UNIVERSITÉ DE NANTES

AJAX

Licence Professionnelle
& **MIAGE M2**

Laurent Guérin / V 1.7 / Février 2012

creative commons

BY NC SA

Introduction

AJAX

A J A X : qu'est-ce que c'est ?

- AJAX n'est pas …
  - Un produit de nettoyage
  - Un club de foot d'Amsterdam

- AJAX n'est pas non plus …
  - Une nouvelle technologie
  - Une norme
- AJAX = **A**synchronous **J**avaScript **A**nd **X**ML
  - → un **nom** pour désigner l'utilisation combinée de **technologies existantes** (cf. article de Jesse James Garrett, février 2005)
  - → une nouvelle façon de concevoir le dialogue « navigateur – serveur » ➔ « ***Web 2.0*** »

BY-NC-SA

AJAX

A J A X : qu'est-ce que c'est ?

- Présentation simple (simpliste ?) :

> « AJAX permet à une page web de solliciter le serveur sans se réafficher entièrement à chaque requête »

- => possibilité d'émettre des requêtes de façon **transparente** pour l'utilisateur
- Les échanges peuvent être **synchrone** ou **asynchrone** (en principe plutôt asynchrone)

BY-NC-SA

AJAX

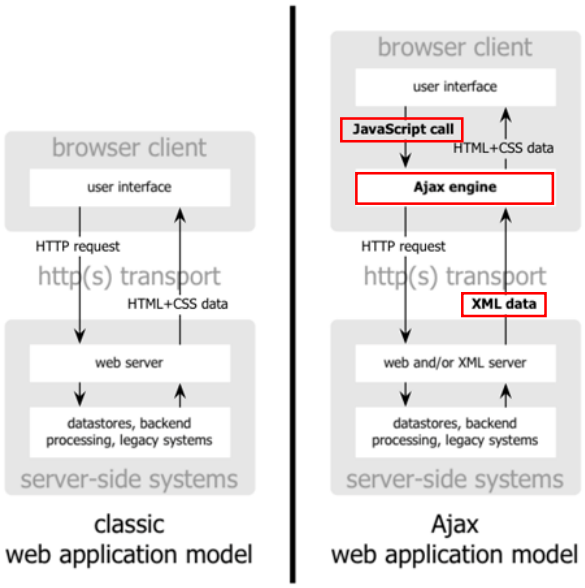
A J A X : qu'est-ce que c'est ?

- AJAX repose sur l'utilisation de l'objet natif « **XMLHttpRequest** » disponible dans les navigateurs les plus utilisés (Firefox, Internet Explorer, Safari, …)
- XMLHttpRequest permet d'effectuer des requêtes sur un serveur HTTP à partir de la page courante sans provoquer un rechargement de page …
  ➔ nombreuses possibilités :
  - effectuer des « requêtes transparentes »
  - retrouver des concepts « client / serveur »
  - améliorer l'ergonomie
  - etc…

BY-NC-SA

AJAX

Modèle « AJAX »



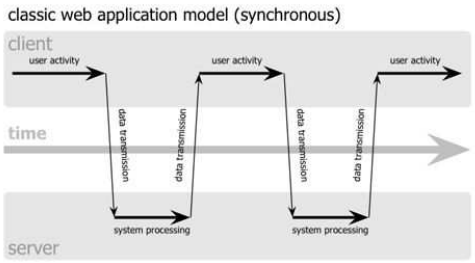
Modèle de dialogue avec le serveur **beaucoup plus flexible** que le modèle Web traditionnel

BY-NC-SA

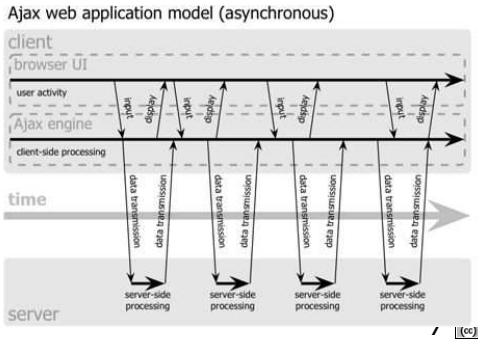
AJAX

Modèle « AJAX »

Requêtes HTTP classiques :
- assez « lourdes »
  (retour = page complète)
- synchrones
- visibles
  (réaffichage de la page)



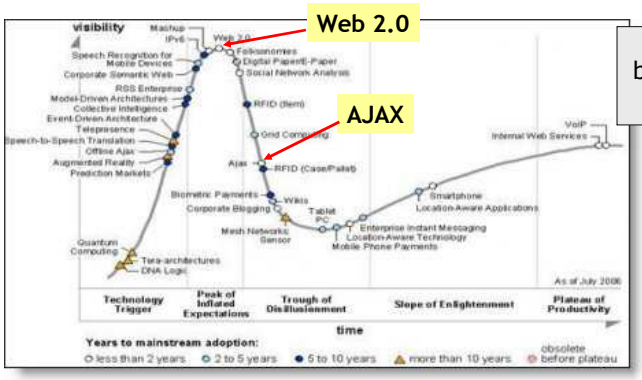
Requêtes **AJAX** :
- plus « fines »
- plus nombreuses
- asynchrones
  (utilisateur non bloqué)
- invisibles



AJAX

AJAX et Web 2.0 : effet de mode ?

