Développement Web

Communication Client-Serveur

3I017

Laure Soulier - slides de Sylvain Lamprier

Sorbonne Université

Communication Client-Serveur

- Client :
  - Initie l'échange avec une **requête**
  - Mis en suspend lors du traitement de la requête
- Serveur
  - A l'écoute du client, traite sa requête et renvoie une **réponse**
  - Tourne en permanence (administrateurs système h24)
  - Peut interagir avec plusieurs clients en même temps
  - Mode de gestion des requêtes : itératif, concurrent

Communication Client-Serveur

- Protocole HTTP
  - Permet de demander et télécharger des pages Web stockées sur un serveur distant
    - Contenu HTML
    - Feuilles de style CSS
    - Fonctions Javascript
    - Contenu multimedia
- Navigateur client
  - Interprète le contenu HTML/CSS et construit l'arbre DOM du document correspondant
  - Traite les scripts Web appelés en réaction à divers évènements
  - Charge et affiche les contenus mulimedia (images, videos, etc...)

Communication Client-Serveur

1. Le protocole HTTP
2. Communication par génération
3. Communication par insertion
4. Problème de persistance des données
5. Communication asynchrone

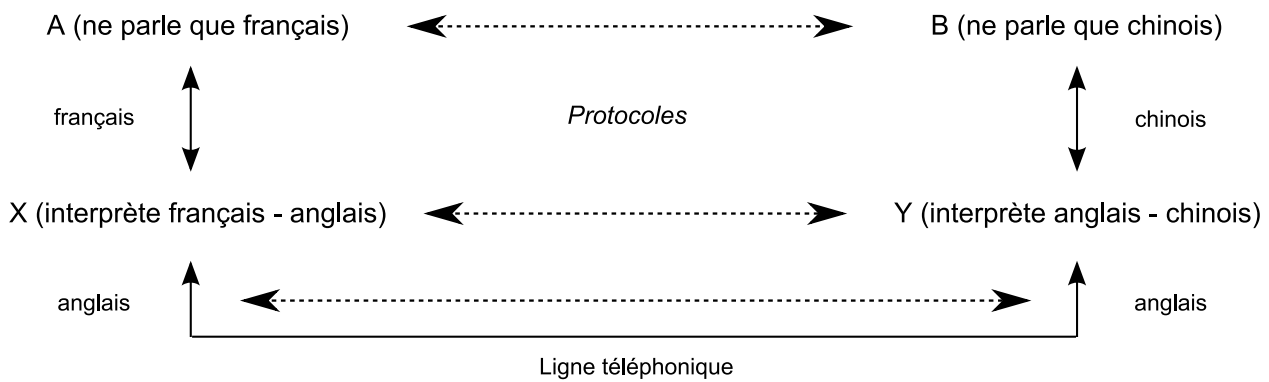
Le protocole HTTP

Les protocoles

- Protocole = Langage de communication
  - ⇒ Ensemble de règles et de procédures à respecter pour émettre et recevoir des données sur un réseau
  - ⇒ Différents protocoles selon ce que attend de la communication :
    - FTP : Échange de fichiers
    - HTTP : Transfert de pages Web
    - SMTP : Transfert de courrier electronique
    - ICMP : Gestion des erreurs de transmission
    - ...
- Deux grandes catégories de protocoles :
  - Les protocoles orientés connexion :
    Contrôle de transmission des données
  - Les protocoles non orientés connexion :
    La machine émettrice envoie des données sans prévenir la machine réceptrice, et la machine réceptrice reçoit les données sans envoyer d'accusé de réception à la première

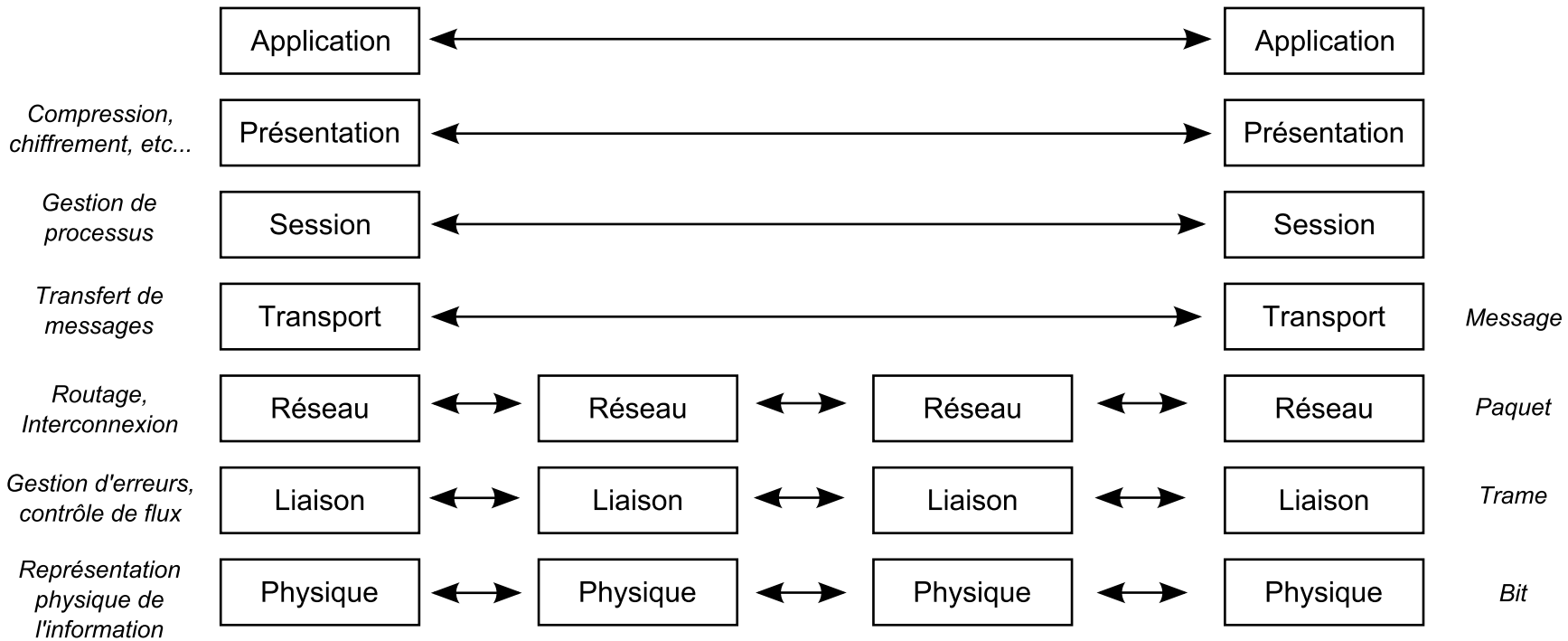
Les protocoles

- Les protocoles respectent un modèle en couches (ensemble de couches empilées) :
  - Chaque couche dialogue avec la couche juste au-dessus et celle juste au-dessous :
    - ⇒ Elle fournit des services à la couche au-dessus
    - ⇒ Elle utilise les services de la couche en-dessous
  - Chaque couche encapsule les données venant de la couche du dessus en y ajoutant ses propres informations
- Analogie : Communication entre deux personnes ne parlant pas la même langue



