# Webdevelopment

Lucas Belpaire

May 2019

# 1 For the love of the Web

## 1.1 The Web and universality

### 1.1.1 Origin and aims

**While Berners-Lee was working at CERN he drafted 'Information Management: A Proposal', what was the goal of the paper, what was needed to get to accomplish goal?** The goal of the paper was to facilitate information sharing. Needs: heterogeneity, decentralization, live links, ...

**The web strives to be universal, how does it do this?** Through the independence of many factors.

- Anyone can use the Web, regardless of software or hardware
- Developers are free to innovate, they build for the web and standards provide interoperability.

**What entity standardizes Web technologies?** The World Wide Web Consortium (W3C). The W3C brings together members organizations to create standards. Every members voice counts, preferably through consensus. There are working groups which publish recommendations, which are considered for web standards (CSS, XML, RDF).

**Communication evolved together with the Web. Explain.** The web is democratic, this means that everyone can read and write information. Thus, blogs emerged as a medium to spread thoughts. This made many users transition from consumers to 'prosumers'.

**Education evolved together with the Web. Explain.** Wikipedia started in 2001, soon Massive Open Online courses followed (Khan Academy, Coursera, ...).

**Business evolved together with the Web.** The web made online deliveries possible (Pizza Hut). Many businesses can now exploit the long tail.

### 1.1.2 Technical Foundations

**What is the difference between the Internet and the Web?** The Internet is a communication network between different machines in the world, it is a global communication network interconnecting devices (TCP, IP, DNS). The Web is a layer of interlinked resources accessible through the internet, it is an information space on top of the internet (HTML, HTTP, URL).

**How is Web linking implemented?** Web linking is decentralized, implemented as one-way links embedded in documents. Text and hyperlinks are the essentials inside of HTML (not shared). Styling, media, scripts are the essentials outside of HTML (shareable).

**Opinion: why could the Web succeed?** It was the only hypertext system simple enough to scale the world. ('Individual links are allowed to break so the entire Web does not.')

**What helped accelerate the Web's growth?** It's universality: HTML and HTTP where OS-independent, it provides compatibility at an unprecedented scale.

## 1.2 Opportunities and challenges

### 1.2.1 The web for machines

**Why does the Web's architecture makes it hard to locate specific content. What are solutions?** The Web is decentralized, we cannot possibly know where everything is located. 'Crawlers' from search engines traverse the Web and create a centralized index of content.

**Machines have only limited possibilities on the "human" Web. What is a possible solution?** Web APIs expose functionality to an automated client. A Web API is a programmable interface to a certain Web platform. We can build client-side applications in- or outside the browser that leverage external functionality. Web APIs let machines execute scripts, but not explore and process content. Machines do not "know" what Web content means.

**Tim Berners-Lee and others proposed a vision of intelligent Web agents. Explain this vision** Read paper!

By adding annotations to existing pages, machines can interpret them and do things for us.

A network of knowledge is created by linking different parts together.

Our personal devices will combine data and services on our behalf.

**Linked (Open) Data aims to bootstrap the Semantic Web vision. Why and how?** Why: the early Semantic Web suffered from a chicken-and-egg problem. (Nobody built applications, because there was no data → Nobody published data, because there were no apps).

How: Berners-Lee proposed Linked Data: principles to follow to publish data. When there's data, apps will follow.

### 1.2.2 Threats to universality

**What are the threats to universality?**

- Native Apps
- Centralization

**How are native apps threatening universality?** Native apps essentially undo all progress on device-independence. A generic browser made it possible consume the world's knowledge.

Because of native apps, innovation becomes expensive and exclusive.

**Centralization from multiple angles is threatening the Web. Which angles, and how does it hurt diversity, innovation and choice?**

- Browser vendors
- Search engines
- Platforms (especially social networks): you need a specific account to use the Web.

Developers depend on centralized platforms for data and identity. People lose control of their data and cannot easily switch to other apps. Thus innovation cannot attract locked-in customers.

# 2 Web Architecture & Technologies

**The Web consists of 3 separate, but connected, inventions. Which ones?**

1. URL (Uniform Resource Locator): A URL uniquely identifies a resource.
2. HTTP (HyperText Transfer Protocol): HTTP allows us to retrieve a representation of a resource through a URL.
3. HTML (HyperText Markup Language): An HTML document can represent a resource, and link to other resources through their URL.

## 2.1 Core Web Standards

### 2.1.1 URL

**What is a Web URL?** A Web URL uniquely identifies and locates a resource anywhere in the universe.

A string is a unique identifier if at most one entity corresponds to it.
A string is a unique locator if at most one location corresponds to it.

**What does a well-chosen URL do?** It combines identification and location.

**Which parts does the uniform structure of an HTTP URL contain?** http://host/path?search#fragment

- host: identifies the machine
- path: identifies the resource within the machine
- search: optionally refines the resource
- fragment: optionally identifies a part of the resource

**How does an HTTP URL provide the instructions to obtain a representation of the resource?** http://host/path?search#fragment

1. The client looks up the host's IP address: the client uses DNS for this.
2. The client requests /path?search: the server generates a response in a server-specific way.
3. The client finds #fragment: fragments are defined by the representation format.

### 2.1.2 HTTP

**What is HTTP?** HTTP is a protocol to transfer representations from a server to a client.

HTTP standardizes how clients send a request for a representation of a resource through its URL.

HTTP standardizes how servers reply with a response that can contain a representation.

**After resolving the server's IP address, the client can send an HTTP request. What is the structure of this request?**

- The request starts with a request line: indicates method, request URL path, HTTP version
- The request can contain header fields: including Host, Accept, User-Agent
- The request can optionally contain a body (if the methods allows it)

**HTTP has a limited number of methods. Give 5 widely known methods.**

1. GET: transfer a representation
2. HEAD: transfer only status and headers
3. POST: perform a resource-specific operation
4. PUT: replace all representations
5. DELETE: remove all representations

**When is an HTTP method safe?** An HTTP method is safe if it is read-only. This means that the client does not request a state change of the resource.

**Which HTTP methods are safe?** GET and HEAD.

**When is an HTTP method idempotent?** An HTTP method is idempotent if repetitions don't alter the outcome. The client can thus execute an idempotent request 1 or more times; the result remains the same.

**Which HTTP methods are idempotent?** Safe methods: GET and HEAD, and PUT, DELETE.

**How can one server host multiple websites?** Because the client sends the hostname to the server, one server can host multiple websites. Although the client resolves it to an IP address, the hostname is also sent to the server, which enables the server to pick the right website.

**There is no one-to-one mapping between server IP addresses and domains. What does this enable?** One website can be hosted by multiple servers.

One server can host multiple websites.

**When a server receives a request, it generates a response. What is the structure of this response?**

- The response starts with a status line: indicates HTTP version, status code, reason phrase
- The response can contain header fields: including Content-Type, Content-Length
- The response can optionally contain a body: depending on status code, contains the actual document contents.

**What are the 5 HTTP categories of status codes to indicate how the request was handled?**

1. 100-199: info, the client may continue
2. 200-299: success, request understood & accepted
3. 300-399: redirection, further action needed
4. 400-499: client error, the request cannot be fulfilled
5. 500-599: server error, the server failed to fulfill

### 2.1.3 HTML

**What is HTML?** HTML is a markup language that captures the structure of documents. HTML divides a document into elements, which are indicated by opening and closing tags. Opening tags can have key/value attributes.

An element consists of its tags, attributes, and child nodes (elements and/or text).

The HTML specification restricts what elements can be used and where. A few dozen tags exist for different element types

HTML documents can contain hyperlinks and other hypermedia controls.

HTML documents can embed or use other media documents.

