Architecture des échanges de données sur le *web*

Yannis Delmas

Architecture des échanges de données sur le *web*

Yannis Delmas

Table des matières

Chapitre 1. Avant-propos

1. Objectifs du cours

Les webmestres et chefs de projet *web* doivent souvent arbitrer entre diverses solutions techniques, ils doivent établir des cahiers des charges pour des réalisations de logiciels externalisés. Ce cours vise à présenter les principaux critères techniques de choix. Il présente ensuite les principales données techniques de l'hébergement *web* qui permettent d'effectuer certains choix stratégiques dans la conception éditoriale d'un site *web*.

2. Prérequis

Les connaissances et compétences suivantes sont prérequises pour ce cours.

- Navigation *web* experte. Utilisation avancée d'un moteur de recherche.
- Utilisation d'un site de vente en ligne avec personnalisation. Utilisation d'un environnement numérique de travail. Utilisation d'un bureau virtuel (webmail au minimum).
- Conception d'une page *web* statique. Notions de programmation côté client (cf. infra pour une définition).
- Notions de fonctionnement d'une application en réseau. Connaissance du fonctionnement général d'Internet.

3. Statut de ce document

Ce document n'est qu'un support de cours, il ne prétend pas être autonome. Le cours correspondant est délivré dans le cadre du mastère *Web éditorial* de l'Université de Poitiers [http://www.univ-poitiers.fr/]. Ce document a été conçu au format DocBook [http://docbook.sourceforge.net/] pour être consulté sous forme de pages web (XHTML, cette version) ou sous forme imprimée (PDF, cette version).

Ce document, destiné aux étudiants du mastère *Web éditorial* de l'Université de Poitiers [http://www.univ-poitiers.fr/], est mis à disposition de tous. Son usage est gratuit dans le cas de formations délivrées par un organisme à but non lucratif (institution gouvernementale, association). En cas de formation dans un cadre à but lucratif ou pour tout autre usage, prendre contact avec l'auteur [http://yannis.delmas-rigoutsos.nom.fr] ou ses ayant-droits.

Chapitre 2. Organisation générale du service *web*

1. Le protocole HTTP

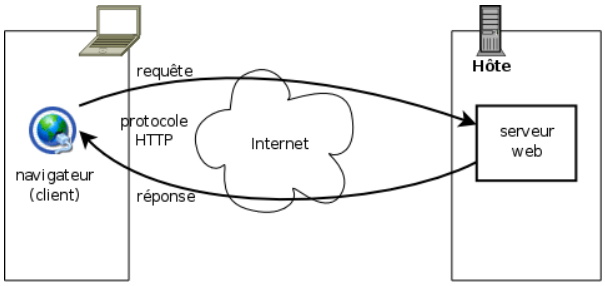
1.1. Architecture client-serveur

Le service web fonctionne selon le mode *client-serveur* pur (Figure 2.1) :

1. Un client se connecte à un serveur.
2. Le client formule une requête HTTP au serveur.
3. Le serveur répond à la requête : soit par un document (page web, image etc.) par tout moyen, soit en indiquant qu'il y a erreur (formulation incorrecte de la requête ou données non disponibles).
4. L'échange reprend à l'étape 2 ou se termine (et peut ensuite reprendre à l'étape 1).

En particulier un serveur ne propose ni ne demande jamais rien à un client, du moins pas directement : il ne peut le faire qu'au moyen d'une réponse à une requête formulé par un client.

Figure 2.1. Communication client-serveur web en HTTP



Ce type d'échange, bien construit, permet une totale transparence au réseau : chaque document peut se trouver n'importe où sur Internet, aussi bien sur la même machine que le client qu'aux antipodes. Dans la plupart des cas, les documents servis étant indépendants (en termes informatiques) les uns des autres, peut importe le serveur qui fournit la donnée. Ainsi, les gros sites tels que celui de Google [http://www.google.fr] correspondent-ils, *via* une adresse unique, à plusieurs (logiciels) serveurs, tournant sur plusieurs (ordinateurs) hôtes différents. Différents éléments d'une même page (HTML, images, CSS) peuvent donc être fournis par plusieurs serveurs différents ; l'assemblage est réalisé par le navigateur.

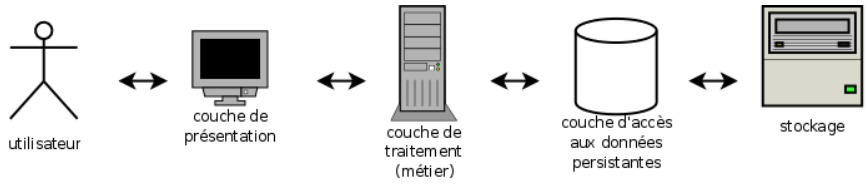
Cette structuration divise le travail de restitution documentaire (au sens le plus large) en deux domaines nettement distincts : le *côté client* et le *côté serveur*. Le client va devoir rendre les pages reçues avec tous leurs éléments incorporés, qui atteignent parfois de grandes complexité (voir, par exemple, Globz.com [http://www.globz.com/]). Il n'aura à sa disposition que certaines ressources de la machine cliente. Le logiciel client fait généralement appel à 1) des possibilités de programmation du client lui-même, susceptibles de modifiers les documents une fois reçus, le plus souvent en JavaScript, 2) d'exécution de programmes incorporés appelés applets, généralement programmés en Java, ou, de plus en plus maintenant, 3) d'appel à des briques logicielles additionnelles (plug-ins) permettant de lire/jouer tel ou tel type de données (Flash, vidéo, MP3 etc.). De leur côté les serveurs de sites professionnels se limitent rarement à la fourniture de pages statiques (documents conçus tels quels par des graphistes ou intégrateurs humains). Ces serveurs (exemple : voyages-sncf.com [http://www.voyages-sncf.com]) mobilisent souvent un grand nombre de ressources complexes afin de produire les pages, dites “dynamiques” demandées par les internautes. D'une façon similaire aux logiciels clients, les serveurs pourront produire ces pages “dynamiques” 1) en utilisant des programmes externes de calcul (CGI-bin), 2) en faisant appel à des briques logicielles additionnelles (plug-ins de serveur, appelées servlet) ou 3) en utilisant un langage de programmation (plus ou moins rudimentaire ou évolué) intimement associé au logiciel serveur (SSI, JSP, PHP, perl, ASP etc.).

Quelle que soit la méthode, les serveurs font généralement appel à d'autres types de ressources que de simples fichiers “statiques”. Dans la quasi-totalité des cas il s'agit de bases de données relationnelles ou d'annuaires.

1.2. Architecture trois-tiers

Avant l'avénement du web, les applications informatiques étaient le plus souvent de deux types : application monolythique (p.ex. traitement de texte, banc de montage vidéo...) ou application client-serveur. Dans ce second cas un client, appelé maintenant « client lourd », se connectait à un serveur, qui centralisait les données et l'essentiel de leur traitement. Ces services étaient, le plus souvent, organisés autour d'une (très grosse) base de données. Rentrent encore dans ce cadre, par exemple, les applications de courrier électronique, associant un logiciel de messagerie (client urd) à des serveurs de boîtes et autres relais de courriers (serveurs).

