

PC3R

Cours 06 - Web: Design et Communication

Romain Demangeon

PC3R MU1IN507 - STL S2

11/03/2021

Plan du Cours 6

- ▶ Design des applications webs: généralités.
- ▶ Protocole HTTP
- ▶ Approche Service (SOAP - RPC)
- ▶ Approche Ressources (REST)

Programmation Réticulaire en PC3R

- ▶ Pourquoi la programmation web ?
 - ▶ style particulier de programmation concurrente par passage de messages,
 - ▶ notions nécessaires dans plusieurs carrières,
 - ▶ était couvert en Master STL par l'UE de M2 DAR,
- ▶ Web en PC3R:
 - ▶ condensé du cours de DAR,
 - ▶ les notions et le design prennent sur la technique
 - ▶ projet DAR → microprojet PC3R (TMEs 6,7,8,9, 10)
- ▶ Plan des cours:
 - ▶ Cours 06: Design des applications / HTTP / Services vs. Ressources
 - ▶ Cours 07: Programmation serveur (Servlets) / BDD / ORM
 - ▶ Cours 08: Programmation client / Javascript

"Réticulaire"

- ▶ Réticulaire: (*wiktionnaire*): qui ressemble à un réseau.
- ▶ Du latin *reticulum*: filet à petites mailles.
 - ▶ donne *rêts*, *réticule*, *rétiaire* ...
- ▶ tentative de traduction de *Web*



- ▶ Caractéristiques:
 - ▶ Architecture Client/Serveur.
 - ▶ Applications Vs. Programmes.
 - ▶ Web 2.0:
 - ▶ services, interactivité,
 - ▶ web social (l'utilisateur crée du contenu).

Rappel: Modèle Client-Serveur

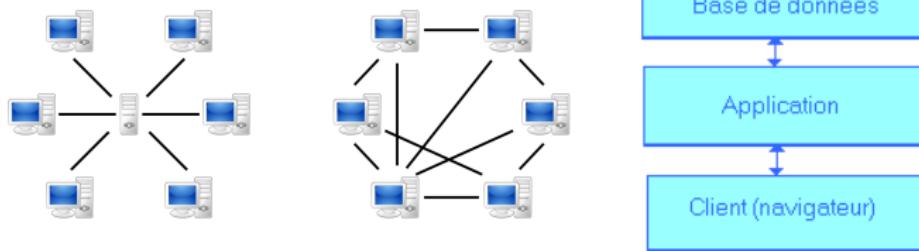
- ▶ Modèle de **communication** entre **programmes** à travers un **réseau**.
- ▶ Modèle asymétrique:
 - ▶ les **Clients** (applications, browsers, programmes ou serveurs) envoient des requêtes,
 - ▶ les **Serveurs** (puissance de calcul) traitent les requêtes et répondent.



- ▶ **Avantages:**
 - ▶ centralisation des données,
 - ▶ centralisation de la puissance de calcul (clients légers),
- ▶ **Inconvénients:**
 - ▶ centralisation des connexions,
 - ▶ peu robuste.

Architectures Client-Serveur

- ▶ **Mainframe**: machine dédiée au centre du réseau.
- ▶ **Peer-to-peer**: chaque agent joue le rôle de client ou de serveur.
- ▶ **Architecture 2-niveaux**: client-serveur classique à travers le Internet.
- ▶ **Architecture 3-niveaux**: division entre:
 - ▶ serveur de **données**
 - ▶ serveur **d'application**.
- ▶ **Architecture *n*-niveaux**: **composants** répartis sur plusieurs "niveaux" (*Web / Business / Data*) s'exécutant sur des serveurs différents.



Vocabulaire : Application Web

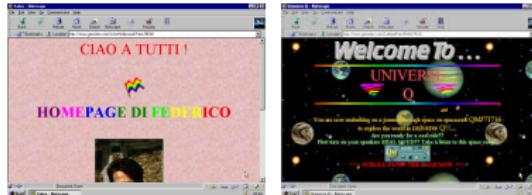
Définition (tentative)

Application **client/serveur** utilisant un **navigateur** comme environnement **d'exécution** du programme client, proposant un service **interactif** à travers une **connection** avec des serveurs sur le **Internet**.

- ▶ Site Web: propose du contenu à partir de données **statiques**
- ▶ Application Web: propose du contenu **à la carte** basée sur des requêtes paramétrées
- ▶ "Valeur ajoutée" des applications:
 - ▶ gère les **utilisateurs** (authentification, sessions),
 - ▶ gère la **sécurité**,
 - ▶ architecture **répartie** (charge sur le client)
 - ▶ application **monopage** + requêtes **asynchrones**.

Age d'or du réticulaire

- ▶ depuis 20 ans: croissance du secteur web.
 - ▶ omniprésence des navigateurs (PC, puis téléphones),
 - ▶ medium par défaut de communication/interaction.
- ▶ Web de plus en plus dynamique
 - ▶ d'abord: pages statiques,
 - ▶ puis: interactions dynamiques (formulaires),
 - ▶ puis: langages de scripts clients (JS)
 - ▶ puis: interactivité totale (AJAX, HTML5)



Name	Value
Name	
Sex	<input type="radio"/> Male <input checked="" type="radio"/> Female
Eye color	green
Check all that apply	<input type="checkbox"/> Over 6 feet tall <input type="checkbox"/> Over 200 pounds
Describe your athletic ability:	
Enter my information	



Exemples d'applications

A screenshot of the Wikipedia homepage. The top navigation bar includes links for 'Article', 'Talk', 'Read', 'Edit', 'View history', and 'Search'. Below the search bar, there's a link to 'Create account' and 'Log in'. The main content area features the article 'Web application' from the 'Wikipedia, the free encyclopedia' section. The article text discusses web applications as client-side programs running in a web browser. A sidebar on the right contains a box for additional citations and a note about the article being challenged and removed in January 2018. Below the article, there's a section titled 'A web-based application' with a sub-section on 'Interactions'. At the bottom of the page, there's a large advertisement for OUI SNCF, featuring a train and travel-related information.

- ▶ différence de design.
 - ▶ des caractéristiques communes.

Caractéristiques des applications modernes

- ▶ **Interface ergonomique:**
 - ▶ éditeur de texte, *drag n'drop*, raccourcis claviers.
- ▶ **Multimedia:**
 - ▶ audio, vidéo, jeux.
- ▶ **Avantages:**
 - ▶ facile à déployer, mettre à jour,
 - ▶ interopérabilité client,
 - ▶ charge de travail, espace mémoire réduits pour le client,
 - ▶ multiplateforme (téléphones, tablettes, consoles, télévisions),
- ▶ **Inconvénients:**
 - ▶ interface limitée (moins avec HTML5),
 - ▶ dépendant des navigateurs,
 - ▶ dépendant d'une connection (moins avec HTML 5),
 - ▶ déplacement du rapport de force vers les entreprises:
 - ▶ collection de données,
 - ▶ vers un monde informatique propriétaire.