Les protocoles normalisés de l'ISO (International Standards Organisation)

Modèle OSI
(Open Systems Interconnection)



Le Protocole HTTP

- HyperText Transfer Protocol (HTTP)
  - Protocole développé pour la communication client-serveur dans le cadre du Web
  - Protocole de la couche application du modèle OSI
  - S'appuie sur le protocole TCP pour le transport des données
  - Utilise le port 80 (443 pour HTTPS)
- Manipulation de 2 types d'objets :
  - Un objet requête : HttpRequest
  - Un objet reponse : HttpResponse

L'objet Request

- Requête HTTP = texte envoyé au serveur par le navigateur :
  - Une ligne introductive contenant :
    - La méthode utilisée
    - L'url demandée
    - La version du protocole utilisé par le client (généralement HTTP/1.0)
  - Une partie d'en-tête contenant un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la requête et/ou le client
  - Une partie corps de la requête contenant les données à transmettre (dans le cas d'une transmission d'informations par la méthode POST)

Objet Request

```
METHODE URL VERSION
EN-TETE : Valeur
.
.
.
EN-TETE : Valeur
Ligne vide
CORPS DE LA REQUETE
```

L'objet Request

- Méthode = Commande informant le serveur sur l'action à effectuer concernant l'URL spécifiée
  - GET : demande le téléchargement d'une ressource (url spécifiée)
  - HEAD : demande des informations sur une ressource
  - POST : permet d'envoyer des données à la ressource
  - PUT : permet de remplacer ou d'ajouter une ressource sur le serveur
  - DELETE : permet de supprimer une ressource du serveur
- Pour l'envoi de données à partir d'un formulaire, on peut en fait utiliser GET ou POST
  - GET : Les informations sont ajoutées en fin de l'URL demandée (après "?")
  - POST : Les informations sont insérées dans le corps de l'objet Request (plus sûr)

Envoi de données à partir d'un formulaire

```
<form action="main.php" method="get">

<form action="http://localhost:8080/LI260/ConnexionServlet" method="post">
```

L'objet Request

- En-têtes de requête
  - Accept : Type de contenu accepté par le browser (text/html, text/plain, text/xml, application/json, audio/mpeg, ...).
  - Authorization : Données d'authentification du client auprès du serveur
  - Content-Type : Type de contenu du corps de la requête (application/x-www-form-urlencoded ou multipart/form-data)
  - Cookie : Cookies préalablement enregistrés par le serveur avec une entête de réponse Set-Cookie
  - Referer : URL de la page à partir de laquelle la requête a été effectuée
  - User-Agent : Logiciel formulant la requête
  - ...

L'objet Response

- Réponse HTTP = texte envoyé du serveur au navigateur :
  - Une ligne de statut contenant :
    - La version du protocole utilisé
    - Le code de statut
    - La signification du code
  - Une partie d'en-tête contenant un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la réponse et/ou le serveur
  - Une partie corps de la réponse contenant les données demandées

Objet Response

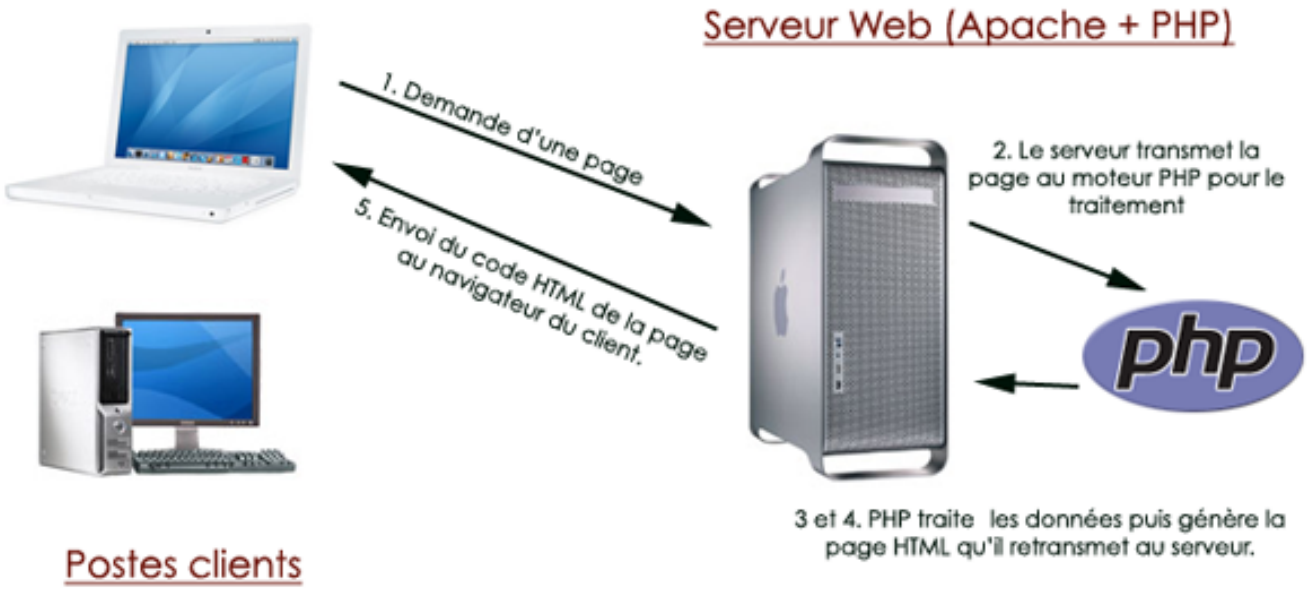
```
VERSION-HTTP CODE EXPLICATION
EN-TETE : Valeur
.
.
.
EN-TETE : Valeur
Ligne vide
CORPS DE LA REPONSE
```