Figure 2.2. Architecture trois-tiers



Avec le web s'est développé un autre mode d'organisation (antérieur) : l'architecture trois-tiers (c'est à dire “trois acteurs”). Ce mode consiste à séparer le traitement des données en trois couches “superposées” bien distinctes (Figure 2.2) :

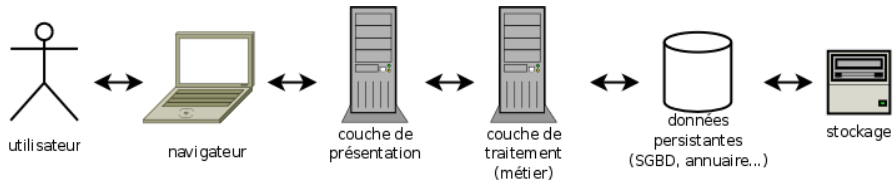
- La couche de *présentation* interagit avec l'utilisateur : elle reçoit ses demandes et lui fournit les informations sous une forme compréhensible par lui (au travers d'une *interface* - IHM).
- La couche de *traitement* des données met en oeuvre ce qui relève du “métier” ou, plus généralement, de la fonction des données. Par exemple, pour une réservation de train, il s'agit de relever les places disponibles, les préférences du voyageur, son paiement, etc. et, de là, effectuer la réservation proprement dite : fournir un billet, l'envoyer, marquer une place comme occupée, etc.
- La couche d'*accès aux données* persistantes s'occupe d'entreposer les données pérenne sous une forme utilisable par les traitements ultérieurs. Il s'agit dans la plupart des cas d'un serveur gestionnaire de base de données (*SGBD*).

Chaque couche n'interagit, dans un tel modèle, qu'avec les couches adjacentes, le plus souvent sur un modèle client-serveur. Ceci fait qu'une couche supérieure (à gauche sur la figure) peut, pour une même opération, interagir avec plusieurs agents ou interagir de nombreuses fois avec un même agent de la couche immédiatement inférieure (à droite sur la figure).

Cette architecture assure d'abord une plus grande modularité : les modifications d'une application se font le plus souvent couche par couche. En particulier l'interface (IHM) n'est pas intriquée aux traitements proprement dits (ou, en tout cas, le moins possible).

Cela permet, en outre, de répartir la charge d'un service sur plus de machines. En particulier, elle permet de rendre accessible un même ensemble de donnée à une population beaucoup plus étendue. Une telle organisation est impérative au delà d'une certaine charge. Pour les services les plus importants, on détaille même parfois plus que trois couches (*multi-tiers* ou *n-tiers*).

Figure 2.3. Architecture n-tiers web



Dans le cas du *web* (Figure 2.3), le découpage en couches est souvent un peu différent. La couche de présentation se décompose en deux couches bien distinctes : le rendu de l'interface, réalisé au moyen d'un simple navigateur, et le calcul de l'interface et de la présentation des données, généralement réalisé par un logiciel implanté sur le serveur *web*. Les pages *web* qui réalisent cette interface, ou par extension le navigateur, sont appelés *client léger*. Ceci est léger pour trois raisons : cela permet une très grande souplesse, en particulier d'interopérabilité (ne dépend pas ou peu du système choisi par l'utilisateur), cela demande l'échange de moins de données et cela n'impose pas d'installer sur la machine de l'utilisateur un quelconque logiciel (avec toutes les difficultés de compatibilité et de maintenance que cela présente).

Toutefois, précisons que pour la plupart des applications *web* courantes, la couche de présentation proprement dite et la couche de traitement sont encore largement intriquées, en pratique.

1.3. *Web services* et AJAX

Ces deux grands types d'architecture subissent actuellement deux grandes évolutions : l'architecture en *web services* et la programmation AJAX.

Les *web services* [services *web*/services “maillés”] sont un nouveau mode de conception (d'urbanisation) des plus grands systèmes d'information. Ils viennent répondre à la difficulté de conception et surtout d'évolution des plus grosses applications métiers, dont le fonctionnement ne peut plus être intégralement perçu par un analyste. Cette approche consiste à ne plus concevoir le traitement des données comme un monstre monolythique mais comme un ensemble d'agents, beaucoup plus petits et plus spécialisés qui s'appellent les uns les autres d'une façon très similaire à la relation entre un client et un serveur *web*. Ces différents agents peuvent alors être conçus de façon modulaire : un développeur donné se concentrera sur la programmation d'un ensemble particulier d'agent sans avoir à se soucier des interactions avec la totalité de l'application.

Ces services s'échangent le plus souvent leurs données dans un format XML et utilisent le protocole HTTP (selon nombre de protocoles distincts : SOAP, WDSL, XML-RPC, etc.). Ceci justifie le terme « *web* » et permet de passer les pare-feu qui ne manquent souvent pas puisque puisque ces services permettent d'organisent des échanges d'informations B-to-B (business-to-business : principalement relations entre entités internes et relations fournisseurs-commanditaire), ou B-to-C (business-to-client). Aujourd'hui, le plus souvent, tous ces agents intéragissent en temps réel, de façon synchrone.

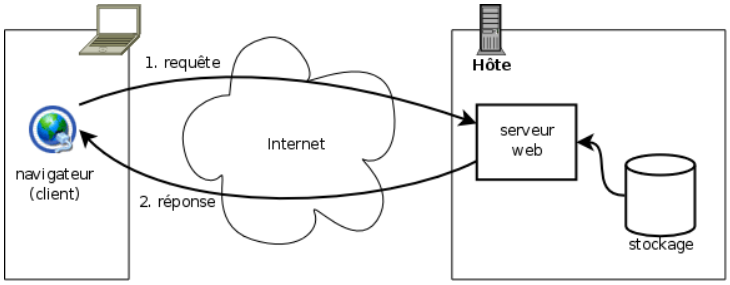
Le deuxième type d'évolution, AJAX, introduit en 2005, touche aujourd'hui tous les sites et services *web* de conception récente. Ce mode de programmation consiste à ne plus se contenter, pour l'interaction entre l'utilisateur et une application informatique centralisée, d'échanger des pages. Cette opération est, en effet, trop lourde quand il s'agit seulement de mettre à jour une liste déroulante ou de proposer une aide à la saisie. AJAX signifie « *Asynchronous Javascript And XML* » [« XML et Javascript asynchrone »]. L'idée est de faire en sorte qu'un programme Javascript implanté dans une page web, côté client, va pouvoir lui-même agir comme un client en demandant des données XML à une certaine URI (sur un mode parent des *web services*). Cette opération est asynchrone au sens où elle n'interomp pas, par ailleurs, le fonctionnement de la page *web*, qui continue à réagir aux interactions habituelles. Il s'agit, le plus souvent de donner des compléments d'information à l'utilisateur, de faire de l'aide à la saisie, de présenter hiérarchiquement de très gros volumes de données ou, simplement, d'ajouter des gadgets à une page web.

