Sommaire

1.1. Présentation générale

AJAX (*Asynchronous JavaScript And XML*) est un ensemble de technologies permettant de construire des applications Web de nouvelle génération, comparables en fonctionnalités aux applications natives Windows ou Mac OS. Inventé en 2005 par *Jesse J. Garrett*, AJAX a été popularisé par des sites innovant tels que Google Suggest, Google Maps, les webmails de Google et Yahoo, writely, iRows ou encore netvibes. Ajax est devenu au fil du temps, un terme générique qui recouvre désormais plusieurs types d'interactions HTTP (HyperText Transfert Protocol) entre le JavaScript et un serveur Web. AJAX n'est pas en lui même une technologie, mais un ensemble de technologies existantes, combinées de façon nouvelle.

1.1.1. Les objectifs poursuivis par AJAX

Les buts recherchés par AJAX sont la diminution des temps de latence, l'apport de nouvelles fonctionnalités et l'augmentation de la réactivité de l'application Web. Les applications Web qui possèdent ces qualités sont appelées Rich Internet Application (RIA). Ainsi AJAX permet aux développeurs des pages Web de :

Mettre à jour ou "rafraîchir" (une partie de) la page sans changer de page

En fait, dans une application Web classique, lorsque l'utilisateur clique sur un lien ou valide un formulaire, le navigateur envoie une requête au serveur HTTP, qui lui retourne en réponse une nouvelle page, qui remplace purement et simplement la page courante, comme l'illustre la figure 1.1

Dans la plupart des sites Web, les pages ont généralement des parties communes. Il s'agit notamment des liens vers les actions possibles sur le site, lesquels conservent le contexte et permettent à l'utilisateur de savoir où il en est et d'accéder rapidement aux informations ou aux actions possibles. Ces liens forment une sorte de menu, situé le plus souvent en haut ou à gauche des pages. Par exemple, pour un site de commerce, le panier de l'utilisateur est visible sur toutes les pages ou est accessible depuis un lien (souvent sous la forme d'une icône représentant un Caddie).

Avec le Web classique, ces parties communes sont envoyées avec chaque réponse HTTP. Lorsque le volume de données des éléments communs est faible par rapport à celle des élé-

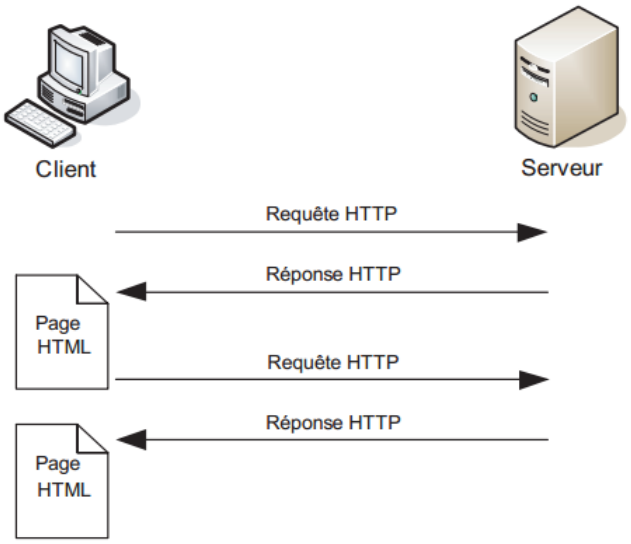

FIGURE 1.1 – Communication Client/Serveur en Web classique.

ments propres à la page, c'est sans grande conséquence. Dans le cas contraire, ce fonctionnement consomme inutilement une partie de la bande passante et ralentit l'application. Par exemple, si l'utilisateur modifie la quantité d'un article dans son panier, seuls deux ou trois petites portions de la page devraient être mises à jour : la quantité (un champ de saisie) et le nouveau montant de la ligne et du total. Le rapport est dans ce cas totalement disproportionné.