L'objet Response

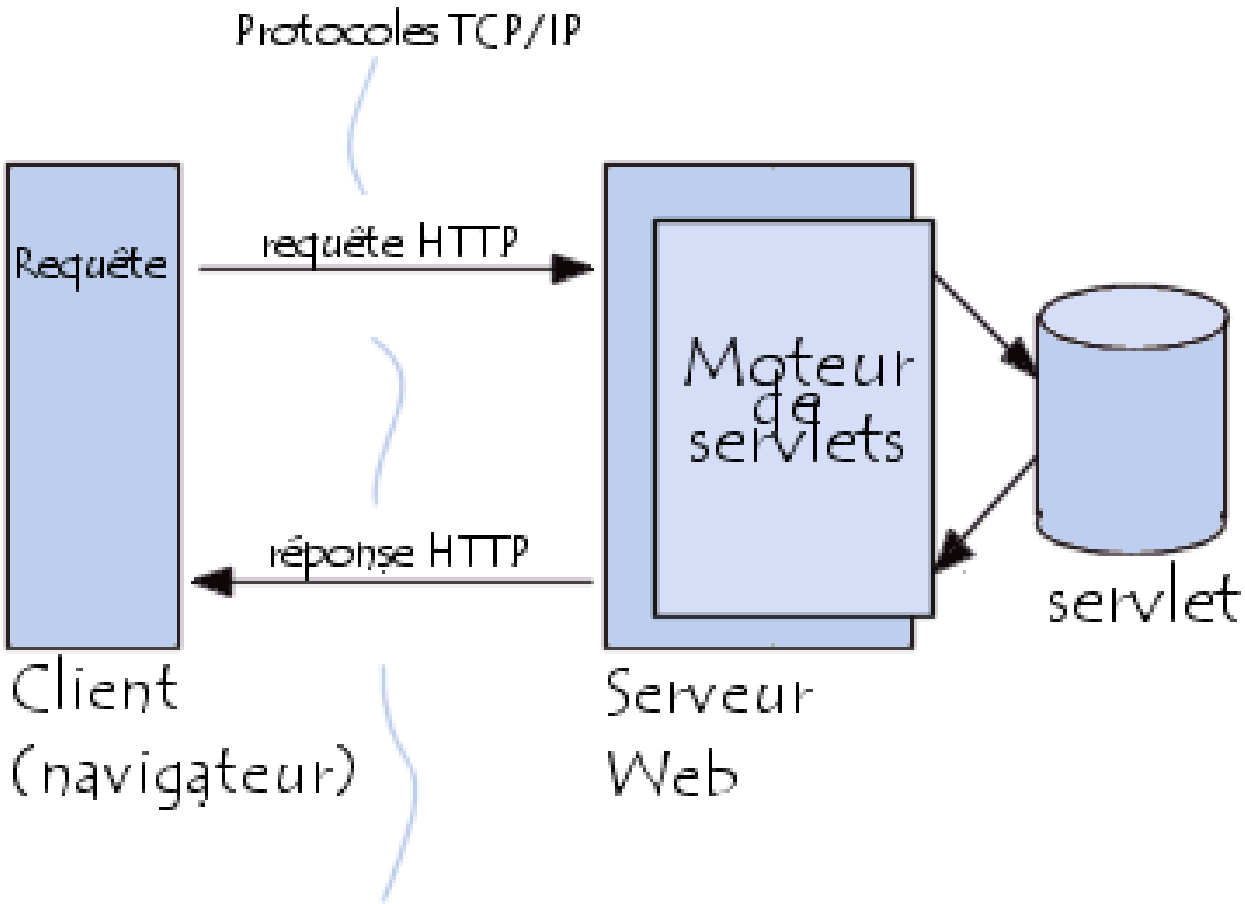
- En-têtes de réponse
  - Content-Type : Type de contenu du corps de la réponse (text/html, text/plain, text/xml, application/json, audio/mpeg, ...).
  - Last-Modified : Date de dernière modification de la ressource retournée
  - Location : En cas de redirection, contient l'adresse à charger par le navigateur
  - Server : Caractéristiques du serveur ayant envoyé la réponse
  - Set-Cookie : Cookies à enregistrer chez le client
  - User-Agent : Logiciel formulant la requête
  - ...

Génération de contenu

Communication par génération de contenu



Communication par génération de contenu



Communication par génération de contenu

- HTML pas prévu pour accepter des informations externes
  - ⇒ Une des solutions est alors de demander au serveur de générer le contenu d'une nouvelle contenant le résultat du traitement d'informations transmises.
- Exemple de scénario
  1. L'utilisateur saisit ses identifiants mots de passe dans un formulaire puis valide
  2. Les informations saisies sont transmises (de préférence en POST) au serveur
  3. Le serveur vérifie la validité des identifiants envoyés (par le biais d'une base de données)
  4. Selon la validité des informations fournies, le serveur génère une nouvelle page HTML (content-type=text/html).
  5. Si les identifiants sont valides, le serveur inclut des informations sur l'utilisateur connecté dans la nouvelle page
  6. La nouvelle page est retournée au client et chargée par le navigateur

Génération de page HTML

```
####################
 Page de Connexion
####################

<form action="www.lip6.serveur.fr/LoginServlet" method="POST">
<input type="text" name="login" />
<input type="password" name="pass" />
<input type="submit" value="Envoyer" />
</form>

####################
   LoginServlet
####################

Récupération des informations dans l'objet HTTP Request
Déclaration du type du contenu de l'objet HTTP Response : text/html
Vérification des login / mots de passe dans la base de données
Si (valide) alors
        Génération d'une page HTML contenant les infos de connexion
        (par méthodes de type print("<HTML> <HEAD> ... </BODY></HTML>"))
Sinon
        Génération d'une page HTML contenant le message d'erreur
FinSi
```

Communication par génération de contenu

- Serveur de génération de pages HTML
  - On passe de pages en pages générées dynamiquement
  - Permet de se déplacer sur des pages contenant les informations dont on a besoin
- Mais...
  - Il peut s'avérer difficile / fastidieux d'écrire l'ensemble des pages avec des print(), surtout si on a de nombreux cas à gérer
  - Le contenu n'est pas séparé des traitements
    - ⇒ Travail à plusieurs difficile
    - ⇒ Code rapidement très complexe
    - ⇒ Évolutions / Modifications coûteuses

Insertion de contenu

Communication par insertion de contenu

- Plutôt que de générer des pages HTML entières
  - ⇒ L'idée est de ne générer qu'une sous-partie de la page à retourner, le reste restant fixe
  - ⇒ Les pages Web contiennent des marqueurs informant des zones à générer dynamiquement
- Deux possibilités
  1. Intégrer un appel à un programme externe de génération (servlet dans notre cas) dans la page
  2. Directement écrire du code serveur dans les pages Web

Communication par insertion de contenu

- Intégration d'un appel à un programme externe
  - SSI (pour Server Side Includes) permet de faire appel à une servlet externe pour remplir une partie de la page

Fichier SSI : bonjour.shtml

```
<HTML>
<HEAD>
<TITLE>SSI</TITLE>
</HEAD>
<BODY>
<SERVLET CODE=Bonjour CODEBASE=http://localhost:8080/>
<PARAM NAME="nom" VALUE="Julien">
Si vous lisez ce texte, c'est que votre serveur Web ne supporte pas
les Servlets utilisées via SSI.
</SERVLET>
</BODY>
</HTML>
```

Communication par insertion de contenu

- SSI
  - Utile pour afficher du contenu stocké sur le serveur (ou des résultats de traitements)
  - Permet de ne pas à avoir à écrire l'ensemble de la page à partir de la servlet
- Mais...
  - Rigide
  - Le serveur doit supporter les directives SSI