2. Mise en oeuvre d'une réponse HTTP

2.1. Les serveurs web délèguent

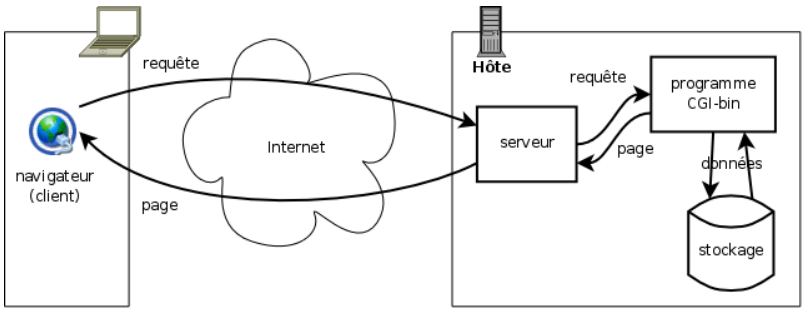
Pour aller plus loin dans l'utilisation des possibilités du web nous devons maintenant aller plus loin dans le processus de traitement d'une requête HTTP.

Figure 2.4. Traitement (simple) d'une requête de page statique



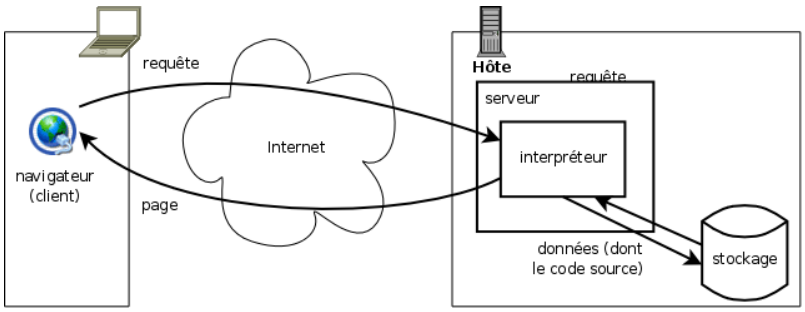
Quand une page statique, ou plus généralement un document statique, est demandé au serveur, celui-ci se contente, pour l'essentiel d'interpréter l'URL et de fournir le fichier demandé au client en précisant son type MIME (Figure 2.4).

Figure 2.5. Traitement d'une requête avec traitement externe



Toutefois, dès 1993 est apparu le besoin de faire calculer dynamiquement des pages pour fournir des services interactifs (et non plus simplement de publication) à ceux qu'on n'appelait pas encore les internautes. On trouva alors une solution assez simple (Figure 2.5) : utiliser des programmes de traitement en respectant un protocole d'interaction avec le serveur web appelé CGI (Common gateway inteface) [interface de passerelle généraliste]. Les programmes ainsi utilisés au travers de l'interface CGI sont appelés *programmes CGI* ou *CGI-bin*. Le principe de fonctionnement de CGI est très simple : le serveur fournit au programme CGI la requête HTTP telle qu'il l'a reçue sur un port de communication appelé son « entrée standard » (ainsi que quelques autres informations, cf. infra), ensuite le programme CGI renvoie sur un port appelé « sortie standard » la réponse HTTP à renvoyer au client (type MIME et document proprement dit, principalement - cf. détails ci-dessous).

Figure 2.6. Traitement d'une requête avec traitement interne



Ce dispositif, à l'usage, s'est avéré trop consommateur de ressources pour les serveurs, les programmes de traitement étant sans cesse lancés puis arrêtés. On a alors développé des modules de production de pages intégrés aux serveurs sous forme de plug-ins ou de module interne. Un cas particulier très fréquent de tels modules est celui des interpréteurs de scripts (PHP, ASP etc.) qui permettent d'effectuer de nombreux traitement de manière légère et aisée à programmer (Figure 2.6). Cette facilité de développement, en particulier pour accéder à des bases de données, doublée d'une très grande flexibilité, a rendu ces langages interprétés extrêmement populaires.

2.2. Lecture d'une URI

Dans le cadre de cette organisation du travail rappelons les différentes parties d'une URI et quoi est destiné à qui.

La forme générale des URI [RFC2396 [http://www.faqs.org/rfcs/rfc2396.html]] est la suivante (les crochets désignent une partie facultative) :

```
[http[s]://hôte[:port]][/chemin][?requête][#fragment]
```

Ces différentes parties se lisent ainsi :

- `http` / `https` : C'est le protocole employé par le client pour contacter le serveur. Il est connu des deux.
- Hôte (et numéro de port) : Cette information est transmise au serveur afin qu'il sache quel site est demandé. En effet plusieurs sites web peuvent être mutualisés sur un même serveur.
- Chemin : Il s'agit en général du chemin d'un fichier sous la racine web du site demandé. Toutefois, dans le cas des pages dynamiques, le “chemin” peut être en réalité une URI virtuelle communiquée à un programme pour calculer un contenu sans que celui-ci corresponde à un fichier. C'est presque toujours le cas des URI associées à un gestionnaire de contenu (WCMS, Wiki, blog, ENT, etc.). Cette partie de l'URI n'est pas sensée être utilisée par les programmes côté client.
- Requête : Informations transmises par le client aux programmes côté serveur pour calculer une réponse dynamique. Ces informations sont présentes directement dans une hyperréférence (un lien) ou dans un formulaire employant la méthode GET. Cette partie de l'URI n'est pas utilisée côté client.
- Nom de fragment : Cette partie (à partir du dièse) ne fait pas à proprement parler de l'URI et n'est pas transmise au serveur. Il s'agit d'un nom de fragment à l'intérieur du document désigné par l'URI proprement dite. En XHTML, ce nom est donné à l'aide de l'attibut `id`. Elle est utilisée par les navigateurs pour positionner l'affichage autant que possible au niveau du début du fragment en question (p.ex. pour une table des matières).

Théoriquement [RFC2616 [http://www.faqs.org/rfcs/rfc2616.html]] taille des URI utilisable dans une requête HTTP n'a pas de limite. En pratique, du fait de la limitation de certains serveurs et clients, il est recommandé de se limiter à 255 caractères. On limitera donc l'usage de la méthode GET pour les formulaires aux cas de tests sur de petits volumes de données. Pour tous les autres cas, on emploiera la méthode POST (décrite ci-après) qui permet de joindre à une requête HTTP toute information utile.

Mentionnons également une autre source d'information, sur laquelle nous reviendrons plus en détail plus bas : les cookies, qui servent principalement à identifier un client particulier auprès d'un serveur ou d'un ensemble de serveur, en particulier quand il s'agit de mettre en oeuvre une session de travail (une interaction longue demandant identification de l'utilisateur).