- « Hype Cycle for Web Technologies »
  (Gartner Group **2006**)



- Ne pas confondre **AJAX**, **Web 2.0** et **RIA** …

AJAX

AJAX et RIA

- **R.I.A.** : « **R**ich **I**nternet **A**pplication »
  - Application Web disposant d'une « ergonomie riche »
  - Objectif : retrouver des ergonomies comparables à celles des applications « client lourd » tout en conservant les avantages du Web
- AJAX n'est pas directement lié à RIA
  - AJAX favorise la mise en place d'ergonomies riches,
  - Mais n'apporte pas les « composants DHTML » nécessaires (bibliothèques de « widgets »)
- Les frameworks « côté client » fournissent des solutions à différents niveaux : JavaScript, AJAX, RIA, etc…

BY-NC-SA

AJAX

AJAX et Web 2.0

- **Web 2.0 = notion floue**
  - Aucune définition (terme créé par l'éditeur O'Reilly)
  - Désigne une **nouvelle génération de sites web** : version 2.0 (par opposition au « Web 1.0 » ?)
- Caractéristiques du « Web 2.0 »
  - Syndication de contenu (fils **RSS** et **Atom**)
  - « ***mashup*** » (mutualisation de données d'origines différentes)
    => services web (SOAP,REST), transformation XML, …
  - Réseaux sociaux (« ***social networking*** »), blogs, Wiki, bookmarks, « nuages de tags », micro-formats, …
  - Ergonomie et look amélioré (grâce à **AJAX**)

BY-NC-SA

AJAX

AJAX : les contraintes

- AJAX = mise à jour dynamique et partielle de la page courante, donc …
- Problème de **référencement** par les moteurs de recherche (parties dynamiques non visibles par les moteurs d'indexation)
- Pas de gestion de l' **historique de navigation** (l'URL ne change pas)
- Pas de marquage de page dans les **favoris**

BY-NC-SA

AJAX

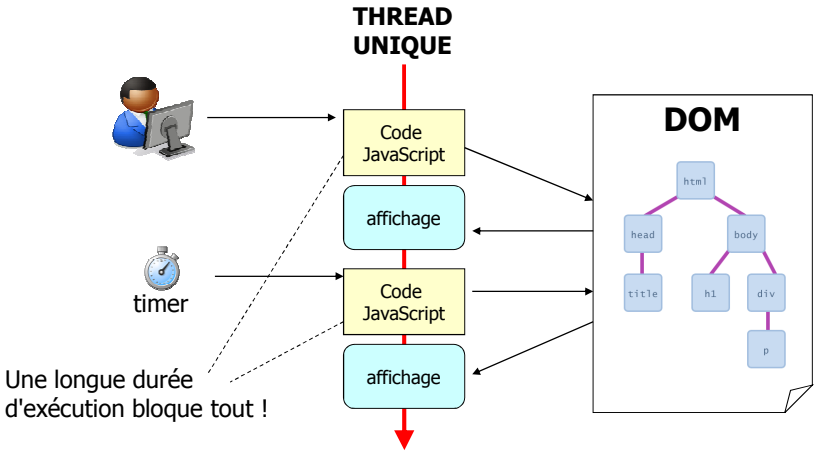
Fonctionnement du navigateur

- Les navigateurs sont "single-threaded"
  - Pour chaque page (ou onglet) il n'y a qu'**un seul thread** (et une seule file d'événements)
- Ce thread unique est partagé par
  - l'exécution du **code JavaScript**
  - l'**affichage** des composants graphiques ("browser UI" / "browser rendering")
- Donc …
  - le navigateur n'exécute jamais deux handlers d'événement à la fois
  - il ne déclenche jamais un timer tant que du code JavaScript est en cours d'exécution

BY-NC-SA

AJAX

Fonctionnement du navigateur



Exécution de code JavaScript limitée par les navigateurs :
FireFox : 10 s, Safari : 5 s, IE : 5 million statements, etc…

BY-NC-SA

L'objet XMLHttpRequest

AJAX

Objet XMLHttpRequest

- Objet du navigateur, utilisable en JavaScript
  - **Objet natif** sous Firefox/Mozilla, Safari, Opera, …
  - **ActiveX** sous I.E.5 et I.E.6
- Souvent appelé « **XHR** »
- Normalisé par le W3C :
  - « working draft » **W3C** (avril 2006)

  → objet standard natif dans tous les navigateurs

  → plus d'ActiveX, même dans IE (à partir d' IE 7)
- Constat :
  - Objet de bas niveau => à encapsuler
  - => framework « Javascript/AJAX » indispensable !

BY-NC-SA

AJAX

Objet XMLHttpRequest

- Création d'une instance de **XMLHttpRequest** :

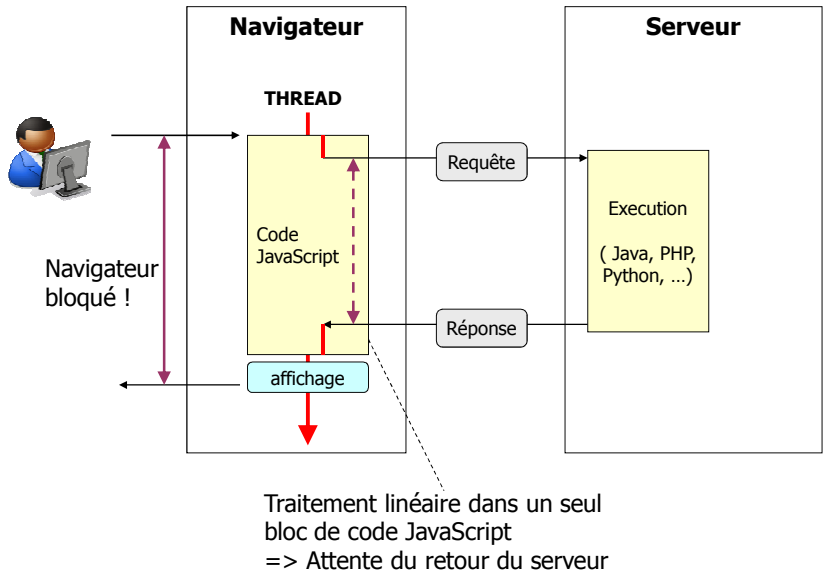
```
var oXHR = null ;
if (window.ActiveXObject)
{
    //--- ActiveX ( Internet Explorer 5 et 6 )
    oXHR = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest)
{
    //--- Objet natif ( Firefox, Mozilla, Safari, IE 7+, ..)
    oXHR = new XMLHttpRequest();
}
```

- Ensuite deux modes d'utilisation :
  - **Synchrone**
  - **Asynchrone**

BY-NC-SA

AJAX

Principe d'une requête synchrone



BY-NC-SA

AJAX

Objet XMLHttpRequest

- Requête **synchrone** :