⇒ Utile pour petites insertions dynamiques (comme JavaScript mais côté serveur)

Communication par insertion de contenu

- Écrire du code serveur dans les pages Web
  - Possible avec de nombreux langages (PHP, ASP, CGI, ...)
  - Pour Java : JSP
- JSP (pour Java Server Pages)
  - Pages HTML classiques (extension .jsp) mais contenant des balises < %...% > permettant d'inclure du code JAVA
  - Le code HTML et les parties de code JAVA de la page jsp sont inclus dans un servlet de génération de contenu HTML au moment du premier appel à la page

Insertion de contenu : les JSP

- Deux objets principaux :
  - request : l'objet représentant la requête venant du client.
  - out : l'objet représentant le flux d'impression en sortie
- Différentes directives (encadrées par <%@...%>)
  - Pour l'import de classes :
    <%@*import* = " *monpackage*.* "%>
  - Pour l'extension de servlets :
    <%@*extends* = " *maClassePersoHttp*" %>
- Expressions (encadrées par : <%=...%>)
  - Simples évaluations directement écrites sur la sortie
  - Permettent d'éviter l'écriture println()
  - Exemple : <%= *request*.*getParameter*(" *parametre*")%>
- Déclarations (encadrées par : <%!...%>)
  - Permet de déclarer des variables globales (qui persistent tant que l'on n'arrête pas le serveur)
  - Exemple : <%!*int nbVisites* = 0; %>

Communication par insertion de contenu : JSP

JSP

```
<HTML>
<HEAD>
<TITLE> JSP</TITLE>
</HEAD>
<BODY>
<%
        Code Java
%>
</BODY>
</HTML>
```

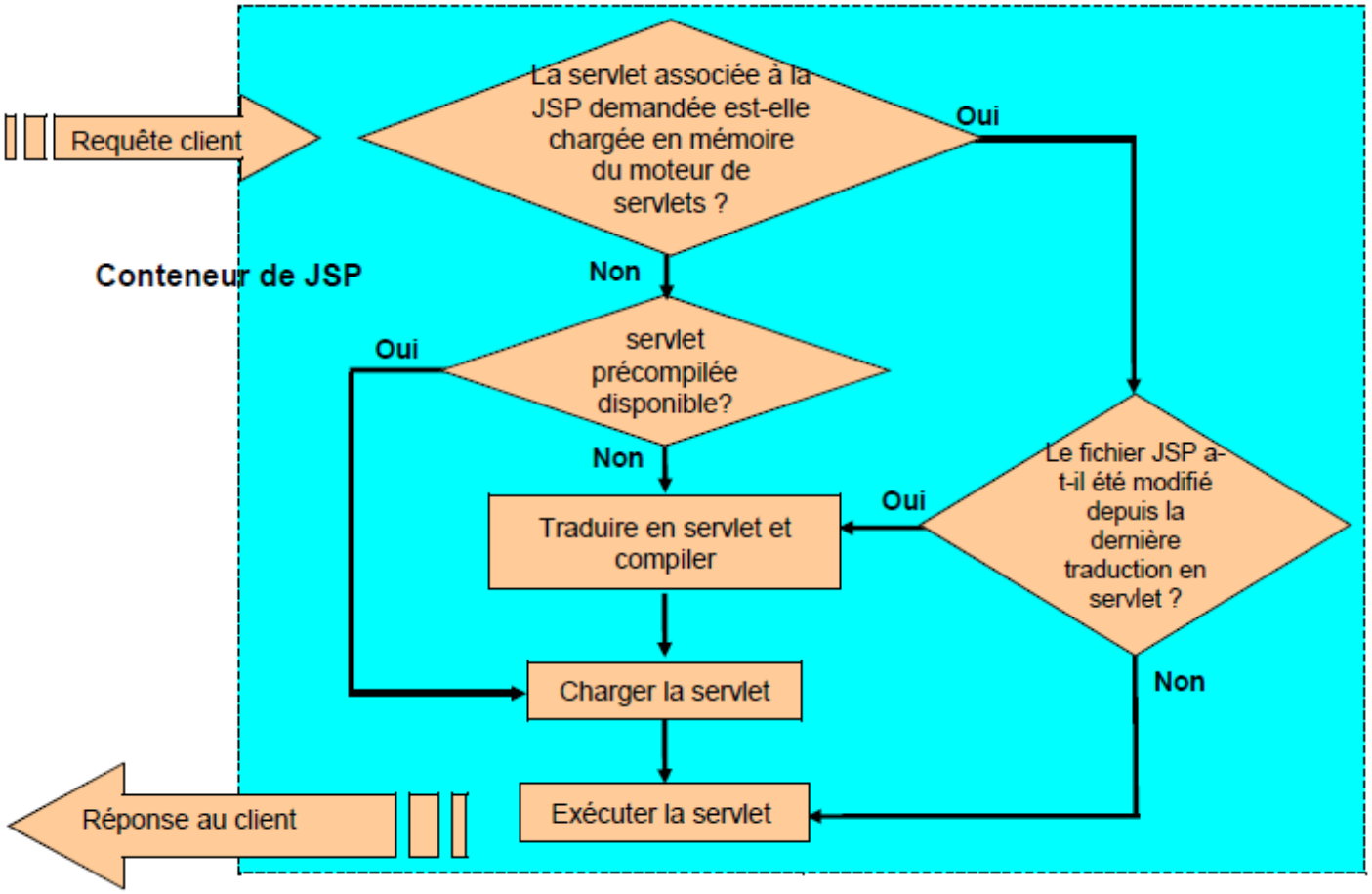
Exemple de JSP

```
<HTML>
<HEAD>
<TITLE> JSP</TITLE>
</HEAD>
<BODY>
<% if (request.getParameter("nom") == null) {
      out.println("Bonjour monde !");
   } else {
      out.println("Bonjour " + request.getParameter("nom")  + " !");
   }
%>
</BODY>
</HTML>
```

Autre exemple JSP

```
<%-- Importation d'un paquetage (package) --%>
<%@page import="java.util.*"%>
<html>
<head><title>Page JSP</title></head>
<body>

<%-- Déclaration d'une variable globale à la classe --%>
<%! int nombreVisites = 0; %>

<%-- Définition de code Java --%>
<% //Il est possible d'écrire du code Java ici
    Date date = new Date();
    // On peut incrémenter une variable globale pour compter le nombre
    // d'affichage, par exemple.
    nombreVisites++;
%>
<h1>Exemple de page JSP</h1>
<%-- Impression de variables --%>
<p>Au moment de l'exécution de ce script, nous sommes le <%= date %>.</p>
<p>Cette page a été affichée <%= nombreVisites %> fois!</p>
</body>
</html>
```

Communication par insertion de contenu



Communication par insertion de contenu

- JSP
  - Utile pour afficher du contenu stocké sur le serveur (ou des résultats de traitements)
  - Utile pour envoyer du contenu au serveur
  - Permet de générer facilement des pages Web dynamiques
- Certainement l'une des meilleures solutions (AJAX excepté) mais...
  - Lenteur de réponse lors du premier appel (traduction et compilation nécessaires)
  - Code relativement complexe
  - Séparation peu évidente entre contenu et traitements
  - Comme les SSI, rechargement global d'une page même pour des modifications minimes (communication synchrone)

Communication Client - Serveur

- Problème de persistance des données
  - Un certain nombre de données sont chargées dans la page du navigateur
    - Identifiant de connexion
    - Informations en provenance du serveur
    - Informations en provenance de l'utilisateur
    - Structures de données construites par le client

  ⇒ Comment faire en sorte que ces informations ne soient pas perdues après communication avec le serveur ?
    - Que ce soit dans le cadre d'une communication par génération ou par insertion, on change de page et donc les infos de la page de départ sont perdues...