3. Pages “dynamiques”

3.1. Les technologies en présence

Les logiciels côté client et côté serveur n'ont pas accès, par principe même, aux ressources de l'autre côté. De plus leurs rapports, plus ou moins intimes, avec le client ou le serveur, leur donnent ou non accès aux ressources de ces applications. Voici un résumé synoptique des possibilités de chacun.

	côté client		**côté serveur**	
	langage intégré	**incrustation**	**langage intégré**	**plug-in / module**
technologie	DHTML : Javascript (+CSS, +DOM)	flash, applet Java, activeX	PHP, ASP, JSP (java interprété)	CGI, servlet Java
programme	script intégré à la page (reçue du serveur)	fichier autonome (envoyé par le	fichier de script permettant au serveur de produire une page	programme exécutable indépendant

	côté client		**côté serveur**	
	langage intégré	**incrustation**	**langage intégré**	**plug-in / module**
		serveur au navigateur)		
exécution	interprété par le client	joué par un plug-in du client	interprété par le serveur (ou un module/plug-in)	exécutable autonome
produit	animation d'une page du navigateur (en général, comportements en réponse à des événements)	animation/ interactivité de l'applet ou “incrustation”(limitée à celle-ci)	création d'une page dynamique renvoyée par le serveur au client	
domaine d'action	Applis limitées à ce qui concerne le client, l’utilisateur, sa machine. Mais ça peut donner, même en DHTML, des choses très complexes (ex.: outils bureautiques de Google).		Interface de n’importe quelle appli client-serveur (trois-tiers, client léger).	
ressources utilisables issues du client	objets du client : pages, fenêtres, cookies, historique, navigateur	objets de l’“incrustation”, *une* fonction javascript, modèle du navigateur	peu, ce que le navigateur veut bien transmettre : URI demandée, données de formulaire, cookies pertinents, URI page précédente, modèle du navigateur, langues du lecteur	
ressources utilisables côté serveur	rien directement (pour avoir des données il faut demander une nouvelle URI ou utiliser AJAX)		certains objets du serveur, fichiers sur la machine serveuse, données issues de tout serveur accessible	fichiers sur la machine serveuse, données issues de tout serveur accessible

3.2. Choisir une technologie côté client

Voici les techologies existant aujourd'hui, avec leurs avantages et inconvénients.

DHTML : programmation Javascript/ECMAscript

De très loin le plus courant. Peut être très standard si elle s'appuie sur les méthodes DOM pour la modification de la structure HTML et sur des feuilles de styles pour les changements d'aspect. À l'aide de ces méthodes on peut (presque) tout faire en terme d'interface (IHM). Les méthodes AJAX apportent désormais des méthode permettant d'assurer le même confort d'utilisation que sur des clients “lourds”. Pour cette raison, les autres méthodes tendent à disparaître.

Si ces méthodes sont employées : portable (fonctionne partout), développement rapide, fiable, sûr.

Sinon : développement facile mais ni fiable ni portable : ne fonctionne pas sur tous les navigateurs (aujourd'hui ou demain).

DHTML à l'aide de VBscript

N'est, pour ainsi dire, plus utilisé que par Microsoft pour Windows.

Animation *flash*

Avantages : très graphique (la mode est passée, mais encore nécessaire pour certains sites), très fiable, sûr, développement rapide

Inconvénients : peu accessible (aux humains et aux robots des moteurs) : copier-coller limité, impression impossible, etc.

Applets Java

Avantages : toutes les possibilités d'un client lourd, fiable, sûr

	Inconvénients : haute technicité (demande de vrais informaticiens développeurs) Technologie en fin de vie (au profit du DHTML).
Contrôle ActiveX	Plug-in de navigateur utilisable comme un simple objet incrusté dans une page. Avantages : a accès à toutes les fonctionnalités de Windows Inconvénients : risqué (accès à toutes les fonctionnalités de Windows) même si une méthode de signature des contrôles ActiveX vise à contenir ces risques, limité au système Windows, complexe/coûteux en développement

3.3. Choisir une technologie côté serveur

Voici les *principales* technologies, avec *quelques* éléments de choix statégique. Le jeu est beaucoup plus ouvert que pour les technologies côté client.

PHP	Extrêmement populaire. Ceci demande donc d'être très vigilant sur les failles de sécurité, vite exploitées. Il s'agit d'un langage interprêté par le serveur mais il est possible de mettre en place un dispositif de compilation à la volée pour améliorer les performances. Avantages : portable, extrêmement flexible, développement rapide (particulièrement pour les bases de données et formulaires), support important (du fait de la popularité) Inconvénients : moins performant que du compilé (mais voir ci-dessus), développements peu fiables s'ils ne sont pas conduits par de très bons développeurs
Servlet Java	Technologie réservée aux plus gros services. C'est le standard des grands systèmes d'information professionnels. Le standard de référence pour les briques d'ENT (environnements numériques de travail). Avantages : très flexible, fiable, rapide Inconvénients : demande une infrastructure spécifique, gourmand en ressources, complexe/coûteux en développement
ASP	Méthode interprêtée principalement utilisée sur les serveurs Microsoft. Le langage de script est principalement le VBscript mais le Jscript (Javascript de Microsoft) et le perl sont également possibles (encore que très peu utilisés). Cette méthode est sourtout adaptée aux “routards” de Windows, qui auront peu à apprendre pour effectuer des développements simples. Avantages : flexible, développement rapide Inconvénients : pas portable (spécifique Microsoft), moins performant que du compilé, souvent peu fiable (car souvent pas produit par des programmeurs spécialistes du web, qui se tournent plus vers Java et PHP)
JSP	Méthode interprêtée similaire à PHP et ASP mais utilisant le langage Java.
Exécutable compilé CGI	A aujourd'hui largement disparu au profit de PHP et des servlets Java. Avantages : extrêment flexible (programmé dans n'importe quel langage compilable : C, C++, Pascal etc.), *peut* être très portable, très performant (vitesse d'exécution)

	Inconvénients : complexe/coûteux en développement, *peut* être très peu portable, peut poser des problèmes de sécurité, délai au démarrage, peut être très gourmand en ressources
Exécutable interprèté CGI	Le plus souvent dans les langages Perl ou Python, mais tous les langages exécutables en terminal texte sont possibles. Réservé à quelques niches historiques.
ActiveX	Cette technologie existe également côté serveur, mais elle est extrêmement peu utilisée (si elle l'est encore). Dans sa forme actuelle, n'est adaptée qu'à des niches technologiques.
SSI (server-side include)	Langage de pré-traitement extêmement rudimentaire. Ancètre des PHP, ASP et autres JSP. A, de fait, disparu.

Chapitre 3. HTTP et organisation des contenus

1. Introduction

Les requêtes et les réponses HTTP ont toutes deux la même forme générale : une série de méta-informations envoyées en entête suivie d'un corps de message. Dans le cas des réponses, le corps de message est le document transmis lui-même, éventuellement encodé. Pour avoir une idée de l'en-tête de requête envoyé par un client à un serveur on peut utiliser le petit code PHP suivant :

```
foreach(apache_request_headers() as $nom => $valeur) {
    echo "$nom:\t$valeur\n";
}
```

Pour connaître l'en-tête de réponse renvoyé par un serveur pour une page donnée, le plus simple est d'utiliser l'extension « web developper » de Firefox et de sélectionner dans le menu : « information > view response headers ».

Voici un exemple d'en-tête et un exemple de réponse HTTP, que nous allons ensuite analyser, point par point, pour ce qui concerne un chef de projet web.