## 2.2 Web Architecture

### 2.2.1 Clients

**Name some different clients the Web supports.**

- Interactive browsers

- Applications
- Crawlers
- Embedded devices and sensors (Web of Things)

**Which core technologies does a client need to support?**

- Networking technologies: TCP/IP, DNS
- The HTTP protocol: many libraries exist, often combined with networking support
- One or more representation formats: not necessarily (only) HTML

**How do browsers offer an interactive environment for general-purpose website consumption?** They render HTML elements as interactive controls. They typically support syles, media, and scripts. Standards ensure consistency. The main differences between browsers are features and preferences.

**How can Web applications perform HTTP requests?** Web applications perform HTTP requests using browser scripting functionality.

A script on a webpage can make HTTP requests. The server typically returns JSON or XML responses, which the script then transforms. Scripts can be triggered automatically or through user actions.

**How can desktop and mobile applications create HTTP requests?** Very similarly to the way web applications create them, but they user their own infrastructure, as they don't have the browser's (in particular, they usually don't set cookies). They typically request JSON (or XML), but HTML is not uncommon.

Opinion: Is there a real need then for a native app?

**What are crawlers?** Crawlers process and/or index webpages, and follow links to others.

Thus they extract, process, and intex text contents. They also analyze some structured annotations (this improves search results). Using links, they can discover other pages.

### 2.2.2 Servers

Important: The HTTP protocol does not attach meaning to URL paths and query strings.

**Web servers exist in many kinds and have many possible implementations. Give some examples.**

- File servers: for static files
- Application Servers: for interactive sites, for editable content (Content Management Systems).
- Proxies: delegate requests to other Web servers.

**What does a static file server do?** A static file server maps HTTP URLs to internal file URLs. A config file usually assigns a root folder per domain. Folders map to possible index files. Custom permissions and rules can be set.

**What does an application server do?** An application server uses server-side code to generate pages on demand. The request is parsed by an application framework, which exposes the URL, method and headers.

Implementors can react to specific URLs or patterns, typically generating responses using templates.

### 2.2.3 Intermediaries

**Give some intermediaries that may exist between a client and a server**

1. Browser
2. Browser cache
3. Provider proxy
4. Cloud cache
5. Reverse proxy
6. App server

**What does HTTP enable, in contrast to many other protocols?** HTTP enables transparent intermediaries.

**What enables HTTP to be transparent?** HTTP can be transparent because of its standardized uniform interface. Caching is possible with headers such as Cache-control and ETag. The intermediary fetches the requested item and keeps it in the cache until it expires.

Repeated requests for the item are served from cache.

**How do standardized method semantics make the caching work?** Repeated GET requests can be cached, because GET is 'safe'.

If POST or PUT are used on a resource, a subsequent GET must not be read from cache.

**What roles can intermediaries play in an HTTP interaction?**

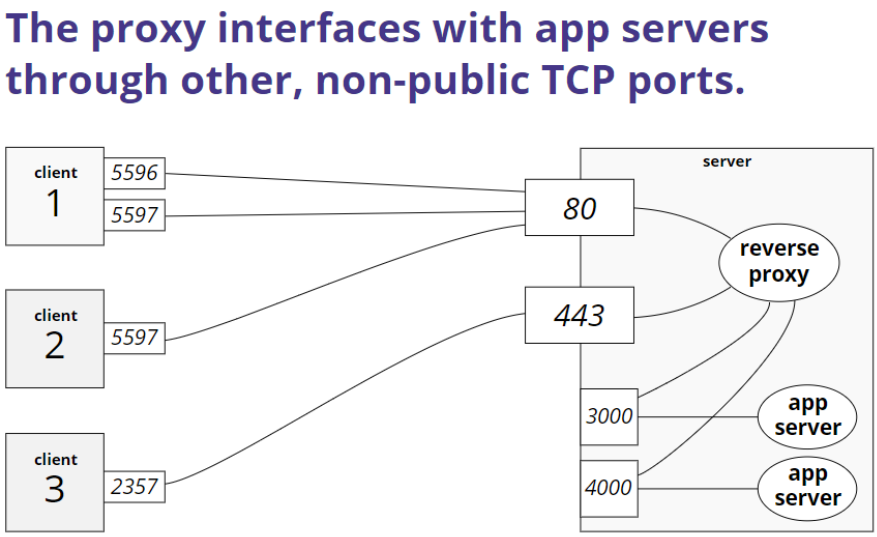
- Caching: to improve performance and availability
- Security: to handle authorization and authentication
- Routing: to redirect toward the right server
- Load balancing: to distribute load over servers
- Anonymizing: to bypass identification or logging

**When are proxies forward and when reverse?** This depends on their position in the network.

Forward proxy: in the network of the client, typically used for caching, possibly for security/anonymity purposes.

Reverse proxy: in the network of the server, typically used for caching, also for routing, hiding/abstracting remote architecture.

**A device can list on only one TCP port 80, how can we combine application servers?** Configure servers to run on internal ports (such as 3000, 4000, 5000) instead of port 80. Run a reverse proxy on port 80, which dispatches requests to other servers.




## 2.3 Beyond the Core

### 2.3.1 HTTPS

**HTTP nodes send plaintext over TCP, which means intermediaries can read it. What are the consequences?** The privacy of your requests

is not guaranteed. The privacy of server responses is not guaranteed. The integrity of server responses is not guaranteed.

**What is HTTPS?** HTTPS is a secure extension of the HTTP protocol. Simply use HTTP over TLS (Transport Layer Security) precisely as you would use HTTP over TCP.

TLS is a cryptographic protocol for Internet communication. It relies on asymmetric key encryption.

**In which different parts of the network can HTTPS be applied?** Consider the use case in which an application server lives behind a reverse proxy.

1. Only the proxy could use HTTPS: Typically, server and proxy operate in a trusted network. Client-proxy communications are secured.
2. The server and the proxy could use HTTPS: These are two different encrypted communications. No proxy can sit 'in between' an encrypted communication!

**What does setting up HTTPS involve?**

- Request SSL certificate from a certificate authority
- Configure your Web server with the certificate (private key stored safely, public key shared with clients).
- When the certificate expires, request a new one.

**Why are HTTPS URLs different?** So clients know what protocol to use.

### 2.3.2 HTTP/2

**What is HTTP/2?** HTTP/2 is an update of the HTTP protocol that addresses a couple of key bottlenecks.

**Why is an update necessary?** Webpages consist of more and more items that increasingly become larger.

Several limitations of HTTP have a negative impact on load times.

**What limitations of HTTP have a negative impact on load times?**

- TCP connections are not used optimally: connections are costly due to the three-way handshake, limit the connections per host to avoid exhaustion.
- HTTP is very latency-sensitive.
- Sequential handling of requests blocks pipelines: HTTP pipelining allows issuing multiple requests on a single TCP connection without waiting for responses to arrive.

**What are some workarounds websites use to circumvent HTTP's limitations?**

- Inlining: Embed scripts, images, and/or styles inside HTML documents instead of linking them to reduce number of items.
- Concatenation: Combine several scripts, images, or styles into a single file.
- Sharding: Distribute resources across different domains to bypass connection restrictions.

**What are HTTP/2's important goals for improvement?**

- Reduce sensitivity to latency.
- Eliminate the need for multiple connections.
- Fix HTTP pipelining and line blocking.
- Maintain interfaces, content, and URLs.
- Be backward compatible for HTTP 1.1 clients.

**HTTP/2 is a binary protocol that sends frames over multiplexed streams. Why switch to binary? What does multiplexing ensures?** Switching to binary makes framing easier. Multiplexing ensures objects don't block each other (frames from many streams flow over the same connection).

Note: additional compression reduces overhead.

**HTTP/2 always goes over TLS in practice, even though the standard also allows TCP. Why?**

- With regular TCP on port 80, clients would need to upgrade from HTTP 1.x to HTTP/2 (overhead necessary for backwards compatibility).
- With TLS, Application-Layer Protocol Negotiation is embedded within the TLS handshake (no protocol latency).
- FireFox and Chrome will only support TLS.

### 2.3.3 Security

**Why is protocol-level security insufficient to provide application-level security?** Protocol-level security only guarantees privacy and integrity from one endpoint to another. But the Web evolved from a document system to a distributed application platform. Thus applications themselves also need mechanisms to secure information.