Avec AJAX, il est possible de ne mettre à jour qu'une partie de la page, comme l'illustre la figure 1.2. Les requêtes HTTP sont envoyées par une instruction JavaScript en réaction à une action de l'utilisateur. La réponse HTTP est également récupérée par JavaScript, qui peut dés lors mettre à jour la page courante, grâce à DOM et aux CSS, qui constituent ce qu'on appelle le HTML dynamique. Plusieurs requêtes peuvent ainsi être émises depuis une même page, laquelle se met à jour partiellement à chaque réponse. Le cas extrême est constitué par une application réduite à une seule page, toutes les requêtes étant émises en AJAX.

Dans l'exemple précédent, lorsque l'utilisateur change la quantité d'un produit dans son panier, une requête HTTP est envoyée par JavaScript. À réception de la réponse, seules les trois zones concernées sont mises à jour. Le volume transitant sur le réseau est ainsi réduit (drastiquement dans cet exemple), de même que le travail demandé au serveur, qui n'a plus à reconstruire toute la page. La communication peut dés lors être plus rapide.

Communiquer de manière asynchrone avec le serveur

Avec AJAX, la communication avec le serveur via JavaScript peut être asynchrone. La requête est envoyée au serveur sans attendre la réponse, le traitement à effectuer à la réception de celle-ci étant spécifié auparavant. JavaScript se charge d'exécuter ce traitement quand la réponse arrive. L'utilisateur peut de la sorte continuer à interagir avec l'application, sans être bloqué par l'attente de la réponse, contrairement au Web classique. Cette caractéristique est aussi importante que la mise à jour partielle des pages.

Pour un site de e-commerce par exemple, si l'ajout d'un produit au panier est réalisé en AJAX, l'utilisateur peut ajouter un premier article, puis un second, sans devoir attendre que le premier soit effectivement ajouté. Si la mise à jour des quantités achetés est également

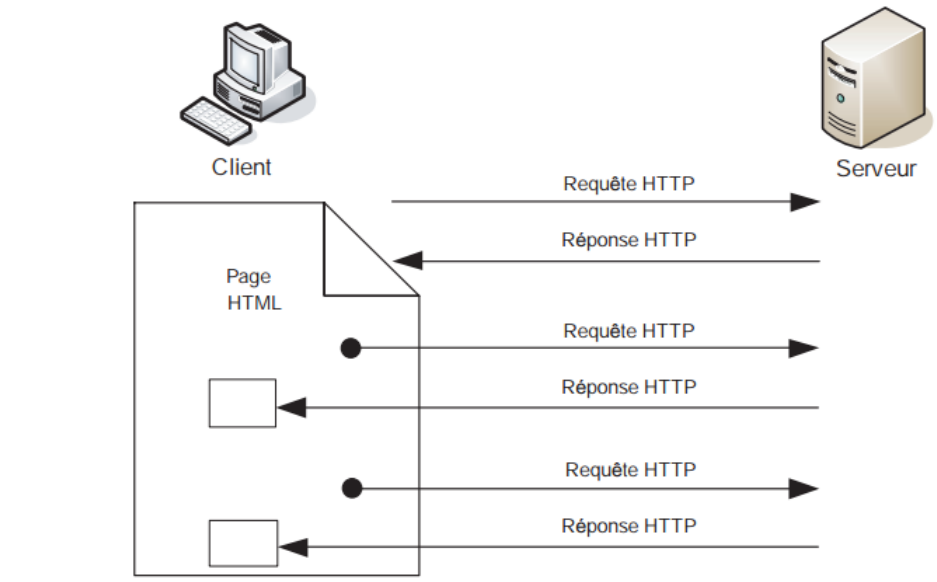

FIGURE 1.2 – Communication Client/Serveur avec AJAX.

réalisée en AJAX, l'utilisateur peut, sur la même page, mettre à jour cette quantité et continuer à ajouter des articles (ou à en enlever).

Les requêtes peuvent ainsi se chevaucher dans le temps, comme l'illustre la figure 1.3. Les applications gagnent ainsi en rapidité et réactivité.