Histoire du Web

► Internet:

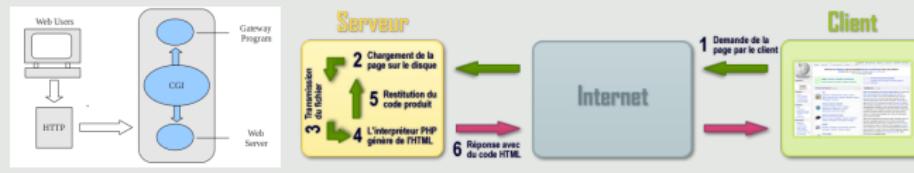
► Web:

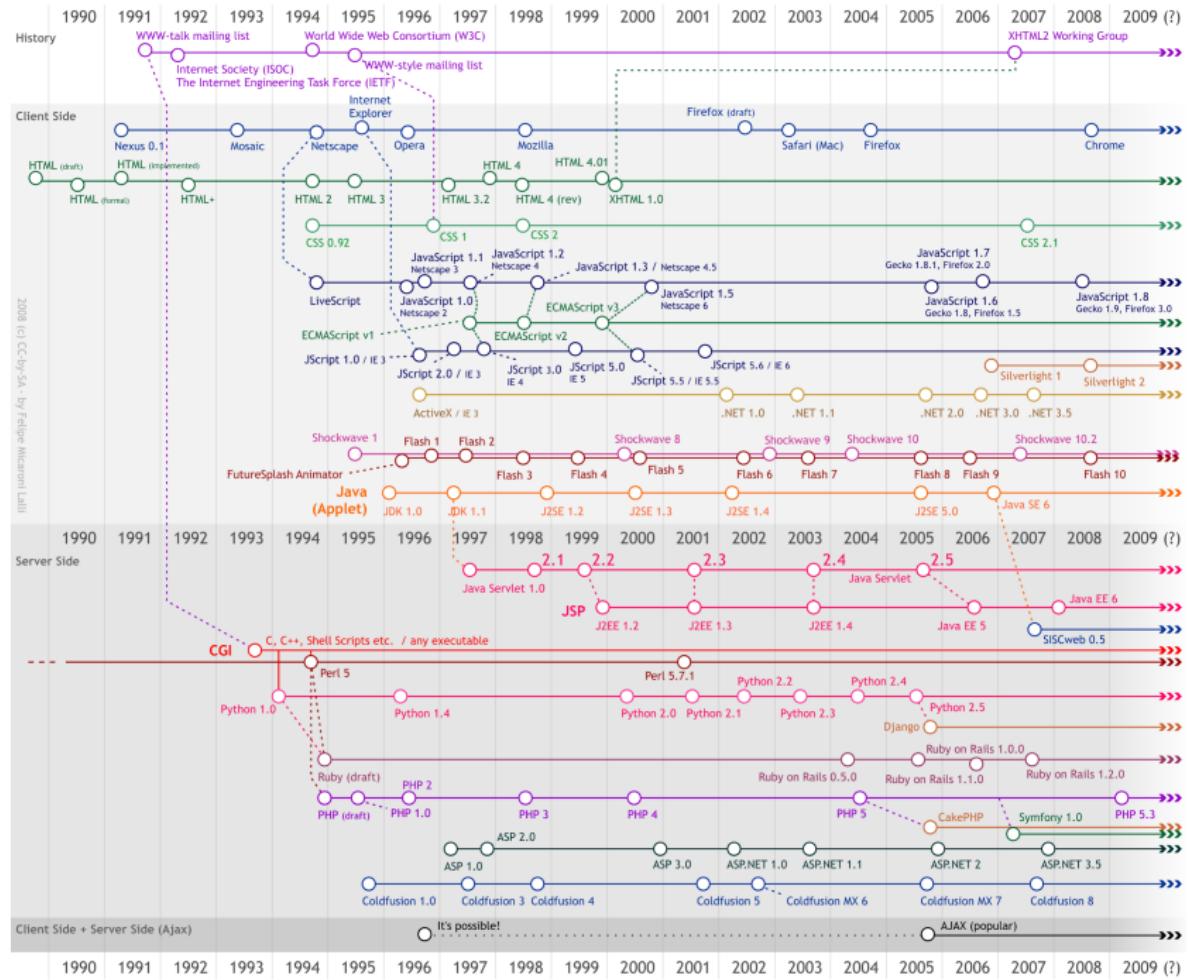
Histoire du Web

- ▶ **Internet:** réseau mondial
 - ▶ medium du web, de l'email, du chat, du FTP, de SSH ...
- ▶ **Web:** système hypertexte public
 - ▶ application de Internet - port 80

Evolutions du Web

- 1993 CGI (génération de contenu par un programme)
- 1995 PHP 1.0 (pages web dynamiques)
- 1995 JavaScript (langage de script pour client)
- 1999 Servlet Java (CGI-like de haut niveau)
- 2005 AJAX (page dynamiques asynchrones)
- 2008 HTML5 - draft





Transfert de charge

1970 Terminaux légers, tout se passe sur le **serveur**,

1980 Ordinateurs personnels, calculs sur les **clients**,

- ▶ **serveurs** d'entreprise,
- ▶ communications **directes**,
- ▶ applications installées en **local**,
- ▶ **administration système**.

1990 Clients légers (navigateurs), logique dans le **serveur**.

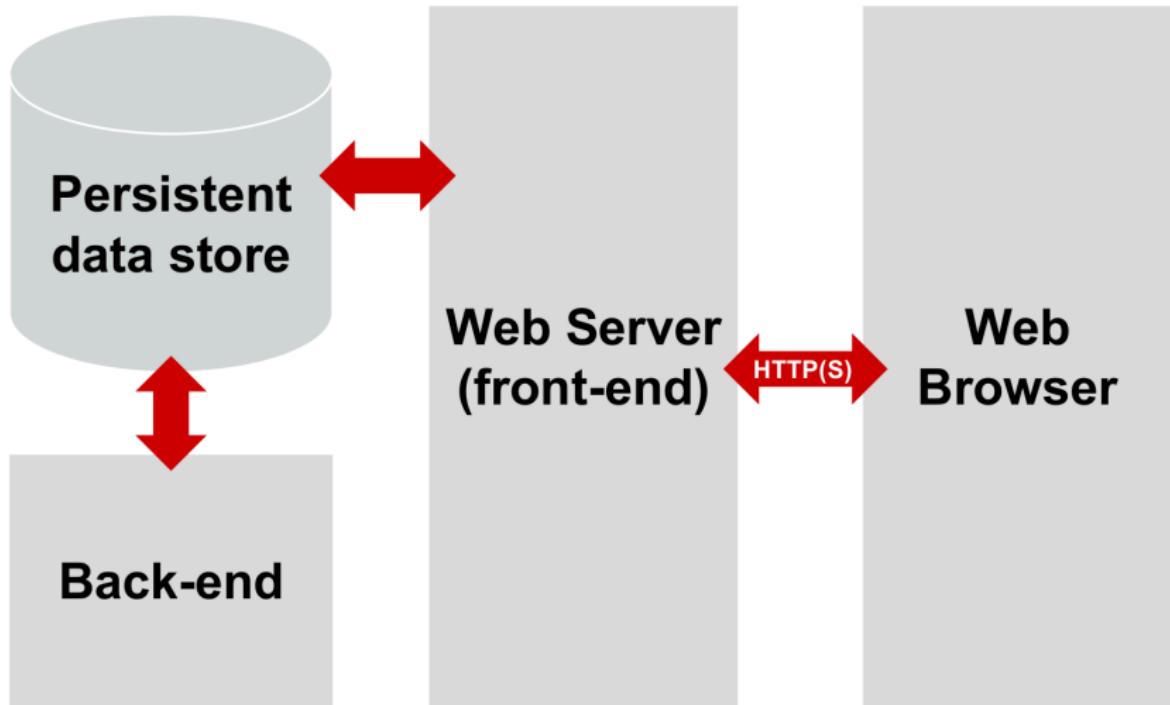
- ▶ **Internet**: calculs scientifiques, mails,
- ▶ **Web statique**,
- ▶ début du Web **dynamique** (génération de pages)