**What are 3 common attack types?**

- Code injection: executing malicious scripts on the server.
- Cross-site scripting: executing malicious scripts on the client.
- Cross-origin access: accessing information from other websites

**Give attack, cause and defence for code injection.** With code injection, clients send input designed to execute on the server. Example: SQL injection.

- Attack: executable code passed in an URL or POST body parameter.
- Cause: improper input validation by the server.
- Defense: never trust raw client input, validate input ranges, escape values before passing to scripts.

relevant xkcd

**Why do servers always need to (re)-validate?** Because client-side validation only offers usability. You can bypass by changing the HTML at runtime or constructing the HTTP request in another way.

**Give atack, cause and defense for Cross-Site Scripting (XSS).** With Cross-Site Scripting (XSS), a client script steals information from a webpage.

- Attack: trick another site into executing code in its own space, exposing or changing user data.
- Causes: improper input validation, trusting third-party content, social engineering.
- Defense: validate input, add token field on forms.

**Give attack, cause and defense for cross-origin access.** With cross-origin access, a page requests third-party resources through Javascript.

- Attack: request resources from another site with JavaScript and read their contents.
- Causes: the existence of XMLHttpRequest being logged in another website.
- Defense: browsers block cross-origin requests by default.

**Why are cross-origin requests only a problem within browsers?** Nothing happens when any other script or app requests 'https://mybank.com/onlinebanking/.' (there are no cookies). When the browser requests the same page, the page might contain personal information (the browser maintains cookies). As a result, browsers block cross-origin requests.

**What does Cross-Origin Resource Sharing (CORS) enable?** CORS enables others to access your pages. Activate CORS to enable Web APIs access from within other browser-based Web applications. The browser adds an Origin header to requests. If the server allows requests from that origin, it adds Acces-Control-Allow-Origin.

# 3 Web APIs

**What is an affordance?** An affordance is a property of an object that explains and aids its usage. In software, affordances strongly affect usage. (Many people use Word, few can use Vim). Affordances are of crucial importance on the Web.

**What is a Web API?** A Web API is a server-side HTTP interface exposing data and functionality to clients.

If well designed, a Web API provides the affordances for humans and/or machines to interact with it. The world of Web APIs and technologies is highly heterogeneous (While the human Web reuses many technologies, Web APIs suffer from a high rate of reinvention).

## 3.1 The REST architectural style

### 3.1.1 The REST constraints

**The REST architectural style consists of 5 types of mandatory constraints. Name them.**

1. Client-server constraints
2. Statelessness constraint
3. Cache constraints
4. Layered system constraints
5. Uniform interface constraints

**Explain the client-server constraints.** The server only listens for requests, clients send requests via a connector.

This realizes a separation of concerns. Separating user interface from data storage $\rightarrow$ increases scalability by simplifying the server, improves portability of the interface across platforms, enables independent evolution of client and server.

**Explain the stateless constraint.** REST architecture constrain communication to be stateless

Each client request must contain all information necessary for the server to understand it (the server doesn't remember anything from previous requests). Stateless requests can be interpreted independently from each other.

Note: statelessness applies to application state. Servers still maintain resource state.

Application state captures the position of a client in the interaction, and differs per client.

Resource state captures properties of data entities on the server, and are the same for all clients.

**What are the advantages and disadvantages of statelessness?** Advantages:

- Visibility: Intermediaries understand each request without having to know any others.
- Reliability: Recovery from partial failures is easier because no request history is needed.
- Scalability: No data must be kept between requests. Each request can be handled by a different, independent server.

Disadvantages:

- Bandwidth: Requests might need to repeat information that was sent previously but not kept.
- Control: Clients become responsible to transition consistently from one state into another. They cannot necessarily be trusted to make correct and allowed transitions.

**Explain cache constraints.** A server response indicates explicitly whether or not it is cacheable.

Advantages: caching improves network efficiency, scalability, and user-perceived performance.

Disadvantages: reliability can be reduced if cached data is stale.

**Explain the layered system constraints.** The layered system constraints allow for flexible hierarchical layers. Intermediaries can be inserted transparently, shielding off complexity and inner details.

Advantages: layering increases flexibility.

Disadvantages: layering can introduce overhead and latency.

**The uniform interface constraints are unique to the REST architectural style. Which 4 uniform constraints does REST introduce?**

1. Identification of resources.
2. Resource manipulation through representations.
3. Self-descriptive messages.
4. Hypermedia as the engine of application state.

**What is a resource? (identification of resources)** A resource is the main unit of information. Anything that can be named is a resource.

Note: Information resources can have zero or more representations. Depending on the needs and capability of a client, a resource can be represented differently.

**What is an identifier? (identification of resources)** A valid identifier corresponds to a unique resource. But any resource can have multiple identifiers.

**A resource is a conceptual relationship. Explain.** The value of a resource may change over time. A resource is a function of time that associates a definition with a value.

This allows late binding. Refer to the concept, whether or not it exists and is final.

**Resource manipulation through representations. Explain.** Information resources can have zero or more representations. Depending on the needs and capabilities of a client, a resource can be represented differently.

**Self-descriptive messages. Explain.** Each message in a REST interaction should be self-descriptive. This means that:

1. Intermediaries can interpret messages.
   (a) Statelessness and explicit cacheability.
   (b) Limited number of standardized operations.
2. Response (and request) bodies are expressed in explicit, agreed-upon media types.

**Hypermedia as the engine of application state. Explain.** The interaction is driven by hypermedia controls inside of responses.

Responses contain hypermedia controls such as hyperlinks and forms. They afford the next steps a client can take (indicate the possible steps and explains how to take them). Thus, rather than executing a pre-defined script, the client follows the server's lead.

Note: hypermedia defines REST interactions, minimizing the client-server contract. Client and server can evolve independently.

### 3.1.2 REST applied to the Web.

**How does the Web implement the client-server constraints?** The Web consists of clients and servers. Clients make HTTP requests and servers listen and respond to HTTP requests.

Note: The 'client' and 'server' labels are roles, and relative to the interaction.

**How does the Web implement the stateless constraint?** The HTTP protocol is designed in a stateless way.

Servers don't remember clients between requests. Each HTTP request must contain all headers, even those that remain the same.

The client-server relation only exists during the request and processing of the response.

**How does the Web implement the cache constraints?** HTTP realizes caching with intermediaries and explicit caching headers.

HTTP responses explicitly indicate cacheability.

HTTP methods semantics define caching influences (GET does not change a resource, POST can change a resource).

**How does the Web implement layered system constraints?** HTTP allows transparent insertion of layers of proxies.

Reverse proxies act as a gateway to application servers. There is no stateful client-server connection, so intermediaries can transparently answer.

HTTP can wrap legacy systems.

**How does the Web implement the uniform interface constraints? Does it support all constraints?** The uniform interface is implemented by a combination of URL, HTTP, and HTML.

The Web supports all 4 constraints:

1. Identification of resources.
2. Resource manipulation through representations.
3. Self-descriptive messages.
4. Hypermedia as the engine of application state.

**How does the Web implement the identification of resources?** Web resources are conceptual relations uniquely identified by HTTP URLs.

An HTTP URL points to at most one resource. The concept pointed to by an URL shouldn't change. The value and representations retrieved when looking up an URL might change over time.

**How does the Web implement 'resource manipulation through representation'?** Clients obtain different representations through HTTP content negotiation.

If a URL identifies a resource, clients need a mechanism to obtain a representation. With content negotiation, a client and server agree on the best mutually understandable media type. The client indicates its preferences with Accept, the server advertises its choice with Content-type.

**How does the Web implement self-descriptive messages?** Well-defined HTTP methods & headers and media types enable self-description.

HTTP provides a limited number of methods (this enables the intermediaries to know the semantics of all methods). HTTP provides an extensible set of headers. The Content-Type header details the media type.