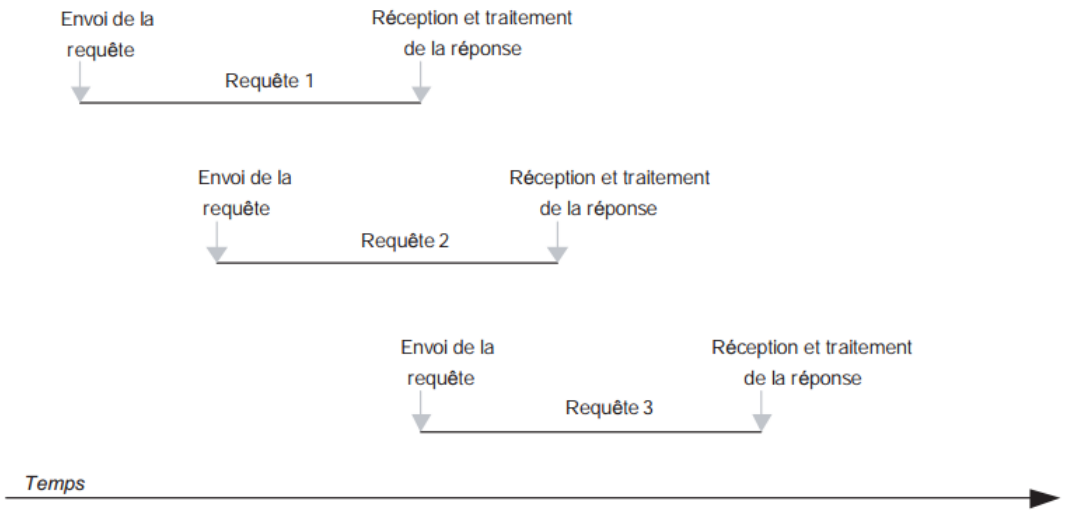

FIGURE 1.3 – Exécution parallèle des requêtes asynchrones par AJAX.

Le parallélisme des requêtes HTTP en AJAX est très utile dans la plupart des cas. Il exige cependant parfois de prendre certaines précautions afin d'éviter de possibles incohérences. Avant d'utiliser des requêtes asynchrones, il convient donc de vérifier qu'elles n'induisent aucun problème fonctionnel.

Obtenir des effets similaires à ceux des applications "Desktop"

L'utilisation de la mise à jour partielle et de la communication via des requêtes asynchrones sur une page Web peuvent permettre d'obtenir sur celle-ci, des effets similaires à ceux des applications natives Windows ou Mac Os.

1.1.2. Quelques exemples pour lesquels AJAX est utile

AJAX est adapté à certains besoin mais pas à d'autres. Pour certaines situations il se révèle utile, voire précieux. Nous présentons ici certaines de ces situations.

Validation et mise à jour d'une partie de la page

Nous avons déjà illustré cette situation avec notre exemple précédent. AJAX se révèle utile pour une telle situation car l'application va plus vite puisque le nombre de requêtes et la taille de la réponse est réduit et le déroulement est plus fluide pour l'utilisateur.

Tri, filtrage et réorganisation de données côté client

Il est très fréquent d'afficher à l'utilisateur les résultats d'une recherche sous forme de tableau, en lui permettant de trier les résultats par un clic sur une colonne, comme dans l'explorateur de Windows, les logiciels de messagerie et bien d'autres applications. Il est aussi parfois utile de lui permettre de filtrer les résultats suivant tel ou tel critère.

Il arrive assez fréquemment que ces résultats ne dépassent pas la centaine ou un volume de l'ordre de 50 Ko, par exemple quand les données correspondent à une sous-catégorie telle que salariés d'un département dans une application de ressources humaines, produits d'un rayon dans une gestion de stock, etc. Comme les données sont sur le client, ce serait du gâchis que de lancer chaque fois une requête au serveur pour réaliser les opérations de tri, de filtrage et/ou de réorganisation. Il est possible de proposer une technique de tri et/ou de filtrage instantané basé sur l'aspect XML de AJAX.