Communication Client - Serveur : Persistance des données

- Différentes possibilités
  - Utilisation de cookies
    - \+ Persistance possible même après redémarrage du client
    - – Problème pour les informations volumineuses
    - – Certains utilisateurs refusent les cookies
    - – Détournement de cookies
  - Utilisation de l'adresse IP
    - \+ Permet de savoir d'où vient la requête
    - – Uniquement pour le suivi d'utilisateur
    - – Problème de partage d'IP / d'ordinateur
    - – Problème avec utilisateur à IP non fixes
  - Renvoi des informations à chaque communication
    - Utilisation de l'URL pour GET et de champs de formulaire cachés pour POST
    - \+ Bonne maitrise des informations échangées
    - – Gestion difficile (le renvoi systématique peut s'avérer complexe / fastidieux)
    - – Forte consommation de bande passante

Communication Client - Serveur : Persistance des données

- Différentes possibilités
  - Utilisation de window.name
    - \+ Propriété modifiable et pas rechargée lors du changement de page
    - \+ Possibilité de stocker des objets complexes dans cette propriété
    - – Différents onglets / fenêtres ne partagent pas cette propriété
    - – Sécurité : Informations stockées disponibles à d'autres sites ouverts dans le même onglet
  - Authentification HTTP
    - Le serveur demande un certificat pour accorder l'accès à une page web
    - \+ Permet un bon suivi de l'utilisateur
    - – Problème de partage de compte
    - – La demande de certificat peut effrayer l'utilisateur
    - – Uniquement pour le suivi d'utilisateur

Communication Client - Serveur : Persistance des données

- Différentes possibilités
  - Utilisateur d'objets locaux partagés
    - Cookies flash gérés par Adobe Flash Player
    + Flash Player est un plugin très répandu
    + La taille limite pour un objet local partagé est de 100 kB
    + Le contrôle des cookies flash est distinct des contrôles des cookies classiques
    – Flash non supporté sur de nombreuses tablettes / smartphones
    – Flash plus ou moins voué à disparaître
  - Sauvegarde dans le navigateur
    - Certains navigateurs permettent de sauvegarder des données dans des objets persistants
    + Prévu dans HTML 5
    – Varie encore fortement selon les navigateurs

Communication Client - Serveur : Persistance des données

- Différentes possibilités
  - Utilisation de sessions
    - Session HTTP : Table de hachage permettant d'enregistrer toutes sortes d'informations côté serveur
    - L'objet Session correspondant à une navigation en cours est repéré par un identifiant contenu dans l'objet Request
    + Permet de sauvegarder des objets complexes construits par le serveur pour une navigation donnée
    – Nécessité de stocker l'id de session quelque part (par défaut dans un cookie créé automatiquement par le serveur)
    – Ne permet pas de sauvegarder les données du client (sauf si on les envoie au serveur)

Communication Client - Serveur : Persistance des données

- Différentes possibilités
  - Ne pas changer de page
    - Si l'on ne change jamais de page, on n'a pas de problème de transmission de données entre pages
    - Avec des techniques de communication asynchrone, il est possible de ne recharger qu'une partie de la page
    - \+ Toutes les informations et structures chargées par le client restent valides, tout au long de la navigation
    - – Concerne uniquement les structures côté client (mais possibilité d'utiliser des sessions)

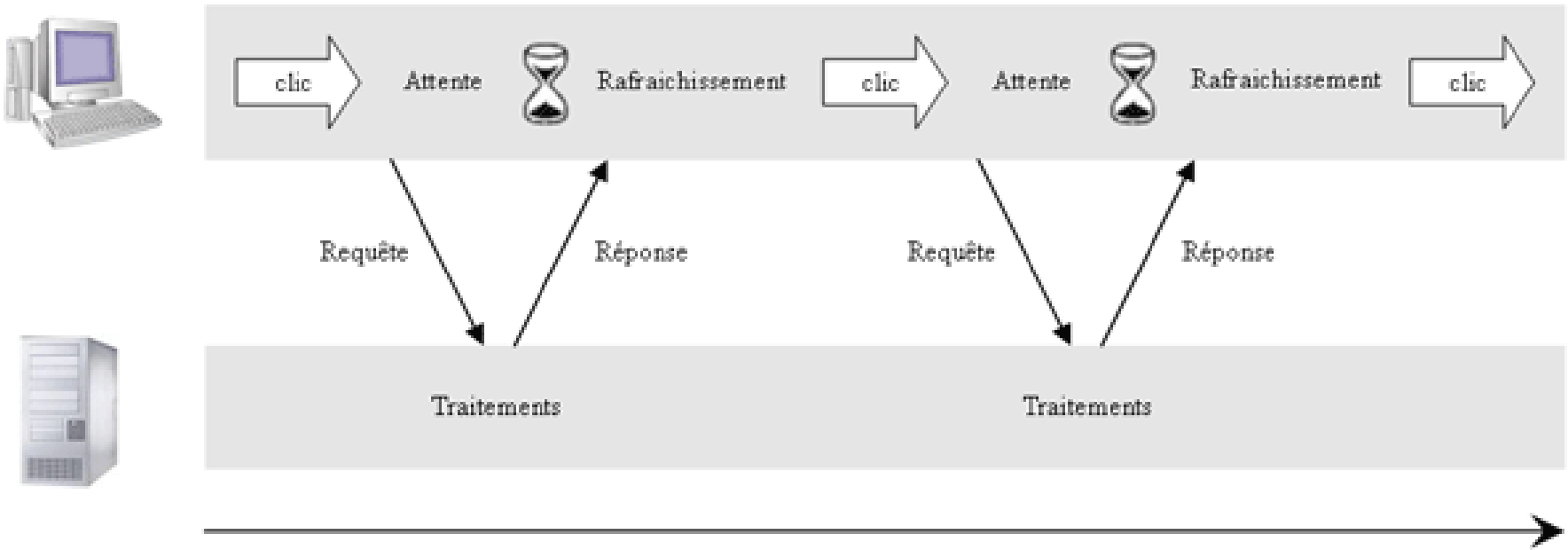
Communication asynchrone

Communication Client - Serveur

- Communication classique
  1. Envoi de données au serveur
  2. Le serveur traite les données, génère une réponse et la retourne au client
  3. Le client charge la réponse reçue