2000 Retour de la logique dans les clients ("Web 2.0").

- ▶ appels **asynchrones**

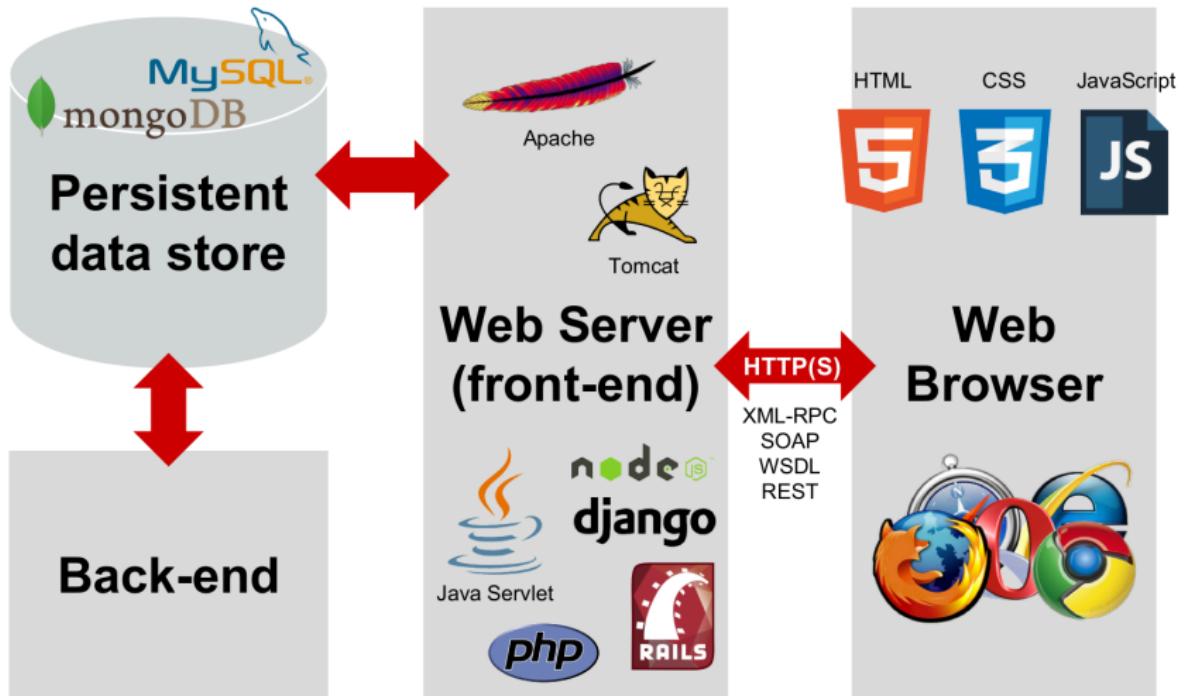
2010 Applications mobiles.

Architecture classique d'une application web



- ▶ Navigateur
 - ▶ interface utilisateur,
 - ▶ état à court terme,
 - ▶ peut implémenter de la logique (confiance ?),
 - ▶ communique avec le serveur web via HTTP(S),
 - ▶ exécute du code HTML, CSS, JS.
- ▶ Serveur Web
 - ▶ répond aux requêtes,
 - ▶ sans état,
 - ▶ lit et écrit dans le serveur de données,
 - ▶ responsable de la logique,
 - ▶ comporte un serveur/*container* et un système pour la logique.
- ▶ Serveur de Données
 - ▶ état de l'application web,
 - ▶ point de synchronisation.
- ▶ Back-end
 - ▶ logique du serveur indépendant du client.

Technologies

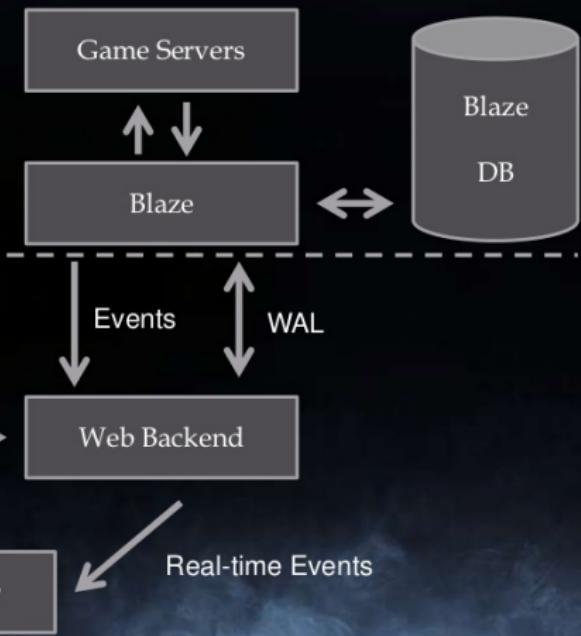


Architectures plus complexes

TECHNOLOGY

Blaze Details

- › Uses Web Access Layer (WAL)
- › WAL client generated from TDF
 - › TDF is a API definition language
- › Blaze events (XML over HTTP)



Exemple d'architecture de l'**application web** associée à un jeu vidéo.

Evolution du Web

Tendances actuelles de la recherche:

- ▶ Frontière Client/Serveur: Ocsigen, Hop, node JS, ...
- ▶ Omniprésence de JS: compilateurs vers JS, ...
- ▶ Meta-données: collecte, stockage, traitement, apprentissage, ...
- ▶ Sécurité: identification, usurpation, ...

Design Classique vs. SPA

Design classique:

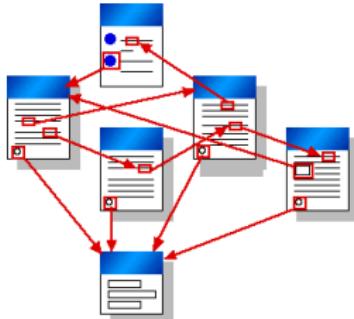
- ▶ L'utilisateur se **déplace** dans différents écrans.
 - ▶ chaque écran correspond à une **fonctionnalité** ou un **état**.
- ▶ Le serveur produit des **écrans** à la demande à partir de **modèles**.
 - ▶ des composants de l'écran sont obtenus de manière **asynchrones**.
- ▶ Le serveur sert de l'**HTML** et des **contenus asynchrones** (JSON, XML, images) qui peuplent certains écran..
- ▶ La commande **back** du navigateur fait revenir au dernier **écran** visité.

Design **SPA**:

- ▶ L'utilisateur reste devant un **unique écran**.
 - ▶ les fonctionnalités proposées **s'intègrent** à cet écran.
 - ▶ les composants de l'écran sont obtenus de manière **asynchrones**.
- ▶ Le serveur sert du **JSON** (ou XML) (et des ressources multimedia).
- ▶ La commande **back** n'a pas de sens.