Aide à la saisie

Il est fréquent de rencontrer sur le Web des champs de saisi (dans les formulaires de recherche par exemple) pour lesquels : lorsque l'utilisateur commence à saisir une expression, à chaque caractère saisi, l'application interroge le serveur de façon asynchrone, récupère les expressions les plus demandées commençant par l'expression saisie et les affiche sous forme de liste. L'utilisateur peut dés lors sélectionner l'une d'elles, qui se place dans le champ de saisie.

Autres situations intéressantes

- **La production des diaporamas et autres applications documentaires** ;
- **Le chargement progressif de données volumineuses** (Facebook, Google Plus, Twitter) ;
- **L'édition WYSIWYG (What You See Is What You Get) de documents** dans laquelle AJAX est nécessaire pour la sauvegarde ;
- **L'édition coopérative WYSIWIS (What You See Is What I See) de documents** dans laquelle AJAX est utile pour la réplication et la synchronisation des éditions des différents co-auteurs.

1.2. Mise en oeuvre de AJAX

Le mécanisme AJAX est mis en oeuvre grâce aux technologies JavaScript et XML (eXtensible Markup Language) :

— JavaScript est utilisé pour écouter les évènements déclenchés par l'internaute sur le DOM (Document Object Model) [1] puis, réagir en envoyant une requête (asynchrone ou non) au serveur et mettre à jour le DOM lorsque c'est nécessaire.
— XML est utilisé comme format d'échange entre le client et le serveur. En réalité, il est possible d'utiliser d'autres formats pour l'échange des données entre les clients et les serveurs en AJAX : parmi ces formats, les plus courants sont le HTML et le JSON (JavaScript Object Notation).

En JavaScript, la nouveauté est l'utilisation d'un objet dédié pour l'envoi des requêtes HTTP aux serveurs et l'échange du code XML : il s'agit de l'objet `XMLHttpRequest` (`XHR`). L'objet `XHR` est embarqué dans presque tous les navigateurs récents bien que son comportement varie en fonction de ceux-ci.

Comme vous vous doutez certainement, pour utiliser AJAX il faut créer en JavaScript un objet `XHR`.

1.3. Création d'un objet XMLHttpRequest

Le code suivant permet de créer un objet `XHR` dans une fonction JavaScript :

```
1  function ajax(){
2      if(window.XMLHttpRequest){
3          var xhr = new XMLHttpRequest();
4          // Envoi d'une requête et traitement de la réponse
5      }else{
6          if(window.ActiveXObject){ // Pour Internet Explorer
7              var xhr = new ActiveXObject("Microsoft.XMLHTTP");
8              // Envoi d'une requête et traitement de la réponse
9          }else{
10             alert("Votre navigateur ne peut pas envoyer des requêtes HTTP via XHR");
11         }
12     }
13     return false;
14 }
```

Comme vous pouvez le remarquer, sous Internet Explorer (ligne 6), l'objet `XHR` est présent sous la forme d'un `ActiveX` dénommé `Microsoft.XMLHTTP` (ligne 7).

1.4. Les propriétés et méthodes d'un objet XMLHttpRequest

Le tableau ci-dessous récapitule les propriétés et les méthodes de l'objet `XHR`.

1. Rappel : le DOM est la représentation abstraite d'un document HTML manipulée par le navigateur.

PROPRIETE OU METHODE	DESCRIPTION
open("méthode","url",flag)	Ouvre la connexion avec le serveur. **méthode** -> "GET" ou "POST" **url** -> l'url à laquelle on va envoyer notre requête. Si la méthode est GET, on met les paramètres dans l'url. **flag** -> true si l'on veut un dialogue asynchrone, sinon, false
setRequestHeader("nom","valeur")	Assigne une valeur à un header HTTP qui sera envoyé lors de la requête. Par exemple, pour un POST : nom -> "Content-Type" valeur -> "application/x-www-form-urlencoded"
send("params")	Envoi la requête au serveur. Si la méthode est GET, on met **null** en paramètre. Si la méthode est POST, on met les paramètres a envoyer, sous la forme : "nomparam1=valeurparam1&nomparam2=valeurparam2"
abort()	Abandonne la requête.
onreadystatechange	Ici, on va lui affecter une fonction a nous qui sera exécutée à chaque "changement d'état" de notre objet.
readyState	C'est cette propriété qu'on va tester dans le onreadystatechange. Elle représente l'état de l'objet et peut prendre plusieurs valeurs : 0 -> Non initialisé. 1 -> Ouverture (open() vient de s'exécuter). 2 -> Envoyé (send() vient de s'exécuter). 3 -> En cours (des données sont en train d'arriver).
	4 -> Prêt (toutes les données sont chargées).
status	Le code de la réponse du serveur. 200 -> OK. 404 -> Page non trouvée. ...
statusText	Le message associé à status.
responseText	La réponse retournée par le serveur, au format texte.
responseXML	La réponse retournée par le serveur, au format dom XML.