Exemple (fictif) d'entête de requête HTTP :

```
GET /cours_prog_web/test-entete.php?a=b&c=d+e HTTP/1.1
Host: yannis.delmas-rigoutsos.nom.fr
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.8) Gecko/20061201 Firefox/2.0.0.8 (Ubuntu-fe
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0
Accept-Charset: UTF-8,*
Accept-Encoding: gzip,deflate
Accept-Language: fr,fr-fr;q=0.9,en;q=0.8,en-us;q=0.6,el;q=0.5,de;q=0.4,it;q=0.3,nl;q=0.1
Connection: keep-alive
Keep-Alive: 300
Cookie: session=2348f5a1c0b11ddcb3250fc7c10f0944
Referer: http://yannis.delmas-rigoutsos.nom.fr/documents/index.html
```

Exemple (fictif) d'entête de réponse HTTP :

```
HTTP/1.1 200 OK
Date: Thu, 22 Nov 2007 12:35:27 GMT
Server: Apache
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
Content-Type: text/html; encoding=UTF-8
Content-Encoding: gzip
Content-Length: 11222
X-Powered-By: PHP/5.1.6-1
Set-Cookie: session=2348f5a1c0b11ddcb3250fc7c10f0944
```

2. La requête HTTP proprement dite

Le protocole du web [RFC2616 [http://www.faqs.org/rfcs/rfc2616.html]] définit trois principales méthodes de requêtes HTTP, c'est à dire trois principaux types de requêtes HTTP. Ces requêtes permettent aux logiciels clients web (généralement des navigateurs) de commander des opérations aux logiciels serveurs web.

La méthode la plus courante est la méthode *GET*. Elle demande simplement à récupérer le contenu associé à une URI. Les données transmises sont dans l'URI même. Les autres informations de la requête ne sont donc que les cookies et la configuration du client. Comme on le voit sur l'exemple ci-dessus, l'URI est décomposée : d'une part le serveur (si le port était indiqué, il serait à la suite dans la ligne `Host`), d'autre part la suite de l'URI.

La méthode *POST* sert principalement pour les formulaires. En effet l'URI est peu commode pour transmettre des informations nombreuses et/ou volumineuses : tous les clients et serveurs ne supportent pas des URI trop longues

et, surtout, cela peut gêner l'utilisateur, s'il souhaite se repérer ou enregistrer la page. En revanche, les informations envoyées par la méthode POST en dehors de l'URI ne seront pas enregistrées dans un signet. La méthode POST envoie les données en les plaçant dans le corps de la requête HTTP, à la suite de l'entête.

La troisième méthode usuelle, *HEAD*, n'est pas directement accessible aux utilisateurs. Fonctionnellement, il s'agit d'une version de la requête GET qui doit engendrer les mêmes types de traitements, hormis la production du document lui-même. Le serveur se contentera donc de renvoyer un entête de réponse, sans contenu. Ceci permet aux caches et proxies, notamment, de tester la présence, la date et le type des documents, plus généralement, de recevoir toutes les méta-informations les concernant.

Ces trois méthodes fournissent l'essentiel des méthodes utilisées par le web. Il en existe toutefois d'autres, notamment une extension du protocole HTTP, appelée WebDAV (*Web-based Distributed Authoring and Versioning*) ou DAV permettant d'utiliser HTTP pour servir des fichiers, en lecture et écriture. La grande nouveauté de ce système, qui a vocation à supplanter un jour le FTP, est de garder trace de versions successives d'un document et, à terme, de permettre le travail simultané de plusieurs utilisateurs sur un même document. Aujourd'hui, malgré les affirmation des éditeurs la plupart des principales plateformes DAV ne sont pas encore irréprochables. DAV est très bien adapté au stockage de documents pour un environnement numérique de travail (il est en place, par exemple, sur l'ENT de l'Université de Poitiers).

3. La langue d'interaction

```
Accept-Language: fr,fr-fr;q=0.9,en;q=0.8,en-us;q=0.6,el;q=0.5,de;q=0.4,it;q=0.3,nl;q=0.1
```

Le premier élément d'entête auquel nous nous intéresseront est la langue. On voit, dans l'exemple de requête ci-dessus, que les navigateur transmettent systématiquement aux serveurs la liste des langues indiquées par l'utilisateur dans la configuration du navigateur par ordre décroissant de préférence (pour Firefox 2 : édition > préférences > avancées > général > langues).

Ceci veut dire que les serveurs web peuvent utiliser cette indication pour s'adapter aux préférences de l'utilisateur, pour les pages statiques comme pour les pages produites dynamiquement.

Il est extrêment décevant de voir que cette fonctionnalité très simple à mettre en oeuvre et source d'un grand confort pour l'utilisateur est encore fort peu utilisée par les éditeurs de sites web, qui laissent le “soin” à l'utilisateur de cliquer sur un icone de langue, dans le meilleur des cas (mais plus souvent un icone de pays, ce qui est parfois gênant). On comparera avec le grand confort proposé, par exemple, par Google, qui offre une interface dans la langue préférée, quel que soit le pays ciblé (p.ex. sur www.google.gr [http://www.google.gr]).

Quand on utilise le serveur web Apache l'aiguillage linguistique automatique est extrêmement simple à mettre en oeuvre. Par exemple, si les pages `index.html.en` et `index.html.fr` existent, l'une de ces deux pages sera fournie au client quand il demandera `index.html` (à condition qu'il n'existe pas de fichier portant ce nom, bien entendu).

Ceci veut également dire qu'il est possible (et même simple) de faire des statistiques pour suivre la langue principale des visiteurs d'un site. Ceci peut être extrêmement instructif quand il s'agit de décider dans quelles langues doit être édité le site. Pour une étude plus fine il est, toutefois, conseillé de s'appuyer également sur des statistiques géographiques (cf. infra).

4. L'encodage des caractères, du document, de la transmission

```
Accept-Charset: UTF-8,*
```

Pour que le contenu textuel d'une page soit lisible, il est essentiel qu'il soit établi dans un jeu de caractères lisible sur l'ordinateur des visiteurs. Ceci demande le concours de plusieurs éléments.

- Le serveur et le client doivent s'entendre sur le codage des caractères employé dans la page. Pour cette raison le client envoie une indication `Accept-Charset` listant ses possibilités. Aujourd'hui tous les navigateurs reconnaissent UTF-8 l'encodage le plus courrant du jeu de caractères Unicode, qui inclut potentiellement toutes les formes d'écriture (présentes et une bonne partie des écritures passées). Dans notre exemple le navigateur indique en outre `*`, ce qui suggère qu'il peut se débrouiller pour comprendre tous les encodages de caractères.

- Le système doit être capable d'afficher tous les caractères employés dans la page. En effet, ce n'est pas parce que l'encodage est reconnu par le navigateur que le système de l'utilisateur dispose du dessin de tous les caractères que cet encodage décrit. Par exemple, si l'on se connecte sur un site chinois (ex.: www.google.cn [http://www.google.cn]) et que les fontes indiquées par la feuille de style de la page ne comprennent pas ces caractères, l'affichage sera très dégradé (généralement les caractères manquants sont remplacés par des points d'interrogation).
- La page doit être effectivement dans l'encodage qu'elle indique. Il n'est pas rare de voir des pages mélanger allègrement plusieurs encodages.