HTTP: Création

- ▶ HTTP: HyperText Transfer Protocol
- ▶ Hypertext:
 - ▶ noeuds (textes) liés entre eux par des hyperliens (permettant le passage d'un document à un autre), présentation non-linéaire d'information
 - ▶ Projet Xanadu - Ted Nelson 1960
- ▶ Protocole lié au Web, inventé en 1989-1990 (en même temps que HTML) par Sir Timothy John Berners-Lee (W3C)
- ▶ 1991 : HTTP 0.9
- 1996 : HTTP 1.0 (RFC 1945) (serveur virtuels, cache, identification)
- 1997 : HTTP 1.1 (RFC 2068) (pipeline, type de contenu)
- 1999 : HTTP 1.1 amélioré (RFC 2616)



Caractéristiques

- ▶ protocole de la **couche application** (comme FTP, IMAP, SSH)
- ▶ protocole **asymétrique** (client-serveur):
 - ▶ le **client** soumet une **requête** au **serveur**,
 - ▶ le **serveur** envoie une **réponse**.
- ▶ pas d'**état**.
 - ▶ les requêtes sont **indépendantes** les unes des autres.
 - ▶ les requêtes **n'identifient pas** (directement) le client.
- ▶ clients HTTP:
 - ▶ principalement **navigateurs**,
 - ▶ "aspirateurs de sites",
 - ▶ robots d'indexations.
- ▶ **ports** 80 pour HTTP, 443 pour HTTPS
- ▶ **indirections** possibles:
 - ▶ **tunnel** (transmission)
 - ▶ **gateway** (modification du protocole)
 - ▶ **proxy** (modification des requêtes, cache local)

Requête HTTP

Structure

1. ligne de **commande**: <method>_<URI>_HTTP/<version>
2. **en-tête** de requête: <name> : _<value>
3. ligne vide
4. **corps** de la requête (si nécessaire)

Exemple

```
GET /www.toto.com?id=1&name=titi HTTP/1.1
```

```
User-Agent: Mozilla/5.0
```

```
Accept: text/html
```

```
EOF
```

URI: Uniform Resource Identifier - identifie une ressource de manière permanente (URL ⊆ URI)

Universal Resource Locator

chaîne de caractères identifiant les ressources du Web

- ▶ adresse web, adresse réticulaire, adresse universelle

```
scheme://[login:pwd]domain[:port]/path/name  
          [?query_string] [#fragment_id]
```

- ▶ scheme: **protocole** utilisé (http par exemple)
- ▶ login, pwd: si authentification requise (peu sécurisé)
- ▶ domain: nom de **domaine** (ou adresse IP)
- ▶ path: **chemin** absolu
- ▶ name: **nom** de la ressource (optionnel, penser à index.html)
- ▶ query_string: chaîne de **requête** traitée par la page web
- ▶ fragment_id: **signet** ou balise à l'intérieur de la page web

Méthodes de Requêtes HTTP

- ▶ **GET**: récupère une ressource (URI).
- ▶ **HEAD**: même chose que GET, mais sans corps de réponse (permet de récupérer des informations sur une ressource sans télécharger la ressource elle-même).
- ▶ **POST**: demande au serveur d'accepter l'entité envoyée (dans le corps de requête) comme **subordonnée** à la ressource identifiée par l'URI.
 - ▶ annoter une ressource, étendre une base de données, soumettre un formulaire, *uploader* des données.
 - ▶ l'effet dépend du serveur web.
- ▶ **OPTIONS**: renvoie les méthodes HTTP supportées pour l'URI spécifiée.

POST /enquete.php HTTP/1.1

Host: www.upmc.fr

Content-Type: application/x-www-form-urlencoded

Content-Length: 43

prenom=Annie&nom=Cordy&netudiant=0123456789

Méthodes de Requêtes HTTP

Autres méthodes (entre autres, utilisées par REST):

- ▶ **PUT**: demande au serveur de stocker l'entité envoyée comme ressource identifiée par l'URI.
- ▶ **DELETE**: supprime la ressource stockée à l'URI fournie.
- ▶ **PATCH**: modifie partiellement la ressource stockée à l'URI fournie.
- ▶ **CONNECT**: utilisée avec un proxy qui peut dynamiquement devenir un tunnel SSL.
- ▶ **TRACE**: méthode de test qui demande au serveur de renvoyer le message envoyé par le client.

Sûreté et Idempotence

- ▶ Les méthodes **OPTIONS** et **TRACE** n'ont pas le droit d'avoir des effets de bords sur le serveur.
- ▶ Une méthode HTTP est dite **sûre** quand elle ne change pas **l'état** du serveur. Elles ne doivent pas faire plus que **récupérer** des ressources.
 - ▶ Les méthodes sûres sont: **GET** et **HEAD**, (**OPTIONS**, **TRACE**).
 - ▶ un serveur peut générer des effets de bords en réceptionnant un **GET** ; ce qui compte c'est que le client ne l'a pas requis.
- ▶ Certaines méthodes sont considérées comme **idempotentes**. Elles renvoient le même résultat après N requêtes identiques. Les méthodes idempotentes sont: **GET**, **HEAD**, **PUT**, **DELETE**, (**OPTIONS**, **TRACE**).

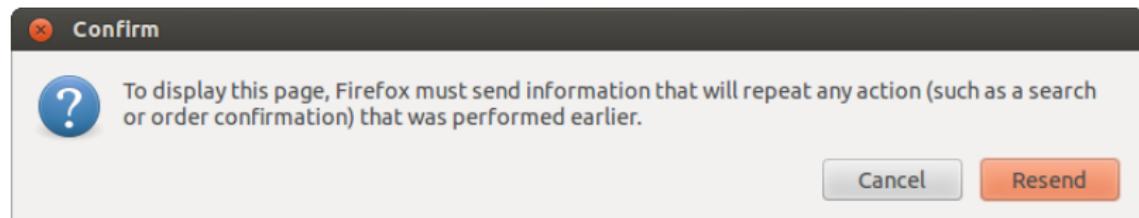
Mathématiques

Une fonction est **idempotente** quand plusieurs applications successives donnent le même résultat:

$$\forall n > 1, \forall x, f^n(x) = f(x)$$

Méthodes non-sûres

- ▶ les méthodes POST, PUT, DELETE étaient à l'origine destinées à la gestion des [fichiers](#).
- ▶ Elles peuvent être dangereuses utilisée avec un site web [statique](#).
- ▶ Elles sont prises en charge directement dans les API [RESTful](#).
- ▶ Les navigateurs préviennent au rechargement d'une page qui implique l'envoi d'une méthode POST:



Headers HTTP

- ▶ Formats attendu pour la réponse.

Accept: text/plain

Accept-Language: en-US

Accept-Encoding: gzip, deflate

- ▶ Contrôle du cache du navigateur ou d'un serveur proxy:

Cache-control: no-cache

Cache-control: max-age=600

If-Modified-Since: Wed, 28 May 2014 08:40:00 GMT

- ▶ Proxy

Host: en.wikipedia.org:80

Max-Forwards: 10

Proxy-Authorization: Base Um9tYWluOnNhdWNpc3NlOTQ=

Via: 1.0 pontneuf, 1.1 pontmarie.fr (Apache/1.1)

En-têtes de requête HTTP

- ▶ **Contenu** de l'entité contenue dans le message.