**How does the Web implement 'hypermedia as the engine of the application state'?** HTML (and other hypermedia) documents can contain hypermedia controls.

In HTML documents, links can be marked up. As a consequence, users almost never have to touch the address bar.

## 3.2 Sustainability of information

### 3.2.1 Web APIs with and without REST

**Opinion: your website is the API. Explain** All you need are additional representations.

- A resource should relate an URL to a concept.
- A resource can have many representations under a single URL.
- Each message should be self-descriptive and stand on its own.
- The interaction should be driven by hypermedia.

See slides for examples.

**What principle do servers offering a hypermedia API enable?** They enable the follow your nose principle. The term 'hypermedia API' is sometimes used to indicate APIs that follow all REST principles.

A client should be able to follow its nose around the API from any starting point. This requires a pre-coded understanding of the hypermedia content type and its links/form types.

### 3.2.2 Designing Web APIs for the long term

**What is an important reason to implement a Web API that follows REST?** REST architectures aim for sustainability, offering constants in a changing world. Follow REST to design for the long term.

## 3.3 Web API technologies

### 3.3.1 Non-REST Web services

**What is SOAP?** The Simple Object Access Protocol (SOAP) transfers XML messages on top of HTTP.

A SOAP message consists of 3 main XML elements.

1. Envelope: root with (optional) header and body.
2. Header: modular extension mechanism.
3. Body: details of a method invocation.

**What does SOAP provide?** SOAP provides language-independent programming over a network.

SOAP enables fast cross-language development. SOAP does not inherit the Web's benefits (no caching, no visibility, no evolvability).

### 3.3.2 Web API descriptions

**What do Web API descriptions capture?** Web API descriptions capture machine interpretable details about an API.

**What are the uses of Web API descriptions?** There exist 3 broad uses for descriptions.

1. In-band: resource and hypermedia control details.
2. Interface: structural properties.
3. Functional: effect and/or purpose characterization.

**Wat is WDSL?** The Web Services Description Language (WDSL) describes SOAP service structure.

A WSDL description is an XML document that lists a service's methods and parameters.

A WSDL description and SOAP wrapper can be generated automatically from program code.

Code to interact with the described SOAP service can be generated from the WSDL.

**What is OWL-S?** OWL-S (Semantic Markup for Web Services) enriches WSDL with functional aspects.

An OWL-S description captures the semantics of a single SOAP operation using RDF.

An OWL-S description consists of three parts.

1. Profile: high-level functionality for discovery.
2. Grounding: connection to SOAP parameters.
3. Model: in-depth functional relations and constraints.

OWL-S is very powerful for RPC (remote procedure call), but remains unused.

**What is WADL?** The Web Application Description Language is the WSDL of non-SOAP Web APIs.

WADL descriptions detail structural properties (support for XML and JSON). A WADL description is an XML document that details HTTP methods and parameters, as well as representation structure.

WADL enables automated code generation (also hard-coded).

**On what do the most recent Web API description formats focus?** Most recent Web API descriptions formats focus purely on developer support.

Opinion: most formats are not fundamentally different: they are tools, pick the one you like the most.

**Opinion: structural descriptions boost development but don't fundamentally change Web APIs.** They encourage uncontrolled growth but hamper a sustainable Web API ecosystem.

They promote thight coupling.

In essence, they have little to do with the Web.

**How do hypermedia-based formats handle descriptions?** Hypermedia-based formats place descriptions inside of the message.

JSON itself doesn not support links or forms. The Hypertext Application Language (HAL) defines links on top of JSON.

**What is Hydra?** Hydra is an RDF vocabulary to describe hypermedia controls and API structure.

Hydra contains hypermedia building blocks to enhance representations with links and forms. In addition to the structure of a REST Web API, Hydra can also describe the semantics of resources.

# 4 The Semantic Web & Linked Data

## 4.1 Linked Data

### 4.1.1 The Linked Data principles

**What are the 4 principles Tim Berners-Lee proposed to publish Linked Data?**

1. Use URIs as names for things.
2. Use HTTP URIs so people can look up those names.
3. When someone looks up a URI, provide useful information using the standards.
4. Include links to other things, so people can discover more.

**Opinion: The Linked Data principles resemble the REST uniform interface constraints. Explain.**

- Linked Data Principles → REST uniform interface constraints
- Use URIs as names for things → Uniquely identify resources
- Use HTTP URIs so people can look up those names → Provide representations of those resources to clients
- When someone looks up a URI, provide useful information using the standards → Each message you send should be self-describing
- Include links to other things, so people can discover more → hypermedia controls must afford next steps

**Why use URIs? What do they do?** Information & non-information resources should be uniquely identifiable. using HTTP URIs ensures that anybody can look up the resource.

An HTTP URI of a resource can be dereferenced: use an HTTP client to retrieve a representation. This relies on the double role of an HTTP URI as identifier and locator.

Note: dereferencing is a core principle of Linked Data (the act of retrieving a representation of a resource identified by a URI is known as dereferencing that URI).

**Dereferencing a URI should lead to useful information about that resource. What is useful information?** Useful means the information is available using the standard technologies (RDF, SPARQL). Useful also means the information provides explanations and/or context for the resource.

Define the resource in terms of concepts the client already knows or can look up.

**How do we create a Web of data?** By including links to other resources.

- Links connect a resource to known concepts.
- Links give meaning to data.
- Links allow exploration of related data.

### 4.1.2 Linked Data on the Web

**An immense amount of Linked Data is available on the Web for reuse. What does this mean on the structural level and on the content level?** On the structural level, hundreds of vocabularies (like foaf) can provide the building blocks to model your data.

On the content level, thousands of datasets provide identifiers and data of individuals.

**What is the open-world assumption?** No Linked Data set is ever complete.

Relational databases use highly rigid structures. They strive for complete data. But with Linked Data , no source has all of the truth. Other sources might have more data on a subject. The absence of a fact does not imply its falsehood. A fact has 2 possible states: true and unknown.

**What are the 'Dublin Core terms'?** The Dublin Core terms are a set of 15 common metadata properties.

Each property is generic, and hence applicable in many cases. Thus many applications use the Dublin Core terms.

**What is Schema.org?** Schema.org is a single vocabulary that covers many different fields.

Created and maintained by major search engines, it mainly provides discovery data for search.

Its concepts are defined rather loosely. Advantage: this makes it flexible to use for developers. Disadvantage: machines cannot derive much knowledge from it.

**What is Open Graph?** Open Graph is similar to Schema.org, and mainly used for Facebook integration.

## 4.2 The Semantic Web

### 4.2.1 RDF

**What is RDF?** The Resource Description Framework is a model for data interchange on the Web.

RDF is a standardized way to represent Linked Data. It has different concrete syntaxes.

The RDF model defines RDF datasets.

- An RDF dataset has a default graph and $\geq 0$ named graphs.
- An RDF graph is a set of RDF triples.
- An RDF triple consists of a subject, predicate, and object.

**What are the 3 types of RDF terms?**

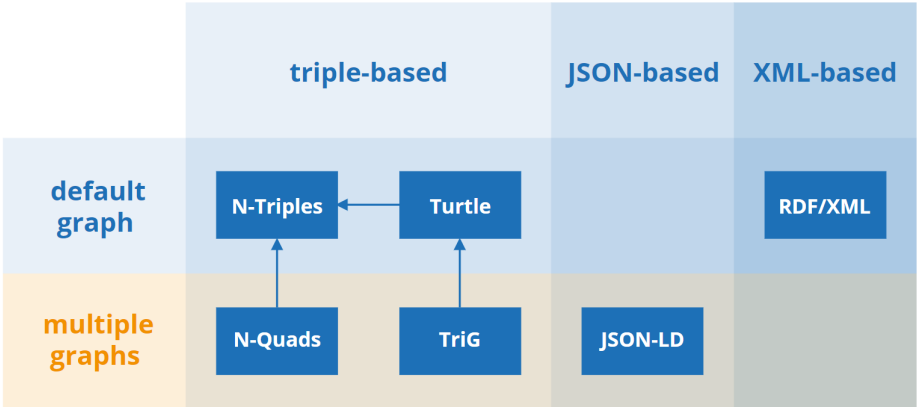
1. Named node: a resource, identified by an IRI. for subjects, predicates, objects.
2. Blank node: an unnamed resource. for subjects and objects.
3. Literal: a value, with a datatype (IRI) or language. for objects only.