1.5. Un exemple concret

Pour cet exemple nous supposons que nous disposons d'un dossier `Ajax` contenant les fichiers `ajax.js` (fichier JavaScript mettant en oeuvre l'envoi des requêtes via `XHR`), `index.html`, `response.txt`, `response.json` et `response.xml`. Les fichiers `response.xxx` nous permette de simuler différents formats de réponses qu'un serveur peut fournir. Les contenus des fichiers sont les suivants :

1.5.1. Les fichiers ressources

Contenu du fichier index.html

```
1  <!doctype html>
2  <html>
3      <head>
4          <meta http-equiv="content-type" content="text/html; charset=utf-8" />
5          <title>Mise en oeuvre de AJAX</title>
6          <script type="text/javascript" src="ajax.js"></script>
7      </head>
8      <body>
9          <a href="#" onclick="return ajax();">Charger Response</a>
10         <div id="response">
11
12         </div>
13     </body>
14 </html>
```

Il s'agit là d'une page HTML simple qui inclut le fichier JavaScript `ajax.js` (ligne 6). Dans le corps de la page il y'a un lien sur lequel, l'évènement `click` déclenche l'exécution d'une fonction JavaScript dénommée `ajax` (ligne 9). Nous avons aussi préparé une zone (lignes 10-12) pour recevoir du contenu.

Contenu du fichier response.txt

```
1  Hello World!
```

Contenu du fichier response.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <response>
3      <message>
4          <id>1</id>
5          <exp>Philippe</exp>
6          <contenu>Bla bla bla</contenu>
7      </message>
8      <message>
9          <id>2</id>
10         <exp>Laurent</exp>
11         <contenu>Blo blo blo</contenu>
12     </message>
13 </response>
```

Ce code XML représente une liste de messages renvoyés par le serveur. Pour chaque message on a son identifiant, son expéditeur et son contenu.

Contenu du fichier response.json

```
[
    {
        id: 1,
        exp: "Philippe",
        contenu: "Bla bla bla"
    },
    {
        id: 2,
        exp: "Laurent",
        contenu: "Blo blo blo"
    }
]
```

Ce code JSON est en quelques sortes l'équivalent du code XML précédent. On peut constater que le format JSON a le mérite d'être simple et plus compact que le XML. Cependant, il est moins expressif.

1.5.2. Chargement du contenu du fichier response.txt à l'aide de AJAX

Nous nous intéressons à présent au contenu du fichier `ajax.js`. Nous allons y définir la fonction `ajax` suivante :