Content-Length: 348

Content-Type: text/html; charset=utf-8

Content-MD5: Q2h1Y2sgSW50ZWdyXR5IQ==

- ▶ **Meta**-données:

Date: Wed, 28 May 2014 08:40:00 GMT

User-Agent: Mozilla/5.0

Referer: http://en.wikipedia.org/wiki/Main_Page

- ▶ Données liées à une **session**

Cookie: Version=1; Skin=new;

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Connection: keep-alive

DNT: 1

Réponses HTTP

Structure similaire à la requête:

- ▶ Commande:

HTTP/<version>_<status>_<reason>

- ▶ En-têtes:

<name>:<value>

- ▶ Ligne vide, puis **corps** de réponses (pas toujours)

HTTP/1.1 200 OK

Content-Type: text/html; charset=UTF-8

<html>

...

</html>

Codes statut

- ▶ 1XX: Information
 - ▶ 100:

Codes statut

- ▶ 1XX: Information
 - ▶ 100: *Continue*
 - ▶ 118:

Codes statut

- ▶ 1XX: Information
 - ▶ 100: *Continue*
 - ▶ 118: *Connection timed out*
- ▶ 2XX: Succès
 - ▶ 200:

Codes statut

- ▶ 1XX: Information
 - ▶ 100: *Continue*
 - ▶ 118: *Connection timed out*
- ▶ 2XX: Succès
 - ▶ 200: *OK*
 - ▶ 201: *Created*
 - ▶ 202: *Accepted*
 - ▶ 204: *No Content*
- ▶ 3XX: Redirection
 - ▶ 301:

Codes statut

- ▶ 1XX: Information
 - ▶ 100: *Continue*
 - ▶ 118: *Connection timed out*
- ▶ 2XX: Succès
 - ▶ 200: *OK*
 - ▶ 201: *Created*
 - ▶ 202: *Accepted*
 - ▶ 204: *No Content*
- ▶ 3XX: Redirection
 - ▶ 301: *Moved Permanently*
 - ▶ 303: *See Other*
 - ▶ 307: *Temporary Redirect*

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400:

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400: *Bad Request*
 - ▶ 403:

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400: *Bad Request*
 - ▶ 403: *Forbidden*
 - ▶ 404:

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400: *Bad Request*
 - ▶ 403: *Forbidden*
 - ▶ 404: *Not Found*
 - ▶ 408:

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400: *Bad Request*
 - ▶ 403: *Forbidden*
 - ▶ 404: *Not Found*
 - ▶ 408: *Request Timeout*
 - ▶ 418:

Codes statut

- ▶ 4XX: Erreur Client
 - ▶ 400: *Bad Request*
 - ▶ 403: *Forbidden*
 - ▶ 404: *Not Found*
 - ▶ 408: *Request Timeout*
 - ▶ 418: *I'm a teapot* (poisson d'avril RFC 1998)
- ▶ 5XX: Erreur Serveur
 - ▶ 500: *Internal Server Error*
 - ▶ 501: *Not Implemented*
 - ▶ 502: *Bad Gateway ou Proxy Error*
 - ▶ 503: *Service Unavailable*
 - ▶ 505: *HTTP Version Not Supported*

En-têtes de réponses HTTP

- ▶ **Contenu** de l'entité contenue dans le message.

Content-Type: text/html; charset=utf-8

Content-Encoding: gzip

Content-Language: da

Content-Length: 348

Content-Location: /index.html

- ▶ Contrôle du **cache**:

Cache-Control: no-cache

ETag: "737060cd8c284d8af7ad3082f209582d"

Expires: Thu, 01 Dec 1994 16:00:00 GMT

Last-Modified: Tue, 15 Nov 1994 12:45:26 +0000

En-têtes de réponses HTTP

- ▶ Information de [Proxy](#):

Via: 1.0 fred, 1.1 example.com (Apache/1.1)

- ▶ [Meta](#)-données:

Allow: GET, HEAD

Date: Tue, 15 Nov 1994 08:12:31 GMT

Location: http://www.w3.org/pub/WWW/People.html

Retry-After: 120

Server: Apache/2.4.1 (Unix)

- ▶ [Session](#):

Set-Cookie: UserID=X; Max-Age=3600; Version=1

Connection: keep-alive

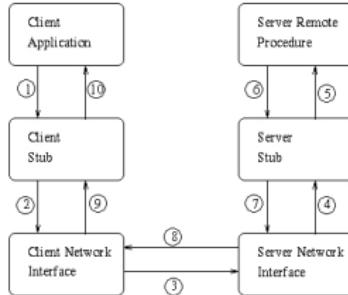
DNT: 1

(Cours 04) Remote Procedure Call

Définition

Un Appel de Procédure Distante (RPC) est un protocole réseau permettant à un programme informatique de faire appel à une procédure (ou routine, ou méthode) s'exécutant dans un autre espace d'adresse (un ordinateur distant sur un réseau commun) à l'aide d'un serveur d'applications, sans que le développeur du programme initial code explicitement les interactions réticulaires.

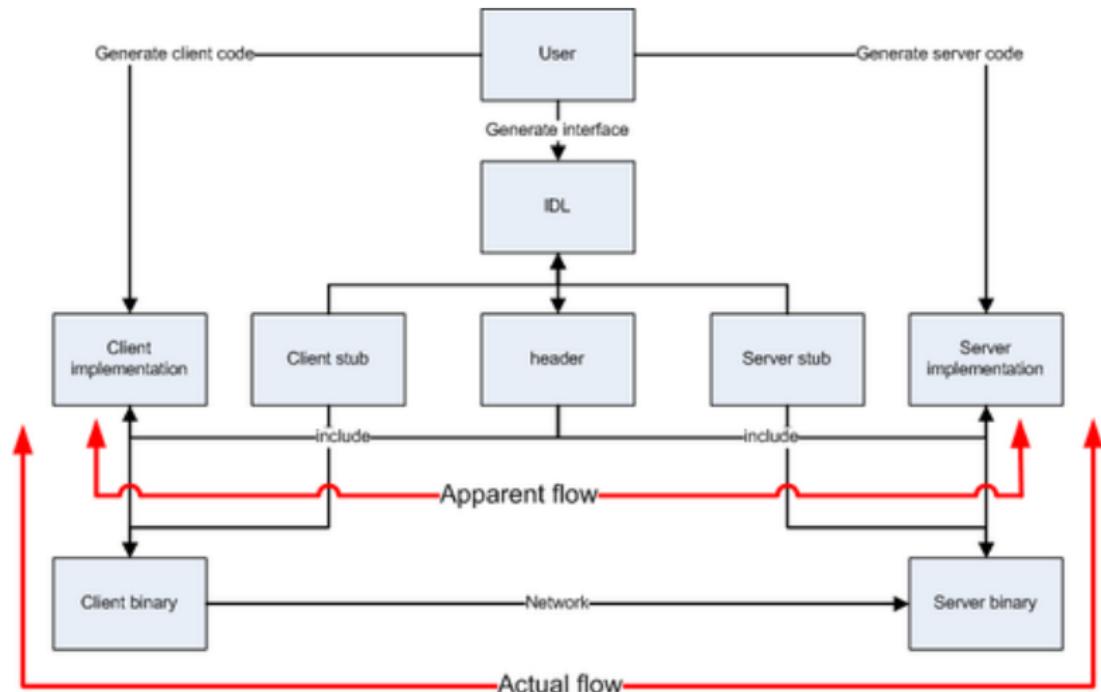
- ▶ Maître-mot: Transparency.
- ▶ Idée datant de 1976.
- ▶ Paradigme de concurrence par passage de messages.