Méthode http : "GET" ou "POST" **URL**

```
//--- Initialisation la requete (Methode, URL, flag)
oXHR.open("GET", "add?p1=5&p2=10", false);
```

false = synchrone

```
//--- Envoi de la requête au serveur
oXHR.send(null); // null : pas de contenu à envoyer
```

Traitement / serveur
Attente de la réponse ...
Navigateur bloqué

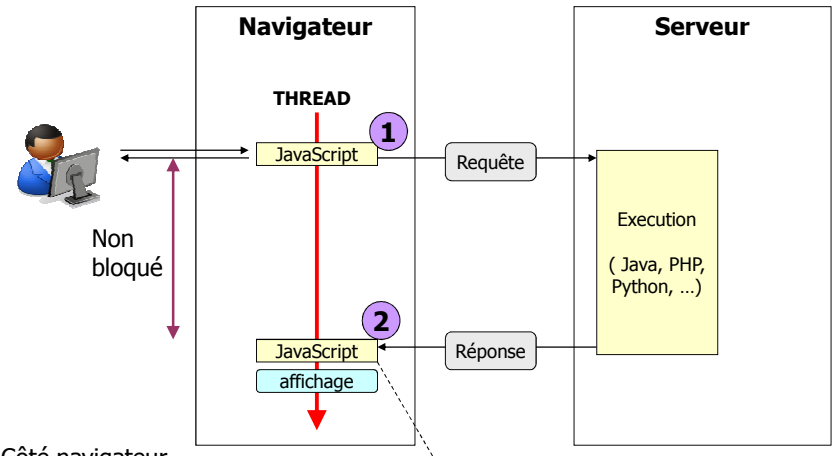
```
//--- Réponse au format TEXTE
var sText = oXHR.responseText ;
alert(oXHR.responseText);

//--- Réponse au format document XML (arbre DOM)
var oXml  = oXHR.responseXML ;
```


BY-NC-SA

AJAX

Principe d'une requête asynchrone



Côté navigateur
deux traitements séparés :
. 1) envoi de la requête
. 2) traitement de la réponse

Traitement de la réponse par une **fonction "callback"**

BY-NC-SA

AJAX

Objet XMLHttpRequest

- Requête **asynchrone** :

```
function processAjaxResponse() {
  // Exploitation de la réponse
}
```

```
//--- Initialisation la requete
oXHR.open("GET", "add?p1=5&p2=10", true );

//--- Affectation d'une « callback method »
oXHR.onreadystatechange = processAjaxResponse;

//--- Envoi de la requête au serveur
oXHR.send(null); // Send sans contenu
```

true = asynchrone

Fonction JavaScript « callback »

NB : Référence sur un objet de type « fonction » et non appel d'une fonction => pas de **( )**

Envoi au serveur

Changements d'état
readyState = 2, 3, **4** (**4** = réponse reçue)

Appel de oXHR.**onreadystatechange** → **Fonction callback**

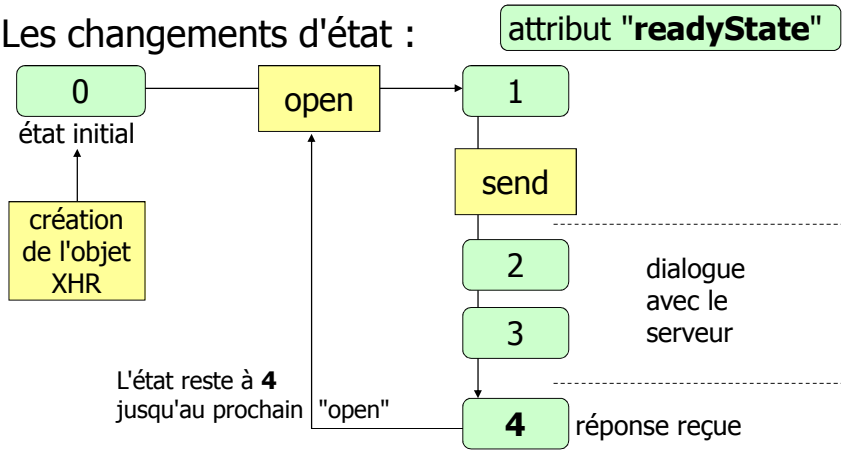
Le traitement JavaScript s'arrête là (provisoirement)
Le navigateur reprend la main (non bloqué)

BY-NC-SA

AJAX

Objet XMLHttpRequest

- Les changements d'état :



Le handler d'événement **onreadystatechange** est appelé à chaque changement d'état (1,2,3,4)

(appel de la fonction affectée par **onreadystatechange** = ***unefonction***)

BY-NC-SA

AJAX

Objet XMLHttpRequest

- Traitement de la réponse (par la fonction « **callback** » indiquée à l'objet XMLHttpRequest avant le « send ») :

```
//--- Méthode appelée à chaque changement d'état (1,2,3 & 4)
function processAjaxResponse()  {
    //--- Si l'état est 4 ( 4 = "COMPLETE" = réponse reçue )
    if ( oXHR.readyState == 4 )  {
        if ( oXHR.status == 200 )  {
            //--- Réponse au format TEXTE
            var sText = oXHR.responseText ;
            //--- Réponse au format document XML (arbre DOM)
            var oXml  = oXHR.responseXML ;
        }
    }
}
```

TEXTE

DOM XML

Contenu HTML (DIV,..)

```
document.getElementById("maTable").innerHTML = sText ;
```

API DOM

```
sValue = oXml.getElementsByTagName('prix').item(0).firstChild.data ;
document.getElementById("prix").value = sValue ;
```

Valeur d'un champ

BY-NC-SA

AJAX

Objet XMLHttpRequest & CallBack

- La "callback function" n'est pas une méthode de l'objet XHR => pour accéder aux attributs de l'objet elle doit avoir une référence sur cet objet.
- Différentes solutions
  - Stocker l'objet XHR au niveau "window", c'est-à-dire en tant que **variable globale** (donc accessible de partout) → pas très "propre" → à éviter !
  - Créer une **fonction anonyme** qui appelle la callback en lui passant l'objet XHR en paramètre

```
function myCallback(oXHR) {
   if ( oXHR.readyState == 4 )
   etc ...
}
oXHR.onreadystatechange = function() { myCallback(oXHR) } ;
```