  ⇒ Navigation peu fluide

  ⇒ Persistance des données difficile

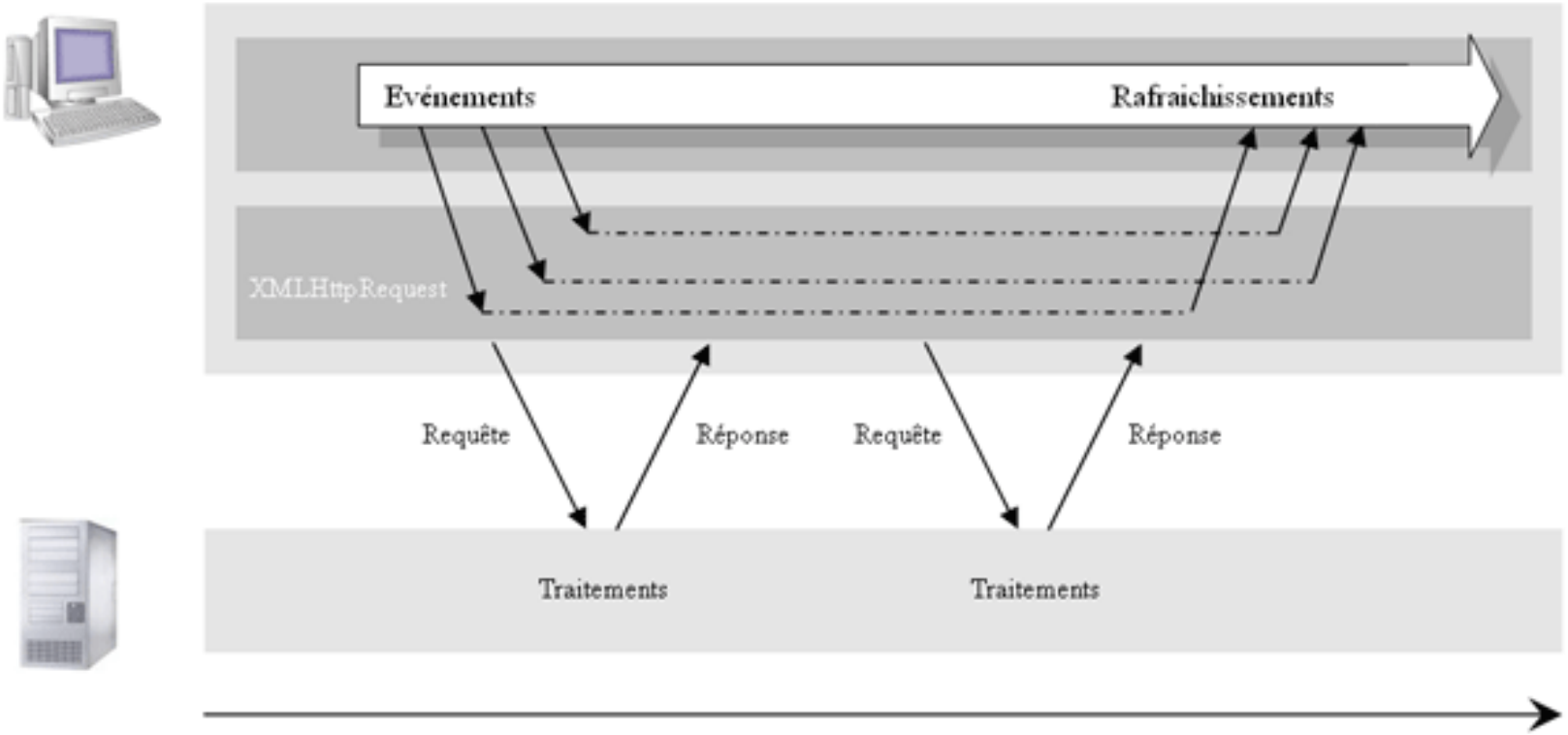


Communication Client - Serveur

- Communication asynchrone
  1. Un objet intermédiaire permettant la communication est créé par le client.
  2. Un gestionnaire de réponse lui est associé.
  3. Un script de la page client envoie les données à l'objet intermédiaire
  4. L'objet intermédiaire se charge de former la requête et de la transmettre au serveur
  5. Le serveur retourne une réponse au format attendu par le navigateur (xml,text,json,html...)
  6. Le gestionnaire de réponse traite les informations reçues et réalise les modifications nécessaires dans l'arbre DOM.

Communication Client - Serveur

- Communication asynchrone



⇒ Pas besoin d'attendre le traitement de la requête pour continuer la navigation

⇒ Permet de conserver le contexte de la requête (données précédemment chargées par le navigateur)

Communication asynchrone

- Première solution : utilisation de <iframe></iframe>
  - La balise iframe permet de charger le contenu d'un fichier (renseigné par src)

1. Ajout d'un attribut target au formulaire pointant vers un iframe invisible
2. Au moment de la soumission du formulaire, la ressource pointée par l'attribut action est appelée normalement
3. Mais le résultat est chargé dans l'iframe pointée par target plutôt que dans la fenêtre principale

⇒ Pas de rechargement global / Pas de perte de contexte

- Si l'on affecte une fonction à l'évènement onload de l'iframe, on peut appliquer un traitement selon les éléments retournés par la ressource

⇒ Modification de certaines données de la page principale

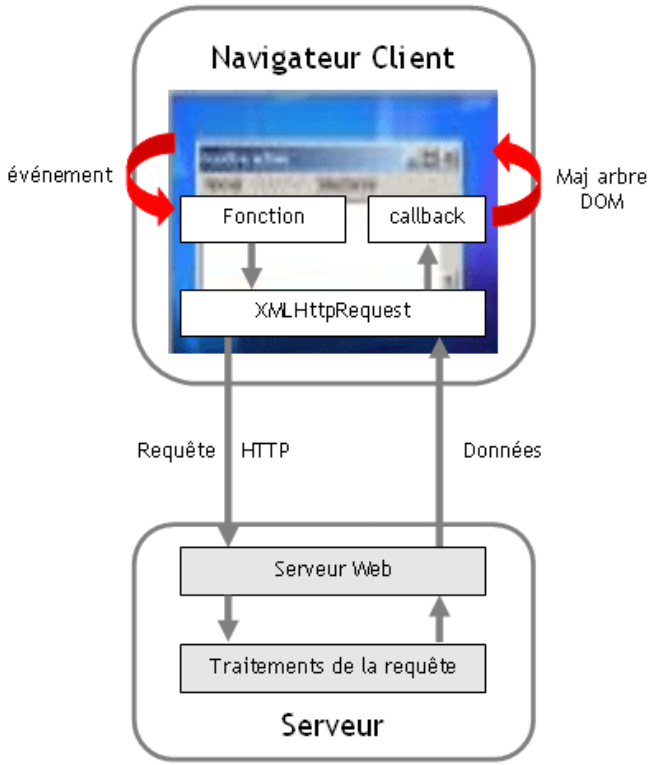
Exemple d'asynchrone avec <iframe> : Envoi de fichier

```
<iframe id="uploadTrg" name="uploadTrg" height="0"
          width="0" frameborder="0" scrolling="yes"></iframe>

<form id="myForm" action="AddFileServlet" method="post"
        enctype="multipart/form-data"  target="uploadTrg">

     File: <input type="file" name="file">

     <input type="submit" value="Submit" id="submitBtn"/>

</form>

<script type="text/javascript">
$("iframe").load(function(){
   alert("Upload ok");
});
</script>
```

