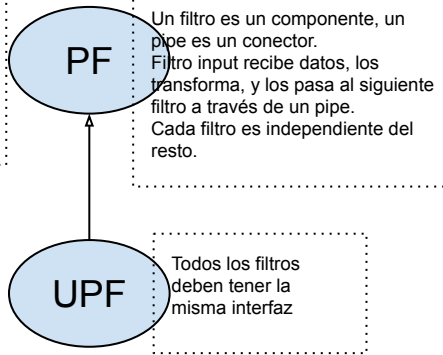
Replication



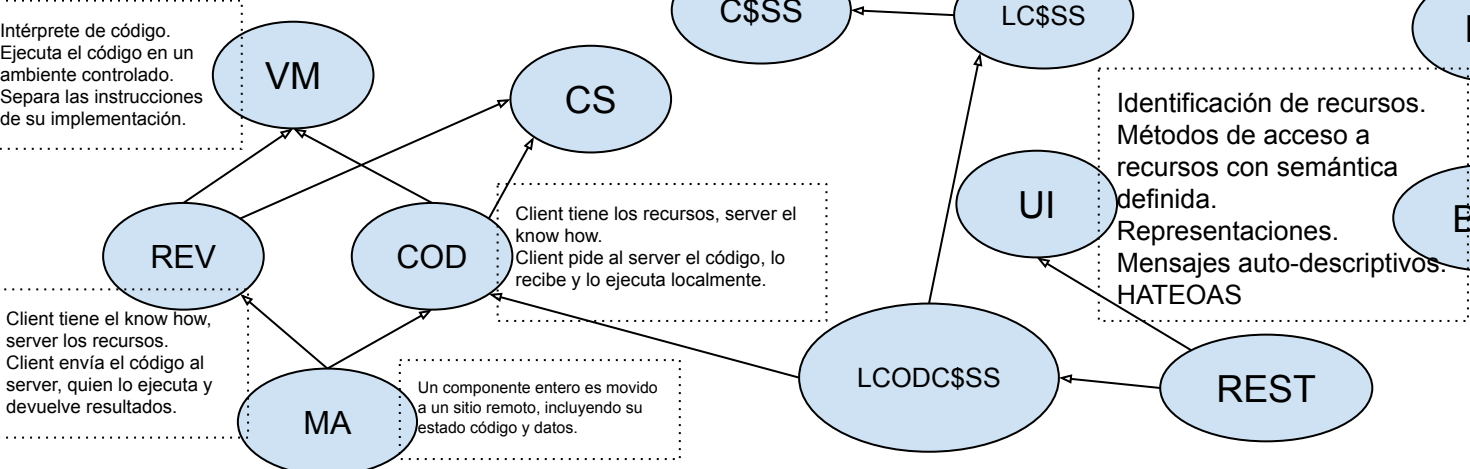
Hierarchical



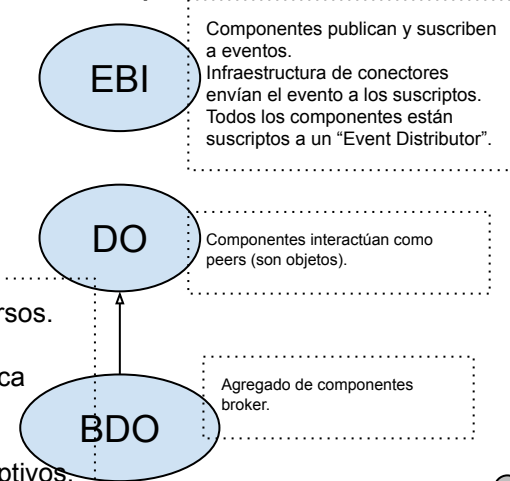
Data flow



Mobile Code



Peer to peer



Legend

○

 Estilo de Arquitectura

→

 "Hereda" tácticas de

⋮

 Tácticas

¿REST APIs?

La frustración de Roy Fielding

Mon
20 Oct
2008

REST APIs must be hypertext-driven

Posted by Roy T. Fielding under software architecture, web architecture

[51] Comments

I am getting frustrated by the number of people calling any HTTP-based interface a REST API. Today's example is the [SocialSite REST API](#). That is RPC. It screams RPC. There is so much coupling on display that it should be given an X rating.

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?



¿Y en qué le estamos errando, Roy?

“A REST API should spend almost all of its descriptive effort in **defining the media type(s) used for representing resources and driving application state**, or in defining extended relation names and/or hypertext-enabled mark-up for existing standard media types.

Any effort spent describing what methods to use on what URIs of interest should be entirely defined within the scope of the processing rules for a media type (and, in most cases, already defined by existing media types). *[Failure here implies that out-of-band information is driving interaction instead of hypertext.]*”

“A REST API **must not define fixed resource names or hierarchies (an obvious coupling of client and server)**. Servers must have the freedom to control their own namespace.

Instead, allow servers to instruct clients on how to construct appropriate URIs, such as is done in HTML forms and URI templates, by defining those instructions within media types and link relations. *[Failure here implies that clients are assuming a resource structure due to out-of band information, such as a domain-specific standard, which is the data-oriented equivalent to RPC’s functional coupling].*”

¿Algo más?

A REST API should be entered with **no prior knowledge beyond the initial URI (bookmark) and set of standardized media types** that are appropriate for the intended audience (i.e., expected to be understood by any client that might use the API).

From that point on, **all application state transitions must be driven by client selection of server-provided choices that are present in the received representations or implied by the user's manipulation of those representations.**

The transitions may be determined (or limited by) the client's knowledge of media types and resource communication mechanisms, both of which may be improved on-the-fly (e.g., code-on-demand).
[Failure here implies that out-of-band information is driving interaction instead of hypertext.]

EL problema

The “Semantic Challenge”

Algunas soluciones parciales

- Domain-Specific design: nuevos media types
 - Ej.: vnd.amundsen.application/maze+xml
- Collection Pattern: application+json, AtomPub
- Hypermedia “puro” (diseños genéricos)
 - HTML
 - HTML Microformats y HTML Microdata
 - Desventaja: Diseñado para el dominio de “human readable documents”
 - HAL: Hypertext Application Language
 - application/hal+json y application/hal+xml
 - Siren
- Profiles (RFC 6906)
 - permiten a los clientes aprender extra app semantics dentro de otro media type (ej HTML)
 - `<link href = "http://microformats.org/wiki/hcard" rel="profile" >`
 - `application/collection+json; profile = "http://www.example.com/profile"`
 - `<div itemscope itemType = "http:// schema.org/ Person" >`
 - Ejemplo de Profile formats: XMDP, ALPS

Qué vimos

- World Wide Web como app sobre Internet
- Historia, requerimientos, y atributos de calidad de la Web
- Arquitectura inicial de la Web
- Agregando tácticas hasta llegar a la Web moderna
- La esencia de la Web moderna es REST
- REST
- Application State
- Hypermedia
- ¿Cómo sería una API REST para clientes máquinas?

¿Consultas?

Feedback:

<http://bit.ly/arq-soft-feedback-clases>

Guillermo Rugilo

guille.rugilo@gmail.com