BY-NC-SA

AJAX

Objet XMLHttpRequest & CallBack

- Techniquement, la "callback function" peut être utilisée à la fois pour des appels synchrones et asynchrones...
- Mais en **synchrone**, l'appel de la callback varie d'un navigateur à l'autre (ou même d'une version à une autre) :
  - un seul appel (quand readyState = 4)
  - un appel à chaque changement d'état (1,2,3,4)
  - "undefined" → erreur
- => Utiliser "**onreadystatechange**" uniquement pour les appels asynchrones

BY-NC-SA

AJAX

Objet XMLHttpRequest multi-instances

- Il est possible de créer **plusieurs instances** d'objets XHR dans une même page (dans le même objet "window" du navigateur).

Exemple : création d'un tableau de XHR

```
var XHRs = new Array() ;
function initXHR(n) {
    var oXHR = new XMLHttpRequest() ;
    XHRs[n] = oXHR ;
}
for ( var i=0 ; i < 5 ; i++ ) {
    initXHR(i) ;
}
```

- NB : un navigateur ne peut ouvrir qu'un nombre limité de "sockets" avec un serveur web.
- En cas d'utilisation simultanée les appels sont empilés par le navigateur.

BY-NC-SA

AJAX

Objet XMLHttpRequest en résumé ...

- XMLHttpRequest permet ...
  - De travailler en **Synchrone** ou **Asynchrone**
  - De récupérer une réponse au format **TEXTE** ou **XML**
- L'utilisation du terme **AJAX** est souvent inappropriée.
  - **A**synchronous → recommandé, mais pas toujours
  - **J**avascript → forcément
  - **X**ML → souvent remplacé par : texte, JSON, formats propriétaires, etc..
- De nombreux « frameworks AJAX » n'utilisent pas XML !

BY-NC-SA

AJAX

Objet XMLHttpRequest

Méthodes et attributs	*Description*
open(method, URL [, asyncFlag [, user [, password]]])	Initialisation d'une requête - method : « **GET** », « **POST** » (+ « HEAD », …) - URL : « http://xxx/xxx » - **asyncFlag** : « true » **asynchrone** « false » **synchrone**
setRequestHeader(name, value)	Défini un « header » dans la requête HTTP
send (content)	Envoi de la requête au serveur - content : données postées (dans le corps de la requête http) ou « null »
abort()	Abandon de la requête en cours
readyState *(attribut)*	0 → uninitialized = état initial (avant "open") 1 → loading = état après "open" 2 → loaded 3 → interactive 4 → complete = état final (réponse exploitable)
onreadystatechange *(événement)*	Handler d'événement « changement d'état » de la requête (états : cf. attribut « **readyState** »)

BY-NC-SA

AJAX

Objet XMLHttpRequest

Attributs & méthodes utilisables **après réception** de la réponse

Méthodes et attributs	*Description*
responseText *(attribut)*	Contenu de la réponse sous forme de **texte**
responseXML *(attribut)*	Contenu de la réponse sous forme d'un **arbre DOM** NB: sous IE l'arbre DOM n'est fourni que si le type de réponse est "**`text/xml`**", par exemple : Servlet : response.setContentType("text/xml");
status *(attribut)*	Statut HTTP (numérique) ex : 200 (OK), 404 (not found), …
statusText *(attribut)*	Message/texte correspondant au statut HTTP
getResponseHeader(« header »)	Retourne un « header » si header inexistant retourne NULL (FireFox) ou une chaine vide "" (IE)
getAllResponseHeaders()	Retourne une collection (« Map ») de « headers »

BY-NC-SA

Objet XMLHttpRequest et requête HTTP

AJAX

xhr.open ("POST", " /servlet/test ", true|false)

xhr.setRequestHeader("name", "value")

```
POST /servlet/test HTTP/1.0
Accept: */*
Connection: Keep-Alive
Host: www.societe.com
User-Agent: Generic

xxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx
xxx ( DATA ) xxx
xxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx
```

Request

Headers

Data
(POST, ...)

Type compréhensible par le serveur qui va traiter la requête : **XML**, **JSON**, **CSV**, format spécifique, ...

xhr.send ("xxxxxxxxxx")
ou
xhr.send (null) → pas de données (méthode « GET »)

BY-NC-SA

Objet XMLHttpRequest et réponse HTTP

AJAX

xhr.status xhr.statusText

```
HTTP/1.1 200 OK
Date: Wed, 19 May 2000 13:12:34 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.7
Etag: "2da0dc-2870-374039cd"
Accept-Ranges: bytes
Content-Length: 10352
Connection: Close
Content-Type: text/xml; charset=iso-8859-1

<root>
<tag1>
 ...
</tag1>
</root>
```

Status Code

Headers

xhr.getResponseHeader(name)
xhr.getAllResponseHeaders()

Body

xhr.responseText → body « tel quel »
ou
xhr.responseXML → arbre XML (DOM)

Si le contenu de la réponse est du **XML** (`Content-Type : text/xml`)
le parser XML du navigateur est utilisé pour construire un **arbre DOM**

BY-NC-SA

AJAX

Synchrone ou asynchrone ?