(Cours 04) Suite d'actions RPC

1. le programme client appelle la **souche client**
 - ▶ appel **local** standard, les paramètres sont mis sur la pile de manière standard
2. la souche client **conditionne** (*marshalling*) les paramètres dans un message et fait **un appel système** pour envoyer le message.
3. le système d'exploitation du client **envoie** le message de la machine client à la machine serveur
4. le système d'exploitation du serveur **reçoit** le message et les passe à la **souche serveur**.
5. la souche serveur **déballe** (*unmarshalling*) les paramètres du message.
6. la souche serveur **appelle la procédure** sur le serveur avec les paramètres.
7. la réponse de la procédure **suit le chemin inverse**.

(Cours 04) Schéma de RPC



RPC à travers les âges

- 1976 Description de RPC dans RFC707
- 1981 Courier de Xerox
- 1984 RPC de Sun Network File System (puis Unix/Windows)
- 1997 Java RMI (API pour l'appel distant en Java)
- 1998 XML-RPC (appel encodé en XML transmis en HTTP)
- 1998 SOAP
- 2001 WSDL (Web Service Description Language)
- 2001 UDDI (Universal Description Discovery and Integration)
- 2005 JSON-RPC

- ▶ **Implémentations:** Courier, Sun RPC, RMI,
- ▶ **Formats d'appel:** XML-RPC, JSON-RPC
- ▶ **Annuaire de Services Webs:** UDDI

- ▶ le client envoie une **requête HTTP** au serveur,
 - ▶ le corps de la requête est un **document XML** spécifiant un appel unique à une méthode (nom de la méthode et arguments).
- ▶ le serveur renvoie une **réponse HTTP** dont le corps contient un **document XML** (avec une valeur de retour unique).
- ▶ on peut encoder des **structures de données** à l'intérieur des documents XML.

Requête XML-RPC: Exemple

```
<?xml version="1.0"?>
<methodCall>
    <methodName>calculatrice.findOrder</methodName>
    <params>
        <param>
            <value><i4>347</i4></value>
        </param>
        <param>
            <value>
                <array>
                    <data>
                        <value><i4>28</i4></value>
                        <value><string>Jean-Paul</string></value>
                    </data>
                </array>
            </value>
        </param>
    </params>
</methodCall>
```

Requête XML-RPC: Exemple

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Saucisse</string></value>
    </param>
  </params>
</methodResponse>
```

JSON-RPC

- ▶ le client envoie une **requête HTTP** au serveur,
 - ▶ le corps de la requête est un **document JSON** spécifiant un appel unique à une méthode (nom de la méthode et arguments).
- ▶ le serveur renvoie une réponse dont le corps contient un **document JSON** (avec une valeur de retour unique).
- ▶ on peut encoder des **structures de données** à l'intérieur des documents **JSON**.

JSON est un format populaire, car il est traité facilement (directement) par JavaScript.

Exemple de JSON-RPC

- ▶ Interaction simple:

```
--> {"method": "echo", "params": ["Hello JSON-RPC"], "id": 1}
<-- {"result": "Hello JSON-RPC", "error": null, "id": 1}
```

- ▶ Application de chat:

```
...
--> {"method": "postMessage", "params": ["Bonjour tous!"], "id": 99}
<-- {"result": 1, "error": null, "id": 99}
<-- {"method": "handleMessage", "params": ["Jeanne", "asv ?"], "id": null}
<-- {"method": "handleMessage", "params": ["Bob", "allez ++"], "id": null}
--> {"method": "postMessage", "params": ["C bi1 Caramail"], "id": 101}
<-- {"method": "userLeft", "params": ["Bob"], "id": null}
<-- {"result": 1, "error": null, "id": 101}
...
...
```

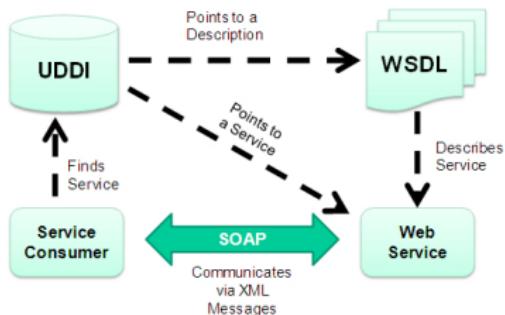
Exemple de JSON-RPC

- ▶ Avec des paramètres:

```
{"method": "verifierCommande",
 "params": [
    {
        "nom": "Cordy",
        "prénom": "Annie",
        "id": 1234567890
    },
    {
        "quantite": 8,
        "nom": "saucisse",
        "montant": 45.50
    }
],
"id": 1234}
```

Approche Service: UDDI, WSDL, SOAP

- ▶ Bob propose des **Services Webs** hébergés sur un serveur.
- ▶ Bob publie les descriptions de ses services en **WSDL** à l'annuaire **UDDI**
- ▶ Alice a besoin d'un service qu'elle cherche dans l'**UDDI** en lisant les descriptions **WSDL**.
- ▶ Alice interagit avec le service de Bob en utilisant le protocole **SOAP**.



Simple Object Access Protocol

Protocole permettant la transmission de messages entre objets distants (invoquer des méthodes d'objets distants), généralement par HTTP/HTTPS (mais peut se faire par SMTP).

► Avantages:

- ▶ Possibilité d'utiliser HTTP (facilite l'utilisation de proxies et de pares-feu)
- ▶ Neutralité (peut-être utiliser sur n'importe quel transport),
- ▶ Indépendance du langage et de la plate-forme,
- ▶ Extensibilité (par exemple ajouter de la sécurité).

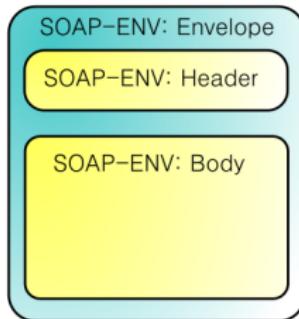
► Inconvénients

- ▶ léger alourdissement du trafic par l'utilisation d'XML (sur de gros volumes de données),
- ▶ approche service: description de la manière dont les applications communiquent: dépendance du couple client/serveur (contrairement à une approche ressource).

Format des messages SOAP

Les messages SOAP sont des documents [XML](#) contenant:

- ▶ *Envelope*: identifie le document XML comme étant un message SOAP (obligatoire).
- ▶ *Header*: contient des informations d'en-têtes (optionnel).
- ▶ *Body*: contient le corps de l'appel et de la réponse (obligatoire).
- ▶ *Fault*: inclus dans *Body*, contient des informations sur d'éventuelles erreurs (optionnel).



Exemple de requête SOAP via HTTP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body
        xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>

</soap:Envelope>
```

Requête d'une **méthode** qui va chercher le prix d'une action.