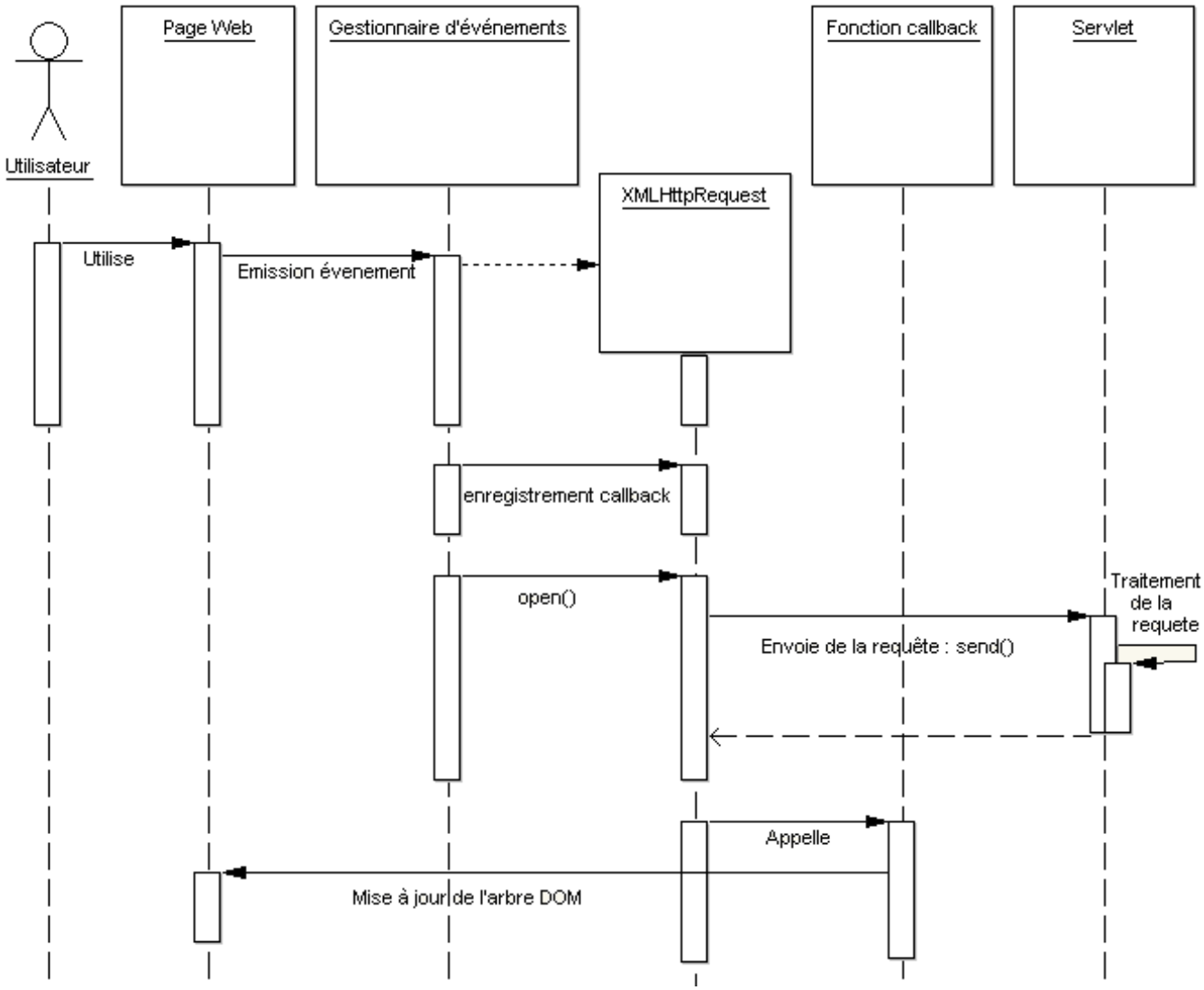
- Limitations de l'asynchrone par iframe :
  - Réponses de requêtes simultanées peuvent se chevaucher
  - Pas de contrôle évident du status de la communication

Communication Client - Serveur

- AJAX (Asynchronous Javascript and XML)
  - Objet central : l'objet Javascript XMLHttpRequest défini dans tous les navigateurs récents
  - Initialement prévu pour communiquer au format XML mais ce n'est pas une obligation.



Communication asynchrone



Communication Asynchrone : AJAX

- 3 grandes étapes
  - Création de l'objet XMLHttpRequest
  - Génération de la requête
  - Gestion de la réponse

Création de l'objet XMLHttpRequest

```
function getXMLObject()  //XML OBJECT
{
   var xmlHttp = false;
   try {
     xmlHttp = new ActiveXObject("Msxml2.XMLHTTP"); //Anciens navigateurs Microsoft
   }
   catch (e) {
     try {
       xmlHttp = new ActiveXObject("Microsoft.XMLHTTP"); // Pour IE 6.0 et +
     }
     catch (e2) {
       xmlHttp = false;
     }
   }
   if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
     xmlHttp = new XMLHttpRequest();        //Pour Mozilla, Opera, etc...
   }
   return xmlHttp;
}
env.xmlhttp = new getXMLObject(); // Enregistrement de l'objet
```

- 3 grandes étapes
  - Création de l'objet XMLHttpRequest
  - Génération de la requête
  - Gestion de la réponse

Génération de la requête

```
if(env.xmlhttp) {
  env.xmlhttp.open("POST","LoginServlet?",true);
  env.xmlhttp.onreadystatechange  = handleConnect;
  env.xmlhttp.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
  env.xmlhttp.send("login="+logname+"&password="+pass);
}
else{alert("AJAX Probleme : xmlhttp introuvable");}
```

- 3 grandes étapes
  - Création de l'objet XMLHttpRequest
  - Génération de la requête
  - Gestion de la réponse

Gestion de la réponse

```
function handleConnect(){
        if (xmlhttp.readyState == 4) {
             if(xmlhttp.status == 200) {
               var rep=xmlhttp.responseXML.documentElement;
               alert(rep.getElementsByTagName('status')[0].firstChild.data);
              }
             else {
                alert(xmlhttp.status+" Error during AJAX call. Please try again");
             }
        }
}
```