- **Synchrone** :
  - Navigateur bloqué jusqu'au retour de la réponse
  - Événements « empilés » par le navigateur
  - Impossibilité d'afficher une information visuelle pour indiquer à l'utilisateur qu'un traitement est en cours (problématique de rafraîchissement de l'écran après interprétation du bloc JavaScript)
- **Asynchrone** :
  - L'utilisateur peut continuer à utiliser le navigateur pendant une requête
  - Prévoir le cas des plusieurs requêtes émises en parallèle (traitement des réponses dans le désordre)
  - Prévoir un moyen de bloquer l'utilisation du navigateur

BY-NC-SA

Types de données échangées

AJAX

Transport des données

- Qu'est-ce que l'on transporte ?
  - Des « **données** » (des « objets »)
    - ex : XML, POST http
  - Des « **vues** » (données déjà mises en forme par le serveur)
    - ex : HTML, XHTML, XUL, …
- Les objectifs :
  - Disposer de format de données **indépendants des langages de développement**
  - Véhiculer des données **compréhensibles par les « deux côtés »** (client et serveur)
  - Eviter une analyse de flux (« parsing ») trop complexe (temps de réponse, empreinte mémoire, …)

BY-NC-SA

AJAX

Transport des données : XML

- En principe **AJAX => XML**
- **Avantages :**
  - Standard
  - Indépendant des langages
  - Utilisable pour …
    → des « données » (XML de données)
    → des « vues » (XML de présentation : XHTML, …)
  - Permet de transporter des objets
- **Inconvénients :**
  - Nécessite un « parser » XML côté serveur (côté client parsing DOM automatique)
  - Objets => mapper/wrapper XML

BY-NC-SA

AJAX

Transport des données : XML

- 1) dialogue **intégralement en XML**

Exemple : Echanges d'une « action Telosys » :

- Requête en XML (POST) :

```
<request  screen-name="S001"  screen-id="0"  action="get" >
</request>
```

Flux XML dans le corps de la requête http

- Réponse en XML :

```
<response screen-name="S001" screen-id="0" action="get" >

<return code="0" message="Ok" >
</return>

<data>
<employe Matricule="23" Nom="Toto" Prenom="Marcel" Contrat="0"
        DateNais="2005-01-03" Cadre="0" Agence="0" Salaire="0.0" />
</data>

</response>
```

Objet Java sérialisé en XML

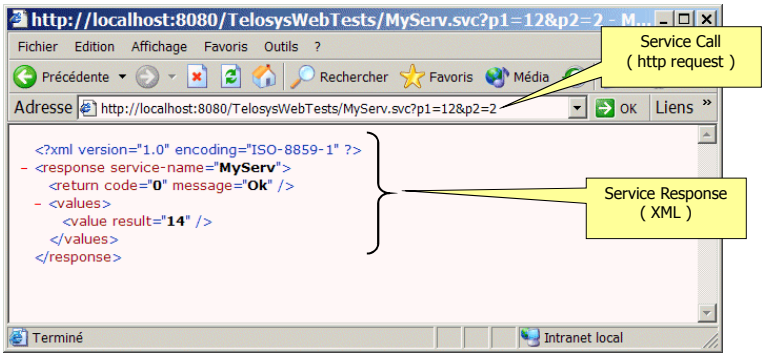
BY-NC-SA

AJAX

Transport des données : XML

- 2) dialogue **partiellement en XML**
  - Requête (GET ou POST) via une URL standard
  - Réponse en XML

Exemple : Appel d'un « service Telosys » :



BY-NC-SA

AJAX

Transport des données : Texte

- Le corps de la requête/réponse http peut contenir **tout type de texte**
- Données dans un format quelconque (texte brut, format spécifique, « CSV », ...)
- Du code JavaScript (ex. Google Suggest) :

eval (http.responseText)

Code JavaScript

- Des données au format « **JSON** » ...

BY-NC-SA

AJAX

Transport des données : JSON

- **JSON** : **J**ava**S**cript **O**bject **N**otation (http://json.org/)
  - permet de représenter des éléments imbriqués
  - plus léger que du XML
  - natif JavaScript => ne nécessite pas de "parser"
- Exemple :

```
{
  "fullname": "Bob Dylan",
  "org": "Music Consulting",
  "emailaddrs": [
    {"type": "work", "value": "bob@foo.biz"},
    {"type": "home", "pref": 1, "value": "bob@foo.tv"}
  ],
  "telephones": [
    {"type": "work", "pref": 1, "value": "+1 214 555 1212"},
    {"type": "fax", "value": "+1 214 555 1213"},
    {"type": "mobile", "value": "+1 214 555 1214"}
  ]
}
```

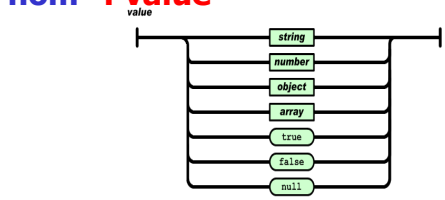

BY-NC-SA

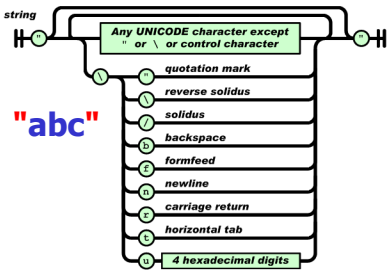
AJAX

JSON - Syntaxe

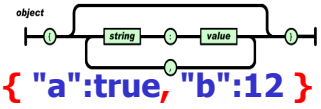
Une syntaxe simple, indépendante de tout langage de programmation, qui permet de représenter :
des **chaînes de caractères**, des **nombres**, des **booléens**, des **objets** et des **tableaux**

"nom" : value

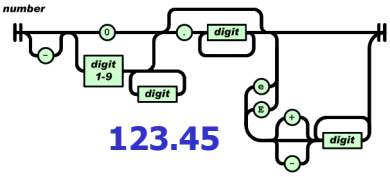




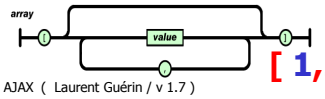
"abc"



{ "a":true, "b":12 }



123.45



[1,2,3]

BY-NC-SA

AJAX

JSON - Langages de programmation

- JSON est utilisable avec pratiquement tous les langages de programmation (via des librairies dédiées) : Java, C#, PHP, Python, etc…
- Pour **JavaScript** JSON est natif :
  - Définition littérale d'un objet JavaScript

```
var obj = { "fullname":"Bob", "age":32 };
```

  - Conversion d'une chaine de caractères JSON en objet

```
var obj = eval( '(' + jsonText + ')' );
var obj = JSON.parse( jsonText );          *
```