Le respect des encodages est absolument essentiel quand il s'agit de pages mettant en oeuvre une application, web, en particulier les WCMS. En effet, si l'encodage de la page n'est pas parfait, l'envoi des données de formulaire, donc également les données ensuite sauvegardées dans l'application risque d'être incorrect.

```
Accept: text/xml,application/xml,application/xhtml+xml,text/html,text/plain,image/png,*/*
Accept: image/gif, image/x-xbitmap, image/jpeg, application/x-shockwave-flash, */*
```

Autre type d'encodage, celui du document : dans quel format est-il composé ? Là encore le navigateur doit commencer par indiquer les formats qu'il est capable de comprendre. Les format compris sont mis sous la forme de types MIME. Dans le premier exemple ci-dessus, le client indique qu'il comprend XML, XHTML, HTML, le texte brut, les images PNG et qu'il essaiera de se debrouiller pour tout autre format. En pratique les formats textuels sont souvent sous-entendus, comme dans le second exemple, et le navigateur précise seulement des formats d'images. Internet explorer indique parfois aussi, quand la suite MSOffice est installée, les types `application/vnd.ms-powerpoint`, `application/vnd.ms-excel` et `application/msword`.

```
Content-Type: text/html; encoding=UTF-8
Content-Type: text/html
```

Toute réponse HTTP comprend, au minimum, l'indication du format de son contenu. Le plus souvent les serveurs web déterminent le format d'un fichier statique en observant son extension (`.html`, `.png` etc.). Pour cette raison, il est absolument essentiel que cette extension soit correcte, sinon l'affichage sur le navigateur a toute chance d'être désastreux. Quand il s'agit d'un document textuel, l'encodage des caractères est parfois précisé, comme dans le premier exemple ci-dessus.

```
Accept-Encoding: gzip,deflate
```

Observons enfin dans l'entête de requête l'indication `Accept-Encoding`. Il ne s'agit plus là du codage d'un document à proprement parler mais de l'encodage de la transmission du document. Dans cet exemple le navigateur indique qu'il comprend les encodage `gzip` et `deflate`, qui sont deux algorithmes de compression de données (le premier est surtout utilisé sous Unix et le second est utilisé par le format populaire ZIP), en plus de la transmission simple qui consiste envoyer le fichier tel quel. Quand il le peut le serveur utilise ces possibilités de compression pour réduire sa bande passante de sortie (au prix d'une charge de calcul, bien sûr). Ceci permet parfois de gagner aussi en rapidité, quand le temps de compression/décompression est inférieur au gain de temps de transmission (surtout utiles pour les gros fichiers qui se compressent bien et les connexions bas-débit). Bien entendu, le fichier est décompressé à l'arrivé par le navigateur pour être affiché correctement à l'utilisateur.

5. Le suivi des utilisateurs

Outre les données de configuration, le client transmet au serveur des informations qu'il stocke de façon plus ou moins permanente : les *cookies* [biscuits].

Le web étant mise en oeuvre au moyen d'un protocole purement client-serveur utilisant des requêtes séparées les unes des autres : il ne permet pas, en soi, le travail suivi sur un ensemble de données au sein d'une application, ce qu'on appelle une *session de travail*. Il y a deux moyens de contourner cet écueil : Le premier est de passer comme un relais, de page en page, les informations pertinentes sous la forme de champs de formulaire (éventuellement cachés). Une variante consiste à réécrire toutes les URI internes de la page pour leur ajouter des informations à retransmettre au serveur à la prochaine demande de page. Ces méthodes sont assez efficaces tant que les données sont peu nombreuses et qu'il n'est pas nécessaire de prolonger la session de travail au-delà d'une succession de pages qui s'enchaîne. Elle ne permet pas, en revanche, de personnaliser l'interface ou l'accès à un site, de façon plus permanente. Le second moyen consiste à stocker des informations dans le client et à les renvoyer systématiquement au serveur dans certaines conditions. C'est le principe des *cookies*, définis par Netscape en 1996.

Un *cookie* peut être considéré comme la donnée des informations suivantes (toutes facultatives, sauf le contenu) :

- *Contenu*.- Il est de la forme `nom=valeur`. La valeur ne peut contenir certains caractères (espace, virgule, point-virgule), pour cette raison, elle est souvent codée comme une URL ou sous forme de suite de chiffres et lettres. La limite de ce contenu est de 4 kio.

- *Date de péremption*.- Elle est écrite de la façon suivante : `expires=Wdy, DD-Mon-YYYY HH:MM:SS GMT`. Si aucune date de péremption n'est donnée, le *cookie* disparaît à la fin de l'utilisation du navigateur (fin de session du client). Notons que cette date est exprimée en temps universel, de façon à pouvoir être comprise de la même façon par le client et le serveur, où qu'ils soient. Le jour de la semaine est normalement en anglais et abrégé sur trois lettres, de même que le mois, de la façon standard (jours : `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun` ; mois : `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, `Dec`).

- *Chemin de validité*.- Écrit `path=/debut/de/chemin`, il permet de restreindre l'envoi du *cookie* à certaines URI sur les serveurs de destination. Dans notre exemple, le *cookie* sera transmis à `/debut/de/chemin.html`, à `/debut/de/chemin`, à `/debut/de/cheminement/etc`, mais pas à `/debut/de/autre`. Par défaut, le chemin de validité est `/` : le *cookie* est renvoyé à tous le site.

- *Domaine de validité*.- Écrit `domain=.acme.com`, il indique aux serveur(s) de quel(s) domaine(s) Internet doit être communiqué ce *cookie*. Dans cet exemple il sera renvoyé à `nimportequoi.acme.com` mais pas à `www.xacme.com`. Dans la mesure du possible, il faut essayer d'être précis et éviter les indications trop générales (d'ailleurs interdites par la norme). Quand le domaine de validité n'est pas précisé, il est sous-entendu qu'il s'agit du serveur émetteur.

- *Demande de sécurisation*.- Si le *cookie* comporte l'attribut `secure`, alors il ne sera renvoyé que sur des connexions sécurisées (HTTPS). S'il ne le comporte pas, le cookie pourra être émis aussi bien en HTTP qu'en HTTPS.

Le point important, concernant les *cookies*, est qu'il sont définis et renvoyés dans les entête des requètes et réponses HTTP. En conséquence, quand un *cookie* est proposé par un serveur à un client, celui-ci ne sera disponible que pour les requêtes suivantes. Concrètement, les opérations se déroulent ainsi :

- Demande d'une page par le client (pas encore de *cookie* défini).

- Le serveur renvoie une première page et, simultanément, définit un *cookie* avec l'indication `Set-Cookie` dans l'entête de la réponse HTTP.

- Le client affiche cette première page et, simultanément, enregistre le *cookie*.

- Dans toutes ses demandes suivantes de page (ou n'importe quelle autre URI) le client renvoie le *cookie* en question (sous réserve des mécanisme expliqués ci-dessus) à l'aide de l'indication `Cookie` de l'entête de requête HTTP.

  ...

- Le serveur peut définir un *cookie* supplémentaire, changer la valeur d'un *cookie* ou en supprimer à l'occasion d'une nouvelle réponse HTTP. Il n'a que cette méthode pour le faire.

- Le client transmettra l'ensemble des *cookies* pertinent à chaque demande d'URI. Il change ou ajoute des *cookies* quand il en reçoit l'ordre par le serveur.

Pour observer les *cookies* envoyé à un navigateur on peut configurer ce dernier pour qu'il signale chaque demande de *cookie* (tous les navigateurs ne le permettent pas - ceci est vite pénible tant les *cookies* sont nombreux). Avec Firefox, il est possible d'installer l'extension « view cookies » qui liste tous les cookies concernés par une page donnée (menu « informations sur la page », onglet « cookies »).