```
function ajax(){
    if(window.XMLHttpRequest){
        var xhr = new XMLHttpRequest();
        xhr.open("get", "response.txt", true);
        xhr.onreadystatechange = function(){
            if(xhr.readyState == 4 && xhr.status == 200){
                document.querySelector("#response").innerHTML = xhr.responseText;
            }
        }
        xhr.send(null);
    }else{
        if(window.ActiveXObject){
            var xhr = new ActiveXObject("Microsoft.XMLHTTP");
            // Reprendre les lignes 4 à 10
        }else{
            alert("Votre navigateur ne peut pas envoyer des requêtes HTTP via XHR");
        }
    }
    return false;
}
```

Cette fonction :

1. Crée un objet `XHR` ;
2. Initialise une requête destinée à récupérer la ressource `response.txt` par la méthode `GET` et de manière asynchrone (ligne 4) ;
3. Dans les lignes 5 à 9, nous définissons la fonction de callback (celle qui sera appelée à chaque fois que l'état de la requête changera). Nous utilisons le concept des fonctions anonymes. La fonction anonyme que nous définissons vérifie si la requête est complète (`xhr.readyState` vaut alors `4`) et si elle s'est achevée avec succès (`xhr.status` vaut alors `200`). Le cas échéant, elle récupère la réponse (`xhr.responseText`) qu'elle

insère dans le DOM de la page HTML comme contenu HTML de l'élément ayant `response` comme valeur de l'attribut `id`.

4. Enfin, la fonction envoie la requête au serveur grâce à la méthode `send` de l'objet `XHR`.

1.5.3. Chargement du contenu du fichier response.json à l'aide de AJAX

Nous proposons une version de la fonction `ajax` qui traite les résultats sous le format JSON :

```
1  function ajax(){
2      if(window.XMLHttpRequest){
3          var xhr = new XMLHttpRequest();
4          xhr.open("get", "response.json", true);
5          xhr.onreadystatechange = function(){
6              if(xhr.readyState == 4 && xhr.status == 200){
7                  var arrMess = eval(xhr.responseText);
8                  var firstMess = arrMess[0];
9                  document.getElementById("response").innerHTML = firstMess.exp;
10             }
11         }
12         xhr.send(null);
13     }else{
14         if(window.ActiveXObject){
15             var xhr = new ActiveXObject("Microsoft.XMLHTTP");
16             // Reprendre les lignes 4 à 12
17         }else{
18             alert("Votre navigateur ne peut pas envoyer des requêtes HTTP via XHR");
19         }
20     }
21     return false;
22 }
```

En fait, nous nous servons de la fonction `eval` pour convertir la chaine de caractère JSON retournée par le serveur en un objet JavaScript (ligne 7). L'objet étant un tableau, nous récupérons son premier élément avec le code de la ligne 8. Cet élément est aussi un objet ayant les champs `id`, `exp` et `contenu` comme présenté dans le fichier `response.json`. Nous mettons ensuite le DOM à jour en y insérant le nom de l'expéditeur du premier message.

1.5.4. Chargement du contenu du fichier response.xml à l'aide de AJAX

Lorsque le format de réponse retourné par le serveur est le XML, la réponse du serveur se trouve dans la propriété `responseXML` de l'objet `XHR`. Nous avons le code suivant :

```
1  function ajax(){
2      if(window.XMLHttpRequest){
3          var xhr = new XMLHttpRequest();
4          xhr.open("get", "response.xml", true);
5          xhr.onreadystatechange = function(){
6              if(xhr.readyState == 4 && xhr.status == 200){
7                  var xmlResp = xhr.responseXML;
8                  var firstExp = xmlResp.getElementsByTagName("exp").item(0);
9                  var data = firstExp.firstChild.data;
```

```
10                  document.getElementById("response").innerHTML = data;
11              }
12          }
13          xhr.send(null);
14      }else{
15          if(window.ActiveXObject){
16              var xhr = new ActiveXObject("Microsoft.XMLHTTP");
17              // Reprendre les lignes 4 à 13
18          }else{
19              alert("Votre navigateur ne peut pas envoyer des requêtes HTTP via XHR");
20          }
21      }
22      return false;
23  }
```

1.6. Synthèse

L'utilisation de `XMLHttpRequest` permet d'améliorer l'interactivité entre notre page HTML et l'utilisateur. Il faut toujours prévoir une solution alternative à AJAX vu que son utilisation dépend fortement de JavaScript qui peut être désactivé chez certains clients.

Faire de l'AJAX pour en faire ne sert à rien, il faut que ça apporte vraiment un plus pour l'utilisation de votre site ou de votre application.

Une autre façon de bien se servir de AJAX est l'utilisation des frameworks JavaScript tels que JQuery, Rico et Prototype.