  - Conversion d'un objet en chaine de caractères JSON

```
var jsonText = JSON.stringify(obj);        *
```

NB : **eval** se contente d'évaluer (exécuter) le code JavaScript
=> peut être risqué si la chaine de caractère contient du code malveillant
=> utiliser de préférence "**JSON.parse**" (plus sécurisé)

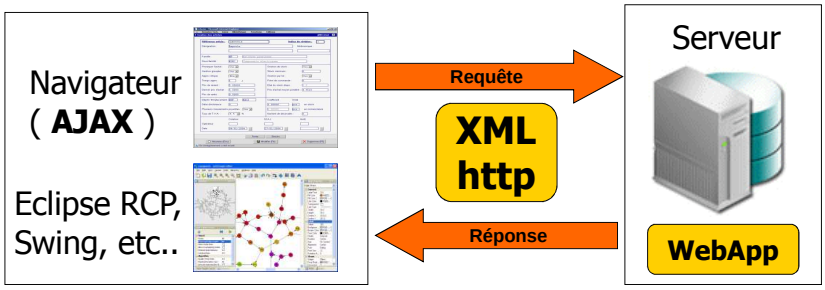
* Ajouté au prochain standard ECMAScript

BY-NC-SA

AJAX

Transport des données

- Certaines applications nécessitent de disposer d'une partie de l' IHM en « client lourd »
- ➔ les architectures doivent prévoir l'utilisation de **postes de travail hybrides**
  => privilégier des formats standards
  (XML, JSON …)



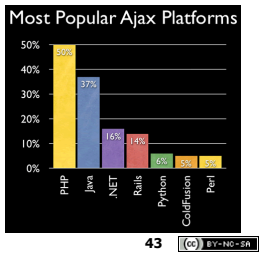
BY-NC-SA

Frameworks AJAX

AJAX

Frameworks AJAX

- Une offre pléthorique → à classifier :
- **Open Source** ou **Propriétaire**
- « **côté client** » **uniquement**
  - Orienté présentation (effets visuels, drag & drop, widgets, …) + appels AJAX
  - Exemples : *Prototype*, Jquery, *DOJO* (JSON-RPC), *script.aculo.us*, *OpenRico* (XML-RPC)
- « **côté client** » **et** « **côté serveur** »
  - AJAX – **Java** : *DWR*, *Echo2* (XML-RPC) , *Telosys* (XML-RPC)
  - AJAX – **PHP** : *CPAINT*
  - AJAX – **Ruby** : *Ruby on Rails*
  - AJAX – **.Net** : *ATLAS* (Microsoft)

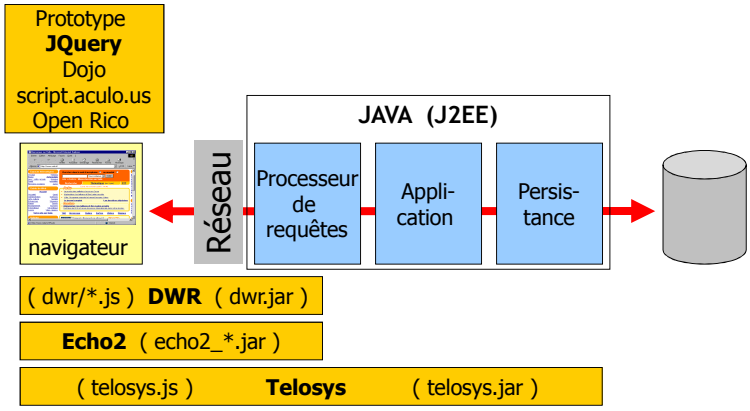


BY-NC-SA

AJAX

Frameworks AJAX

- Positionnement de différents frameworks :



BY-NC-SA

AJAX

Frameworks AJAX

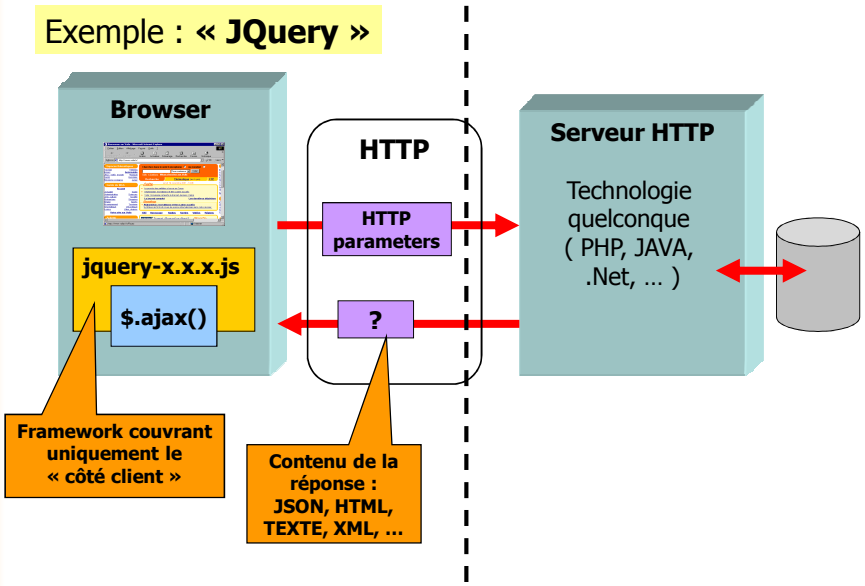
- Quelques critères de choix
  - Non intrusif
  - Respect des standards
    - JavaScript : ECMAScript
    - Format des données : XML, JSON, …
    - Java : Objets « *pure POJO* »
  - Minimum de code (globalement & minimum de code JavaScript en particulier)
  - Intégration ou pas avec la couche de présentation
    => lien avec la technologie serveur
    (ex : taglibs JSP)
  - Préserver l'interopérabilité
    => échanges au format XML (ou JSON)

BY-NC-SA

AJAX

Framework « côté client »

Exemple : « **JQuery** »



BY-NC-SA

AJAX

Exemple avec JQuery

- Un appel AJAX avec **JQuery** repose sur **$.ajax()**.