**An RDF dataset has one default graph and zero or more named graphs. What is a default graph? What is a named graph?** The default graph is an RDF graph, an RDF graph is a set of triples. The default graph can be empty.

A named graph is an RDF graph identified by an IRI.

Note: not all concrete syntaxes support named graphs, but all syntaxes support the default graph.

**How do you choose the right RDF syntax?** You choose the right RDF syntax based on graph support and client technology.




**What does RDFa allow?** RDFa allows extending generic HTML and XML documents with RDF triples. Instead of performing content negotiation, RDFa embeds triples inside other formats.

RDFa extends existing markup with new attributes: property, resource, content.

### 4.2.2 RDFS

**What is RDFS?** RDF Schema is an RDF vocabulary to model RDF vocabularies. RDFS defines classes, properties, and datatypes that are used to define vocabularies. RDFS defines concepts in two namespaces (rdf, rdfs).

**What are ontologies?** Practitioners in the RDF wordl often refer to vocabularies as ontologies. Strictly speaking, a vocabulary is a set of words; an ontology a set of concepts and their relations. The W3C states that there is 'no clear division'.

**What does RDFS do?** RDFS defines the basic building blocks to construct RDF vocabularies.

- Describing resources
- Describing classes
- Describing properties

**What is a functional property?** A functional property restricts the objects for a given subject to be identical.

**What is rdfs:label?** rdfs:label is a property that gives a human-readable name to a resource. Resources typically have $\leq 1$ label per language. Label is not a functional property.

**What is rdfs:comment?** rdfs:comment is a property that clarifies human-readable meaning and usage. Resources typtically have $\leq 1$ comment per language. rdfs:comment is not a functional property.

**What is rdfs:seeAlso?** rdfs:seeAlso is a property to express some link between two resources.

Opinion: the meaning of this property is rather vague, essentially, all regual Web links have type rdfs:seeAlso.

**What is rdf:type?** rdf:type is a property stating that a resource is an instant of a class. Resources can (and do) have multiple classes.

**What is rdfs:Resource?** rdfs:Resource is a class of which everything is an instance.

**What is rdfs:Class?** rdfs:Class is a class for resources that conceptually define a set of things.

**What is rdf:Property?** rdf:Property is a class for resources that can be used as triple predicates.

**What is rdfs:Literal?** rdfs:Literal is a class for resources that have literal value.

**What is rdfs:subClassOf?** rdfs:subClassOf is a property stating that all members of a class belong to another.

**What is rdfs:domain?** rdfs:domain is a property that states the class of possible subjects of a property. Properties can have multiple domain restrictions and subjects need to satisfy all of them!

**What is rdfs:range?** rdfs:range is a property that states the class of possible objects of a property. Properties can have multiple range restrictions and objects need to satisfy all of them!

**What is rdfs:subPropertyOf?** rdfs:subPropertyOf is a property stating a property is more specific than another. If the subproperty holds for a subject and object, the less specific property also holds.

Note: knowledge of RDFS will help you understand most vocabularies.

#### 4.2.3 OWL

**What is OWL?** The Web Ontology Language (OWL) provides concepts for detailed ontologies.

RDFS captures basic ontological relations, but lacks several common and important concepts. OWL extends RDFS with advanced concepts. RDFS and OWL are used side by side.

OWL defines additional constraints for individuals, properties, and classes.

OWL defines its own version of resources and classes. The class of everything is owl:Thing (similar to rdfs:Resource), the class of classes is owl:Class (subclass of rdfs:Class).

**Can you assume that 2 different IRIs necessarily point to different resources?** No. owl:sameAs indicates two resources are the same. owl:differentFrom indicates two resources differ.

**Explain ow:DataTypeProperty.** Properties taking only literal values as object are instances of owl:DataTypeProperty.

**Explain owl:ObjectProperty.** Properties taking only non-literal values as object are instances of owl:ObjectProperty.

**Explain owl:inverseOf.** Inverse properties express a triple in the opposite direction. One property is the owl:inverseOf another if it asserts the same relation from object to subject. Why is this needed? Ontologists typically pick one property direction. Different ontologies might choose different directions, owl:inverseOf allows to connect such properties.

**What is a functional property?** A functional property restricts the objects for a given subject to be identical. If any subject can at most have one unique value for some property, it's an owl:FunctionalProperty.

**Does OWL contain properties for symmetry, reflexivity, and transitivity?** yes, owl:(As)SymmetricProperty, owl:(Ir)ReflexiveProperty, owl:TransistiveProperty.

## 4.3 Querying and Reasoning

### 4.3.1 SPARQL

**What is SPARQL?** SPARQL is a query language. Select specific data from an RDF dataset. Insert, change or delete data in an RDF dataset.

**What is the SPARQL protocol?** The SPARQL protocol is a Web API definition for querying in the SPARQL language over HTTP. A SPARQL endpoint executes SPARQL queries sent by clients through HTTP, and replies with their results.

**The SPARQL language defines forms a query can take. What are the 4 read-only query forms?**

- SELECT: find values that satisfy conditions
- CONSTRUCT: create triples that satisfy conditions
- ASK: check whether data exists.
- DESCRIBE: show information about a resource.

**What is a Basic Graph Pattern (BGP)?** BGP is the main building block of a SPARQL query. It is a set of triple patterns. Their syntax is a superset of Turtle.

**What is a triple pattern?** A triple pattern is a triple in which each of the components can be a variable. Variables start with a question mark (?name).

**What is a solution mapping?** A solution mapping is a mapping from a set of variables to a set of RDF terms. We use the term 'solution' where it is clear.

A SPARQL query engine finds solution mappings. Variables and blank nodes are mapped to IRIs, blank nodes, or literals according to dataset triples.

**In addition to BGPs, SPARQL queries can contain modifiers. Give 4 modifiers.**

1. LIMIT: only return the first n results.
2. OPTIONAL: specifies a left join.
3. FILTER: selects based on an expression.
4. ORDER BY: sorts results based on an expression.

**What is the purpose of the SPARQL protocol?** Sending queries and receiving results. The server is typically an RDF database (triplestore) with a SPARQL engine. The client sends a query using a URI template, the server replies in a standardized format (XML, JSON, CSV/TSV, RDF for CONSTRUCT/DESCRIBE).

### 4.3.2 RDFS, OWL, and N3 inferencing

**Explain the reasoning of the Semantic Web.** Semantic Web reasoning is an agent's ability to verify and discover facts.

Linked Data provides a body of knowledge. Query engines let clients select specific facts. Reasoning allows clients to combine knowledge from different sources and draw conclusions.

**Some reasoners are tailored to a task, others can/need to be extended. What is the difference?** Tailored to a task: reasoner with built-in knowledge. Can tackle a problem withouth configuration. Can have internal optimizations for certain cases.

Need to be extended: reasoner without built-in knowledge. Can be extended with inference (=a conclusion reached on the basis of evidence and reasoning. ) steps. Can explain in-depth why a step was taken.

**What is the difference between RDFS, OWLS reasoners and rule-based reasoners?** RDFS reasoners can make entailments (=deductions) based on the RDFS semantics. OWL reasoners can make entailments based on the OWL semantics. For both reasoners the you load the data together with its vocabularies. The reasoning is built-in, but not your vocabulary.

Rule-based reasoners allow you to choose and define your own rules. The internal knowledge is limited to rule evaluation and sometimes built-in functions (math, dates, ...). Rules that implement specific ontological concepts are often available and reusable.