- ▶ `xmlns:` espace de noms XML

Exemple de réponse SOAP via HTTP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPriceResponse>
            <m:Price>34.5</m:Price>
        </m:GetStockPriceResponse>
    </soap:Body>

</soap:Envelope>
```

Réponse de la **méthode** appelée précédemment.

SOAP vs. JSON

- ▶ SOAP est capable de représenter des **graphes**, pas seulement des arbres, ou des enregistrements (objets),
- ▶ les messages SOAP peuvent être envoyés à **plusieurs** destinataires,
- ▶ SOAP peut **encrypter** des parties des messages afin qu'une partie seulement des destinataires puissent les lire (difficile en JSON).
- ▶ les messages SOAP sont **robustes**: en cas de coupure de connexion, SOAP envoie à nouveau le message.
- ▶ les spécifications **WSDL** sont plus neutres et universelles que les descriptions JSON (qui sont centrées sur les navigateurs).
- ▶ JSON est très **simple**.



Web Services Description Language

Grammaire XML permettant de décrire un service web - plus particulièrement, son interface publique.

2001 Version préliminaire 1.1

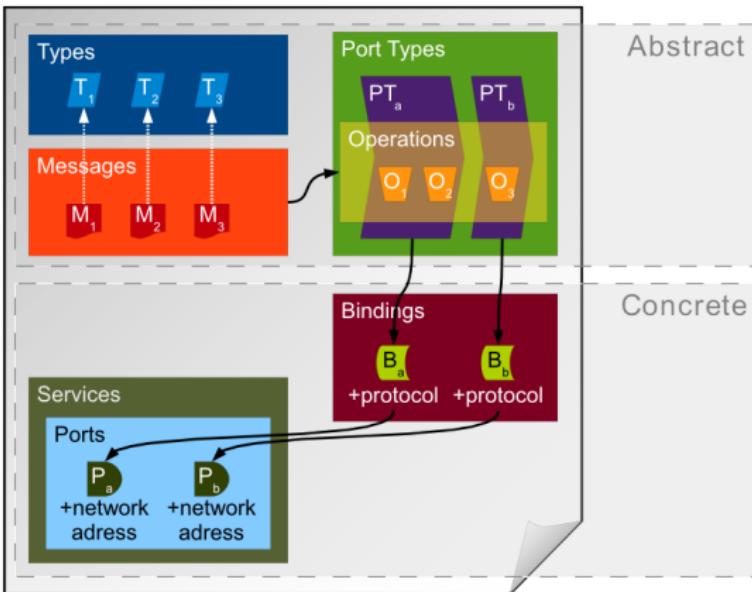
2007 Version 2.0 approuvée par le W3C

- ▶ donne une description analysable par une machine qui liste:
 - ▶ le protocole de communication (SOAP),
 - ▶ le format des messages communiqués,
 - ▶ les méthodes du service qui peuvent être invoquées,
 - ▶ la localisation du service.
- ▶ soutenu par IBM et Microsoft.

Structure du message

- ▶ *Service*: contient un ensemble de fonctions système exposées aux protocoles (*ports*),
- ▶ *Endpoint*: définit l'adresse de *connexion* au service (souvent une URL),
- ▶ *Binding*: spécifie l'interface, définit le style de liaison (RPC/document) et de transport (SOAP).
- ▶ *Interface*: définit le *service*, les *opérations* qui peuvent être effectuées, et les *messages* qui peuvent être échangés.
- ▶ *Operation*: définit les *actions SOAP* et la manière dont le message est encodé. Equivalent d'une méthode ou d'une fonction.
- ▶ *Type*: Décrit les données. Le langage schéma XML (XSD) est utilisé.

Structure du message



Exemple de code WSDL

```
<message name="GetStockPriceRequest">
  <part name="StockName" type="xs:string"/>
</message>

<message name="GetStockPriceResponse">
  <part name="Price" type="xs:double"/>
</message>

<portType name="GetStockPrices">
  <operation name="GetStockPrice">
    <input message="GetStockPriceRequest"/>
    <output message="GetStockPriceResponse"/>
  </operation>
</portType>
```

Partie **abstraite**: type de service et messages.

Exemple de code WSDL: liaison SOAP

```
<binding type="glossaryTerms" name="b1">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation>
    <soap:operation
      soapAction="http://example.com/GetStockPrice"/>
    <input><soap:body use="literal"/></input>
    <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

Partie **concrète**: liaison SOAP du service.

Universal Description Discovery & Integration

- ▶ UDDI est:
 - ▶ un **annuaire** indépendant des plateformes, basé sur l'XML, utilisé par les entreprises (au niveau mondial) qui peuvent publier leurs services **sur le web**,
 - ▶ un **mécanisme** pour enregistrer et trouver des applications implémentant des services webs.
- ▶ Initiative **libre** (2000).
- ▶ Utilisation la plus **fréquente**:
 - ▶ les entreprises l'utilisent pour lier dynamiquement les systèmes des clients à des implémentations.
- ▶ composé de:
 - ▶ pages **blanches**: informations sur l'entreprise (numéro de téléphone, adresse)
 - ▶ pages **jaunes**: classification de l'entreprise/du service basée sur les taxonomies standards (SIC, NAICS, UNSPSC). Potentiellement plusieurs pages jaunes (une par service) pour une page blanche.
 - ▶ pages **vertes**: description technique de l'accès aux services (information sur les liaisons). Potentiellement plusieurs pages vertes (une par liaison) par page jaune.

REpresentational State Transfer

Style d'architecture logicielle, ensemble de contraintes architecturales appliquées aux composants et données d'un système **hypermedia**.

- ▶ Introduit en 2000 par Roy Fielding (dans sa thèse).
- ▶ Ignore l'implémentation des composants et la syntaxe des protocoles et se concentre sur le rôle des composants, les contraintes d'interactions avec d'autres composants et leur interprétation des données.
- ▶ Approche Ressource: requêtes et réponses sont construites autour du transfert de représentation de ressources.
 - ▶ une **ressource** est un concept que l'on peut interroger/récupérer/accéder à (par exemple, une **base de données**),
 - ▶ une **représentation** est un document qui capture un état de la ressource (par exemple, un **document XML**).

Contraintes de style REST

Ensembles de **contraintes** à satisfaire pour être considéré ***RESTful***:

- ▶ **Client/Serveur:** une interface uniforme sépare les clients des serveurs.
 - ▶ **Séparation** du travail:
 - ▶ les clients ne s'occupent pas du **stockage**,
 - ▶ les serveurs ne s'occupent pas d'**UI**.
 - ▶ Développement et remplacement **indépendant**.
- ▶ **Pas d'état:** pas de contexte client stocké sur le serveur entre les requêtes.
 - ▶ Etat de la session est stocké sur le client.
 - ▶ Chaque requête contient toutes les informations client requises.
- ▶ **Cachable:** Les clients peuvent mettre des réponses **en cache**. Les réponses doivent contenir des informations sur leur cachabilité.

Contraintes de style REST

- ▶ **Système stratifié:** un client ne peut pas deviner s'il est connecté à un serveur final ou à un **intermédiaire**, ce qui permet **d'équilibrer la charge** et de mettre en place des **politiques de sécurité**.
- ▶ **Code à la demande (optionnel):** les serveurs peuvent modifier les fonctionnalités d'un client en envoyant du code executable.
- ▶ **Interface Uniforme:** indépendance entre clients et serveurs grâce à une interface simplifiant et découplant l'architecture (permettant aux composants d'évoluer indépendamment).