Exemple :

```
$.ajax({
  url: "req1.php", // url qui va être appelée
  type: "GET",     // methode HTTP ( "GET" par défaut )
  cache: false,    // pas de mise en cache
  headers: { "my-header" : "value" }, // HTTP header
  success: function(data, textStatus, oXHR){ // si requête OK
    afficher(data); // appel fonction avec données reçues
  },
  error: function(oXHR, textStatus, errorThrows){ // si erreur
    // traiter l'erreur
  }
});
function afficher(data){
  $("#contenu").empty(); // vide DIV
  $("#contenu").append(data); // met "data" dans DIV
}
```

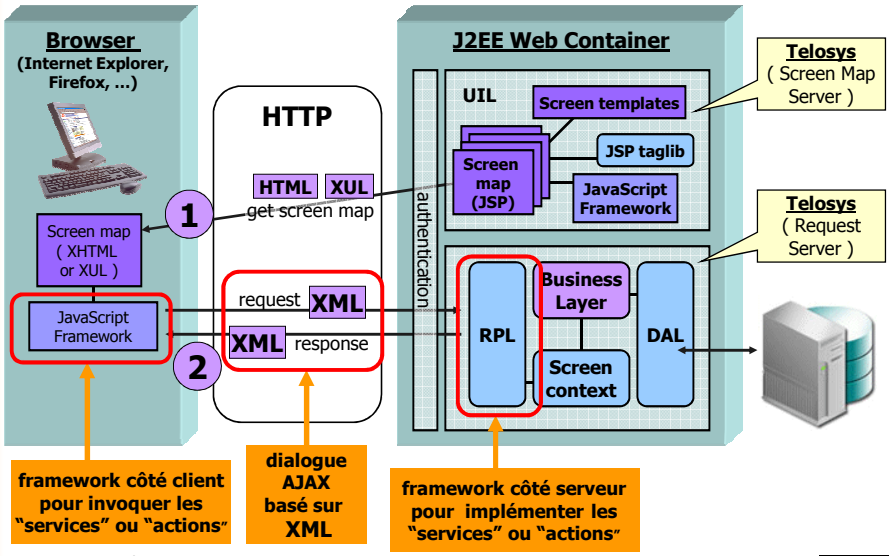
Il existe aussi des formes "courtes et spécialisées" pour des requêtes spécifiques :
$.get() **$.post()** **$.load()** **$.getJSON()** **$.getScript()**

(cc) BY-NC-SA

AJAX

Framework « global »

Exemple : « **Telosys** »



(cc) BY-NC-SA

AJAX

Exemple avec Telosys

- Appel d'un service de type RPC implémenté sous la forme d'une classe Java côté serveur

```
var addService = telosys.getService("Add", processAddResponse);
// Appel du service "Add" implémenté en Java côté serveur
addService.call("12","3");

function processAddResponse( oResponse ) {
  if ( oResponse.reportsErrors() ) {
    // Si la réponse comporte des erreurs : les afficher
    oResponse.showAllErrors();
  }
  else {
    alert ( "OK : Result = " + oResponse.getValue("result"));
    // Valorise le champ "result" avec la valeur renvoyée
    fwkSetValue("result", oResponse.getValue("result"));
  }
}
```


BY-NC-SA

Limites et contraintes

AJAX

Les limites de XHR

- L'objet XHR n'a pas de visibilité (et donc de contrôle) sur tous les échanges entre le navigateur et le serveur.
- Exemples : codes de redirection (302) ou d'authentification (401)

2 échanges http, mais XHR n'en voit qu'un

BY-NC-SA

AJAX

Les limites de XHR

- Sous **Internet Explorer 4**, **5** et **6** XHR est un **ActiveX**, donc ...

Objet "normal" à partir d'I.E. 7

  - Si l'utilisation des ActiveX n'est pas autorisée par la configuration du navigateur => pas d'AJAX.
  - Il ne s'agit pas d'un véritable objet JavaScript => on ne peut pas lui ajouter des attributs dynamiquement.

    Exemple :

```
oXHR.myAttribute = "toto" ;
```

    → Possible sous FireFox
    → Impossible sous I.E. 4, 5, 6

BY-NC-SA

AJAX

Traitements (granularité & localisation)

- Ne pas « abuser » des requêtes AJAX
  - Penser à la **montée en charge** …
  - Contrôles champ par champ → suffisant
  - Contrôles sur saisie de caractères → excessif !
- Localisation des traitements :
  - Est-il judicieux de coder des « règles métier » en JavaScript ?
  - Equilibre à trouver :
    - Contrôles de surface → côté client (JavaScript)
    - Règles métier → côté serveur (Java)

BY-NC-SA

AJAX

HTML dynamique & Javascript

- Valorisation du contenu d'un élément de la page (DIV, …) Exemple :

```
document.getElementById("myDiv").innerHTML = sText ;
```

- Seule la « présentation » (HTML & CSS) est prise en compte
- Le code Javascript n'est pas interprété lors de l'affectation, les fonctions et objets restent donc inconnus => inutilisables
- Contournement :
  récupérer les tags <script> contenus dans la réponse et forcer leur exécution

BY-NC-SA

AJAX

Sécurité

- Pour des raisons de sécurité, les requêtes AJAX ne peuvent être adressées qu'à une URL dont
  - le **protocole** (http)
  - le **host**
  - le **port** (80, 8080, ..)

  sont identiques à ceux de l'URL de la page d'origine (page qui émet la requête).
- Pour éviter les attaques de type « **cross-site scripting** »

Exemple :

```
XHR.open("GET","http://autreserveur",true);
```

→ lève une exception

BY-NC-SA

AJAX "Cross-Domain"

AJAX

AJAX "Cross-Domain"

- Des échanges de données entre une page web et un serveur différent du serveur d'origine sont souvent utiles ...
- Mais XHR ne le permet pas !