Les premiers sites de vente en ligne ont pu utiliser des cookies pour stocker le panier d'un visiteur, toutefois, la méthode a très vite montré ses limites avec l'augmentation du nombre de produits disponibles. Pour implémenter réellement une session de travail d'application web, une méthode plus élaborée s'est vite imposée : le suivi de session. Il consiste à mettre en place le pendant des cookies du côté des serveurs web, sous la forme d'objets appelés *sessions*. Une telle “session” n'est guère plus qu'un fichier ou un jeu d'enregistrements associés à un utilisateur le

temps de sa session de travail sur une application web. Comme ce fichier stocke généralement la quasi-totalité des informations nécessaire, il n'est plus nécessaire de mettre dans les cookies de l'utilisateur qu'un numéro de session. Ce mécanisme est, aujourd'hui, totalement automatisé dans PHP, ce qui permet aux programmeurs d'utiliser des sessions sans s'inquiéter de gérer le fichier de session ou le cookie enregistrant le numéro de session.

Le deuxième type d'usage des cookies est d'enregistrer les préférences de l'utilisateur pour un site donné (observer, par exemple, les cookies de Google et, en particulier, leur évolution quand on visite la page « préférences »). Jusqu'au début des années 2000, ceci était généralement suffisant. Toutefois, depuis, l'usage de la personnalisation des sites s'est considérablement développé, du moins pour les sites les plus dynamiques. Un cookie n'est plus, dans ce cas, suffisant pour stocker toutes les données qui concernent un utilisateur. Sur ces sites, ce qui concerne un utilisateur donné doit être placé dans une base de données : ces sites tendent de plus en plus, dans leurs technologies, à se rapprocher de véritables environnements numériques de travail. Les sites de vente en ligne les plus élaborés relient bien entendu ces données à un logiciel de gestion de la relation client (GRC) : les achats précédents ou les pages lues fournissent de précieuses indications sur les goûts et les intérêts d'un prospect, qui permettent même, à l'idéal, de lui faire des suggestions d'achats.

Pour n'importe quel site web d'ampleur, il est essentiel de bien connaître ses usagers. Un premier niveau consiste à pister anonymement ses visiteurs, en observant les pages (ou suites de pages) visitées. Un simple cookie de session suffit pour cela (au minimum pour ne pas lui asséner dix fois la même publicité). Le niveau suivant consiste à établir des profils de consultation, idéalement en relation avec un logiciel de GRC, et à s'adapter à ces profils (connaître le client est bien, le satisfaire est mieux). L'évolution actuelle est celle de la personnalisation. Le degré le plus simple de personnalisation, passif, est de permettre à l'utilisateur de personnaliser ses service, en particulier de mettre au premier plan ceux qu'il utilise le plus volontiers. Mais il peut également s'agir de méthodes actives de suggestion de services. Par exemple (Amazon), on pourra observer que de nombreux lecteurs de tel ouvrage ont également acheté tel autre ; une méthode active consiste alors à proposer un avertissement, une page, un e-mail, etc. de suggestions d'achat. Ces méthodes révolutionnent aujourd'hui le commerce culturel en donnant accès au plus grand nombre à des ouvrages d'audience restreinte. Elles augmentent ainsi considérablement le maché potentiel de ces produits ; c'est le phénomène de « longue traîne » [*long tail*], en partie au détriment de la vente de masse (cf. cours. E. Leguay).

Chapitre 4. Principaux autres outils

Nous avons évoqué précédemment les principaux outils issus des techniques d'analyse des requêtes HTTP par les serveurs. Nous allons maintenant évoquer d'autres outils stratégiques, plus généralement externes (ou détachés de la simple utilisation du protocole HTTP).

1. Statistiques d'accès au site

Pour administrer les contenus d'un site, il est absolument essentiel d'en connaître les visiteurs. Quand ceux-ci sont nombreux, ceci passe nécessairement par l'utilisation d'outils statistiques.

1.1. Adapter la plateforme technique

Le premier niveau, élémentaire, d'adaptation aux utilisateurs est l'adaptation “volumétrique” à leur nombre. Celui-ci, certes basique, est absolument nécessaire. Les principaux éléments à prendre en compte sont les suivants :

- Le *nombre de requêtes*, en crête et en moyenne. Ceci est nécessaire pour dimensionner le serveur web, en terme de nombre d'accès simultanés en particulier, mais également de nombre de serveurs en attente de connexion. Ce nombre de requêtes se fait en suivant les statistiques “internes” produites par le serveur web lui-même.

- La *bande passante* (en crête), c'est à dire la quantité d'information (le volume de données) devant être émise chaque seconde. Ceci est nécessaire pour dimensionner le raccordement à Internet. Quand l'hébergement est opéré par un grand fournisseur d'hébergement, ceci ne pose généralement pas problème.

- Le *trafic mensuel* (total), c'est à dire la quantité d'information (le volume de données) émise durant chaque mois. En effet, les contrats d'hébergement fixent souvent une limite supérieure. Il faut toujours garder une marge de sécurité. Un événement particulier peut attirer de nombreux visiteurs, qu'il faut pouvoir accueillir sans être "coupés" par l'hébergeur. Ceci doit être régulièrement suivi. En cas d'auto-hébergement, ceci ne pose pas de problème, en soi.

- La *charge du ou des serveurs*. Plus un serveur est sollicité, plus il met de temps à répondre, soit du fait du volume de traitements demandé, soit du fait de la saturation de sa mémoire de travail. Ces deux paramètres doivent être largement dimensionnés par un technicien compétent, le plus souvent d'après les trois informations mentionnées précédemment mais également d'après le type de services à fournir. Un site dynamique (WCMS, p.ex.) demande ainsi plus de puissance de calcul et plus de mémoire centrale qu'un site statique similaire. Un système de *cache* performant permet, à l'inverse, de réduire considérablement la charge de calcul d'un serveur. Les sites web les plus importants devront être publiés à l'aide de plusieurs serveurs utilisant les mêmes données (*redondance*). Dans ce cas, il est généralement utile de faire appel à un professionnel de l'hébergement web. Quand le service web est stratégique pour son éditeur, il peut être également utile de disposer d'une politique de *scalabilité*, permettant d'augmenter la puissance disponible en cas de besoin ponctuel. Tous ces paramètres sont déterminants dans le choix d'une plateforme technique.