Communication Asynchrone : AJAX

- JQuery propose une fonction $.ajax(obj) simplifiant grandement la mise en œuvre d'AJAX
- Le paramètre obj est un objet contenant les attributs suivants :
  - type : methode GET ou POST
  - data : chaine contenant les données à envoyer (ou objet avec attributs simples), à la manière des informations ajoutées à l'URL par GET
  - dataType : type de données que l'on attend en retour ("text", "json", "xml", "html", "script")
  - error : Fonction à appeler en cas de problème au cours de la communication
  - success : Fonction à appeler en cas de succès de la communication (cette fonction doit attendre un argument qui contiendra les données de réponse formatées selon dataType)
  - url : url de la ressource à interroger (Servlet dans notre cas)

AJAX avec JQuery

```
$.ajax({
    type: "POST",
    url: "AjoutSupContactServlet",
    data:"id="+env.actif+"&id_util="+user.id+"&contact="+((user.contact)?0:1),
    dataType:"json",
    success: function(rep){
            if ((rep.error==undefined) || (rep.error==0)){
                if (user.contact){
                        alert(user.login+" retiré de votre liste de contacts");
                        user.contact=false;
                }
                else{
                     alert(user.login+" ajouté à votre liste de contacts");
                     user.contact=true;
                 }
            }
            else{
                if (rep.error==1){
                        alert("Problème base de données");
                }
                else{
                        alert("Problème serveur");
                }
            }
    },
    error: function(jqXHR, textStatus, errorThrown){
            alert(textStatus);
    }
});
```

Communication Asynchrone : AJAX

- A noter
  - Pas de redirection avec AJAX
    - Avec AJAX, l'émission de la requête et la gestion de la réponse sont réalisées sur la même page
    - AJAX ne gère donc pas les redirections vers d'autres pages
    - ⇒ Se connecter sur connexion.html et obtenir la réponse sur main.html est alors impossible en AJAX
    - ⇒ Utiliser pour cela d'autres techniques (telles que JSP)
  - Upload de fichier difficile
    - Par sécurité, les navigateurs interdisent à Javascript de récupérer le fichier renseigné dans un input de type file
    - ⇒ Utilisation de XMLHttpRequest alors impossible
    - ⇒ Passer par des iframes

Communication Asynchrone

Discussion intéressante sur les avantages / inconvénients de l'utilisation de XMLHttpRequest plutôt que de passer par des iframes :

http://ajax.sys-con.com/node/188390

Communication client-serveur avec React

- Librairie axios :
  `https://www.npmjs.com/package/axios`

Communication synchrone

```
axios.get("url/url-pattern").then(res => {
        const persons = res.data;
        this.setState({ persons });
})

ou
const user = {name: this.state.name};
axios.post("url/url-pattern",{name}).then(res => {
        const persons = res.data;
        this.setState({ persons });
})

ou

axios.delete("url/url-pattern/?name"+{this.props.name}).then(res => {
        const persons = res.data;
        this.setState({ persons });
})
```

Communication asynchrone

```
getUsers = async () => {
    let res = await axios.get("https://reqres.in/ap
    let { data } = await res.data;
    this.setState({ users: data });
};
```

Récupération du JSON par le client

Javascript : Le format JSON

- JSON (JavaScript Object Notation)
  - Format de données textuel
  - Dérivée de la construction littérale d'objets
- Format JSON est composé :
  - d'ensembles de paires nom / valeur ⇒ les objets
  - de listes ordonnées de valeurs ⇒ les tableaux

Javascript : Le format JSON

- Le format JSON = chaine de caractères correspondant à la formation littérale d'un objet

  ⇒ Nécessite de disposer :

  - D'un parser : texte JSON ⇒ objet
  - D'un serializer : objet ⇒ texte JSON

Javascript : Parser du JSON

- Parser : la fonction eval(string)
  - Permet d'interpréter une chaîne de caractères
  - Puisque le format JSON = chaine de construction littérale, *eval*('('+*json_text*+')') construit l'objet correspondant au texte contenu dans *json_text*
- Mais :
  - eval est une fonction générique permettant d'évaluer n'importe quelle chaîne représentant du code Javascript

  ⇒ Problèmes de sécurité car du code nuisible peut être exécuté lors de la transformation du texte JSON

Faille de sécurité

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\":\"json\"}";

var obj=eval("("+json_texte+")");  // construction de l'objet

// JSON tronqué lors du transfert :
json_texte="function(){alert('Hack!')})(";
```

Javascript : Parser du JSON

- Depuis 2009, les navigateurs intègrent un support JSON comportant un fonction *parse(json_text, reviver)*
  - json_text : chaîne JSON à transformer
  - reviver (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la construction de l'objet

  ⇒ Méthode spécifique n'interprétant pas d'autre code qu'une chaîne de construction JSON

JSON.parse

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\":\"json\"}";

// construction de l'objet
var obj=JSON.parse(json_texte);

Pour un parsing fonctionnant sur n'importe quel navigateur :
var obj = typeof JSON !='undefined' ?
            JSON.parse(json_texte) : eval('('+json_texte+')');

// JSON tronqué lors du transfert :
json_texte="function(){alert('Hack!')})(";

// Affiche "SyntaxError: JSON.parse: unexpected keyword"
```

Javascript : Serializer en JSON

- Le support JSON comporte égalemment une fonction *stringify(objet, replacer);*
  - objet : objet à transformer en chaîne JSON
  - replacer (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la structure de l'objet pour spécifier un traitement spécial

JSON.stringify

```
//Objet à serializer en JSON
var obj=new Object();
obj.g="1";
obj.f="json";

var json_text=JSON.stringify(obj);
// json_text contient "{\"g\":\"1\",\"f\":\"json\"}"
```

⇒ Attention : JSON.stringify ne serialise pas ce qui est dans le prototype

Exemples utilisation replacer et revival : objet Date()

```
var obj={g:1, r:new Date()};
obj; // Affiche "({g:1, r:(new Date(1329482734849))})" sur la console

json_text = JSON.stringify(obj, function (key, value) {
                return this[key] instanceof Date ? 'Date(' + this[key]
});

obj2=JSON.parse(json_text, function (key, value) {
                var d;
                if (typeof value === 'string' &&
                   value.slice(0, 5) === 'Date(' &&
                   value.slice(-1) === ')') {
                        d = new Date(value.slice(5, -1));
                        if (d) {
                                return d;
                        }
                }
                return value;
});

obj2; Affiche "({g:1, r:(new Date(1329482734000))})" sur la console We
```

Javascript : JSON

Pour s'assurer que JSON est bien pris en charge par le navigateur :

```
<!-- Au cas ou pas JSON sur browser -->
<script type="text/javascript" src=
    "https://github.com/douglascrockford/JSON-js/blob/master/json2.js"
</script>
```