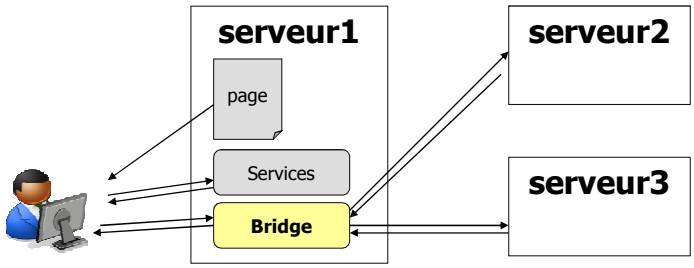
- Il faut donc utiliser d'autres moyens
- Par exemple :
  - Un "**bridge**" (ou "**proxy**") côté serveur
  - La technique "**JSON with Padding**" (**JSONP**)
    (ex : API Yahoo, Google Analytics, etc...)

BY-NC-SA

AJAX

AJAX "Cross-Domain" - Bridge/Proxy

- Puisque le navigateur ne peut pas invoquer directement un autre serveur, il suffit d'utiliser le serveur d'origine comme intermédiaire pour invoquer les services souhaités



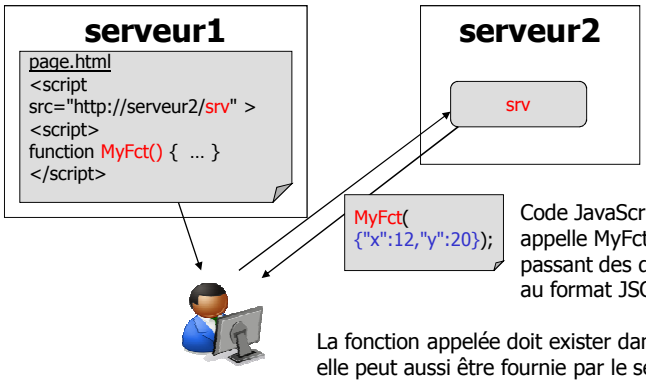
Fonctionnement correct et sécurisé, mais ...
. doublement des requêtes (perfs, trafic réseau,...)
. le "serveur 1" doit avoir accès à tous les autres serveurs

BY-NC-SA

AJAX

AJAX "Cross-Domain" - JSONP

- Cette technique repose sur une astuce assez simple : les fichiers JavaScript importés à l'aide d'une balise <script> peuvent provenir d'un autre domaine.



La fonction appelée doit exister dans la page, elle peut aussi être fournie par le serveur2

BY-NC-SA

AJAX

AJAX "Cross-Domain" - JSONP

- Comment utiliser cette technique à la demande, chaque fois que nécessaire ?
- Il suffit de créer dynamiquement des tags <script> dans le DOM.
  Exemple :

```
function periodicChecker() {
   eval("var old = document.getElementById('myScript')");
   if (old != null) {
     old.parentNode.removeChild(old);
     delete old;
   }
   var head = document.getElementsByTagName("head")[0];
   var script = document.createElement('script');
   script.id   = 'myScript';
   script.type = 'text/javascript';
   script.src  = "http://www.domain2.com/serv?param=abc";
   head.appendChild(script);
 }
```


BY-NC-SA

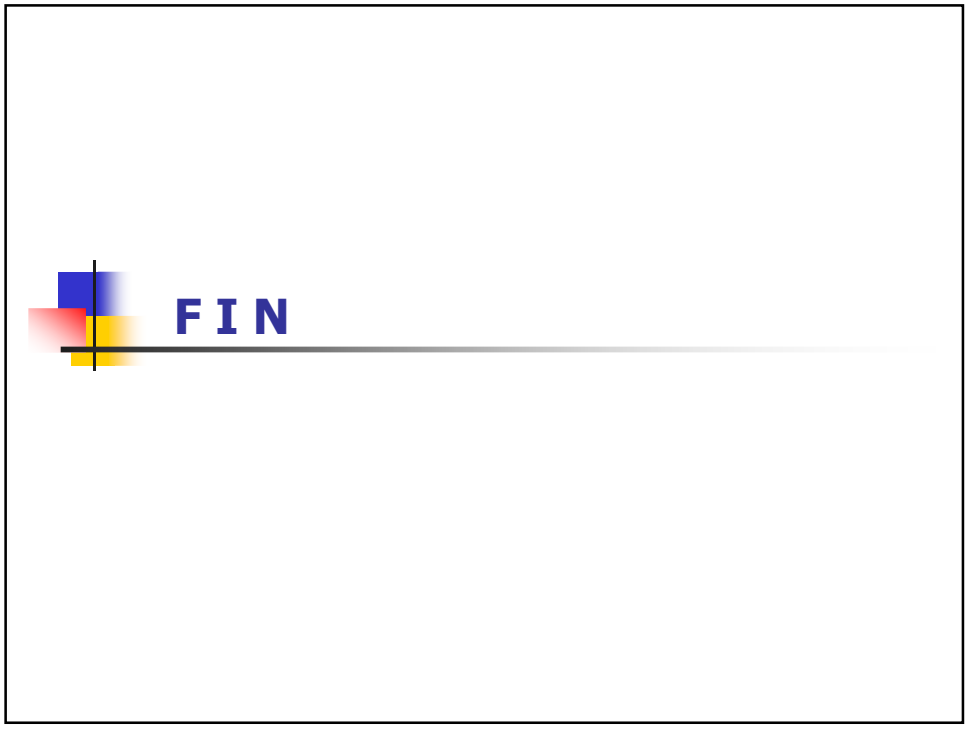
Ressources utiles

AJAX

Liens

- AJAX :
  - Ajax: A New Approach to Web Applications (de **J.J. Garrett**)
    http://www.adaptivepath.com/publications/essays/archives/000385.php
- W3C :
  - XMLHttpRequest Object (Working Draft – avril 2006)
    http://www.w3.org/TR/XMLHttpRequest/
- Outils & frameworks :
  - Telosys : http://www.telosys.org/
  - DWR : http://getahead.ltd.uk/dwr/
  - JQuery : http://jquery.com/
  - Dojo : http://dojotoolkit.org/
  - Echo2 : http://www.nextapp.com/platform/echo2/echo/
  - Prototype : http://www.prototypejs.org/

BY-NC-SA


FIN