Interface Uniforme REST

- ▶ **Identification des ressources:** chaque ressource est identifiée dans les requêtes, par exemple par une URI,
- ▶ **Manipulation des ressources à travers leurs représentations:** quand un client possède la **représentation** d'une ressource, il a assez d'information pour **modifier ou supprimer** la ressource.
- ▶ **Messages autodescriptifs:** chaque message contient assez d'informations pour son traitement (quel parseur utiliser, par exemple). Les réponses indiquent leur cachabilité.
- ▶ **HATEOAS** (l'hypermedia comme moteur d'état d'application): les clients effectuent des **transitions d'état** uniquement à travers des actions identifiées dans l'hypermedia du serveur (par exemple par hyperliens)
 - ▶ les clients n'effectuent pas d'actions qui ne sont pas prévues dans les représentations.

API Web RESTful

Une API (Interface de Programmation Applicative) RESTful (ou Service Web RESTful) est une API implémentée de manière **RESTful** utilisant l'**HTTP**.

C'est une **collection** de ressources avec les aspects suivants:

- ▶ une **URI de base** pour l'API `http://saucisse.com/resources/`,
- ▶ un **type de media Web** pour les données supportées par l'API (souvent JSON, mais aussi XML, images),
- ▶ des **méthodes HTTP** standard supportées par l'API (GET, PUT, POST et DELETE),
- ▶ l'API doit être basé sur l'**hypertexte**.

API Google Maps

```
http://maps.googleapis.com/maps/api/geocode/json?  
address=4+Place+Jussieu+Paris&sensor=false
```

```
{
  "results" : [
    {
      "address_components" : [...],
      "formatted_address" : "4 Place Jussieu,
        Université Pierre et Marie Curie, Université
        Jussieu, 75005 Paris, France",
      "geometry" : {
        "location" : {
          "lat" : 48.8464111,
          "lng" : 2.3548468
        }
      },
      ...
    }
  ]
}
```

- ▶ méthode GET,
- ▶ JSON comme format média,
- ▶ <http://maps.googleapis.com/maps/api/geocode/> comme URI de base.

API Google Drive

```
GET https://www.googleapis.com/drive/v2/files/fileId  
POST https://www.googleapis.com/upload/drive/v2/files  
PUT https://www.googleapis.com/upload/drive/v2/files/fileId  
DELETE https://www.googleapis.com/drive/v2/files/fileId  
POST https://www.googleapis.com/drive/v2/files/fileId/copy
```

- ▶ actions du clients modifiant la ressource.
- ▶ méthodes GET, POST, PUT, DELETE.

API Twitter

```
GET https://api.twitter.com/1.1/search/tweets.json?
q=%23freebandnames&since_id=24012619984051000
&xmax_id=250126199840518145&result_type=mixed&count=4
```

```
{
  "statuses": [
    {
      "coordinates": null,
      "favorited": false,
      "truncated": false,
      "created_at": "Mon Sep 24 03:35:21 +0000 2012",
      "id_str": "250075927172759552",
      "entities": {
        "urls": [
        ],
        "hashtags": [
          {
            "text": "freebandnames",
            "indices": [
              20,
              34
            ]
          }
        ],
        "user_mentions": [
        ]
      },
      "in_reply_to_user_id_str": null,
      "contributors": null,
      "text": "Aggressive Ponytail #freebandnames",
      "metadata": {
        "iso_language_code": "en",
        "result_type": "recent"
      }
    }
  ]
}
```

Pratiques des APIs RESTful

- ▶ Quatre opérations (CRUD), implémentées par les quatre méthodes HTTP:
 - ▶ *Create*: POST
 - ▶ *Read*: GET
 - ▶ *Update*: PUT
 - ▶ *Delete*: DELETE
- ▶ Succès/Echec récupéré par un code statut HTTP.
- ▶ Ressources identifiées par URI, et représentées and XML, JSON, YAML, ...
- ▶ **RSDL**: équivalent WSDL pour les APIs RESTful

Approche Ressource vs. Approche Service

Modèle d'Interactions:

- ▶ **SOAP** échanges:
 - ▶ le serveur **garde des données de la session**,
 - ▶ les messages n'ont **que leur objet** comme contenu.
- ▶ **REST** opérations indépendantes:
 - ▶ serveur **sans état**,
 - ▶ les messages doivent contenir du **contexte**.

Utilité:

- ▶ **SOAP**: services **transactionnels**.
- ▶ **REST**: échanges de **données** ou de **documents**.

Approches et Versionnage

Quel cadre:

- ▶ **SOAP**: grandes entreprises (services internes).
- ▶ **REST**: monde du Web (API publiques).

Versionnage

- ▶ La question se pose **maintenant**.
- ▶ **Mises-à-jour** impossible (en pratique) à déployer sur tous les clients et tous les serveurs en même temps.
- ▶ Deux approches:
 - ▶ **Compatibilité** ascendante,
 - ▶ Inclusion des **numéro de version** dans les requêtes.

Examen Réparti I : Prévisions

(rien n'est contractuel)

- ▶ Cinq exercices:
 - ▶ un exercice de *Fair Threads* (en C)
 - ▶ contient une difficulté: link / unlink
 - ▶ similaire à ceux du TD02 et de PC2R.
 - ▶ un exercice de canaux synchrones en Go
 - ▶ contient une difficulté: interface{} ?
 - ▶ similaire à ceux des TD03/04 et de PC2R.
 - ▶ un exercice de canaux synchrones en OCaml
 - ▶ contient une difficulté: wrap
 - ▶ similaire à ceux des TD03/04 et de PC2R.
 - ▶ un exercice de modelisation (en Promela)
 - ▶ similaire à ceux du TD05
 - ▶ un autre exercice mystérieux.
 - ▶ mutexes ? multi-langage ? client-serveur internet ?
- ▶ Cours et TD 01-05.
 - ▶ pas de Web.

Mini-Projet

- ▶ couvre entièrement les TME 06-10 (inclus)
- ▶ mini-projet de réalisation d'une (petite) appli web
 - ▶ condensé du projet DAR
- ▶ sujet libre,
- ▶ contraintes à respecter,
 - ▶ notamment, utilisation d'une API Web externe
- ▶ Découpage des TMEs:
 - ▶ 06: design: choix de l'API et du sujet, dossier,
 - ▶ 07: serveur: BDD, servlets, Tomcat, appels à l'API externe,
 - ▶ 08: client: Javascript, appels asynchrones,
 - ▶ 09-10: finitions: interface, fonctionnalités, rapport.
- ▶ Application à rendre (lien *github* / accès *GitLab*, par exemple)
 - ▶ avec un rapport complet,
 - ▶ pour le Dimanche 23 Mai 23h59
(compte, avec les rendus intermédiaires, pour 20% de la note finale)

Conclusion

- ▶ Résumé:
 - ▶ Principes généraux de design des applis webs,
 - ▶ Approche Service,
 - ▶ Approche Ressources.
- ▶ TD / TME:
 - ▶ TD: produire des dossiers d'appli web.
 - ▶ TME: dossier du micro-projet.
- ▶ Séance prochaine:
 - ▶ Web II: Serveurs Web: BDD, Servlets.