Outre l'adaptation volumétrique, il est impératif de s'adapter aux capacités techniques de l'utilisateur. Un premier moyen, simple à mettre en oeuvre, consiste à utiliser l'information `User-Agent` transmise par les clients web, qui précisent leur modèle. Voici quelques exemples :

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.8) Gecko/20061201 Firefox/2.0.0.8 (Ubuntu-fe
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:1.8.1) Gecko/20061010 Firefox/2.0
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
```

Ces indications ne sont pas toujours simples à lire. `Mozilla/n.m` désigne normalement les navigateurs de Netscape, dont les récents Firefox. Le `U` ci-dessus entre parenthèses indique que le client sait utiliser Unicode. Pour une raison étrange, les navigateurs *Internet Explorer* de Microsoft prétendent être des Netscape. Les deux derniers exemples désignent ainsi respectivement un IE 5.5 sous Windows XP et un IE 7 sous Windows Vista.

Les professionnels de la statistique de site web, par exemple Xiti, Médiamétrie ou Google analytics, collectent plus d'informations techniques encore, notamment en utilisant un petit code javascript qui collecte les d'informations à

renvoyer au serveur sur la machine cliente. Parmi ces informations, citons : l'activation de javascript, acceptation de cookies, la résolution d'écran et le nombre de couleurs affichables à l'écran.

1.2. Adapter la langue

Outre les aspects matériels, qui assurent d'être lisible dans de bonnes conditions, il s'agit d'être compréhensible de ses visiteurs. Ceci demande une adaptation linguistique.

Un premier niveau d'analyse consiste à observer les langues préférées des utilisateurs (cf ci-dessus). Ces statistiques peuvent être simplement collectées par le serveur web ou par certaines applications WCMS utilisant cette information. On peut également faire appel à des prestataires extérieurs qui, eux, disposent de bases de données capables d'exploiter l'adresse IP de la machine cliente pour identifier (dans une certaine mesure) la zone géographique des visiteurs.

Attention : adapter la langue ne veut pas seulement dire traduire des contenus. En effet, une culture différente induit une lecture différente, bien au-delà de la langue proprement dite. Il est parfois nécessaire de concevoir des versions d'un site entièrement spécifiques.

1.3. Adapter le contenu

Pour garder un contenu en phase avec ses visiteurs, il est important de suivre leur comportement (statistiquement). Quelles sont les pages qui sont les plus vues parmi les informations fraîches ? parmi les informations à longue durée de vie ? Quelles sont les rubriques les plus fréquentées ? Toutes ces informations existent dans les statistiques d'accès du serveur web.

La réponse (statistique) à ces questions, parmi d'autres, suggère ce qu'il faut privilégier dans le cadre d'une réponse aux attentes du public actuel (mais ceci ne présage, bien sûr, pas d'autres choix stratégiques visant à se développer en direction d'autres publics). Ces réponses permettent également de poser la question de l'ergonomie du site, en particulier de son aborescence : est-elle adaptée à ces préférences générales ? En effet, de nombreux sites ont une arborescence qui reflète plus l'organisation interne de l'éditeur (en tant que société), au lieu d'être conçue comme un outil permettant au visiteur de se repérer efficacement. Pour aller plus loin dans l'élaboration de l'arborescence, il est parfois utile d'utiliser des outils permettant de “pister” l'utilisateur en observant des suites de pages vues au sein du site. Ces outils utilisent des cookies ou l'indication `Referer` de l'entête de requête HTTP, qui indique la source de l'hyperlien ayant conduit à la demande actuelle (s'il ne s'agit pas d'un signet ou d'une entrée manuelle).

Cette même indication Referer est également éminemment utile pour repérer les principaux flux de visiteurs vers un site. Précisément, il s'agit de repérer les sites « afférents » vers un site donnés. Le suivi de ces sites est d'une importance capitale, à plusieurs titres. Le premier est celui de la publicité, qu'elle soit gratuite ou onéreuse : ce suivi permet de mesurer très précisément et en temps-réel son efficacité et, le cas échéant, sa pertinence. Le second est celui de la propagation de notoriété. En effet, un hyperlien ne fait pas que transmettre un flux de visiteurs, il est également suivi par les moteurs qui l'utilisent pour propager une partie de la note de notoriété des sites « afférents » vers le site-cîble. Enfin, ce suivi est important parce que toute anomalie est l'indice de changements dans les liens afférents, qu'il s'agit parfois de corriger vite. N'oublions pas que la politique d'échange ou d'achat de liens d'un site est un élément déterminant de sa fréquentation.

2. Utilisation des moteurs de recherche

Les moteurs de recherche ne sont pas que des outils aux mains des internautes. Il peuvent fournir également des outils extrêmement efficaces à l'éditeur d'un site web.

L'outil le plus simple à disposition consiste à utiliser comme outil de recherche “interne” à son site un grand moteur de recherche. Ceci permet d'une part de profiter de la puissance de ces moteurs et, d'autre part, de vérifier concrètement que l'infrastructure technique du site est compatible avec ce moteur. Cette technologie n'est pas forcément indiquée dans les cas où le site comporte des parties confidentielles mais est bien adaptée aux sites entièrement ouverts au public.

Le deuxième type d'outil consiste à décrire son site d'une manière efficace pour les “araignées” des moteurs de recherche. Le minimum est de disposer d'un fichier décrivant l'accès à ces robots. Pour les sites vivants, il est

aujourd'hui impératif de disposer d'un ou plusieurs *fils RSS*[1] décrivant les nouveautés du site. Ceci est, bien sûr, utile aux robots mais permet surtout aux visiteurs de s'abonner à ces nouveautés et à des sites “amis” d'afficher ces nouveautés sous forme de liens afférents. Enfin, l'idéal est de disposer d'une *carte de site*, décrivant l'ensemble du site pour les robots.

Le troisième type d'outil est beaucoup plus élaboré et s'obtient généralement sous la forme d'une prestation d'une société d'analyse statistique. Il consiste à analyser les mots-clefs utilisés par les visiteurs qui sont arrivés sur le site en provenance d'un moteur de recherche. En effet, les éléments de recherche sont dans l'URI de la page de réponse, donc disponibles dans la donnée `Referer` de l'entête de requête HTTP.

3. Suivi des URL

Le dernier élément sur lequel nous voulons insister est la persistance des URL. Du fait que le web comporte de très nombreuses pages dont le contenu n'est pas, ou pas souvent, remis à jour, il résulte une inertie considérable des URL, dans les signets, dans le web... et par conséquent dans les moteurs de recherche. Pour prendre un exemple parlant continue, à ce jour, à recevoir des notifications d'erreur pour une adresse utilisée pendant une ou deux semaines en 1994 ou 1995.

Tout éditeur de site web doit bien avoir toujours présent à l'esprit ce fait. Il est donc impératif que toute URL qui amène à un site un nombre non négligeable de visiteurs garde reste “active”. Cela signifie que le contenu doit y être maintenu. Si cela venait, un jour à ne plus être souhaitable ou pertinent, il est impératif, alors, de mettre en place une redirection, c'est à dire un dispositif renvoyant les visiteurs de l'URL en question vers une autre adresse, publiant un contenu à jour et pertinent. De telles redirections peuvent être mises en place à titre temporaire ou définitif, selon le besoin. Une redirection définitive demande aux navigateurs de mettre à jour leurs signets automatiquement.

Ces redirections sont également utilisées par les moteurs de recherche. En effet, une notoriété importante d'une URL sur un moteur de recherche fait partie du patrimoine d'un éditeur. Cela constitue même une part très importante de l'actif de certaines sociétés. Il est absolument essentiel de le préserver.

Signalons enfin que quand un même contenu est accessible au moyen de plusieurs URL, il est important que toutes ces URL sauf une soient des redirections, de façon à ne pas “diluer” la notoriété auprès des moteurs de recherche. Une notoriété divisée par deux, par exemple, peut ainsi envoyer de la première page aux oubliettes, même si les robots des moteurs de recherche développent des stratégies d'identification des doublons.

[1] On parle de « fil RSS » ou de « flux RSS » même quand la technologie n'est pas RSS, mais Atom. Cf. cours correspondant.