**What are some examples of rule languages and syntaxes?** Notation3, RIF, ...

**What is Notation3 (N3)?** N3 is a rule-based language defined as a superset of Turtle. N3 adds support for additional constructs (variables, formulas, implications). There are several N3 reasoners (cwm, EYE). Common RDFS and OWL concepts exist as N3 rules.

# 5 Re-decentralizing the Web

## 5.1 The pendulum of (de-)centralization

### 5.1.1 Broader perspective

**Give an historical overview of the pendulum of (de-)centralization.**

- Started centralized: large mainframes provided computational power for an entire organization.
- Personal computing decentralized computational power.
- Even when the internet became popular, data and CPU remained decentralized. Desktop software remained dominant.
- The software-as-service model brought storage and processing to the "cloud". Centralized software platforms reduce friction.

### 5.1.2 On the Web

**Give the three wars that threatened universality.**

1. 1990: browser wars.
2. 2000: search engine wars.
3. 2010: the platform wars.

**Our data has become centralized in a handful of Web platforms. Why is this bad?** This has far-reaching consequences for privacy. The Web's university is visibly threatened ("sign in with facebook to see this content").

**What are the three challenges for the web that Berners-Lee listed?**

1. We need to regain control over our personal data.
2. We need to reduce the spread of misinformation.
3. We need transparency and understanding of political advertising.

**What do these challenges indicate?** A loss of control/agency.

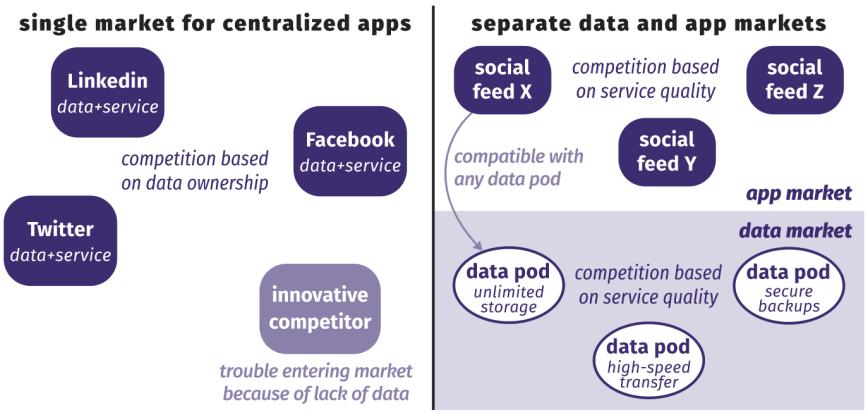
## 5.2 The Solid ecosystem

**What is Solid?** Solid is a way of building Web apps that let people keep control of their data. Solid combines existing W3C standards to define how agents should interact (it builds on top of the existing Web). Like the Web, Solid is for everyone.

**What does Solid aim to do?** Solid aims to restore choice by separating data from apps.

Typical platforms nowadays store data inseparably from an application. By separating data from apps, we create independent choices.

**How does separating app and storage competition drive permissionless innovation?** See figure.




**What are Solid servers?** A Solid server acts as a data pod that stores and guards your data. It is essentially a website with data.

**What are Solid clients?** Solid clients are browser or native apps that read from or write to your data pod.

People give read and/or write permissions for specific pieces of data to: apps, other people, automated agents, ...

Apps deliver an integrated experience: instead of displaying individual webpages or data items, apps interleave data from multiple sources.

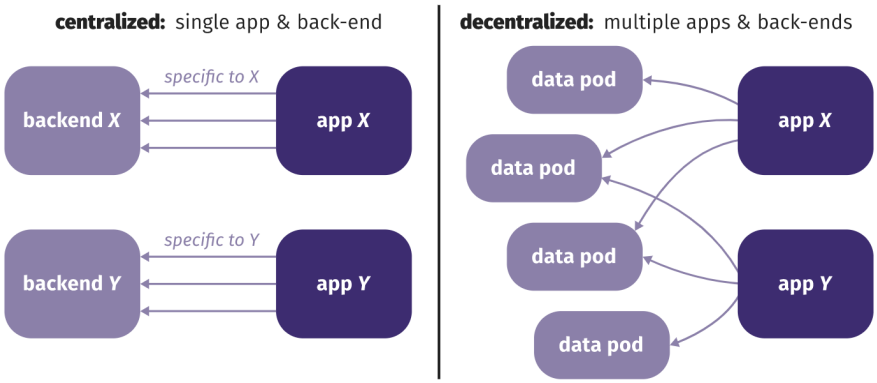
**How is client-server communication governed?** Client-server communication is governed by the Solid specifications.

The specifications are a (currently unstandardized) set of documents on GitHub. They essentially demand compliance with a couple of existing standards (HTTP, LDP, LDN). They suggest specific RDF vocabularies.

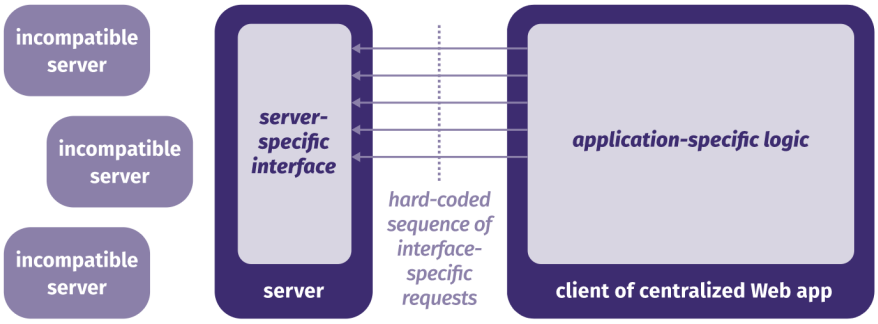
## 5.3 Challenges for the future

### 5.3.1 Decentralized application architecture

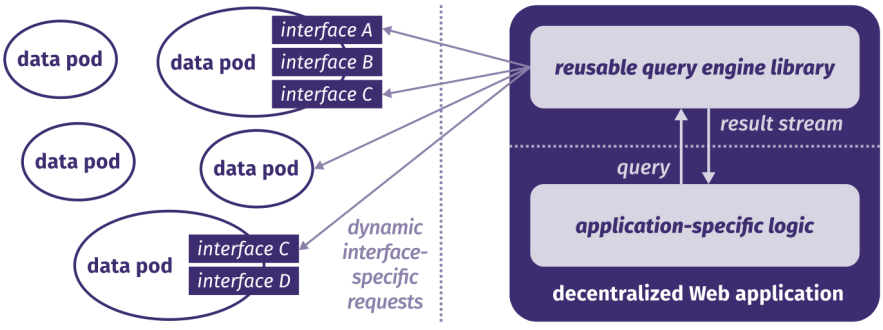
**What is the difference in back-end for centralized and decentralized apps?** See figure.




**What is the difference in building APIs for centralized and decentralized apps?** Centralized:

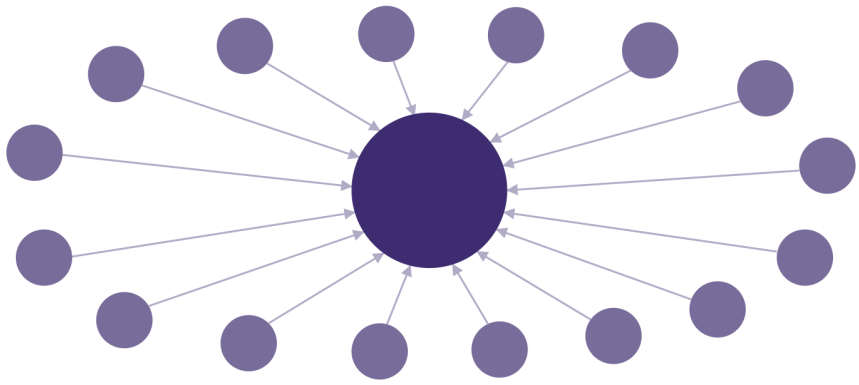



When clients do not bind to HTTP requests, APIs can evolve independently of app logic:

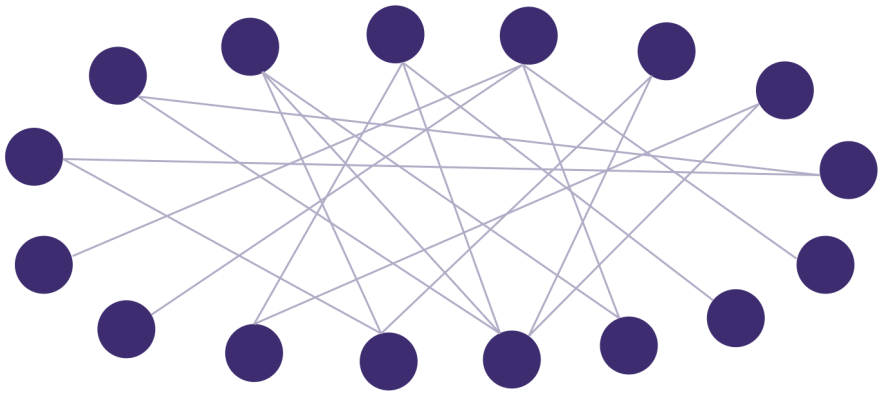


**What is an aggregator?** A website or program that collects related items of content and displays them or links to them.

**What is the different roll for the aggregators in a centralized and decentralized network?** Current networks are centered around the aggregator (centralized).



But in a decentralized network nodes need to become the source of truth. Aggregators serve as a crucial but transparent layer in the network. Their main responsiblity becomes fostering a network between nodes.



**What is the paradox of freedom? Link it to decentralization.** You can only be free if you follow rules.

Decentralization means making your own choices, but unless we agree on some basic things, no one will see the result of our choices.

Agreement can be layered.

### 5.3.2 Linked Data developer experience

**How are interoperability challenges in Solid solved?** Through Linked Data in RDF.

**If we all store our own data, how can we connect it to others' data?** With JSON-LD, every piece of data can link to any other piece of data.

**How can apps share data, without too many prior agreements?** Data shapes and their semantics enable layered compatibility.

**How do we integrate data from multiple data pods?** Different source data can be concatenated (=link (things) together in a chain or series).

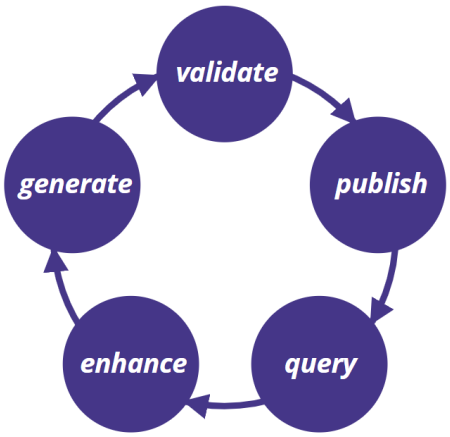
**What is the most crucial factor for Solid success?** The developer experience. Front-end developers build the apps people see. Together with UX designers they bring Solid to people.

**What does LDflex do?** LDflex exposes Linked Data's flexibility without RDF's complexity. LDflex is a domain-specific language for Javascript.

# 6 Linked Data Publishing

## 6.1 Linked Data Life Cycle.

**Give the 'simple' Linked Data life cycle.** See figure.




### 6.1.1 Generate

**What happens in the generation step?** Generation is the step in which we convert non-RDF data to RDF.

**How are most representations on the Web generated?** They are generated by templates. The data resides in a back-end database. The front-end Web application translates database entries via templates into representations (HTML, JSON, ...). The data model from the database easily maps to the target representation.

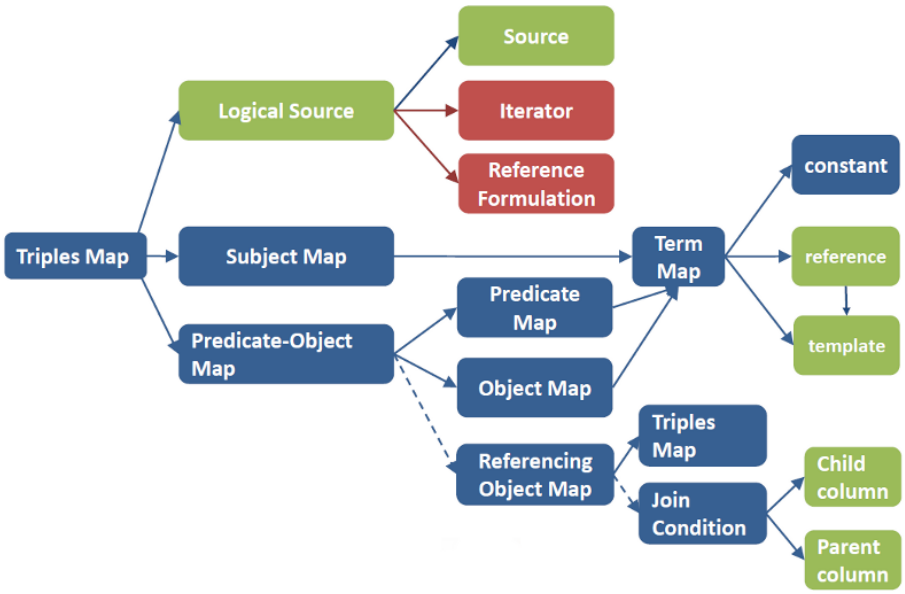
**How can Linked Data be generated in batch?** Through a mapping process:

- A mapping processor takes input sources and a mapping file as input.
- The mapping file explains how input data should be converted to the RDF model.
- Different processors have different features: source type support, interlinking together with mapping, ...

**How can mapping be performed for specific datasets?** By using ad-hoc scripts.

You write custom code to handle your case. Thus the short-term costs might be low, but long-term maintenance can prove difficult. Different but related data sources result in duplicated effort and incompatibilities.

**What is a triples map?** A Triples Map defines rules to generate zero or more RDF triples sharing the same subject. A Triples Map consists of a Logical Source, a Subject Map and zero or more Predicate-Object Maps.




Blue: R2RML's components, green: R2RML's extended components, red: RML's specific components.

**What is R2RML?** R2RML is an RDF vocabulary to describe a mapping of relational data into RDF.

A triples map has access to tables (and queries) through logical tables.

These logical tables are mapped with term maps. Subject maps generate resource identifiers. Predicate-object maps generate the rest of the triple.

Term maps generate individual RDF terms (constant, column value, string template).

**What is RML?** RDF Mapping Language is a generalization of R2RML toward heterogeneous data sources.

RML abstracts different types of logical sources. A source is modelled as an iterator of items. Sources can be databases, CSV, XML, JSON, ... The mechanism is extensible through new vocabularies.

RML term maps use logical sources to create terms. The contents of iterator items are accessible through data-source-specific reference formulations.

RML can map and interlink heterogeneous sources.

Note: there is an example in the slides.

**Is it possible for RML to include other resources and properties?** Yes, the mapping can be extended.

RML mappings can incorporate data from other sources, even in different formats. RML mappings can look up data from Web APIs. Interlinking can happen at mapping time, rather than as a separate step after mapping.

### 6.1.2 Validate

**On which stages can validation be applied?** Validation on input data level can address: spelling mistakes, field overloading, simple datatype checks, ...

Validation on the semantic level can address: domain and range of values, inconsistencies, ...

**What is the difference between validation in databases and validation in ontologies?** Databases only allow for rudimentary constraint validation (elementary value-type checks, referential integrity).

Ontologies allow for much more fine-grained constrained validation. More specific types can be defined, and their definition can be reused. Type checking can also take more factors into account (domain and range, incompatible types, ...). Additional reasoning can further identify problems.

**What can automated validation tell you?** Automated validation tells you whether date makes sense. Violations across triples can be identified, but not always automatically resolved.

**Explain the automated validation process.**

- RDFUnits assesses the quality of a dataset by running automated tests on it.
- Ontologies used in a dataset can be looked up by dereferencing its concepts.
- The constraints in the ontology are transformed into SPARQL queries and then evaluated, which can result in warnings/errors.

**Why validate during the mapping process?** By validating during the mapping process, we detect quality issues before they occur.

### 6.1.3 Publish

**What are the 3 ways of publishing Linked Data on the Web?**

1. Data dump: provide an archive file with the entire dataset.
2. SPARQL endpoint: expose a triple store's query interface.
3. Linked Data documents: browse triples per resource/topic.

**What is a datadump? Advantages, disadvantages?** A data dump places all dataset triples in one or more archive files.

Advantages: they offer the client full flexibility to choose how data is processed.

Disadvantages: dumps need to be downloaded entirely before they can be queried, keeping data-up-to date requires effort

**What is a SPARQL endpoint? Advantages, disadvantages?** A SPARQL endpoint lets clients evaluate arbitrary (read-only) queries on a sever.

Advantages: the clients have direct access to the data they are interested in (little bandwidth is required), data is always up-to date.

Disadvantages: the per-request cost for SPARQL endpoitns is much higer than for tother HTTP servers.

**What is a Linked Data document? Advantages, disadvantages?** Linked Data documents provide per-topic access to a dataset. They follow the Linked Data principles.

Advantages: browsing up-to date datasets is straightforward, query evaluation is possible through link-traversal-based querying.

Disadvantages: the evaluation of SPARQL queries is rather slow, completeness cannot be guaranteed.

### 6.1.4 Query

**On what do the possibilities for query evaluation depend?** On how data is made available.

- Available in RDF: discover the ontologies used.
- Data linked to other data: can/might need to involve other datastes.
- In what interfaces is the data available: the client might need to evaluate (a part of) the query.

**What challenges are introduced by evaluating queries over a federation of interfaces?**

- Which interface has the necessary data?
- How will the query evaluation be coordinated?
- In what order are subqueries executed?

### 6.1.5 Enhance

**What does provenance make possible?** Provenance allows modeling the history trail of facts.

Provenance captures entities, activities, and people involved in producing a resource. Provenance can asses quality, reliability, or trust.

**What is reverse mapping?** Reverse mappings could feed edits back to the original source.

If the RDF triples are generated from raw data, edits to the triples should be ported back. If the mapping file declaratively specifies how a source maps to triples, we might be able to reverse it automatically.

## 6.2 Semantic Web Challenges

**What is the difference between the current generation of agents and the agents envisioned by the Semantic Web?** The current generation of agents only performs preprogrammed acts, but the Semantic Web's goal is to allow this with unknown services and data. Machines should discover services and data and use them without prior knowledge.

**What is the Semantic Web's answer to live querying?** public SPARQL endpoints. A SPARQL endpoint allows clients to send any SPARQL query for evaluation.

But, what could possibly go wrong?

- What if clients send expensive queries?
- What if many clients send medium queries?
- How will you mirror the server's data?

**Discuss the Linked Data availability problem.**

1. Public SPARQL endpoints that exist, lack uptime.
    - High uptime would be possible, but comes with a high server cost.
2. For most datasets, no public endpoint exists.
    - Publishers provide data dumps instead, but these cannot be queried live.

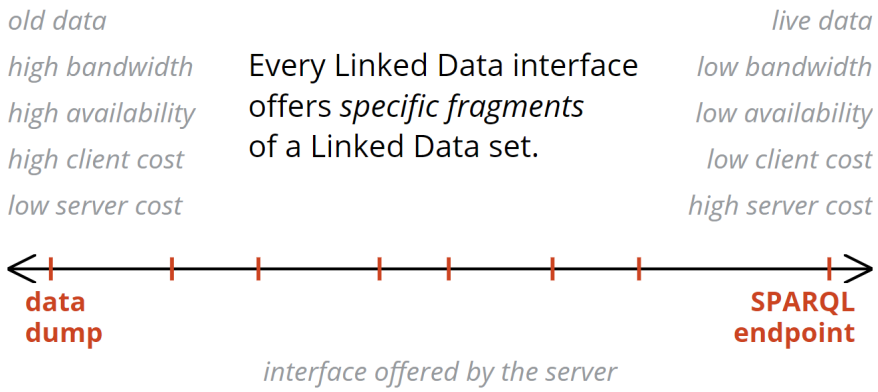
**Opinion: 15 years of Semantic Web research has mostly lead to intelligent servers.** But without a cost model behind it, server intelligence is not scalable (not every Web resource can be created just for you).

Opinion: Instead of trying to be intelligent ourselves, we should enable clients to be intelligent.

## 6.3 Linked Data Publishing

### 6.3.1 Trade-offs for the Semantic Web

**Discuss Linked Data Fragments.** Linked Data Fragments is a uniform view on Linked Data interfaces.




**Each type of Linked Data Fragment is defined by three characteristics. Give for each type the three characteristics.** Linked Data Fragment (general)

- Data: what triples does the fragment contain?
- Metadata: do we know more about the data/fragment?
- Controls: How can we access more data?

Data dump

- Data: All dataset triples.
- Metadata: Number of triples, file size.
- Controls: none.

SPARQL query result

- Data: triples matching the query.
- Metadata: none.

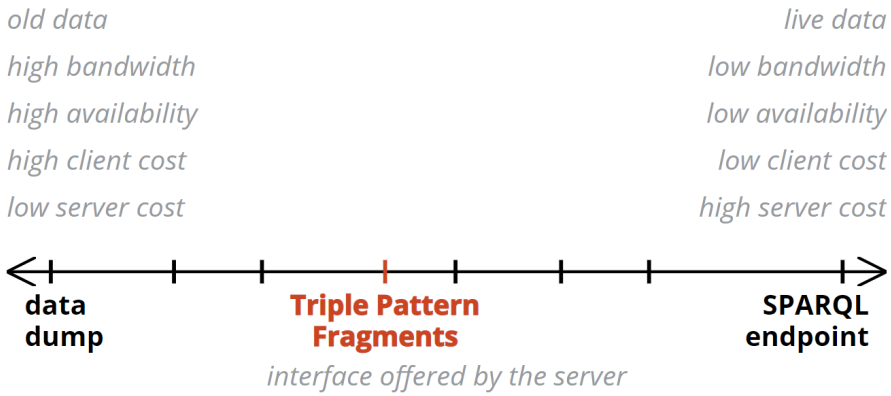
- Controls: none.

Linked Data document

- Data: triples about a topic.
- Metadata: creator, maintainer, ...
- Controls: links to other Linked Data documents.

**Discuss the Triple Pattern Fragment.** A Triple Pattern Fragments interface is low-cost and enables clients to query.

- Data: matches of a triple pattern (paged).
- Metadata: total number of matches.
- Controls: access to all other Triple Pattern Fragments of the same dataset.




**Why are Triple Patterns lightweight?** They do not require a triple store.

The interface can be realized with many back-ends. Since queries are relatively simple, a less expensive data infrastructure is sufficient.

The Header-Dictionary-Triples (HDT) format stores triples in a compressed file.

Note: triple patterns are not the final answer, no interface will ever be!

### 6.3.2 Querying Triple Pattern Fragments

**How can a client evaluate a SPARQL query over a TPF interface?**

- Give the client a SPARQL query, and the URL of any TPF of the dataset.
- It uses the controls inside of the fragment to determine how to access the dataset.
- It reads the metadata to decide how to plan the query.

See example in the slides.

### 6.3.3 Evaluating the Trade-offs

**What did the experiments verify?** Processing everything on the server is costly. Processing everything on the client isn't Web.

Solutions that divide the workload can offer new perspectives, if we accept the trade-offs they bring.

**Opinion: on what should the Semantic Web focus?** On making each of the life cycle phases sustainable.

**Opinion: how can we accelerate the process to get intelligent agents?** Stop building intelligent servers.